DOI: 10.1111/cgf.70226

COMPUTER GRAPHICS *forum*
Volume 0 (2025), number 0, e70226

Hierarchical Differentiable Fluid Simulation

Xiangyu Kong,[1,2,3] Arnaud Schoentgen,[3] Damien Rioux-Lavoie,[3] Paul G. Kry[1] and Derek Nowrouzezahrai[1,2,4]

[1]McGill University, Montreal, Canada
xiangyu.kong@mail.mcgill.ca, {paul.kry, derek.nowrouzezahrai}@mcgill.ca
[2]Mila, Quebec AI Institute, Montreal, Canada
[3]Ubisoft Montreal, Montreal, Canada
{arnaud.shoentgen, damien.rioux-lavoie}@ubisoft.com
[4]Canada CIFAR, Toronto, Canada

Abstract
Differentiable simulation is an emerging field that offers a powerful and flexible route to fluid control. In grid-based settings, high memory consumption is a long-standing bottleneck that constrains optimization resolution. We introduce a two-step algorithm that significantly reduces memory usage: our method first optimizes for bulk forces at reduced resolution, then refines local details over sub-domains while maintaining differentiability. In trading runtime for memory, it enables optimization at previously unattainable resolutions. We validate its effectiveness and memory savings on a series of fluid control problems.

Keywords: differentiable fluid simulation, fluid control, fluid simulation

CCS Concepts: • Computing methodologies → Physical simulation;

1. Introduction

Modern fluid simulations are widely used in many industries, for example visual effects, game development, manufacturing and advertising; however, controlling the behaviour of these simulations remains challenging due to the non-linearity of their governing equations.

Fluid control is an inverse problem where the goal is to compute initial simulation states and control parameters, to guide the simulated output to an approximate of a target state. Differentiable simulation [dSA*18, HAL*19, UBF*20] has recently emerged as a promising solution that uses automatic differentiation (AutoDiff) [BPRS18] to automate the computation of target loss gradients (with respect to simulation parameters). These gradients can then be used with gradient-based optimization to solve the inverse problems. This approach is general, supporting many applications in a single framework and avoiding the need to create handcrafted methods.

Despite the advantages, differentiable simulation has high memory consumption requirements: as simulation resolution increases, the memory required to compute gradients grows non-linearly, especially in 3D scenes. While more powerful hardware and/or larger compute farms can be used to distribute this cost, this is impractical for games and visual effects artists who primarily rely on consumer-grade hardware to produce visually pleasing animations. For instance, more than 40GB of VRAM are needed to optimize for control forces for a 256^3 resolution simulation for 40 timesteps as shown in Figure 1. This memory constraint limits the applicability and utility of differentiable simulation in the gaming and visual effects industries. Our work enables scalable differentiable fluid simulation for such large-scale fluid control tasks.

We draw inspiration from domain decomposition and multi-resolution optimization to address this memory bottleneck. Our contributions include the following:

- an iterative differentiable smoke simulation optimization framework that solves control problems with significantly reduced memory consumption at the expense of higher computation time;
- a subgrid method that mimics full-resolution simulation, maintaining differentiability w.r.t. simulation inputs;
- a hierarchical domain subdivision strategy that further reduces memory consumption for gradient computation, with a trade-off in runtime and slight accuracy degradation in some cases; and
- a PYTHON implementation and performance measurements, benchmarked against an existing differentiable simulator [HTK19].

© 2025 The Author(s). *Computer Graphics Forum* published by Eurographics - The European Association for Computer Graphics and John Wiley & Sons Ltd.
This is an open access article under the terms of the Creative Commons Attribution-NonCommercial License, which permits use, distribution and reproduction in any medium, provided the original work is properly cited and is not used for commercial purposes.

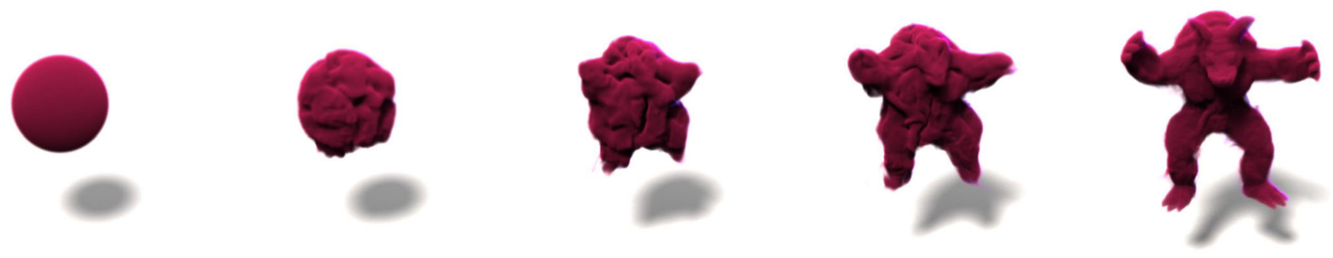

Figure 1: *We optimize space-time forces to drive a grid-based smoke simulation (256^3, 40 frames) to match a 3D armadillo target. We reduce memory consumption by 80% compared to a full resolution baseline with our bulk and iterative subgrid differentiable simulation approach.*

2. Previous Work

We discuss relevant work in fluid simulation and control from the computer graphics literature, before summarizing advances from differentiable simulation and its application to fluid control. We finally review the use of domain decomposition and multi-resolution optimization methods in fluid simulations, since our algorithm draws inspiration from both techniques.

2.1. Fluid Simulation

The field of fluid simulation has enjoyed significant advancements in the past few decades. Lagrangian particle-based methods, such as smoothed particle hydrodynamics (SPH) [MCG03], treat fluids as particle systems and use smoothing kernels to approximate properties carried by each particle. In contrast, Eulerian grid-based methods [FM97b, Sta99] compute the evolution of the fluid quantities at fixed positions in space. Hybrid methods, such as particle in cell (PIC) and fluid implicit particle (FLIP) [Har62, BR86, ZB05], combine the advantages of both representations. In the recent years, new techniques such as flow map based fluids [ZCD*24] and Monte Carlo fluids [RSÖ*22, SBH24] have been explored to introduce new families of methods with distinct sets of properties.

Our work targets the grid-based fluid representation. Specifically, we adapt the stable fluids algorithm [Sta99, Bri15], which is an industry-standard grid-based method renowned for its simplicity and unconditional stability. Our implementation builds on the semi-Lagrangian approach introduced by this algorithm, and in addition, we implement the staggered grid discretization [HW65] to enhance simulation accuracy and prevent null-space problems. While our algorithm leverages this specific implementation, it is readily generalizable to other grid-based fluid simulation methods. For instance, in collocated grids, we can adapt our method to evaluate vector quantities at grid centres instead of at cell face centres.

2.2. Fluid Control

Fluid control is an important yet challenging problem that has countless applications in many fields, ranging from looping animation and keyframe matching in games and visual effects [JWLC23, HTK19, UBF*20], to material shape and property optimization in manufacturing and engineering [MP04, HLM*19]. In the field of computer graphics, fluid control is broadly applied to the problem of target-driven fluid animation [Sch21]. Over the past few decades, extensive advancements have been made in this field. Early work tackles this problem by directly modifying fluid properties such as pressure and velocity [FM97a]. Some later works tackle the problem of target-driven fluid animation by directly computing forces based on the difference between the simulated smoke state and the desired target states [FL04, MM13], and others merge reverse simulations with forward ones to create visually pleasing transitions to achieve the target state [OFEH18, SPM22].

Another popular class of methods that controls the fluid motion is gradient-based optimization methods. Analytical gradients for each fluid operator have been derived [TMPS03] and were later accelerated using the adjoint method [MTPS04]. Recently, further improvements have focused on improving the quality, runtime, and memory consumption of the gradient-based fluid control problems [PM17, TACS21, CLL24]. These methods all rely on the analytically derived gradients for optimization, but a major drawback is that the gradient derivations are tightly coupled with the solver implementation. Any change in the solver requires re-derivation and re-implementation of the gradients, which is a cumbersome and error-prone process. Our method leverages modern AutoDiff frameworks, adopting a differentiable simulation approach that is general and flexible, with gradient computation independent of the underlying solvers.

2.3. Differentiable Simulation

Alongside the recent surge of research in machine learning, the paradigm of differentiable simulation has gained popularity over the past few years. Using automatic differentiation (AutoDiff) [BPRS18], differentiable simulations enable gradient propagation from the output states of physics simulations to the input parameters. These gradients are computed automatically and can be used in gradient-based optimization methods. Moreover, these differentiable simulations can act as modular components that can be easily integrated into machine learning networks and pipelines [JMG*21, dSA*18, LXY*23].

There are many differentiable simulation frameworks. DiffTaichi [HAL*19] accelerates gradient computation using source code transformations and global tracing. PhiFlow [HTK19], specializing in differentiable fluid simulations, supports multiple AutoDiff frameworks including PyTorch [PGM*19], TensorFlow [AAB*16] and Jax [SC20]. While differentiable simulation is generally agnostic to underlying solver implementations, this flexibility comes

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

at the cost of high memory consumption arising from the need to build computation graphs that store operations during solver roll outs [BPRS18]. This memory consumption scales non-linearly with the simulation's degrees of freedom. This behaviour is especially severe for grid-based simulations, where depending on the dimensionality, a linear grid resolution increase results in a quadratic or cubic increase in the degrees of freedom.

Gradient checkpointing [CXZG16] partially addresses this challenge by temporally segmenting the simulation and only constructing computation graphs in each segment; however, it does not address spatial memory constraints and therefore does not scale effectively with simulation resolution. Our work introduces a novel method to tackle this spatiotemporal memory bottleneck, borrowing ideas from domain decomposition and multi-resolution methods.

2.4. Domain Decomposition and Multi-Resolution Optimization

Domain decomposition is a class of methods for solving partial differential equations by dividing larger problems into smaller subproblems, solving each separately before merging local solutions into a global result. Domain decomposition has been successfully applied to fluid simulations in physics [THH21] and graphics [GNS*12, CZY17]. High-performance computing (HPC) and distributed fluid simulations [TAB*96, ZG11, GNS*12, MSQ*18] decompose large simulation domains into sub-domains and solve each sub-domain in parallel on multiple compute units. The subdomains communicate with each other through methods such as halo and ghost boundary cells [ZG11, ZM18] to ensure consistency across domain boundaries. Our method, described in Section 3.2, shares similarities with this strategy; however, these sub-domain communication methods cannot be applied directly to control problems or differentiable simulations, where the gradient computations between the subdomains are undefined due to the discontinuities. Although there has been some exploration of domain decomposition in fluid control [YC17], little prior work addresses these challenges in differentiable simulations. We propose an iterative scheme to resolve this limitation.

Hierarchical multi-resolution methods are another class of approaches commonly used for large-scale optimization problems [KY00, RWA21, XZ23]. This class of methods solve the problem progressively from coarse to fine resolution. In fluid simulations, multi-grid methods are commonly used to accelerate convergence by iteratively computing residual solutions across different resolution levels, using high-frequency local solutions to refine low-frequency global solutions [MST10, PM17].

We are inspired by domain decomposition and multi-resolution optimization. Our approach begins with a coarse optimization, which captures the global fluid motion. It is then followed by a series of iterative sub-domain optimizations to refine the local forces.

3. Method

We target inviscid fluid flows for simplicity, which are governed by the incompressible inviscid Navier–Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\rho^{-1}\nabla p + \mathbf{f} \, , \nabla \cdot \mathbf{u} = 0 \, , \frac{\partial s}{\partial t} + \mathbf{u} \cdot \nabla s = 0, \quad (1)$$

where $\mathbf{u}$ is the velocity, s is the smoke density that gets passively advected along with $\mathbf{u}$, p is the pressure used to satisfy the incompressibility constraint, $\mathbf{f}$ are time-varying external forces applied to the fluid body, and ρ is the fluid density. To properly define the fluid behaviour, we also need to enforce conditions at the domain boundaries as well as initial conditions. When discretized, the simulation takes place over a $n_x \times n_y$ grid and over T time steps. The process of computing the simulation solution state from simulation parameters is referred to as *forward simulation*.

In contrast, a typical fluid control problem involves finding simulation parameters so that solving Equation (1) achieves a user-defined goal, such as a target solution state. For this reason, these types of fluid control problems are referred to as *inverse simulation*, taking solution states to simulations parameters.

To demonstrate our method, we will focus on a subset of fluid control problems called keyframing, although our method generalizes to other types of inverse simulation problems such as optimizing for airfoil shapes and lift in engineering, where instead of optimizing a series of forces, the obstacle itself is optimized. Given a set $\mathcal{K}$ of user-provided keyframes s^{t*} where $t \in \mathcal{K}$, we wish to find a series of time-varying dense force fields $\mathbf{f} \in \mathbb{R}^{n_x \times n_y \times T}$, such that the simulated density fields s^t are as close to s^{t*} as possible.

We can formulate this problem as optimizing parameters $\mathbf{f}$ to minimize an objective Φ (i.e. $\text{argmin}_{\mathbf{f}}\, \Phi(\mathbf{f}, s)$; [CLL24]) with

$$\Phi(\mathbf{f}, s) = \frac{1}{N_K} \sum_{t \in \mathcal{K}} \left\| s^t - s^{t*} \right\|^2 + \frac{w_f}{N_T} \sum_{t=0}^{T} \left\| \mathbf{f}^t \right\|^2 , \quad (2)$$

where the first term minimizes the discrepancy, measured by the mean squared error, between the optimized and target smoke densities at each keyframe t, and the second regularization term prevents excessive control forces from being applied to the simulation. Here, N_K and N_T are normalization terms, and the coefficient w_f is user-defined. In practice, we found that $w_f = 0.1$ leads to a good balance between visual result and optimized force magnitude.

Our work relies on the use of gradient-based optimization methods to minimize this objective function. More specifically, we apply a differentiable simulation approach by using AutoDiff to automatically compute gradients through the construction of the computation graph. While this method is agnostic to the underlying fluid solver used, naively using this approach comes with severe drawbacks. Indeed, to compute the gradient of the objective function with respect to the forces using AutoDiff, the computation graph records all the computations encountered during the simulation. In grid-based simulations, the number of parameters and computations required increases not only linearly as the number of time steps increases, but also quadratically (or cubically in 3D) as the grid resolution increases. These factors contribute to the exponential scaling in the size of the computation graph. For this reason, the high memory consumption is a major bottleneck of differentiable fluid simulation and limits the resolution of the control problem.

To address this problem, we take inspiration from domain decomposition and multi-resolution methods. To do this, we split

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

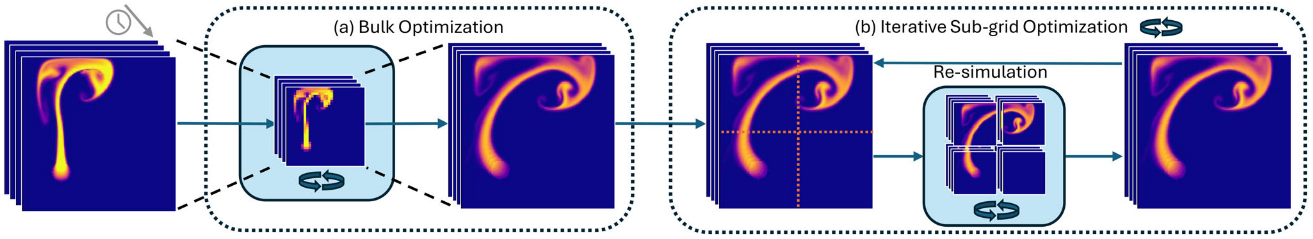

Figure 2: *Our pipeline consists of two main components. First, we perform (a) bulk optimization of control forces on down-sampled simulation trajectory states, and then up-sample and re-simulate the result. We then perform (b) iterative subgrid optimization. Control forces for each subgrid are optimized and then replaced into the full grid and re-simulated to propagate the optimized information to other quadrants over iterations. The optimize-and-re-simulate process is repeated until the resulting full grid converges.*

the optimization into two parts, bulk optimization and subgrid optimizations, as shown in Figure 2. The bulk optimization part operates on a down-sampled resolution to optimize the general bulk motion of the fluid, whereas the subgrid optimization builds on top of it to refine local details. Both of these optimization steps work at a lower resolution than the original problem and hence reduce the memory required to build the computation graph used by AutoDiff. We present our derivation in *2D* and for *inviscid flows*. Our method naturally extends to 3D, and we demonstrate 3D experiments in Section 4.8. Our method focuses on simulations within a closed domain with slip boundary conditions and no obstacles. Further effects could be incorporated into the equations of motion of the fluid, such as viscosity; however, for simplicity, we discuss only inviscid flows in this work.

3.1. Bulk Optimization

The bulk optimization phase is based on the following insight: we can decouple the solution forces $\mathbf{f}^{\star}$ that minimize the objective function Φ into bulk and fine details, which correspond to low- and high-resolution components. Thus, with the bulk optimization process, we aim to optimize a low-resolution force field that drives the fluid's bulk motion to match the keyframes.

At the beginning of the bulk optimization phase, both the keyframe targets and the simulation initial states are down-sampled by a factor k into $(n_x/k) \times (n_y/k)$ by averaging the neighbouring values. These down-sampled keyframes and initial states are then used for optimizing the down-sampled forces. The initial guess of the optimized forces is set to a constant zero field.

For each optimization iteration, the simulation is first rolled out using the down-sampled initial states and forces. Then, the objective function Φ is evaluated using the down-sampled keyframes and the simulated states. Using AutoDiff, the force gradients are computed by back-propagating through the computation graph containing all operations used for simulation and loss evaluations. With a optimization step size α_{bulk} defined by the user, this gradient is then used to update the forces. This process is repeated for a user-defined amount of epochs N_{bulk}. After the optimization converges, we up-sample the forces to the full resolution using bi-linear interpolation to obtain the *bulk* forces $\hat{\mathbf{f}}$ at resolution $n_x \times n_y$.

In practice, we find the optimal value of k depends on the simulation resolution. Larger factors will result in faster runtime and lower memory consumption. If the original simulation resolution is too large, larger k is also beneficial in that it leads to a higher quality optimized bulk motion due to the reduced degrees of freedom; however, if the original resolution is already small enough, higher k leads to lower-quality results after the forces are up-sampled.

As a final step in the bulk optimization, we compute a full-resolution simulation using $\hat{\mathbf{f}}$ to obtain the *bulk* smoke $\hat{s}$, velocity $\hat{\mathbf{u}}$ and pressure $\hat{p}$. This step is necessary as simple up-sampling of these fields would not be physically consistent with the up-sampled bulk force due to the non-linearity of advection and pressure projection. Note that because of this non-linearity and this re-simulation step which is not in the optimization loop, the bulk states will not perfectly match the full-resolution target keyframes; however, building on top of the bulk low-frequency global information, we can optimize for the remaining high-frequency local forces using subgrid optimization.

We note that the final forward simulation computations, while full-resolution, do not contribute to the memory consumption for building the computation graph. For this reason, the memory consumption for bulk optimization is $1/k^d$ compared to the full resolution, where d is the dimension.

3.2. Subgrid Simulation

As we mentioned in the last section, the bulk forces guide the general motion of the fluid to be close to the keyframes. To further improve the results, all while maintaining the low memory consumption, we divide the high-resolution grid into subgrids for a second optimization pass. Our method supports hierarchical division in a recursive manner. We will describe the base case in this section and Section 3.3, and the recursive hierarchical case in Section 3.4.

At the base level, our method divides the bulk grids into 2×2 *valid subgrids* $\bar{s}$, $\bar{\mathbf{u}}$ and $\bar{p}$, corresponding to smoke, velocity and pressure subgrids, each of size $\frac{n_x}{2} \times \frac{n_y}{2}$. Our method can be generalized to $m \times m$ subgrids for $m \geq 2$, and all of the methods described in the following sections still apply; however, in our experiments, we did not find that necessary, especially due to the hierarchical division described in Section 3.4.

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

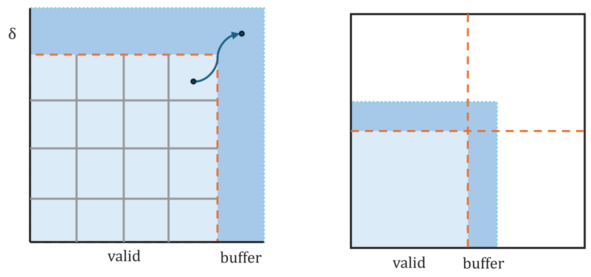

Figure 3: *During advection, back-tracing may exceed interior boundaries and land outside of the valid region (left). To handle this case, we use a buffer region of width δ to ensure the back-trace lands at a region with known values (right).*

3.2.1. *Advection*

The first non-trivial step in the subgrid forward simulation is the advection operator, which encapsulates the phenomenon of quantities being carried through the velocity field. Our advection operator builds upon the widely-used semi-Lagrangian advection scheme [Sta99], with specific handling for cases where the back-traced position computed during advection lands outside of the simulated subgrids.

In a full-resolution simulation, if the back-traced positions exceed the simulation domain, depending on the scene settings, there are different boundary treatments such as nearest-neighbour, reflection or periodic schemes [Sta99, Bri15]. However, in subgrid simulations, if the back-traced positions exceed the interior boundary, as shown in Figure 3 left, the location lies within a neighbouring subgrid, and all of these existing extrapolation treatments will result in inaccurate simulation results. To handle these interior boundaries more robustly, we extract this information from the bulk simulations during subgrid division, as it is available from the forward simulation up-sampling step and does not suffer from extrapolation error.

We extend each valid, divided subgrid with a buffer region, denoted s^δ and $\mathbf{u}^\delta$ correspondingly. Figure 3 right illustrates the valid and buffer region for the bottom-left subgrid. The width of the buffer region δ can be estimated using the bulk velocity as $\delta = \lceil(\max \hat{\mathbf{u}} \times \Delta t)/\Delta x\rceil$, for time step Δt and grid resolution Δx. During subgrid advection, the values inside the valid regions are back-traced and updated, while the buffer regions are read-only and provide information when the back-trace exceeds the interior boundary, as shown in Figure 3 left.

In practice, we use the CFL condition [CFL28] to determine Δt and Δx. To avoid overly large velocity magnitudes that cause high δ, the penalty term in Equation (2) is used to regulate the force magnitude. Throughout our experiments, we did not observe buffer regions of size more than 10% of the full resolution.

3.2.2. *Pressure Projection*

In fluid simulations, the pressure projection step ensures fluid incompressibility and adherence to boundary conditions. For a full-resolution simulation, this step computes pressure using global velocity divergence and boundary conditions, resulting in a divergence-free velocity field after applying the pressure gradient.

Dividing the simulation domain into subgrids, however, creates a loss of global information and introduces unrealistic interior boundaries. To more closely match the subgrid simulation to the full-resolution setting, the ideal interior boundary condition should capture as much global information as possible. As in Section 3.2.1, existing boundary treatments fall short: Neumann interior boundaries impeded fluid flow between subgrids, and free-flow interior boundaries fail to encode information about velocities outside the subgrid, leading to incorrect behaviour.

We apply a Dirichlet interior boundary condition leveraging global information from the bulk optimization in Section 3.1. Following the bulk optimization, we obtain the full resolution bulk smoke $\hat{s}$, velocity $\hat{\mathbf{u}}$ and pressure $\hat{p}$ through re-simulation. The bulk pressure already satisfies the global divergence-free and exterior boundary constraints and will serve as a reference. By applying this boundary pressure as a Dirichlet condition between subgrids, the pressures within each subgrid align closely with the corresponding bulk pressure. Consequently, the updated and stitched subgrid velocities will resemble the bulk fluid flow and satisfy the constraints.

We demonstrate our method using a staggered grid [HW65] spatial discretization scheme. In a staggered grid, scalar quantities are stored at grid centres, while vector quantities such as velocity are stored at grid face centres. Computing the pressure involves solving a discretized Poisson equation

$$\begin{aligned}\frac{\Delta t}{\rho}\left(\frac{4p_{i,j} - p_{i-1,j} - p_{i+1,j} - p_{i,j-1} - p_{i,j+1}}{\Delta x^2}\right)\\ = -\left(\frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{\Delta x} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{\Delta x}\right).\end{aligned} \tag{3}$$

To construct Dirichlet boundary conditions, we approximate the interior boundary pressure p^b on cell faces by linearly interpolating the bulk pressures $\hat{p}$ from adjacent cell centres. This is demonstrated in Figure 4. Without loss of generality, and in 2D, assume the boundary lies between cells (i, j) and $(i+1, j)$. We approximate the boundary pressure as $p^b = (\hat{p}_{i,j} + \hat{p}_{i+1,j})/2$. This boundary pressure serves as the Dirichlet boundary condition on the interior boundary during subgrid simulations.

The subgrid pressure should maintain the same pressure on the interior boundary. We use 'ghost' pressure cells $p^g_{i+1,j}$ to derive the updated Poisson equation when computing subgrid pressures $\bar{p}_{i,j}$. Similarly upholding Dirichlet boundary conditions, the ghost and interior boundary pressure cells should satisfy $p^b = (\bar{p}_{i,j} + p^g_{i+1,j})/2$.

Rearranging and substituting the ghost pressure into Equation (3), the discretized Poisson equation for cell (i, j) is

$$\begin{aligned}\frac{\Delta t}{\rho}\left(\frac{5p_{i,j} - p_{i-1,j} - p_{i,j-1} - p_{i,j+1}}{\Delta x^2}\right)\\ = -\left(\frac{u_{i+\frac{1}{2},j} - u_{i-\frac{1}{2},j}}{\Delta x} + \frac{v_{i,j+\frac{1}{2}} - v_{i,j-\frac{1}{2}}}{\Delta x}\right) + \frac{2\Delta t p^b}{\rho \Delta x^2}.\end{aligned} \tag{4}$$

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

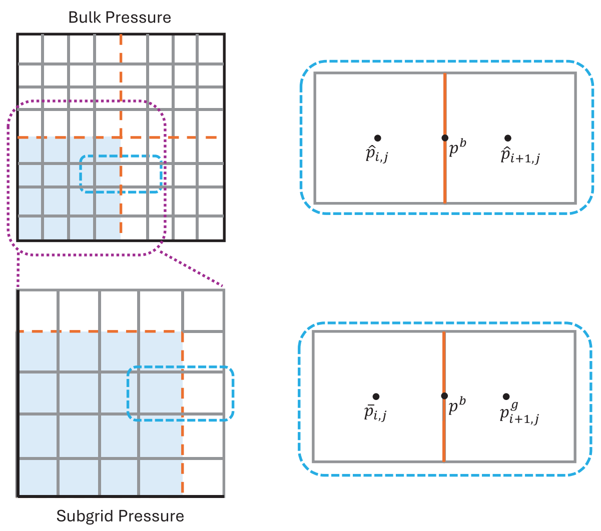

Figure 4: *The interior boundary pressure at the cell face center, p^b, is computed from the bulk pressure $\hat{p}$. The subgrid pressure projection should respect the same pressure at the boundary cell face centre. By using a 'ghost' pressure cell p^g, we can use a Dirichlet boundary condition to approximate the required pressure at the interior cell centre.*

The above derivation generalizes to all cells adjacent to interior boundaries. Finally, by constructing the linear system $A\mathbf{p} = \mathbf{b}$ and solving for the pressures, the velocity is updated using the pressure gradient. The computed pressures and velocities closely match the bulk information, ensuring that the forward subgrid simulation resembles the bulk flow.

We note that interpolating the boundary pressure p^b is necessary. If the pressure is taken directly from the neighbouring cell outside the interior boundary, i.e. setting $p^g_{i+1,j} = \hat{p}_{i+1,j}$, the pressure solve will result in band-like artifacts with a width of two pixels along the interior boundary, and results in inaccurate simulation outcome.

3.3. Subgrid Optimization

In the previous section, we demonstrated how to forward simulate subgrids using bulk information. In this section, we detail how to optimize the subgrids. We first note that all operations we described in the previous section maintain the solver's differentiability. Indeed, once the subgrids are divided, there are no discontinuous operators that block the flow of the gradient. Since the optimization only takes place at the subgrid level, the gradients do not need to be propagated to the full grid and hence the discontinuity at subgrid division does not affect optimization. Therefore, computing the gradients for the subgrids is as simple as full-resolution differentiable fluid simulations.

In subgrid optimization, the up-sampled bulk forces are fixed and serve as a warm start solution. We optimize a series of local correction forces that refine the simulated state on top of the bulk forces.

Algorithm 1. Subgrid optimization

```
1: Given bulk forces $\hat{\mathbf{f}}$, smoke $\hat{s}$, velocity $\hat{\mathbf{u}}$, and pressure $\hat{p}$
2: for $se \in 1$ to $N_{SE}$ do
3:     for $i \in$ randomize (subgrids) do
4:         $\bar{s}^i, \bar{\mathbf{u}}^i, \bar{p}^i =$ GetSubgrid $(\hat{s}, \hat{\mathbf{u}}, \hat{p})$        ▷ Section 3.2
5:         $\bar{\mathbf{f}}^i =$ Optimize $(s^{*,i}, \bar{s}^i, \bar{\mathbf{u}}^i, \bar{p}^i, N_E)$
6:         $\bar{\mathbf{f}} =$ Concat $\bar{\mathbf{f}}^{1..m}$ ▷ Concatenate into full resolution
7:         $\hat{s}, \hat{\mathbf{u}}, \hat{p} =$ Simulate $(s, \mathbf{u}, \bar{\mathbf{f}})$
8: return $\hat{s}, \hat{\mathbf{u}}, \bar{\mathbf{f}}$
```

Empirically, this initialization scheme results in a more stable convergence and better results compared to directly updating the up-sampled bulk forces.

A naive way of optimizing subgrids is to optimize each subgrid individually and stitch the optimized subgrids together; however, with this implementation, each subgrid will not have visibility to their adjacent subgrids' updates. This will result in discontinuous stitched simulated states, and re-simulating using the stitched forces will result in states further away from the target keyframe and visual artifacts.

We overcome this by iteratively optimizing the subgrids in series in a randomized order, as shown in Figure 2 part (b). After a subgrid is optimized for a user-specified amount of subgrid epochs N_E, a global re-simulation takes place to update the bulk grids. By re-simulating, the bulk grid will be updated, and subsequent subgrid optimizations will have new padded regions for advection and interior boundaries for pressure. After all subgrids are optimized, this process takes place repeatedly for a user-controlled super-epoch number of times N_{SE}. This process is described in Algorithm 1. In practice, by tuning the epoch, super-epoch and optimization step size hyperparameters, the repeated looping optimization leads to global quality improvement. Experimentally, the choice of the subgrid optimization ordering does not affect the optimization result. Section 4.5 demonstrates there is negligible difference when the subgrids are optimized in random order, in-order, or by parity (odd grids first, then even).

With our approach, each subgrid is optimized for a total of $N_{sub} = N_{SE} \times N_E$ iterations, and the combination of $\{N_{SE}, N_E\}$ determines the trade-off between quality and runtime. Given the same number of total iterations for each subgrid, smaller N_E will result in more re-simulations, and hence the locally-updated information will be more frequently propagated to other subgrids, thus resulting in better optimization quality; however, the runtime will increase linearly as N_{SE} increases due to the additional re-simulations. On the other hand, larger N_E will result in reduced runtime, but the information between each subgrid will be less frequently propagated, resulting in worse training quality. In the extreme case where $N_{SE} = 1$, our algorithm is equivalent to the naive approach described above. We illustrate how settings of super-epochs and epochs can affect runtime and optimization quality in Section 4.7.2.

To reduce memory consumption, we can optimize each subgrid separately, hence only requiring computation graph construction

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

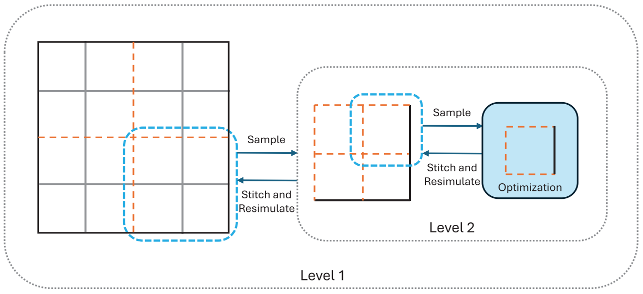

Figure 5: *Hierarchical subgrid optimization. Optimization takes place at the leaf subgrids. After each subgrid is optimized, it is stitched to the parent subgrid and the parent subgrid re-simulates to propagate the changes.*

Algorithm 2. Recursive subgrid optimization

```
1:  Given bulk forces $\hat{\mathbf{f}}$, smoke $\hat{s}$, velocity $\hat{\mathbf{u}}$, and pressure $\hat{p}$
2:  for $i \in$ randomize (subgrids) do
3:      $\bar{s}^i, \bar{\mathbf{u}}^i, \bar{p}^i$ = Subdivide $\hat{s}, \hat{\mathbf{u}}, \hat{p}$
4:      if leaf node level then
5:          $\bar{\mathbf{f}}^i$ = Optimize $(s^{*,i}, \bar{s}^i, \bar{\mathbf{u}}^i, \bar{p}^i, N_E)$        ▷ Section 3.3
6:      else
7:          $\bar{\mathbf{f}}^i$ = Recurse $(\bar{s}^i, \bar{\mathbf{u}}^i, \bar{p}^i)$
8:      $\bar{\mathbf{f}}$ = Stitch $\bar{\mathbf{f}}^{1..4}$
9:      $\hat{s}, \hat{\mathbf{u}}, \hat{p}$ = Simulate $(s, \mathbf{u}, \bar{\mathbf{f}})$
10: return $\hat{s}, \hat{\mathbf{u}}, \bar{\mathbf{f}}$
```

and storage for a $(nx/2, ny/2)$ grid setting. While a re-simulation at full resolution is needed, no computation graph is required.

3.4. Cascade Optimization

To further reduce memory consumption, the iterative divide-and-conquer-style method can be applied at multiple levels recursively. For each level $l = 1, 2, \ldots$, the subgrid optimizations will operate on grids of resolution $(nx/2^l, ny/2^l)$.

For the bulk optimization, the down-sample factor will be adjusted based on the level, such that $k \geq 2^l$. For subgrid optimization, the full resolution grid is divided into 4^l subgrids, and the optimization takes place at the lowest level leaf nodes in a depth-first order, as illustrated in Figure 5. After each subgrid is optimized, the parent subgrid is re-simulated to re-generate the bulk information used for the subsequent subgrid. This recursive subgrid optimization process is illustrated in Algorithm 2. This algorithm is then repeated for the user-specified super-epoch N_{SE} amount of iterations.

4. Experiments and Results

We implement our method in PYTHON and PYTORCH [PGM*19]. For simplicity, the pressure projection linear system is solved using the conjugate gradient algorithm without preconditioning. For optimization, we use the Adam optimizer [KB14] with exponential optimization step size scheduling with $\gamma = 0.95$, and the step

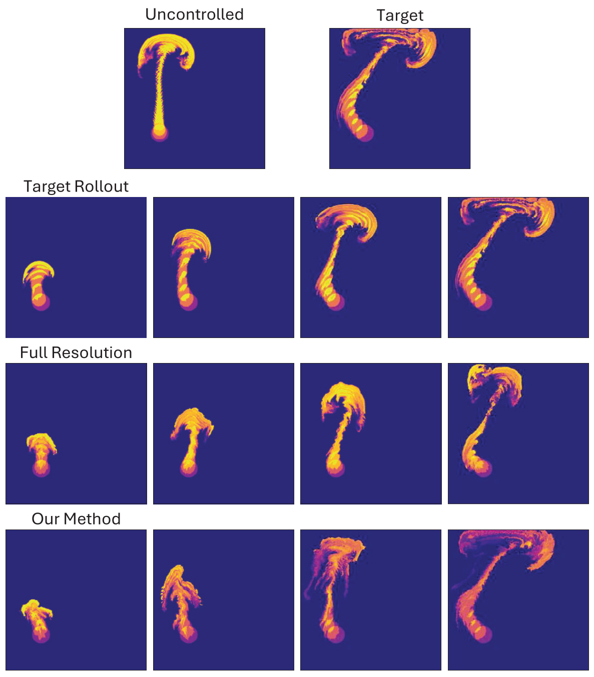

Figure 6: *Top left shows the final frame of a plume simulation that rises due to buoyancy, and top right shows the keyframe target. First row: applying a series of hand-crafted forces, we can painstakingly attain the target final frame. Second row: without any blurring or down-sampling, a full-resolution optimization cannot escape a local minima, failing to match the target. Third row: our method automatically optimizes for forces that result in a closer target match.*

sizes are adjusted based on each experiment. All experiments are run on a compute farm server with Intel(R) Xeon(R) W-2135 CPU and NVIDIA RTX A4000 GPU with VRAM size 16 GB. All data are created on the CPU by default. Relevant grids, down-sampled grids, and sub-grids are transferred from the CPU to the GPU during simulation and optimization, and moved back to the CPU afterwards.

4.1. Rising Plume Final Frame Matching

We first validate our method using a single final keyframe matching example in 1024 × 1024 resolution. In Figure 6, the left image shows a smoke simulation given an inflow injection and buoyancy forces, without any other external forces. The simulation rolls out for 30 frames with a time step size of 0.5. Under the same simulation settings, by applying a series of hand-crafted ground-truth forces, we generate a target frame on the right, with a target roll-out simulation at the top. The optimization goal is to find a space-time varying force field, such that when applied to the simulation, the final frame converges to the target keyframe as much as possible.

We define our **baseline** as the full resolution optimization using differentiable simulation, as described in Section 2.3. The result of our baseline is shown in the second row in Figure 6. Due to the

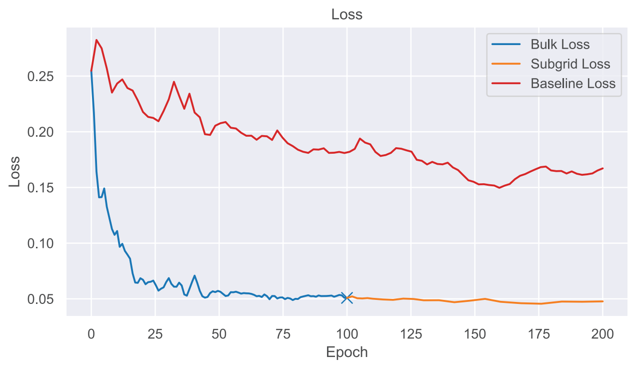

Figure 7: *Loss plot for the plume example at resolution* 1024^2. *We visualize the density mean squared error, instead of the full energy function* Φ. *While the baseline fails to optimize after 200 epochs, our method decreases the loss significantly. For our method, the first 100 epochs correspond to the down-sampled bulk optimization. Later 100 epochs correspond to the subgrid optimization.*

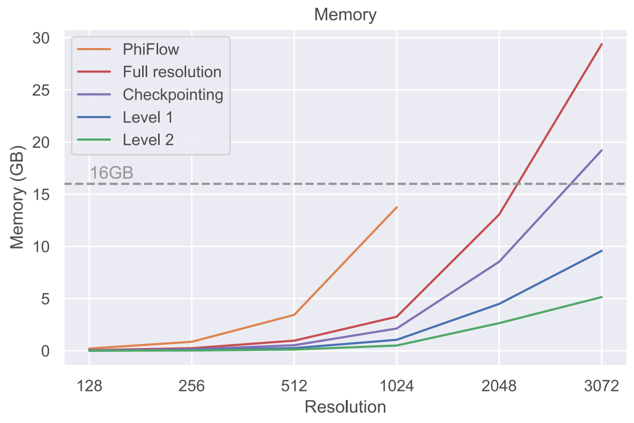

Figure 8: *The memory consumption of optimization against resolutions. Missing data points are due to simulation solves failing to converge. Our method outperforms gradient checkpoinitng, and the reduction in memory increases as subgrid division level increases.*

high resolution and hence degrees of freedom, the full-resolution optimization is very sensitive to perturbations to the step size and easily gets stuck at local minima, resulting in unsatisfactory optimization outcome, with Structural Similarity Index Measure (SSIM) [WBSS04] score 0.7748. In the figure, we showcase the best result we could obtain by repeatedly tuning hyper-parameters. On the other hand, our method does not suffer from this issue thanks to the bulk down-sample optimization process. In this example, the bulk optimization first down-samples by a factor of $k = 8$ to optimize the bulk forces for $N_{bulk} = 100$ epochs with optimization step size $\alpha_{bulk} = 3 \times 10^{-2}$. This helps reduce the degrees of freedom, which not only guides the optimization in the correct direction but also makes our method more stable. The bulk optimization produces an SSIM score of 0.7991. Then, continuing from the bulk state, we carry out subgrid optimization at division level $l = 1$ for $N_E = 4$ and $N_{SE} = 25$ epochs and step size $\alpha_{sub} = 10^{-3}$ to further improves the optimized result, and eventually produces the final row in the figure, with SSIM score 0.8093.

Figure 7 compares the loss between the baseline at full resolution and our method. The bulk loss is measured at each epoch by up-sampling and re-simulating at the full resolution. Similarly, the subgrid loss is measured after each super-epoch, by stitching the optimized force and re-simulating at the full resolution. For a clearer comparison and visualization, instead of the objective function Φ, we plot the pixel-wise mean squared error of the density s and do not include the penalty losses. We first observe that the loss of the baseline optimization aligns with its visual result. Due to the high degrees of freedom, the optimization oscillates and performs poorly. On the other hand, the bulk optimization in the first 100 epoch is able to quickly reduce the loss and optimize the bulk motion of the fluid. Subsequently, by dividing the grid into subgrids and iteratively refining each one of them, the loss is able to decrease further.

We note that the subgrid optimization loss curve oscillates because the re-simulation will propagate local forces to the global domain. Since fluid simulation is highly non-linear, a force that improves a local region may not propagate the same effect to other subgrids.

4.2. Benchmark and Stress Test

Table 1 shows the memory consumption and runtime cost when running the experiment described above under different resolutions. We make comparisons against PhiFlow's implementation [HTK19], our baseline implementation at full resolution, gradient checkpointing [CXZG16] at full resolution, and our subgrid optimization techniques with division levels 1 and 2. Across all these implementations, the pressure projection step uses the conjugate gradient solver with the same maximum iteration count of 500, and the same relative tolerance of 10^{-2}.

We acknowledge that PhiFlow supports different AutoDiff frameworks, and more complex and optimizable boundary conditions, and hence the heavier overhead may not make a fair comparison. We also note that PhiFlow supports just-in-time (JIT) compilation, which if enabled, would greatly improve the runtime performances; however, we did not implement JIT compilation as our main goal focused on reducing the memory bottleneck. With gradient checkpointing, we checkpoint every simulation time step. For simulation with T time steps, this checkpointing scheme will reduce the memory consumption of computing the gradient to $\mathcal{O}(\sqrt{T})$ when computing the gradient. While the simulation and gradient computation (hence optimization) results are the exact same as that of the full resolution, the runtime will increase due to the recomputations between each checkpoint [CXZG16]. Moreover, this checkpointing scheme is not invariant to resolution, and hence the memory consumption will still increase exponentially as the simulation resolution increases.

Figure 8 shows the memory consumption for optimizing with different methods for 100 epochs. We demonstrate that our subgrid method outperforms gradient checkpointing, and reduces the memory consumption compared to full resolution at least by half. Increasing the subgrid recursion level also leads to lower

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Table 1: *Memory and runtime for optimizing the plume rising scene using different optimization schemes for different resolutions. The* **bolded** *data ran out of memory and was re-run on a Quadro RTX 8000 GPU with 48 GB VRAM for validation (runtime not reported due to hardware difference).*

	PhiFlow		Full Optim.		Checkpoint		Level 1		Level 2	
Res.	Mem. (GB)	Time (h:m)	Mem. (GB)	Time (h:m)	Mem. (GB)	Time (h:m)	Mem. (GB)	Time (h:m)	Mem. (GB)	Time (h:m)
128	0.22	1:05	0.06	0:16	0.05	0:40	0.02	0:46	0.01	1:04
256	0.86	1:13	0.25	0:26	0.19	1:01	0.06	1:06	0.03	1:30
512	3.44	1:29	0.97	0:30	0.72	1:05	0.26	1:24	0.12	1:47
1024	13.75	1:58	3.27	0:38	2.86	1:08	1.05	1:33	0.51	2:17
2048	OOM	\	13.06	1:48	11.45	2:44	4.50	2:59	2.65	3:58
3072	OOM	\	**29.38**	\	**19.21**	\	9.58	5:06	5.15	5:46

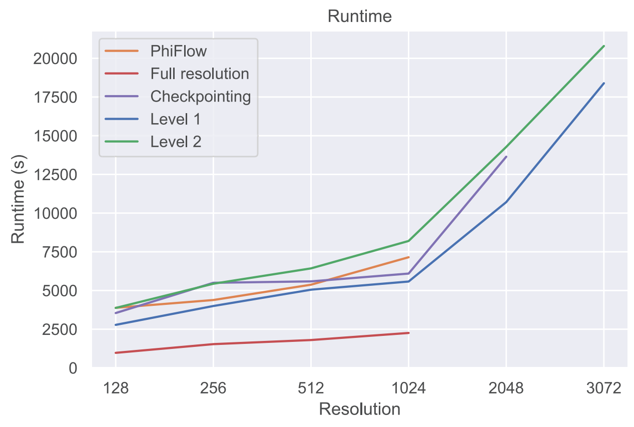

Figure 9: *Equal 100 epoch runtime against resolution. Missing data points are due to out-of-memory on consumer-level graphics card or failed simulation convergence. As resolution increases, the runtime increases for all implementations. Our method provides a new trade-off between memory consumption and optimization runtime.*

memory consumption, but the memory saving diminishes in a logarithmic fashion. The diminishing return is expected since the decrease in degrees of freedom is logarithmic, and the overhead for storing simulation states at high resolution is also not negligible. Note that for resolutions above 2048 × 2048, the PHIFLOW forward simulation *fails to converge*, and hence is not included in our reporting. At resolution 3072 × 3072, both full-resolution optimization and gradient checkpointing *run out of memory*. Our method successfully reduces the memory consumption: at level one recursion, our method uses **32**% of the memory (9.58 GB) of a full-resolution optimization and **50**% of the gradient checkpointing option. With level two recursion, memory usage is further reduced to 5.15 GB (**18**% and **27**% compared to full resolution and gradient checkpointing).

Figure 9 shows the optimization runtime using different methods for 100 epochs. For our subgrid approach, we consider one full-resolution epoch to be equivalent to the combination of one bulk epoch and one epoch for each subgrid. This figure highlights the fact that our method offers a new trade-off between memory consumption and optimization runtime. Another factor that contributes to the runtime is the combination of the number of super-epochs and epochs, as detailed in Section 4.7.2

4.3. 'FLUID' Letter Morphing

We compare our multi-resolution hierarchical approach with the work of Pan and Manocha [PM17], which splits the solution of the optimization into two objectives. We recreate the 'FLUID' example and provide a visual and performance comparison. Each letter is optimized separately, and we optimize on both the original 128 × 128 resolution and a larger 256 × 256 resolution for 40 time steps. The bulk optimization takes $k = 2$ and runs for $N_{bulk} = 100$ iterations with $\alpha_{bulk} = 10^{-2}$. The subgrid optimization divides the domain with $l = 1$, and takes $N_{SE} = 50$ and $N_E = 2$, and $\alpha = 10^{-3}$.

Figure 10 visually compares the result between our method and Pan and Manocha [PM17]. Our method produces fewer artifacts and matches the FLUID target better. As for memory and performance, Pan and Manocha [PM17] reported to have taken 0.06GB with 0.25 h runtime. For the same 128 × 128 resolution, our method takes 0.02GB with 2.68 h runtime, and for the higher 256 × 256 resolution, our method takes 0.08GB with 4.08 h runtime.

4.4. Multi-Keyframe Animation

We apply our method to multiple target keyframes, illustrating how the subgrid optimization can optimize local forces that improve the bulk optimized result. In Figure 11, the fluid morphs between 5 letter keyframes 'GRAPH' over 100 frames on a 256 × 256 domain. The bulk optimization uses $N_{bulk} = 200$ epochs with a step size of $\alpha_{bulk} = 10^{-3}$, and each subgrid optimization uses $N_{sub} = 100$ epochs $\alpha_{sub} = 10^{-4}$. The optimization converges, correctly matching the desired keyframes. We visualize the bulk and subgrid loss plots, which indicate that the subgrid optimization reduces the loss significantly after the bulk optimization converges. We also show the keyframes for the final letters 'H' after both bulk and subgrid optimization converges. Qualitatively, we observe that the converged bulk result correctly produces the general shape of the letters, but the local details such as the edges of the letters are chaotic and irregular. The subgrid optimization, by contrast, successfully optimizes the local forces that enhance the bulk initial results. Consequently, subgrid-refined letters have sharper edges and more distinctly defined shapes compared to the bulk-only results.

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

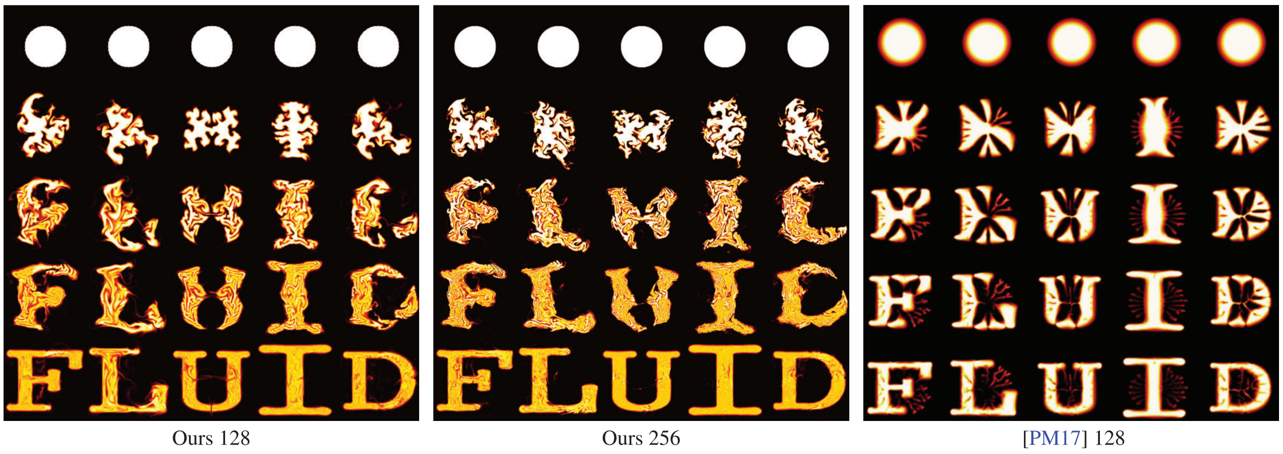

Figure 10: *We compare our method (left two) against the method of Pan and Manocha [PM17] (right) on the FLUID example. We demonstrate results at both resolutions of* 128^2 *and* 256^2*. Each row shows the resulting simulation at frames 0, 20, 30, 35, 40 respectively.*

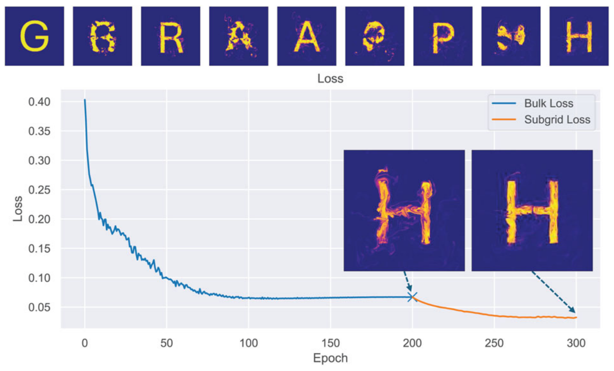

Figure 11: *The optimized forces morph the simulated smoke into 'GRAPH' by defining multiple letter keyframes. The top row is the optimized simulation rollout. Every other image is the reconstruction of a target letter. At the bottom, we show the loss plot and the final 'H' letter after bulk and subgrid optimization finishes.*

4.5. Latte Art

We provide an example of keyframe morphing: given a Latte art image as a target keyframe, starting from a circular initial smoke state, the optimization optimizes a series of forces that drive the density towards the latte art design. The simulation runs at 512×512 for 30 time steps with a step size of 0.5s. The bulk optimization downsamples the grid with a factor of $k = 2$ and optimizes with a step size of $\alpha_{bulk} = 0.05$ for $N_{bulk} = 200$ epochs. Then the subgrid optimization takes place at recursion level $l = 1$, and each subgrid is optimized for $N_{sub} = 200$ epochs with a step size of $\alpha_{sub} = 0.0001$. Figure 12 shows the result rendered in a coffee mug.

Figure 13 shows the loss curve of the 2D latte art morph optimization. We experiment with different subgrid optimization ordering described in Section 3.3. Continuing from the bulk optimization, we investigate alternative subgrid ordering schedules, including in-order and parity (odd followed by even) ordering, with step size $\alpha = 10^{-3}$ and $N_E = 4$, $N_{SE} = 50$ iterations. The result shows that the ordering of the subgrid optimization does not have a significant effect on the optimization result.

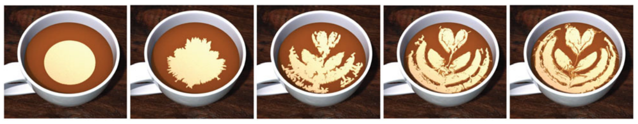

Figure 12: *A series of forces are optimized to morph a circular disk into a latte art. The resultant 2D simulation is then rendered onto a coffee mug to create a latte art effect.*

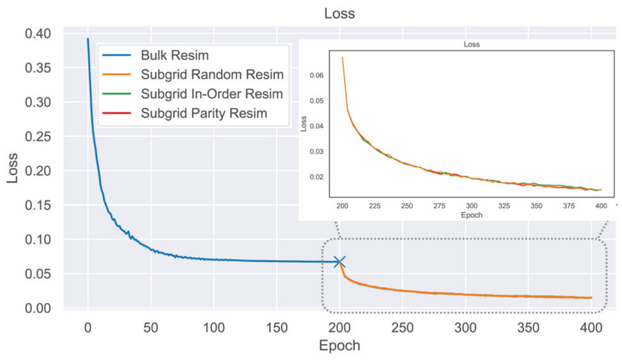

Figure 13: *The latte art morph loss curve with different subgrid optimization ordering scheme, including random order, in-order and parity (even grids first, then odd).*

4.6. Looping Fluid Simulation

Looping fluid simulation is a long-standing challenge in fields such as visual effects and game development. Although recent advances have been made in looping simulations for thin shells and N-body systems [JWLC23], cyclic fluid simulation still remains an open

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

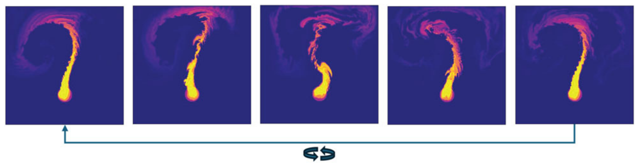

Figure 14: *Looping simulations with external force optimization and 2-level subgrids, matching the final and initial frames.*

challenge. We apply the fluid control framework that we introduced in this paper to tackle this problem by framing it as a keyframe control problem. In this application, arbitrary sinks are created to dissipate the smoke so that injecting the smoke does not fill the scene. First, we run the forward simulation with random forces over a period of time. Then, we take one frame of the simulation and designate it as the start frame of the looping simulation. The goal of the control problem is to compute forces throughout time, such that continuing the simulation from the start frame will result in the end frame as close to the start frame as possible. On top of the smoke density mean squared error loss, we also impose a velocity mean squared error loss in our energy function Φ, such that the animation loops in a smooth motion.

With this formulation, we solve the control problem with our subgrid division at 256×256 for 30 time steps with time step of 0.5 s. We use two levels of subgrid division, with a bulk optimization step size of $\alpha_{bulk} = 0.01$ and $N_{bulk} = 100$ epochs. The subgrid optimization recurses 2 levels, with each subgrid optimized for $N_{sub} = 100$ epochs with a step size of $\alpha_{sub} = 0.001$ (see Figure 14). Both the bulk and subgrid optimizations converge to a simulation end state steered towards the target.

4.7. Design Choice Analysis

4.7.1. *Subgrid Division Level*

We study the effect of subgrid division level l on the optimization using our latte morph optimization as an example. From the same bulk optimization checkpoint, we optimize using level 1 and level 2 subgrid divisions using the same step size $\alpha = 10^{-3}$ and $N_E = 4$, $N_{SE} = 50$. Figure 15 demonstrates the loss graph of the optimizations. We observe that while both division levels are able to significantly decrease the loss further, level 1 subgrid division is able to reduce the loss faster compared to that of level 2. Combining this observation with Figure 8 and Figure 9, we conclude that L1 achieves higher accuracy and lower runtime, though L2 consumes less memory. For this reason, level 1 should be used if there is enough memory.

4.7.2. *Epoch and Super Epoch Benchmark*

One important hyper-parameter in our algorithm is the combination of subgrid epochs N_E and super-epochs N_{SE}. As described in Section 3.3, even though the combination of N_E and N_{SE} does not affect the memory consumption of our algorithm, it determines the trade-off between runtime and quality. Given a fixed amount of total number of epochs per subgrid $N = N_E \times N_{SE}$, a smaller N_E larger N_{SE}

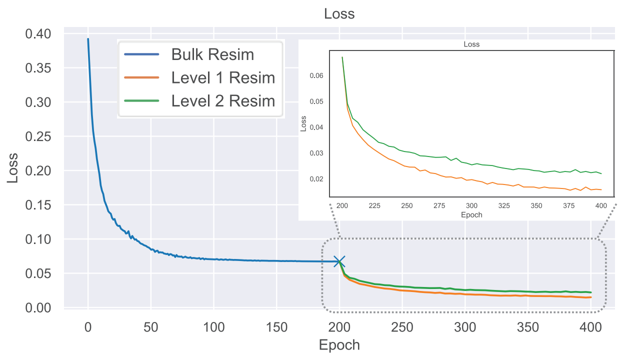

Figure 15: *Optimizing the subgrid using level 1 and level 2 division, continuing from the same bulk checkpoint in the latte art morph. Under the same number of epochs and step size, level 1 optimization results in lower loss.*

Table 2: *Epoch and super-epoch combination comparison. Each subgrid is optimized for 100 epochs in total, hence the optimization runtime remains mostly constant; however, the super-epoch and epoch combination determines the number of re-simulations. More re-simulations result in a longer total runtime. More re-simulations usually lead to lower converged loss.*

$N_{SE} \times N_E$	Total runtime (h:m)	Optimization runtime (h:m)	Re-simulation runtime (h:m)	Re-simulation Loss
1×100	**0:19**	0:18	0:01	0.03597
2×50	0:21	0:18	0:03	0.03581
4×25	0:24	0:18	0:06	0.03559
5×20	0:26	0:18	0:08	0.03552
10×10	0:33	0:18	0:15	0.03479
20×5	0:49	0:18	0:31	0.03486
25×4	0:57	0:18	0:39	**0.03310**
50×2	1:36	0:18	1:18	0.03361
100×1	2:56	0:18	2:38	0.03381

combination indicates more re-simulations, leading to a higher runtime. In this case, the information is exchanged and updated more frequently between subgrids, usually leading to better optimization quality. Conversely, a larger N_E and smaller N_{SE} combination will result in faster training speed due to the reduced re-simulations, but worse results. In this experiment, we investigate the relationship between the two hyper-parameters.

For the test scene in Section 4.1, after 100 iterations of downsampled bulk optimization, we continue subgrid optimization for $N_{sub} = 100$ iterations at step size $\alpha_{sub} = 10^{-4}$, testing multiple N_E and N_{SE} combinations; we report runtime and final loss for each combination in Table 2. We observe that optimization runtime for each super-epoch and epoch combination remains constant, and as N_{SE} increases the number of re-simulations increases (and thus the total runtime). Also note that, at low N_{SE}, the loss is higher due to reduced information propagation; however, as N_{SE} increases, the loss decreases and reaches a minimum with $N_{SE} = 25$ and $N_E = 4$. Further increasing N_{SE} does not further decrease the loss, and the losses are still lower compared than $N_{SE} < 25$.

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

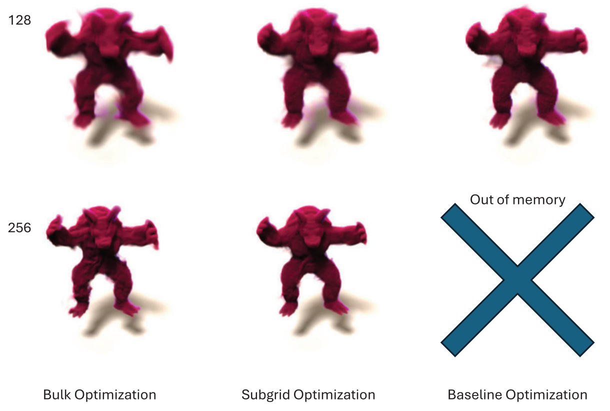

Figure 16: *Top row shows our result after bulk optimization (left) and subgrid optimization (middle), compared against the baseline method (right) at resolution* 128^3 *for 40 time steps. Bottom left shows our method at resolution* 256^3 *for 40 time steps. Baseline fails to optimize due to out of memory error (exceeds 40GB).*

4.8. 3D Implementation

We demonstrate through a morphing example that our method described in Section 3 extends to 3D. We use test scenes with resolution of 128^3 and 256^3 for 40 frames. Starting from a sphere, the goal is to optimize a series of dense force fields that morph the smoke into the target shape. The bulk optimization down-samples with $k = 2$, and uses $N_{bulk} = 100$ iterations with step size of $\alpha_{bulk} = 10^{-3}$. The subgrid optimization is done with $l = 1$, which divides the simulation domain into 8 subgrids, with $N_{SE} = 50$ and $N_E = 4$ and $\alpha_{sub} = 10^{-4}$. As seen in Figure 1, our method optimizes a series of forces that leads to a continuous simulation that eventually reaches the armadillo target.

Figure 16 visually compares the optimized result from bulk, subgrid and baseline optimizations at both resolutions. The baseline optimization runs for $N = 170$ epochs and uses step size $\alpha = 10^{-2}$. We observe the bulk optimization generates global forces that morphs the smoke into the general shape; however, local details such as the claws are of low quality due to the low resolution of the force field. Continuing from the bulk, the subgrid optimization refines locally and produces better results with finer details. At resolution 128^3, the baseline method produces a better result, but takes 14.22GB of memory and was optimized for 1.1 h, whereas our method takes only 2.10GB and was optimized for 4.8 h. At resolution 256^3, the baseline method fails due to out of memory even on a more powerful Quadro RTX 8000 GPU with 48GB of VRAM; however, our method is still able to converge successfully with the consumption of 15.90GB of VRAM with optimization time 31.4 h.

Figure 17 demonstrates the optimization loss curve corresponding to the resolution 128^3. Aligned with the visual demonstration, the bulk optimization decreases the loss by optimizing the low-resolution forces, and the subgrid optimization decreases the loss further by optimizing at high resolution locally. Compared to the baseline of optimizing at full resolution, the quality does not degrade significantly. Figures 18 and 19 showcase the memory consumption and runtime with respect to resolution. Similar to the 2D result, in trading runtime and making small compromises in quality, we reduce the memory consumption drastically.

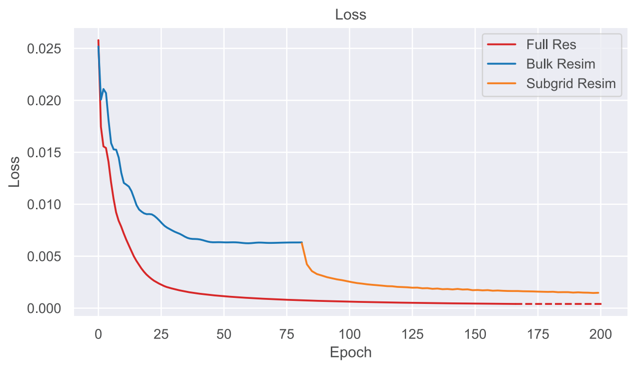

Figure 17: *Loss curve for morphing from a sphere to a 3D armadillo at resolution* 128^3.

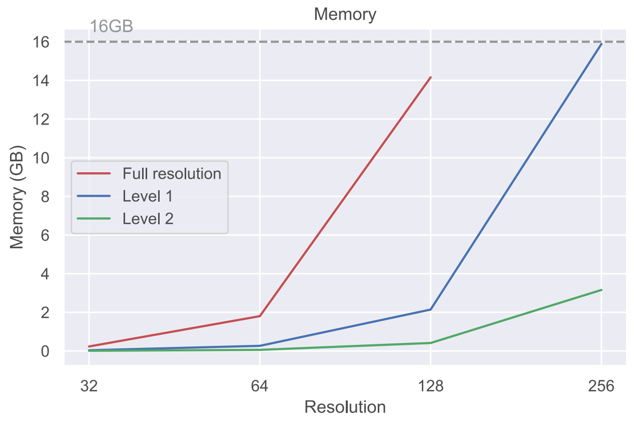

Figure 18: *The memory consumption of the armadillo morph using different optimization methods at several resolutions. The baseline full resolution optimization fails at resolution* 256^3*, even on a more powerful GPU with 40GB VRAM.*

5. Conclusion and Discussion

We presented a hierarchical splitting approach to reduce the memory consumption of differentiable fluid simulation. We demonstrated that our splitting method requires significantly less memory and achieves comparable quality to previous methods, at the expense of additional runtime. Although our method saves memory by trading runtime, it enables optimizations on large scenes where full-resolution optimization was unfeasible due to the memory bottleneck. In terms of optimization quality, in some cases, our method results in less accuracy compared to the baseline full-resolution optimization, and in other cases, our bulk optimization helps reduce the degrees of freedom and facilitates escaping the local minima,

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

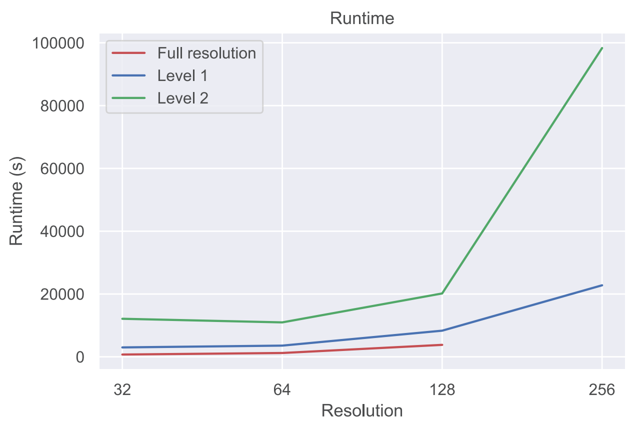

Figure 19: *The runtime of optimizing of the armadillo morph for optimizing 100 epochs at different resolution. The full resolution optimization at* 256^3 *resolution is not reported due to out of memory.*

resulting in better quality. Our work addresses the memory bottleneck and opens many exciting avenues.

Firstly, during subgrid optimization, the re-simulation after each subgrid epoch is necessary to propagate the local forces to the other subgrids, leading to an additional cost in runtime. This is mainly caused by the fact that during simulation and optimization, each subgrid has no knowledge of the neighbouring subgrids' states such as density or forces. A possible direction of research is to improve the coupling between subgrids. Such improvements can be done at the simulation level by coupling with not only pressure, but also velocity, density and forces. Another possible approach is improving at the loss level by injecting forces information from neighbouring subgrids through better penalty terms. Exploring decomposition strategies beyond spatial partitioning, such as frequency-domain approaches inspired by octave-based representations in wave simulations, is also a promising direction for future research.

Secondly, our experiments currently only support box-like domains with easily-described slip boundary conditions. Supporting a wider range of boundary conditions and obstacles into the scene requires extra engineering and introduces an added layer of complexity due to the grid-based data structure in PyTorch. One future avenue of work is to support more general scenes and fluid-obstacle interactions.

Thirdly, in this work, we focused our attention on optimizing space-time varying forces that drive fluid motion; however, the capabilities of differentiable simulation extend far beyond this application. We believe additional parameters such as inflow location, fluid density, temperature, and even obstacle shape, could be optimized by extending this framework. Simpler scalar parameters such as fluid density can be optimized by treating them as input parameters to the differentiable simulator and taking gradients of the loss with respect to them. Optimizing inflow location and obstacle shapes would require further attention to the subgrid discontinuities; and non-axis-aligned obstacles present another layer of challenges. These challenges present a broader scope for further exploration.

Finally, a main trade-off our method makes is the increased runtime. This is due to a multitude of factors. Generalizing our subgrid division scheme from 2×2 to $m \times m$, for level l division, would require m^{d^l} optimizations to take place in series. As shown in Section 4.7.2, depending on the combination between the super-epoch and epoch parameters, the re-simulation runtime is not negligible. One future work direction to address this problem is to come up with better heuristics to determine the balance between the super-epoch and subgrid epoch combination.

Acknowledgements

The authors have nothing to report.

Conflict of Interest

None of the authors has a conflict of interest to disclose.

References

[AAB*16] Abadi M., Agarwal A., Barham P., Brevdo E., Chen Z., Citro C., Corrado G. S., Davis A., Dean J., Devin M., Ghemawat S., Goodfellow I., Harp A., Irving G., Isard M., Jia Y., Jozefowicz R., Kaiser L., Kudlur M., Levenberg J., Mane D., Monga R., Moore S., Murray D., Olah C., Schuster M., Shlens J., Steiner B., Sutskever I., Talwar K., Tucker P., Vanhoucke V., Vasudevan V., Viegas F., Vinyals O., Warden P., Wattenberg M., Wicke M., Yu Y., Zheng X.: TensorFlow: Large-scale machine learning on heterogeneous distributed systems, Mar. 2016. arXiv:1603.04467.

[BPRS18] Baydin A., Pearlmutter B., Radul A., Siskind J.: Automatic differentiation in machine learning: A survey. *Journal of Machine Learning Research 18* (Apr. 2018), 1–43. https://doi.org/10.48550/arXiv.1502.05767.

[BR86] Brackbill J. U., Ruppel H. M.: FLIP: A method for adaptively zoned, particle-in-cell calculations of fluid flows in two dimensions. *Journal of Computational Physics 65*, 2 (Aug. 1986), 314–343. https://doi.org/10.1016/0021-9991(86)90211-1.

[Bri15] Bridson R.: *Fluid Simulation for Computer Graphics* (2nd edition). AK Peters/CRC Press, New York, Sept. 2015. https://doi.org/10.1201/9781315266008.

[CFL28] Courant R., Friedrichs K., Lewy H.: Über die partiellen Differenzengleichungen der mathematischen Physik. *Mathematische Annalen 100*, 1 (Dec. 1928), 32–74. https://doi.org/10.1007/BF01448839.

[CLL24] Chen Y., Levin D., Langlois T.: Fluid control with Laplacian eigenfunctions. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, July 2024), SIGGRAPH '24,

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Association for Computing Machinery, pp. 1–11. https://doi.org/10.1145/3641519.3657468.

[CXZG16] Chen T., Xu B., Zhang C., Guestrin C.: Training Deep Nets with Sublinear Memory Cost, Apr. 2016. arXiv:1604.06174.

[CZY17] Chu J., Zafar N. B., Yang X.: A Schur complement preconditioner for scalable parallel fluid simulation. *ACM Transactions on Graphics 36*, 5 (July 2017), 163:1–163:11. https://doi.org/10.1145/3092818.

[dSA*18] de Avila Belbute-Peres F., Smith K., Allen K., Tenenbaum J., Kolter J. Z.: End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems* (Montreal, Canada, 2018), Bengio S., Wallach H., Larochelle H., Grauman K., Cesa-Bianchi N., Garnett R. (Eds.) (vol. *31*), Curran Associates, Inc., Red Hook, New York.

[FL04] Fattal R., Lischinski D.: Target-driven smoke animation. *ACM Transactions on Graphics. 23*, 3 (Aug. 2004), 441–448. https://doi.org/10.1145/1015706.1015743.

[FM97a] Foster N., Metaxas D.: Controlling fluid animation. In *Proceedings Computer Graphics International* (Hasselt and Diepenbeek, Belgium, June 1997), IEEE Computer Society Press, pp. 178–188. https://doi.org/10.1109/cgi.1997.601299.

[FM97b] Foster N., Metaxas D.: Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (USA, Aug. 1997), SIGGRAPH '97, ACM Press/Addison-Wesley Publishing Co., pp. 181–188. https://doi.org/10.1145/258734.258838.

[GNS*12] Golas A., Narain R., Sewall J., Krajcevski P., Dubey P., Lin M.: Large-scale fluid simulation using velocity-vorticity domain decomposition. *ACM Transactions on Graphics. 31*, 6 (Nov. 2012), 148:1–148:9. https://doi.org/10.1145/2366145.2366167.

[HAL*19] Hu Y., Anderson L., Li T.-M., Sun Q., Carr N., Ragan-Kelley J., Durand F.: DiffTaichi: Differentiable programming for physical simulation. *ArXiv* (Sept. 2019). https://doi.org/10.48550/arXiv.1910.00935.

[Har62] Harlow F. H.: *The Particle-in-Cell Method for Numerical Solution of Problems in Fluid Dynamics*. Tech. Rep. LADC-5288, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), Mar. 1962. https://doi.org/10.2172/4769185.

[HLM*19] He X., Li J., Mader C. A., Yildirim A., Martins J. R. R. A.: Robust aerodynamic shape optimization—From a circle to an airfoil. *Aerospace Science and Technology 87* (Apr. 2019), 48–61. https://doi.org/10.1016/j.ast.2019.01.051.

[HTK19] Holl P., Thuerey N., Koltun V.: Learning to control pdes with differentiable physics. In *International Conference on Learning Representations* (Addis Ababa, Ethiopia, Sept. 2019). https://doi.org/10.48550/arXiv.2001.07457.

[HW65] Harlow F. H., Welch J. E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of Fluids 8*, 12 (1965), 2182. https://doi.org/10.1063/1.1761178.

[JMG*21] Jatavallabhula K. M., Macklin M., Golemo F., Voleti V., Petrini L., Weiss M., Considine B., Parent-Levesque J., Xie K., Erleben K., Paull L., Shkurti F., Nowrouzezahrai D., Fidler S.: gradsim: Differentiable simulation for system identification and visuomotor control. *International Conference on Learning Representations (ICLR)* (2021). https://doi.org/10.48550/arXiv.2104.02646.

[JWLC23] Jia S., Wang S., Li T.-M., Chern A.: Physical cyclic animations. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 6*, 3 (Aug. 2023), 1–18. https://doi.org/10.1145/3606938.

[KB14] Kingma D. P., Ba J.: Adam: A method for stochastic optimization. *CoRR* (Dec. 2014). https://doi.org/10.48550/arXiv.1412.6980.

[KY00] Kim Y. Y., Yoon G. H.: Multi-resolution multi-scale topology optimization—A new paradigm. *International Journal of Solids and Structures 37*, 39 (Sept. 2000), 5529–5559. https://doi.org/10.1016/S0020-7683(99)00251-6.

[LXY*23] Li Z., Xu Q., Ye X., Ren B., Liu L.: DiffFR: Differentiable SPH-based fluid-rigid coupling for rigid body control. *ACM Transaction on Graphics. 42*, 6 (Dec. 2023), 179:1–179:17. https://doi.org/10.1145/3618318.

[MCG03] Müller M., Charypar D., Gross M.: Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar, DEU, July 2003), SCA '03, Eurographics Association, pp. 154–159.

[MM13] Madill J., Mould D.: Target particle control of smoke simulation. In *Proceedings of Graphics Interface 2013* (CAN, May 2013), GI '13, Canadian Information Processing Society, pp. 125–132.

[MP04] Mohammadi B., Pironneau O.: Shape optimization in fluid mechanics. *Annual Review of Fluid Mechanics 36*, Volume 36, 2004 (Jan. 2004), 255–279. https://doi.org/10.1146/annurev.fluid.36.050802.121926.

[MSQ*18] Mashayekhi O., Shah C., Qu H., Lim A., Levis P.: Automatically distributing Eulerian and hybrid fluid simulations in the cloud. *ACM Transactions on Graphics 37* (June 2018), 1–14. https://doi.org/10.1145/3173551.

[MST10] McAdams A., Sifakis E., Teran J.: A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Goslar, DEU, July 2010), SCA '10, Eurographics Association, pp. 65–74.

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

[MTPS04] McNamara A., Treuille A., Popović Z., Stam J.: Fluid control using the adjoint method. *ACM Transactions on Graphics 23*, 3 (Aug. 2004), 449–456. https://doi.org/10.1145/1015706.1015744.

[OFEH18] Oborn J., Flynn S., Egbert P., Holladay S.: Time-reversed art directable smoke simulation. In *Proceedings of the 39th Annual European Association for Computer Graphics Conference: Short Papers* (Goslar, DEU, Apr. 2018), EG, Eurographics Association, pp. 1–4.

[PGM*19] Paszke A., Gross S., Massa F., Lerer A., Bradbury J., Chanan G., Killeen T., Lin Z., Gimelshein N., Antiga L., Desmaison A., Kopf A., Yang E., DeVito Z., Raison M., Tejani A., Chilamkurthy S., Steiner B., Fang L., Bai J., Chintala S.: PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems* (Red Hook, NY, USA, 2019), vol. *32*, Curran Associates, Inc. https://doi.org/10.48550/arXiv.1912.01703.

[PM17] Pan Z., Manocha D.: Efficient solver for spacetime control of smoke. *ACM Transactions on Graphics 36*, 5 (July 2017), 162:1–162:13. https://doi.org/10.1145/3016963.

[RSÖ*22] Rioux-Lavoie D., Sugimoto R., Özdemir T., Shimada N. H., Batty C., Nowrouzezahrai D., Hachisuka T.: A monte carlo method for fluid simulation. *ACM Transactions on Graphics 41*, 6 (Dec. 2022), 1–16. https://doi.org/10.1145/3550454.3555450.

[RWA21] Reinisch J., Wehrle E., Achleitner J.: Multiresolution topology optimization of large-deformation path-generation compliant mechanisms with stress constraints. *Applied Sciences 11*, 6 (Jan. 2021), 2479. https://doi.org/10.3390/app11062479.

[SBH24] Sugimoto R., Batty C., Hachisuka T.: Velocity-based Monte Carlo fluids. In *ACM SIGGRAPH 2024 Conference Papers* (New York, NY, USA, July 2024), SIGGRAPH '24, Association for Computing Machinery, pp. 1–11. https://doi.org/10.1145/3641519.3657405.

[SC20] Schoenholz S. S., Cubuk E. D.: JAX, M.D. a framework for differentiable physics. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, Dec. 2020), NIPS'20, Curran Associates Inc., pp. 11428–11441. https://doi.org/10.48550/arXiv.1912.04232.

[Sch21] Schoentgen A.: *Tools for Fluid Simulation Control in Computer Graphics*. Modeling and Simulation. Université de Poitiers; Université de Montréal (1878-….), 2021. English. ⟨NNT: 2021POIT2287⟩. ⟨tel-03862771⟩.

[SPM22] Sivakumaran G., Paquette E., Mould D.: Time Reversal and Simulation Merging for Target-Driven Fluid Animation. In *Proceedings of the 15th ACM SIGGRAPH Conference on Motion, Interaction and Games* (New York, NY, USA, Nov. 2022), MIG '22, Association for Computing Machinery, pp. 1–9. https://doi.org/10.1145/3561975.3562952.

[Sta99] Stam J.: Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (USA, July 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 121–128. https://doi.org/10.1145/311535.311548.

[TAB*96] Tezduyar T., Aliabadi S., Behr M., Johnson A., Kalro V., Litke M.: Flow simulation and high performance computing. *Computational Mechanics 18*, 6 (Dec. 1996), 397–412. https://doi.org/10.1007/BF00350249.

[TACS21] Tang J., Azevedo V., Cordonnier G., Solenthaler B.: Honey, I shrunk the domain: Frequency-aware force field reduction for efficient fluids optimization. *Computer Graphics Forum 40* (May 2021), 339–353. https://doi.org/10.1111/cgf.142637.

[THH21] Tang H. S., Haynes R. D., Houzeaux G.: A review of domain decomposition methods for simulation of fluid flows: Concepts, algorithms, and applications. *Archives of Computational Methods in Engineering 28*, 3 (May 2021), 841–873. https://doi.org/10.1007/s11831-019-09394-0.

[TMPS03] Treuille A., McNamara A., Popović Z., Stam J.: Keyframe control of smoke simulations. *ACM Transactions on Graphics 22*, 3 (July 2003), 716–723. https://doi.org/10.1145/882262.882337.

[UBF*20] Um K., Brand R., Fei Y. R., Holl P., Thuerey N.: Solver-in-the-loop: Learning from differentiable physics to interact with iterative PDE-solvers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems* (Red Hook, NY, USA, Dec. 2020), NIPS'20, Curran Associates Inc., pp. 6111–6122. https://doi.org/10.48550/arXiv.2007.00016.

[WBSS04] Wang Z., Bovik A., Sheikh H., Simoncelli E.: Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (Apr. 2004), 600–612. https://doi.org/10.1109/TIP.2003.819861.

[XZ23] Xu J., Zheng Z.: Gradient-based simulation optimization algorithms via multi-resolution system approximations. *INFORMS Journal on Computing* (Mar. 2023). https://doi.org/10.1287/ijoc.2023.1279.

[YC17] Yang H., Cai X.-C.: Two-level space–time domain decomposition methods for flow control problems. *Journal of Scientific Computing 70*, 2 (Feb. 2017), 717–743. https://doi.org/10.1007/s10915-016-0263-0.

[ZB05] Zhu Y., Bridson R.: Animating sand as a fluid. *ACM Transactions on Graphics 24*, 3 (July 2005), 965–972. https://doi.org/10.1145/1073204.1073298.

[ZCD*24] Zhou J., Chen D., Deng M., Deng Y., Sun Y., Wang S., Xiong S., Zhu B.: Eulerian-Lagrangian fluid simulation on particle flow maps. *ACM Transactions on Graphics 43*, 4 (July 2024), 76:1–76:20. https://doi.org/10.1145/3658180.

[ZG11] Zaspel P., Griebel M.: Massively parallel fluid simulations on Amazon's HPC Cloud. In *2011 First International Symposium on Network Cloud Computing and Applications* (Toulouse, France, Nov. 2011), pp. 73–78. https://doi.org/10.1109/NCCA.2011.19.

[ZM18] Zheng Y., Marguinaud P.: Simulation of the performance and scalability of message passing interface (MPI) communications of atmospheric models running on exascale supercomputers. *Geoscientific Model Development 11*, 8 (Aug. 2018), 3409–3426. https://doi.org/10.5194/gmd-11-3409-2018.

Supporting Information

Additional supporting information may be found online in the Supporting Information section at the end of the article.

Video S1

14678659, 0, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/cgf.70226, Wiley Online Library on [26/10/2025]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License