Efficient Solver for Spacetime Control of Smoke

Zherong Pan  Dinesh Manocha

Department of Computer Science
the University of North Carolina

We present a novel algorithm to control the physically-based animation of smoke. Given a set of keyframe smoke shapes, we compute a dense sequence of control force fields that can drive the smoke shape to match several keyframes at certain time instances. Our approach formulates this control problem as a PDE-constrained spacetime optimization. In order to compute the locally optimal control forces, we alternatively optimize the velocity fields and density fields using an ADMM optimizer. In order to reduce the high complexity of multiple passes of fluid resimulation during velocity field optimization, we utilize the coherence between consecutive fluid simulation passes. We demonstrate the benefits of our approach by computing accurate solutions on 2D and 3D benchmarks. In practice, we observe up to an order of magnitude improvement over prior optimal control methods.

Additional Key Words and Phrases: Fluid Simulation, Optimal Control

1. INTRODUCTION

Physically-based fluid animations are widely used in computer graphics and related areas. Over the past few years, research in fluid simulation has advanced considerably and it is now possible to generate plausible animations for movies and special effects in a few hours on current desktop systems. In this paper, we mainly deal with the problem of the keyframe-based spacetime control of smoke, a special kind of fluid. Given a set of keyframe smoke shapes, our goal is to compute a dense sequence of control forces such that the smoke can be driven to match these keyframes at certain time instances. This problem is an example of directable animation and arises in different applications, including special effects [Rasmussen et al. 2004] (to model a character made of liquid) or artistic animations [Angelidis et al. 2006] (to change the moving direction of the smoke plume). Some of these control techniques, such as [Nielsen and Bridson 2011], are used in the commercial fluid software.

In practice, the keyframe-based control of fluids is still regarded as a challenging problem. Unlike fluid simulation, which deals with the problem of advancing the current fluid state to the next one by time integrating the Navier-Stokes equations, a fluid controller needs to consider an entire sequence of fluid states that results in a high dimensional space of possible control forces. For example, to control a 3D smoke animation discretized on a uniform grid at resolution $64^3$ with 60 timesteps, the dimension of the resulting space of control forces can be as high as $10^8$. The problem of computing the appropriate control force sequence in such a high dimensional space can be challenging for any continuous optimization algorithm. Furthermore, the iterative computation of control forces would need many iterations, each of which involves solving a 2D or 3D fluid simulation problem that can take hours on a desktop system.

Fluid control problems have been well studied in computer graphics and animation. At a broad level, prior techniques can be classified into proportional-derivative (PD) controllers and optimal controllers. PD controllers [Fattal and Lischinski 2004; Shi and Yu 2005] guide the fluid body using additional ghost force terms that are designed based on a distance measure between the current fluid shape and the keyframe. On the other hand, optimal controllers [Treuille et al. 2003; McNamara et al. 2004] formulate the problem as a spacetime optimization over the space of possible control forces constrained by the fluid governing equations, i.e., the Navier-Stokes equations. The objective function of this optimization formulation consists of two terms: The first term requires the fluid shape to match the keyframe shape at certain time instances, while the second term requires the control force magnitudes to be as small as possible.

Optimal controllers are advantageous over PD controllers in that they are less sensitive to the choice of the parameters and they search for the control forces with the smallest possible magnitude, which usually provides smoother keyframe transitions as well as satisfies the fluid dynamic constraints. Treuille et al. [2003] and McNamara et al. [2004] use a simple gradient-based optimizer to search for control forces. This method is easy to implement, but can be computationally inefficient since each gradient evaluation needs to solve a fluid simulation problem. Such repeated fluid simulations slow down the overall performance. In the original work [Treuille et al. 2003], this issue is alleviated by reducing the dimension of the search space using a set of control force templates. However, this treatment also restricts the amount of fluid-like details in the controlled animations.

We present a new, efficient optimization algorithm for controlling smoke. Our approach exploits the special structure of the Navier-Stokes equations discretized on a regular staggered grid, and solves the optimization problem by finding the stationary point of the first order optimality (Karush-Kuhn-Tucker) conditions [Nocedal and Wright 2006]. However, unlike prior methods [Treuille et al. 2003; McNamara et al. 2004] that only solve for the primal variables, we maintain both the primal and dual variables (i.e., the dual variables corresponding to the fluid equations). We present a new, efficient optimization algorithm for controlling smoke. Our approach exploits the special structure of the Navier-Stokes equations and solves the optimization problem by finding the stationary point of the first order optimality (Karush-Kuhn-Tucker) conditions [Nocedal and Wright 2006]. However, unlike prior methods [Treuille et al. 2003; McNamara et al. 2004] that only solve for the primal variables, we maintain both the primal and dual variables (i.e., the dual variables corresponding to the fluid equations).
Lagrangian multipliers). By maintaining the additional dual variables, we can iteratively update our solution without requiring it to satisfy the Navier-Stokes equations exactly in each iteration. Therefore, we alternatively update the velocity fields and control force fields using the alternating direction method of multiplier (ADMM) [Boyd et al. 2011]. In order to update the control force fields efficiently without fluid resimulation, we present a nonlinear multigrid solver: the full approximation scheme (FAS) [Brandt and Livne 2011]. Using a novel spacetime smoothing operator that takes all the timesteps into consideration, our multigrid can converge within a number of iterations independent of the grid resolution and the number of timesteps.

We have evaluated our approach on several benchmarks. Our benchmarks vary in terms of the grid resolution, the number of timesteps, and the control force regularization parameter. We highlight results with up to 60 timesteps at the resolution of $64^3$. Without using force templates, we allow each component of the velocity field to be controlled. In practice, our algorithm can converge faster than a gradient-based quasi-Newton optimizer [Nocedal and Wright 2006] for similar accuracy. An example of achieved smooth transitions between keyframes is illustrated in Figure 1.

2. RELATED WORK

In this section, we give a brief overview of prior techniques for fluid simulation, multigrid solvers and animation control algorithms.

Fluid simulation has been an active area of research in both computer graphics and computational fluid dynamics. The simulation of fluid is typically solved by a discretized time integration of the Navier-Stokes equations or their equivalent forms. At a broad level, prior fluid simulators can be classified into Lagrangian or Eulerian solvers according to the discretization of the convection operator. In order to model smoke and fire, a purely Eulerian solver [Fedkiw et al. 2001] is the standard technique. In terms of free-surface flow, hybrid Lagrangian-Eulerian representation [Zhu and Bridson 2005] has been mostly used in computer graphics. In our work, we confine ourselves to the control of fluids without free-surface, i.e., smoke or fire. We use [Harlow and Welch 1965] as our underlying fluid simulator.

Multigrid solvers are widely used for fluid simulation. Multigrid is a long-standing concept that has been widely used to efficiently solve linear systems discretized from elliptic partial differential equations (see [Brandt and Livne 2011]). This idea has been successfully applied to fluid simulation [Chentanez et al. 2007; Chentanez and Müller 2011; Zhang and Bridson 2014] to find the solenoidal component of the velocity field. In terms of PDE-constrained optimization and control theory, the idea of multigrid acceleration has been extended to the spatial temporal domain. Borzi and Griesse [2005] proposed a semi-coarsening spacetime multigrid to control the time-dependent reaction-diffusion equation. Hinze et al. [2012] used a spacetime multigrid to solve the velocity tracking problem governed by the Navier-Stokes equations. The nonlinear multigrid used in our method is closely related to [Hinze et al. 2012], which also solves a spacetime optimization problem. However, our method is a spatial-only multigrid with the smoothing operator that handles all the timesteps at once. Moreover, unlike [Hinze et al. 2012], which solves a velocity tracking problem, our formulation is a density tracking problem. Therefore, the multigrid is used as a subproblem solver in our ADMM optimization framework.

Fluid control problems tend to be challenging and computationally demanding. Compared to other kinds of animations, e.g., character locomotion [Mordatch et al. 2012], the configuration space of fluid body is of much higher dimension. Prior work in this area can be classified into two categories: PD controllers [Fattal and Lischinski 2004; Shi and Yu 2005] and optimal controllers [Treuille et al. 2003; McNamara et al. 2004]. PD controllers compute the control forces by considering only the configuration of the fluid at the current and next time instance. For example, in [Shi and Yu 2005], a PD controller is used where the control forces are made proportional to the error between the current fluid shape and the target keyframe shape. Similar ideas are used for controlling smoke [Fattal and Lischinski 2004] and liquid [Shi and Yu 2005; Raveendra et al. 2012]. In contrast, optimal controllers search for a sequence of control forces that minimize an objective function. Prior methods [Treuille et al. 2003; McNamara et al. 2004] typically solve spacetime optimization over a high-DOF search space to compute such control forces. Recently, these two methods have been combined [Pan et al. 2013] by first optimizing for the fluid shape at each keyframe and then propagating the changes to the neighboring timesteps. Fluid control can also be achieved by combining or modifying the results of existing fluid simulation data [Raveendran et al. 2014] or guiding fluid using a designed or captured low-resolution animation [Nielsen and Bridson 2011; Nielsen and Christensen 2010; Gregson et al. 2014].

Our algorithm is based on spacetime optimization, similar to [Treuille et al. 2003; McNamara et al. 2004]. In order to solve this optimization problem, we use the ADMM method [Boyd et al. 2011]. This solver has also been previously used in [Gregson et al. 2014] for fluid capturing and guiding. However, our method differs from these previous works in three ways. First, unlike [McNamara et al. 2004], whose method solves for primal variables only, we use a primal-dual formulation. This treatment does not require the Navier-Stokes equations to be satisfied exactly in each iteration of optimization. Moreover, to solve fluid tracking problem, [Gregson et al. 2014] uses ADMM method to address incompressible constraints in one timestep. While we use ADMM method for spacetime optimization taking all timesteps into consideration. Finally, [Gregson et al. 2014] considers the linear solenoidal constraints as hard constraints. In our work, we take into account the entire nonlinear fluid governing equations as the hard constraints, resulting in a nonlinear subproblem that is solved using a multigrid method.

3. FLUID CONTROL

In this section, we formulate the spacetime fluid control problem based on fluid dynamics (Section 3.1) and optimal control theory (Section 3.2). The set of symbols used throughout the paper can be found in Figure 3, and the subscript $i$ is the timestep index. In general, we are dealing with a dynamic system whose configuration space is denoted as $s_i$ at physical time $\Delta t$. Consecutive configurations $s_i$ and $s_{i+1}$ are related by the partial differential equation denoted as the function $f$: $s_{i+1} = f(s_i, u_i, \Delta t)$, where $u_i$ is the control input. An optimal controller computes a set of control inputs $\{u_i | i = 0, \ldots, N-1\}$ that minimize the objective function denoted as function $E(s_0, \ldots, s_N)$. The overall optimal control problem is specified using the pair of functions $f$ and $E$. In the case of smoke control problems, $f$ is a discretization of the Navier-Stokes equations, and $E$ measures the difference between the smoke and keyframe shapes at certain time instances.
3.1 Fluid Dynamic System

In our work, we restrict ourselves to the control of incompressible fluids without a free surface. Fluids such as smoke and fire, which are commonly used in movies and animations, fall into this category. We omit viscous terms for brevity. Small viscosity can under such discretization can be handled by a slight modification to \( f \), which does not increase the complexity of our algorithm. Following [Harlow and Welch 1965; Pavlov et al. 2011], we discretize the velocity-vorticity version of the Navier-Stokes equations using finite difference scheme and backward Euler time integrator for advection. Our configuration space \( s_i = \{ v_i^T, p_i^T, p_i^T \} \) concatenates the velocity field \( v_i \), the kinetic pressure field \( p_i \), and the density or dye field \( \rho_i \). These scalar and vector fields are discretized on a staggered grid, which has been widely used by previous works such as [Fedkiw et al. 2001]. The transfer function \( f \) under such discretization can be represented as:

\[
\frac{v_{i+1} - v_i}{\Delta t} + \text{Adv}[v_{i+1}] = u_i - \nabla \rho_{i+1},
\]

\[
\nabla \cdot v_{i+1} = 0,
\]

\[
\rho_{i+1} = A[\rho_i, v_i],
\]

where the self-advection operator \( \text{Adv}[\cdot] \) is a discretization of the quadratic operator \( \nabla \times \bullet \times \bullet \) and we assume constant unit fluid density. The pressure field \( \rho_{i+1} \) is identified with the Lagrangian multiplier of the divergence free constraints \( \nabla \cdot v_{i+1} = 0 \). Finally, the operator \( A[\bullet, \bullet] \) is the passive scalar advection operator discretized as:

\[
\rho_{i+1} = e^{A(v_i) \Delta t} \rho_i,
\]

where matrix \( A(v_i) \) is the second order upwinding stencil [Leonard 1979]. By approximating the matrix exponential using Taylor series, the advection operator can be defined as:

\[
A[\rho_i, v_i] = \sum_{k=0}^{\infty} \frac{\Delta t^k}{k!} A(v_i)^k \rho_i.
\]

When \( k \) tends to infinity, this upwinding advection operator is unconditionally stable since \( A(v_i) \) is skew-symmetric, so that \( e^{A(v_i) \Delta t} \) is an orthogonal matrix and \( \| \rho_{i+1} \| = \| \rho_i \| \). In practice, we truncate \( k \) to a finite value. Specifically, we set \( k \) adaptively to be the smallest integer satisfying \( \frac{\Delta t}{k!} A(v_i)^k \rho_i < 10^{-5} \). Although this operator is computationally more expensive than the widely used semi-Lagrangian operator, it generates smoother controlled animations with large timestep size, as shown in Figure 4. This is useful when fewer timesteps are used to reduce the runtime cost.

3.2 Spacetime Optimization

The optimal control of the dynamic system, discussed in Section 3.1, can be formulated as a spacetime optimization over the configuration trajectory \( s_0, s_1, \ldots, s_N \). Our objective function is similar to the ones proposed in prior works [Treuille et al. 2003; McNamara et al. 2004] that try to match \( \rho_i \) to a set of keyframes \( \rho_i^* \) while minimizing the magnitude of control forces \( u_i \). The overall optimization problem can be formulated as:

\[
\begin{align*}
\text{argmin} & \quad E = \frac{1}{2} \sum_{i=0}^{N} c_i \| \rho_i - \rho_i^* \|^2 + \frac{r}{2} \sum_{i=0}^{N-1} \| u_i \|^2 \\
\text{s.t.} & \quad \Delta t_k = f(s_i, u_i, \Delta t),
\end{align*}
\]

where \( c_i = 1 \) if there is a keyframe \( \rho_i^* \) at frame \( i \) and 0 otherwise. \( r \) is the regularization coefficient of the control forces.

Treuille et al. [2003] and McNamara et al. [2004] solve this optimization by eliminating the transfer function \( f \) and plugging them into the objective function. Although this reformulation simplifies the problem into an unconstrained optimization, their new objective function takes a much more complex form, which is a long chain of function compositions. To minimize the new objective function, Treuille et al. [2003] and McNamara et al. [2004] use a general-purpose gradient-based optimizer. A typical gradient-based optimizer such as the quasi-Newton method [Byrd et al. 1995] requires repeated gradient calculation to approximate the Hessian matrix and performs line search to compute the stepsize. Each such gradient calculation requires a fluid resimulation, which becomes the major bottleneck in their algorithm.

3.3 Our Approach

Prior methods require that the solution computed during each iteration should satisfy the Navier-Stokes equations exactly, i.e., is a feasible solution. As a result, each iteration takes considerable running time. In practice, this requirement can be overly conservative because we only need to ensure that the final computed solution at the end of the algorithm is feasible. Thus, we can relax this requirement during the intermediate steps, and only need to ensure that the final solution lies in the feasible domain. This is a well-known idea and has been used by many other numerical optimization algorithms such as the interior point method [Nocedal and Wright 2006].
Based on this observation, we design a new optimization algorithm illustrated in Figure 2. We first notice that our objective function is essentially constrained by two kinds of partial differential equations: the passive advection (Equation 3) governing the time evolution of the density field $\rho_i$ and the incompressible Navier-Stokes (Equation 1 and Equation 2) governing the time evolution of the velocity field $v_i$. We introduce a set of slack variables to break these two kinds of constraints into two subproblems: Advection Optimization (AO) is constrained only by Equation 3 and Navier-Stokes Optimization (NSO) is constrained only by Equation 1 and Equation 2. In order to solve the Advection Optimization (Section 4.1), we use a fixed point iteration defined for its KKT conditions. For the Navier-Stokes Optimization (Section 4.2), we update our solution using the full approximation scheme (FAS) to avoid repeated fluid resimulations. This leads to considerable speedup over prior methods, not only because of the fast convergence of our multigrid solver, but also because the multigrid solver allows warm-starting so that we can utilize coherence between consecutive iterations. In contrast, previous methods use fluid resimulations, which always solve Navier-Stokes equations from scratch, and solve them exactly.

4. SPACETIME OPTIMIZATION

In this section, we present our novel algorithm to solve Equation 5. By introducing a series of slack variables $v_i^*$, we can decompose the overall optimization problem into two subproblems and reformulate Equation 5 as:

$$\text{argmin}_{u_i} \frac{1}{2} \sum_{i=0}^{N} c_i \|\rho_i - \rho_i^*\|^2 + \frac{r}{2} \sum_{i=0}^{N} \|u_i\|^2 + \lambda_i^T (v_i - v_i^*) + \frac{K}{2} \sum_{i=0}^{N-1} \|v_i - v_{i-1}\|^2 \quad (6)$$

s.t. \[ \frac{v_{i+1} - v_i}{\Delta t} + \text{Adv}(v_{i+1}) = u_i - \nabla \rho_{i+1} \]

\[ \rho_{i+1} = A[\rho_i, v_i^*], \quad \nabla \cdot v_i = 0, \]

where we added the augmented Lagrangian term $\lambda_i^T (v_i - v_i^*)$ and the penalty term $K \sum_{i=0}^{N-1} \|v_i - v_{i-1}\|^2$. This kind of optimization can be solved efficiently using the well-known alternating direction method of multipliers (ADMM) [Boyd et al. 2011]. Specifically, in each iteration of our algorithm, we first fix $v_i, \rho_i$ and solve for $v_i^*$. This subproblem is denoted as the Advection Optimization (AO) because the PDE constraints are just passive advection of the density field $\rho_i$. We then fix $v_i^*$ and solve for $v_i, \rho_i$. We denote this subproblem as the Navier-Stokes Optimization (NSO), constrained by the incompressible Navier-Stokes equations. The final step is to adjust $\lambda_i$, according to the constraint violation as: $\lambda_i = \lambda_i + K\beta (v_i - v_i^*)$ where $\beta$ is a constant parameter.

The idea of introducing slack variables to relax hard constraints has been used in several previous works, e.g., for fluid tracking [Gregson et al. 2014] and projective dynamic simulation [Narain et al. 2016]. The advantage of decomposing the problem up is that we can derive simple and effective algorithms to solve each subproblem. Our algorithm directly solves the first order optimality (KKT) conditions of both problems. To solve the AO subproblem, we introduce a fixed point iteration in Section 4.1, while for the NSO subproblem, which is the bottleneck of the algorithm, we introduce the nonlinear multigrid solver (FAS) in Section 4.2.

4.1 Advection Optimization

The goal of solving the AO subproblem is to find a sequence of velocity fields $v_i^*$ to advect $\rho_i$ so that it matches the keyframes, assuming that these $v_i^*$ are uncorrelated. By dropping terms irrelevant to $v_i^*$ from Equation 6, we get a concise formulation for the AO subproblem:

$$\text{argmin}_{v_i^*} \frac{1}{2} \sum_{i=0}^{N} (\rho_i - \rho_i^*)^T C_i (\rho_i - \rho_i^*) + K \sum_{i=0}^{N-1} \|v_i + \lambda_i/K - v_i^*\|^2$$

s.t. \[ \rho_{i+1} = A[\rho_i, v_i^*], \quad \nabla \cdot v_i^* = 0, \]

where we can absorb the augmented Lagrangian term $\lambda_i^T (v_i - v_i^*)$ by setting: $v_i^* = v_i + \lambda_i/K$.

Due to the inherent nonlinearity and ambiguity in the advection operator, an AO solver is prone to falling into local minimum, leading to trivial solutions. We introduce two additional modifications to Equation 7 to avoid these trivial solutions. First, we replace the scalar coefficient $c_i$ with a matrix $C_i$ which could be used to avoid the problem of a zero gradient if the keyframe $\rho_i^*$ is far from the given density field $\rho_i$. Similar to [Treuille et al. 2003; Fattal and Lischinski 2004], we use the idea of Gaussian Pyramid [Adelson et al. 1984] and take $C_i = c_i \Sigma_k G_k^T G_k$ to be a series of Gaussian filters $G_k$ with receding support. Specifically, $G_k$ has a standard deviation of $\sigma(G_k) = 2\sigma(G_{k-1})$. The Gaussian Pyramid makes our method almost resolution invariant, since any local error in the density field will always lead to a non-zero gradient value at every point in the grid domain. We also introduce additional solenoidal constraints on $v_i^*$. Note that this term does not alter the optima of Equation 6 since $v_i = v_i^*$ on convergence. However, it prevents the optimizer from creating or removing densities in order to match the keyframe, which is a tempting trivial solution.

We solve this optimization via a fixed point iteration derived from its KKT conditions. To derive this system we introduce Lagrangian multipliers $\mu_i$ for each advection equation $\mu_{i+1} = A[\rho_i, v_i^*]$ and $\gamma_i$ for the solenoidal constraints, giving a Lagrangian function:

$$L = \frac{1}{2} \sum_{i=0}^{N} (\rho_i - \rho_i^*)^T C_i (\rho_i - \rho_i^*) + \frac{K}{2} \sum_{i=0}^{N-1} \|v_i - v_i^*\|^2 + \sum_{i=0}^{N-1} \mu_i^T (\rho_i - A[v_i, v_{i-1}^*]) + \gamma_i^T \nabla \cdot v_i^*$$

After taking the derivative of the above Lagrangian against $\rho_i, v_i^*$ (primal variables) and $\mu_i, \gamma_i$ (dual variables), respectively, we get the following set of KKT conditions for $0 \leq i \leq N$:

$$\mu_{i+1} = \frac{\partial A[\rho_i, v_i^*]}{\partial \rho_i} \mu_i - C_i (\rho_i - \rho_i^*)$$

$$v_i^* = Q(v_{i+1} = \frac{\partial A[v_i, v_i^*]}{\partial v_i} \mu_{i+1} - \frac{K}{2} )$$

$$\mu_{i+1} = A[\rho_i, v_i^*], \quad \nabla \cdot v_i^* = 0,$$
Algorithm 1 The Fixed Point Iteration: This is used to solve the AO subproblem. The algorithm consists of a forward sweep that updates the density fields $\rho_i$ and a backward sweep that updates $\mu_i$ and $v_i^*$.

**Input:** Initial $v_i, \rho_i, \alpha \in (0, 1)$, and keyframes $\rho_i^*$.

**Output:** Fixed point solution $v_i^*, \mu_i$.

1: $E \leftarrow \infty$
2: for $i = 0, \ldots, N - 1$ do
3: $\triangleright$ Initialization
4: $v_i^* \leftarrow v_i$
5: $v_i^{**} \leftarrow v_i$
6: end for
7: while not converged do
8: $\triangleright$ Forward pass
9: for $i = 1, \ldots, N$ do
10: $\triangleright$ Find primal variables $\rho$
11: Find smallest $k$ such that $\frac{1}{K} A(v_i^{*})^k \rho_i < 10^{-5}$
12: $\mu_i \leftarrow A [\rho_i, v_i^{**}]$
13: end for
14: $\triangleright$ Ensure function value decrease
15: $E_{\text{new}} \leftarrow \frac{1}{2} \sum_{i=0}^{N} \frac{|\rho_i - \rho_i^*|}{\bar{c}_i + \frac{1}{2} \sum_{j=0}^{N} |v_i - v_i^*|^2}$
16: if $E_{\text{new}} < E$ then
17: $E \leftarrow E_{\text{new}}$
18: for $i = 1, \ldots, N - 1$ do
19: $v_i^{**} \leftarrow v_i^*$
20: end for
21: increase $\alpha$
22: else
23: for $i = 1, \ldots, N - 1$ do
24: $v_i^{**} \leftarrow v_i^*$
25: end for
26: decrease $\alpha$
27: goto Line 8
28: end if
29: $\triangleright$ Backward pass
30: set $\mu_{i-1} \leftarrow 0, \mu_N \leftarrow 0$
31: for $i = N - 1 \ldots, 1$ do
32: $\triangleright$ Find dual variables $\mu$
33: $\mu_{i-1} \leftarrow \frac{\partial A[\rho_i, v_i^*]}{\partial v_i} T \mu_i - C_i (\rho_i - \rho_i^*)$
34: $\triangleright$ Find primal variables $v$
35: $v_i^{**} \leftarrow (1 - \alpha) v_i^{**} + \alpha Q(v_i) + \frac{\partial A[v_i^*]}{\partial v_i} T \mu_i$
36: end for
37: end while

way as in Section 4.1, we derive the KKT conditions and assemble them into a set of nonlinear equations:

$$f = \left\{ \frac{\partial K(v_i - v_i^*)}{\partial v_i} + \frac{\partial u_i}{\partial v_i} T \cdot u_i + \frac{\partial u_{i-1}}{\partial v_i} T \cdot u_{i-1} + \nabla \bar{p}_i \right\} \nabla \cdot v_i = 0 \right\}$$

where the partial derivatives are \( \frac{\partial u_i}{\partial v_i} = -\frac{1}{\Delta t} \) and \( \frac{\partial u_{i-1}}{\partial v_i} = \frac{1}{\Delta t} + \frac{\partial A (v_i)}{\partial v_i} \), and the additional variable $\bar{p}_i$ is the Lagrangian multiplier for the solenoidal constraint $\nabla \cdot v_i = 0$. We refer readers to Appendix A for the derivation of Equation 9. In summary, we have to solve for the primal variables $u_i, v_i$ as well as the dual variables $p_i, \bar{p}_i$. Unlike Equation 8, however, we do not differentiate these
two sets of variables and solve for them by iteratively bringing the residual $f$ to zero.

To adopt this idea to solve Equation 9, we introduce a hierarchy of spatial grids $(v^h, \bar{p}^h, u^h, p^h)$, where $h$ is the cell size. We use semi-coarsening in spatial direction only where every coarser level doubles the cell size. We denote the coarser level as $(v^{2h}, \bar{p}^{2h}, u^{2h}, p^{2h})$, and use the simple FAS-VCycle(2,2) iteration to solve the nonlinear system of equations: \(f(v, \bar{p}, u, p) = \text{res} \). See Algorithm 2 for details of the NSO solver.

**Algorithm 2** FAS VCycle($v^h, \bar{p}^h, u^h, p^h, \text{res}^h$): This is used to solve the NSO subproblem. The algorithm is a standard FAS-VCycle with 2 pre and post smoothing (Line 8, Line 28) and 10 final smoothing (Line 3).

**Input:** A tentative solution $(v^h, \bar{p}^h, u^h, p^h)$

**Output:** Refined solution to $f(v^h, \bar{p}^h, u^h, p^h) = \text{res}^h$

1: if $h$ is coarsest then
2:▷ Final smoothing for the coarsest level
3: for $k = 1, \ldots, 10$ do
4: $S(v^h, \bar{p}^h, u^h, p^h)$
5: end for
6: else
7:▷ Pre smoothing
8: for $k = 1, 2$ do
9: $S(v^h, \bar{p}^h, u^h, p^h)$
10: end for
11:▷ Down-sampling
12: for $t = v, \bar{p}, u, p$ and do
13: $t^{2h} \leftarrow R(t^h)$
14: $t^h = t^h - P(t^{2h})$
15: end for
16:▷ Compute FAS residual by combining:
17:▷ 1. the solution on coarse resolution
18:▷ 2. the residual on fine resolution
19: $\text{res}^{2h} = f(v^{2h}, \bar{p}^{2h}, u^{2h}, p^{2h})$
20: $\text{res}^{2h} = \text{res}^{2h} - R(\text{res}^h - f(v^h, \bar{p}^h, u^h, p^h))$
21:▷ VCycle recursion
22: $\text{VCycle}(v^{2h}, \bar{p}^{2h}, u^{2h}, p^{2h}, \text{res}^{2h})$
23:▷ Up-sampling
24: for $t = v, \bar{p}, u, p$ do
25: $t^h = t^h + P(t^{2h})$
26: end for
27:▷ Post smoothing
28: for $k = 1, 2$ do
29: $S(v^h, \bar{p}^h, u^h, p^h)$
30: end for
31: end if

The fast convergence of the geometric FAS relies on a proper definition of the three application-dependent operators: R, P and S. The restriction operator R downsamples a fine grid solution to a coarser level for efficient error reduction, and the prolongation operator P upsamples the coarse grid solution to correct the fine grid solution. We use simple trilinear interpolation for these two operators whether applied on scalar or vector fields. Finally, designing the smoothing operator S is much more involved. S should, by itself, be a cheap iterative solver for $f(v, \bar{p}, u, p) = \text{res}$. Compared with previous works such as [Chentanez and Müller 2011] where multigrid is used for solving the pressure field $p$ only, we are faced with two new challenges. First, since we are solving the primal as well as dual variables, which gives a saddle point problem, the Hessian matrix is not positive definite in the spatial domain, so that a...
Jacobi or Gauss-Seidel (GS) solver does not work. Second, we are not coarsening in the temporal domain, so the temporal correlation must be considered in the smoothing operator.

Our solution is to consider the primal and dual variables at the same time using the Symmetric Coupled Gauss-Seidel (SCGS) smoothing operator [Vanka 1983]. SCGS smoothing is a primal-dual variant of GS. In our case, where all the variables are stored in a staggered grid, SCGS smoothing considers one cell at a time. It solves the primal variables \(v, u\) stored on the 6 cell faces as well as the dual variables \(p, \bar{p}\) stored in the cell center at the same time by solving a small \(14 \times 14\) linear problem \((10 \times 10 \text{ in } 2D)\). Like red-black-GS smoothing, we can parallelize SCGS smoothing using the 8-color tagging (see Figure 5).

The above SCGS solver only considers one timestep at a time. To address the second problem of temporal correlation, we augment the SCGS solver with the temporal domain. We solve the 14 variables associated with a single cell across all the timesteps at once. Although this involves solving a large \(14N \times 14N\) linear system for each cell, the left hand side of the linear system is a block tridiagonal matrix so that we can solve the system in \(O(N)\). Indeed, the Jacobian matrix of \(f\) takes the following form:

\[
\frac{\partial f}{\partial v, p, u, \bar{p}} =\begin{pmatrix}
\begin{bmatrix}
\frac{\partial f}{\partial v_0} & \frac{\partial f}{\partial u_0} & \frac{\partial f}{\partial \bar{p}_0} \end{bmatrix}^T \\
\frac{\partial f}{\partial v_1} & \frac{\partial f}{\partial u_1} & \frac{\partial f}{\partial \bar{p}_1} \\
\frac{\partial f}{\partial v_2} & \frac{\partial f}{\partial u_2} & \frac{\partial f}{\partial \bar{p}_2} \\
\end{bmatrix}
\end{pmatrix},
\]

where the size of each block is \(5 \times 5\) in 2D and \(7 \times 7\) in 3D. Due to this linear time solvability, the optimal multigrid performance is still linear in the number of spatial-temporal variables. The average convergence history for our multigrid solver with a conventional LBFGS algorithm [McNamara et al. 2004] in Figure 6. Our algorithm achieves a stable linear rate of error reduction independent of both the grid resolution and the number of timesteps.

5. RESULTS AND ANALYSIS

Algorithm 3 ADMM Outer Loop

<table>
<thead>
<tr>
<th>Input</th>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>(\Delta t)</td>
<td>0.4 (\sim) 2.0s</td>
</tr>
<tr>
<td></td>
<td>(K)</td>
<td>(10^3)</td>
</tr>
<tr>
<td></td>
<td>(r)</td>
<td>(10^{-4})</td>
</tr>
<tr>
<td></td>
<td>(\beta) for updating (\lambda)</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Equation 8</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>(\epsilon_{\text{FAS}})</td>
<td>(10^{-5})</td>
</tr>
<tr>
<td></td>
<td>(\epsilon_{\text{ADMM}})</td>
<td>(\frac{p_{\text{max}}}{\rho_{\text{max}}})</td>
</tr>
</tbody>
</table>

Table I. Parameters.

Parameter Choice: We use the same set of parameters listed in Table I for all experiments, where \(p_{\text{max}}\) is the maximal density magnitude at the initial frame. In our experiments, the ADMM algorithm always converges in fewer than 50 iterations. Further, running only 2 iterations of Equation 8 in each ADMM loop will not deteriorate the performance.
fact, according to the averaged convergence history of the AO subproblem illustrated in Figure 7, the fixed point iteration Equation 8 usually converges in the first 4 iterations before it wanders around a local minimum. After fine tuning, we found that 2 iterations lead to the best overall performance. In this case, the overhead of solving the AO subproblem is marginal compared with the overhead of solving the NSO subproblem. Finally, unlike fluid simulation, the performance of spacetime optimization does not depend on the timestep size due to our robust advection operator (Equation 4). When we increase the timestep size from 0.4s to 2s for the examples in Figure 8 and Figure 9, which is extremely large, our algorithm’s convergence behavior is about the same. Under this setting, the convergence history of the ADMM outer loop for our first example (Figure 1) is illustrated in Figure 7. The convergence history can be decomposed into two stages. In the first stage, the first term of Equation 5 (keyframe shape matching) dominates, the solver gradually evolves the solution to match the keyframe shape, and the KKT-Residual is not monotonically decreasing. In the second stage, however, the second term (control force regularization) dominates Equation 5 and the KKT-Residual quickly decreases. Since the solutions of consecutive ADMM iterations do not change much, we have also tried to use just a few SCGS smoothing steps, instead of the entire FAS Algorithm 2, to approximately solve the NSO subproblem. In practice, we observed this treatment smoothed out the fluid-like behaviors, when large regularization $r$ is used.

**Benchmarks:** To demonstrate the efficiency and robustness of our algorithm, we used 7 benchmark problems that vary in their grid resolution, number of timesteps, and number of keyframes. The memory overhead and computational overhead are summarized in Table II. All of the results are generated on a desktop PC with an i7-4790 8-core CPU 3.6GHz and 12GB of memory. We use OpenMP for multithread parallelization.

Our first example is five controlled animations matching a circle to the letters “FLUID”. Compared with [Treuille et al. 2003], which uses a relatively small set of control force templates to reduce the search space of control forces, we allow control on every velocity component so that the matching to keyframe is almost exact. After the keyframe, we remove the control force, and rich smoke details are generated by pure simulation as illustrated in Figure 1. However, in the controlled phase of Figure 1, this example seems “too much controlled”, meaning that most smoke-like behaviors are lost. This effect has also been noticed in [Treuille et al. 2003]. However, unlike their method, in which the number of templates needs to be carefully tuned to recover such behavior, we can simply adjust the regularization $r$ in our system to control matching exactness and the amount of smoke-like behaviors. In Figure 8, we generated three animations with two keyframes: first two circles and then a bunny, using $r = 10^{2.3}, 10^{2.6}, 10^{2.8}$ respectively. These animations are also shown in the video. Our algorithm is robust to a wide range of parameter choices. But more iterations are needed for the multigrid to converge for a larger $r$ as shown in Table II. Finally, since we allow every velocity component to be optimized, the resulting animation exhibits lots of small-scale details as indicated in Figure 9, which is not possible with the small set of force templates used in [Treuille et al. 2003].

In addition to these 2D examples, we also tested our algorithm on some 3D benchmarks. Our first example is shown in Figure 10 and runs at a resolution of $64^3/40$. We use two keyframes at frame 20 and 40, and the overall optimization takes about 7 hours. In our second example, shown in Figure 11, we try to track the smoke with a dense sequence of keyframes from the motion capture data of a human performing a punch action. Such an example is considered the most widely used benchmarks for PD-type controllers such as [Shi and Yu 2005]. With such strong and dense guidance, our algorithm converges very quickly, within 5 iterations. Our third example (Figure 13) highlights the effect of regularization coefficient $r$ in 3D. Like our 2D counterpart Figure 8, larger $r$ usually results in more wake flow behind moving smoke bodies. Finally, we evaluated our algorithm on a benchmark with keyframe shapes of varying genera. As illustrated in Figure 12, the initial smoke shape has genus zero, but we use two keyframes, where the smoke shapes...
Our algorithm is approximately an order of magnitude faster than a typical implementation of LBFGS. This is mainly because of the repeated gradient evaluation in each LBFGS iteration, where each evaluation runs the adjoint method result in much lower overhead in terms of fluid resimulation, as compared to our approach based on optimal controllers. In contrast, optimal controllers provide better flexibility and robust solutions as compared to PD controllers. A PD controller tends to be sensitive to the parameters used for the ghost forces. Moreover, its performance also depends on the use of the compressible control forces (see Figure 14). These compressible forces can potentially eliminate the visually appealing vortical fluid motions. On the other hand, an optimal controller always achieves exact keyframe timing, while such exact timing requires fine-tuning the strength of control forces in a PD controller, as shown in Figure 14. Moreover, with an optimal controller, users can easily balance between the exactness of keyframe matching and the amount of fluid-like behaviors based on a single parameter $r$ (see Figure 8).

**Comparison with PD Controller:** We also compared our method with simple tracker type controllers such as PD controller [Fattal and Lischesnk 2004]. To drive the fluid body towards a target keyframe shape using heuristic ghost forces, PD controllers result in much lower overhead in terms of fluid resimulation, as compared to our approach based on optimal controllers. In contrast, optimal controllers provide better flexibility and robust solutions as compared to PD controllers. A PD controller tends to be sensitive to the parameters used for the ghost forces. Moreover, its performance also depends on the use of the compressible control forces (see Figure 14). These compressible forces can potentially eliminate the visually appealing vortical fluid motions. On the other hand, an optimal controller always achieves exact keyframe timing, while such exact timing requires fine-tuning the strength of control forces in a PD controller, as shown in Figure 14. Moreover, with an optimal controller, users can easily balance between the exactness of keyframe matching and the amount of fluid-like behaviors based on a single parameter $r$ (see Figure 8).

**Memory Overhead:** Since fluid control problems usually have a high memory overhead, we derive here an analytical upper bound of the memory consumption $M(n, d, N)$:

$$M(n, d, N) = \left[ \left( n^3 \right) \times \left( 1 + d \right) \times 2 + 2 \right] \times \left[ 1 + \frac{1}{2} + \frac{1}{4} \ldots \right] \times N = 8n^3(1 + d)N,$$

<table>
<thead>
<tr>
<th>Example ($n^3/N$)</th>
<th>Boundary</th>
<th>#ADMM</th>
<th>Avg. AO (s)</th>
<th>Avg. NSO (s)</th>
<th>Total(hr)</th>
<th>Memory(Gb)</th>
<th>Total LBFGS(hr)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Letters “FLUID”(128$^2/40, r = 10^3$)</td>
<td>Neumann</td>
<td>13</td>
<td>10</td>
<td>80</td>
<td>0.25</td>
<td>0.06</td>
<td>4</td>
</tr>
<tr>
<td>Letters “FLUID”(128$^2/80, r = 10^3$)</td>
<td>Neumann</td>
<td>17</td>
<td>21</td>
<td>142</td>
<td>0.76</td>
<td>0.12</td>
<td>9</td>
</tr>
<tr>
<td>Circle Bunny(128$^2/80, r = 10^3$)</td>
<td>Neumann</td>
<td>25</td>
<td>20</td>
<td>130</td>
<td>1.04</td>
<td>0.2</td>
<td>12</td>
</tr>
<tr>
<td>Circle Bunny(128$^2/80, r = 10^4$)</td>
<td>Neumann</td>
<td>37</td>
<td>20</td>
<td>220</td>
<td>2.46</td>
<td>0.2</td>
<td>15</td>
</tr>
<tr>
<td>Circle Bunny(128$^2/80, r = 10^4$)</td>
<td>Neumann</td>
<td>43</td>
<td>20</td>
<td>218</td>
<td>2.84</td>
<td>0.2</td>
<td>16</td>
</tr>
<tr>
<td>Letters ABC(128$^2/60, r = 10^3$)</td>
<td>Neumann</td>
<td>33</td>
<td>16</td>
<td>179</td>
<td>1.78</td>
<td>0.15</td>
<td>14</td>
</tr>
<tr>
<td>Sphere Armadillo Bunny(64$^2/40, r = 10^3$)</td>
<td>Neumann</td>
<td>17</td>
<td>103</td>
<td>1341</td>
<td>6.81</td>
<td>1.34</td>
<td>N/A</td>
</tr>
<tr>
<td>Varying Genus(64$^2 \times 32/40, r = 10^3$)</td>
<td>Periodic</td>
<td>20</td>
<td>82</td>
<td>840</td>
<td>5.12</td>
<td>0.67</td>
<td>N/A</td>
</tr>
<tr>
<td>Human Mocap(64$^2 \times 128/60, r = 10^3$)</td>
<td>Periodic</td>
<td>5</td>
<td>1437</td>
<td>3534</td>
<td>6.9</td>
<td>4.0</td>
<td>N/A</td>
</tr>
<tr>
<td>Moving Sphere(64$^2/60, r = 10^3$)</td>
<td>Neumann</td>
<td>17</td>
<td>630</td>
<td>1792</td>
<td>11.43</td>
<td>2.2</td>
<td>N/A</td>
</tr>
<tr>
<td>Moving Sphere(64$^2/60, r = 10^3$)</td>
<td>Neumann</td>
<td>22</td>
<td>630</td>
<td>1978</td>
<td>15.93</td>
<td>2.2</td>
<td>N/A</td>
</tr>
<tr>
<td>Letters ABC 3D(64$^2/150, r = 10^3$)</td>
<td>Periodic</td>
<td>20</td>
<td>1512</td>
<td>3220</td>
<td>26.28</td>
<td>5.9</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table II.: Memory and computational overhead for all the benchmarks. From left to right: name of example (resolution parameters); the spatial boundary condition; number of outer ADMM iterations; average time spent on each AO subproblem; average time spent on each NSO subproblem; total time until convergence using our algorithm; memory overhead; total time until convergence using LBFGS. By comparing the three “Circle Bunny” examples, we can observe that the number of ADMM outer loops is roughly linear to $\log_2(r)$. More ADMM outer loops are needed, if more fluid-like behaviors are desired. From the two examples of the Letters “FLUID” (Line 1 and Line 2), we can observe that the computational cost of each ADMM outer loop (Avg. AO + Avg. NSO ) is roughly linear in the number of timesteps. This cost is also governed by the number of keyframes. By comparing Line 2 and Line 4, we can observe that the Circle Bunny example which involves two keyframes requires more computation to solve the NSO subproblem.
Fig. 11: We generate the famous example of tracking smoke with a dense sequence of keyframes, which comes from human motion capture data. Our algorithm converges and generates rich smoke drags within 5 ADMM iterations.

Fig. 12: Example of smoke control where the keyframes have varying genera. The initial frame is a sphere (genus 0). The first keyframe located at frame 20 is a torus (genus 1) and the second keyframe located at frame 40 is the shape eight (genus 2). The resolution is $64^2 \times 32/40$ and the overall optimization takes 5hr with $r = 10^3$.

Fig. 13: A moving smoke sphere guided by the 3 keyframes (left). We experimented with $r = 10^3$ (top) and $r = 10^4$ (bottom). Larger regularization results in more wake flow behind moving smoke bodies. The same effect can be observed in Figure 8.

where $n$ is the grid resolution, $d$ is the dimension, and $N$ is the number of timesteps. To derive this bound, note that we can reuse the memory consumed by Algorithm 2 in Algorithm 1, and Algorithm 2 always consumes more memory than Algorithm 1, so that we only consider the memory overhead of Algorithm 2. The first term $n^d \star (1 + d)$ is the number of variables needed for storing a pair of pressure and velocity fields. This number is doubled because we need to store $u_i, p_i$ in addition to $v_i, p_i$ at each timestep. We double it again because we need additional memory for storing the resid in FAS. Finally, the power series is due to the hierarchy of grids. At first observation, this memory overhead is higher than [Treuille et al. 2003; McNamara et al. 2004] since we require additional memory for storing the dual variables at multiple resolutions. However, due to the quasi-Newton method involved in their approach, additional memories are needed to store a set of $L$ gradients to approximate the inverse of the Hessian matrix. $L$ is usually $5 \sim 10$, leading to the following upper bound:

$$M_{LBFGS}(n, d, N) \sim \left( n^d \star (1 + d) \right) \star L \star N = Ln^d(1 + d)N.$$  

In our benchmarks, the memory overheads of our ADMM and LBFGS solvers are comparable.

**Convergence Analysis:** Here we analyze the convergence of our approach and discuss some modifications towards improved convergence of Algorithm 3. We have applied some of these modifications for Line 7 and Line 9 of Algorithm 3, which then takes a slightly more complex form.

For our AO solver (Line 7 of Algorithm 3), we observe that it can be difficult for Algorithm 1 to converge to an arbitrarily small KKT residual in each loop of Algorithm 3. As illustrated in Algorithm 1, one could use a simple strategy that can guarantee function value decreases by blending a new solution with the previous solution and tuning the blending factor in a way similar to the line search algorithm. This modification has low computational overhead since one does not need to apply the costly solenoidal projection operator.

Fig. 14: We deform a smoke ball into a dragon using our method (top left) and PD controller with different parameter settings. Top Right: Without gathering forces, the keyframe shape is not matched. Bottom Left: The animation appears to be oscillatory without viscous forces, resulting in lots of escaped smokes. Bottom Right: A stable, non-oscillatory animation that matches the keyframe well can be achieve by fine-tuning the three parameters: the strength of guiding forces, the strength of gathering forces, and the strength of viscous forces.
Q again after the blending, as the sum of two solenoidal vector fields is still solenoidal. In our benchmarks, this strategy leads to a convergent algorithm with low overhead, but the error reduction rate after the first few iterations can still be slow.

The same analysis can also be used for the NSO solver (Line 9 of Algorithm 3). To ensure convergence of Algorithm 2, we could add a perturbation to the penalty coefficient $K$ in the Hessian matrix Equation 10. Note that as $K \rightarrow \infty$, $v_i \rightarrow v_i^\ast$. Therefore, this strategy essentially makes Algorithm 2 the subproblem solver for the Levenberg-Marquardt algorithm [Nocedal and Wright 2006], which in turn guarantees convergence. As illustrated in Figure 15, Levenberg-Marquardt modification can be necessary when one uses extremely large regularization rates, because we observe that the convergence rate decreases as the perturbation increases. In these settings, however, many more FAS-Vcycles are needed to solve the NSO subproblem and the advantage over the LBGFS solver also decreases.

Finally, for the ADMM outer loop (Line 5 of Algorithm 3), state-of-the-art results showing its convergence rely on strong assumptions of its objective function, such as global convexity. Therefore, our current implementation of outer loop is not guaranteed to converge. However, such guarantee can be provided by using a standard Augmented Lagrangian solver, instead of ADMM solver. Specifically, one can run Algorithm 3 without applying Line 18 until the decrease in function value is lower than some threshold. Further exploration of this option is left as future work.

6. CONCLUSION AND LIMITATIONS

In our work, we present a new algorithm for the optimal control of smoke animation. Our algorithm finds the stationary point of the KKT conditions, solving for both primal and dual variables. Our key idea is to refine primal as well as dual variables in a warm-started manner, without requiring them to satisfy the Navier-Stokes equations exactly in each iteration. We tested our approach on several benchmarks and a wide range of parameter choices. The results show that our method can robustly find the locally optimal control forces while achieving an order of magnitude speedup over the gradient-based optimizer, which performs fluid resimulation in each gradient evaluation.

On the downside, our method severely relies on the spatial structure and the staggered grid discretization of the Navier-Stokes equations. This imposes a major restriction to the application of our techniques. Nevertheless, generalizing our idea to other fluid discretization is still possible. For example, our method can be used with a fluid solver discretized on a general tetrahedron mesh such as [Chentanez et al. 2007; Pavlov et al. 2011] since the KKT conditions are invariant under different discretizations, and the three operators to define FAS stay valid. On the other hand, generalizing our method to free-surface flow or to handle internal boundary conditions can be non-trivial. The distance metric $C_i$ in Equation 7 needs to be modified to make it aware of the boundaries, e.g., Euclidean distances should be replaced with Geodesic distances. However, modifying the NSO solver to handle the boundaries can be relatively straightforward. This is because our multigrid formulation is the same as a conventional multigrid formulation in spatial domain, using simple trilinear prolongation and restriction operators. Therefore, existing works on boundary aware multigrid such as [Chentanez and Müller 2011] can also be applied to our space-time formulation.

In addition, unlike [Treuille et al. 2003; McNamara et al. 2004], which use a set of template ghost force bases to reduce the search space, our method allows every velocity component to be optimized. This choice is application dependent. For matching smoke to detailed keyframes with lots of high frequency features, our formulation can be useful. However, using a reduced set of template ghost forces could help to avoid the popping artifacts illustrated in Figure 4, and at the same time it allows more user control over the applied control force patterns. For example, the use of vortex force templates encourages more swirl motions in the controlled animations. Moreover, from Figure 8, we can see many small-scale escaping smoke parcels using a large $r$. Our controller does not apply control forces on these parcels in order to reduce the magnitude of control forces. If these smoke parcels are undesirable in the final animation, a template-based formulation can be used. Combining the control force templates with our formulation is considered as future work.

In terms of computational overhead, since our optimal controller always solves the space-time optimization by considering all the timesteps, it is much slower than a simple PD controller which considers one timestep at a time. For example, it took more than 26 hours to generate our longest animation with 150 timesteps illustrated in Figure 16. In order to reduce runtime cost, we can use a larger timestep size to reduce the number of timesteps. Fluid simulation with a large timestep size has been addressed in prior work, such as [Lentine et al. 2012]. We handle the large timestep size by using a novel advection operator (Equation 4) with an adaptive order of Taylor expansion. An alternative solution is to use a conventional advection scheme with an adaptive timestep size determined using the CFL condition. A thorough analysis of alternative advection schemes is a good topic of future work. Also, we can lower the spatial resolution in the control phase and then use smoke upsampling methods such as [Nielsen and Bridson 2011] to generate a high quality animation.

Further accelerations to our method are still possible. For example, one can parallelize our algorithm in a distributed environment. Indeed, multigrid is known as one of the most cluster-friendly algorithms. Moreover, meta-algorithms such as multiple shooting [Bock and Plitt 1984] try to break the space-time optimization into a series of sub-optimizations that consider only a short animation segment and are thus faster to solve. Finally, the benefits of both optimal and PD controllers can be combined by borrowing the idea of receding horizon control [Mayne and Michalska 1990]. In these controllers, optimal control is applied only to a short window of timesteps starting from the current one, and the window keeps being shifted forward to cover the whole animation.
APPENDIX

A. KKT SYSTEM OF THE NSO SUBPROBLEM

We derive here the KKT system for the NSO subproblem. Instead of simply introducing the Lagrangian multipliers and following standard techniques as we did for the AO subproblem, we present a derivation based on the analysis of the ghost force $u_i$. We first eliminate the Navier-Stokes constraints by writing $u_i$ as a function of $u_i$ and $v_i$. Next, we plug this function into our objective to obtain:

$$\frac{K}{r} \sum_{i=1}^{N-1} |u_i(v_i, v_{i+1})|^2 + K \sum_{i=1}^{N-1} |v_i - v_i^*|^2.$$ 

Taking the derivative of this objective against $v_i$ and considering the additional solenoidal constraints on $v_i$, we get the first two equations in $f$:

$$\frac{K}{r} \sum_{i=1}^{N-1} |u_i(v_i, v_{i+1})|^2 + \frac{\partial u_i}{\partial v_i}^T u_i + \frac{\partial u_{i-1}}{\partial v_i}^T u_{i-1} + \nabla p_i = 0$$

$$\nabla \cdot v_i = 0,$$

where $p_i$ is the Lagrangian multiplier. Now in order to derive the other two conditions in Equation 9, we need to determine the additional pressure $p_i$. We assert that $p_{i+1}$ is the Lagrangian multiplier of the solenoidal constraints on $u_i$. In fact, if $u_i$ is not divergence-free, we can always perform a pressure projection on $u_i$ by minimizing $|u_i - \nabla p_{i+1}|^2$ to get a smaller objective function value. As a result, $u_i$ must be divergence-free at the optimum with $p_{i+1}$ being the Lagrangian multiplier, and we get the two additional equations of $f$:

$$\frac{v_{i+1} - v_i}{\Delta t} + \text{Adv}[u_{i-1}] - u_i + \nabla p_{i+1} = 0$$

$$\nabla \cdot u_i = 0.$$ 

From these two conditions, we can see that $\frac{\partial u_i}{\partial v_i} = -Q \frac{1}{\Delta t}, \frac{\partial u_{i-1}}{\partial v_i} = Q(\frac{1}{\Delta t} + \frac{\partial \text{Adv}[u_i]}{\partial v_i})$. Here $Q$ is the solenoidal projection operator introduced in Equation 8. However, we can drop this $Q$ because we have $\frac{\partial u_i}{\partial v_i}^T u_i = (\frac{1}{\Delta t} + \frac{\partial \text{Adv}[u_i]}{\partial v_i})^T Q^T u_i$ and $Q^T u_i = \text{Q} u_i = u_i$ by the fact that $u_i$ is already solenoidal.

REFERENCES


James Gregson, Ivo Ihrke, Nils Thuerey, and Wolfgang Heidrich. 2014. From Capture to Simulation: Connecting Forward and Inverse Problems in Fluids. *ACM Trans. Graph.* 33, 4, Article 139 (July 2014), 11 pages. DOI: http://dx.doi.org/10.1145/2601097.2601147


