# Fluid Control Using the Adjoint Method

Antoine McNamara
University of Washington

Adrien Treuille
University of Washington

Zoran Popović
University of Washington

Jos Stam
Alias Systems

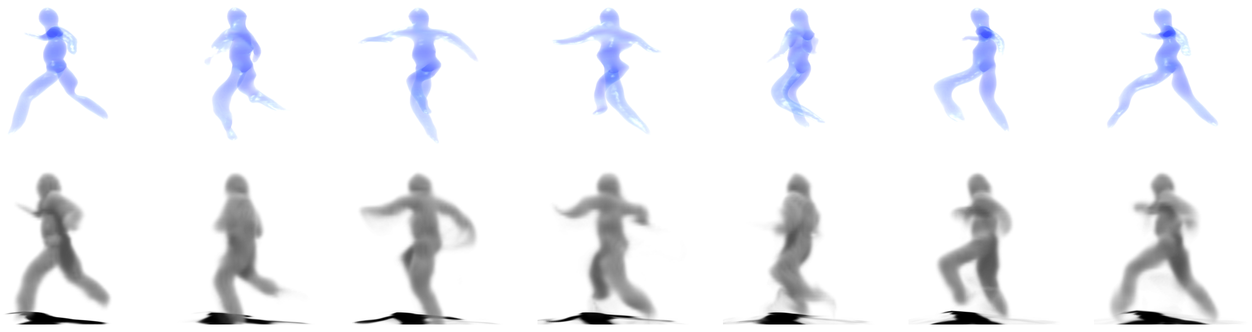

Figure 1: Controlled simulations of a man running, created out of water (top) and smoke (bottom).

## Abstract

We describe a novel method for controlling physics-based fluid simulations through gradient-based nonlinear optimization. Using a technique known as the *adjoint method*, derivatives can be computed efficiently, even for large 3D simulations with millions of control parameters. In addition, we introduce the first method for the full control of free-surface liquids. We show how to compute adjoint derivatives through each step of the simulation, including the fast marching algorithm, and describe a new set of control parameters specifically designed for liquids.

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

**Keywords:** Adjoint Method, Inverse Control, Optimization

## 1 Introduction

In recent years, physics-based animation has become pervasive in computer graphics, producing animations with striking nuance and realism. In particular, significant advances in modeling the dynamics of liquids and gases have yielded stunning animations which could never have been created by hand. Unfortunately, just as with any simulation, the animator cannot freely design the animation: directly editing simulation parameters affects the dynamics in complex and unpredictable ways. Therefore, researchers have begun seeking high-level control methods for complex dynamics.

Nevertheless, the fine-grained control of physically-based simulation has remained out of reach. Such detailed control might easily require hundreds of thousands of free variables; previous systems required a derivative simulation for each variable, making control feasible only at relatively coarse scales. To make full control tractable for large simulations, we need techniques that scale well with the number of control parameters.

In this work, we present such a technique. Our system is given an initial state, either smoke or water, and a set of keyframes provided by the animator. The system then repeatedly simulates the fluid, iteratively solving for external control parameters that help the simulation meet the user's keyframes. This closely follows Treuille *et al.* [2003], except for the crucial gradient calculation.

To compute the gradient, we use an approach from optimal control theory, the adjoint method, which drastically reduces the system's dependence on the number of control parameters. This results in a significant improvement over the state-of-the-art, letting us control simulations of vastly larger scale, for long sequences with many keyframes.

Finally, whereas previous work only dealt with controlling smoke, we also apply this framework to the level set simulation of liquids. Here, the main difficulty lies in formulating the adjoint of the fast marching step. We show how this may be done by observing that the fast marching algorithm is locally continuous. This makes the high-level control of free-surface liquids possible for the first time.

## 2 Related Work

Animating fluid flows of smoke and water has a long history in computer graphics and remains an active area of research. Early procedural work was pioneered by Kajiya and Von Herzen [1984]. Later, Kass and Miller [1990] linearized the equation of water flow to create realtime simulations. Modern physics-based fluid simulation began with the work of Foster and Metaxas, who used the full Navier-Stokes equations to model both water [Foster and Metaxas 1996] and gases [Foster and Metaxas 1997b], producing convincing fluid flows on relatively coarse grids. Shortly thereafter, Stam [1999] addressed the timestep limitations in these earlier techniques by introducing the Stable Fluids algorithm, which combined semi-Lagrangian advection with an implicit viscosity solver. For smoke, Fedkiw *et al.* [2001] extended this approach with a vorticity confinement force to help counteract numerical dampening.

Semi-Lagrangian advection has also proven effective in modeling water simulations. Kunimatsu *et al.* [2001] combined cubic semi-Lagrangian advection with a volume-of-fluid method to create near realtime animations of water. The work of Foster and Fedkiw [2001] and Enright *et al.* [2002] addressed the mass loss inherent in semi-Lagrangian advection by coupling a level set method with particles, currently the state-of-the-art in water simulation for computer graphics. In this work, we use semi-Lagrangian advection, but we do not use the hybrid-particle technique. This is purely for simplicity; we see no reason why our control method should not extend to this more sophisticated model. Finally, researchers have recently turned to pure particle methods, based on smoothed particle hydrodynamics (SPH), to model water flows with promising results [Mueller et al. 2003; Premoze et al. 2003].

Because of the difficulty of editing simulations by hand, computer graphics researchers have also considered how to *control* physics-based simulations. Initial work was pioneered by Barzel *et al.* [1996], who discussed the theoretical underpinnings of control in terms of visual plausibility, and demonstrated an algorithm for controlling pool balls. Later, Popović *et al.* [2000; 2001] and Chenney and Forsyth [2000] separately proposed control paradigms for general rigid-body simulations.

Work on fluid control in graphics was initiated by Foster and collaborators. Foster and Metaxas [1997a] proposed high-level user controls over the fluid parameters. Later, Foster and Fedkiw [2001] controlled the motion of water flow by exactly setting the flow's velocity at specific locations. However, neither of these approaches allows the user to enforce high-level objectives for the simulation.

Very recently, Treuille *et al.* [2003] proposed a new paradigm to control smoke simulations through user-defined keyframes. The approach guides the simulation towards the constraints using a set of control forces whose parameters are computed using a non-linear optimization. Our method is largely based on this work. However, instead of the inefficient forward gradient computation, we use a technique, the adjoint method, that is orders of magnitude more efficient. This allows us to fully control 3D simulations in a fraction of the time it took Treuille *et al.* to compute 2D simulations. We also apply this framework for the first time to the control of water flows. We are not aware of any previous work on the direct control of free-surface liquids.

Concurrently with this work, Fattal *et al.* [2004] demonstrate a system for controlling smoke simulations that also allows the user to give high-level directions. As with our method, this approach adds control variables to the dynamics, but they avoid optimization entirely, instead offering a closed-form solution for the control parameters. While this technique cannot guarantee the optimality of any particular solution, the authors demonstrate very impressive animations computed at roughly the computational cost of an uncontrolled simulation.

Gradient computation with the adjoint method has a long history in optimal control theory [Lions 1971]. Most relevant to us are applications in computational fluid dynamics, where optimization is important in areas such as drag reduction for automobile design and data assimilation for weather forecasting [Ghil et al. 1997].

Adjoint techniques come in two varieties: continuous and discrete. We refer the reader to the overview articles by Bewley *et al.* [2001; 2002] for a thorough exposition of the continuous approach including applications to fluid mechanics. The discrete nature of our problem (voxelized grids, discrete timesteps, etc.) makes it, and many other problems in graphics, particularly suited for the discrete approach. Giles and Pierce [2000] discuss both the continuous and discrete varieties in detail, and we have modeled our derivation of the discrete adjoint method on their own.

## 3 Simulation

Physics-based simulation begins with an initial state $\mathbf{q}_0$ and repeatedly applies a sequence of operations $\mathbf{f}_i$ to the state, so that $\mathbf{q}_{i+1} = \mathbf{f}_i(\mathbf{q}_i)$ for all $i \geq 0$, thus advancing the state through time.

In our simulator, the state $\mathbf{q} = (\mathbf{v}, \rho)$ consists of a grid of velocities $\mathbf{v}$ and a grid $\rho$ representing the fluid material (densities for smoke, a surface level set for water).[1] The functions $\mathbf{f}_i$ model the Navier-Stokes equations that describe fluids.

For smoke simulations, we use the semi-Lagrangian projection model [Stam 1999; Fedkiw et al. 2001]. In each time step, we perform four operations:

$$\text{ADVECT} \rightarrow \text{DIFFUSE} \rightarrow \text{HEAT} \rightarrow \text{PROJECT}.$$

These operations are standard in fluid simulation, and the details are not important for understanding our control model. Briefly, ADVECT transports the materials and velocities through the velocity field, DIFFUSE accounts for viscosity, the HEAT step applies an upwards force proportional to the smoke density, and PROJECT enforces incompressibility.

Water simulation closely resembles smoke simulation, except that the water is represented not as a density field, but as an implicit function whose zero-isocontour defines the surface. [Foster and Fedkiw 2001; Enright et al. 2002] The sequence of operations is also slightly different:

$$\text{ADVECT} \rightarrow \text{DIFFUSE} \rightarrow \text{GRAVITY} \rightarrow \text{PROJECT} \rightarrow \text{REDISTANCE}.$$

The GRAVITY operation applies a downwards force to the water, while REDISTANCE maintains the water surface as a signed-distance function using the fast marching algorithm [Osher and Sethian 1988].

## 4 Control

Having given an overview of the simulation process, we now consider how to control the dynamics.

### 4.1 Control Parameters

To control any system, we must be able to somehow influence the underlying simulation, namely, through a set of external control parameters. We combine all of these parameters into a control vector, $\mathbf{u}$, which encodes all of the external influences the system has over the simulation.

In our fluid system, we found that the most useful controls are the Gaussian wind forces presented in [Treuille et al. 2003]. These allow the system to insert small wind forces to a local region of the velocity field $\mathbf{v}$, scaled with a Gaussian falloff. These added velocities help guide the fluid towards the user's keyframes.

In addition, when controlling level sets, a second type of parameter, which we dub a *source* is also useful. Whereas forces are added to the velocity grid, sources are added directly to the grid $\rho$, also in a local region with the same Gaussian falloff. We do not allow this type of control for smoke because that would let the system solve for the keyframes trivially, removing mass at the smoke's current location and adding it back at the keyframe. For liquids, however, any alteration to the implicit function away from the surface is nullified when the level set is redistanced. Therefore, sources can only affect the fluid interface itself, slightly perturbing it inwards or outwards. We found that sources were crucial to help control level set-based animations, both for matching complex shapes and for preserving the mass of the simulation over time.

Both forces and sources are scaled by their respective control parameters before being applied to the underlying system. Therefore, while we formulate the adjoint method for arbitrary differentiable control, in practice, the forces are linear. In other words the function $\mathbf{f}$ that applies control parameters can be expressed as follows:

[1] We denote the level set by $\rho$, instead of the traditional $\phi$, to avoid ambiguity with our objective function $\varphi$.

$$\mathbf{f}(\mathbf{q}) = \mathbf{q} + M_t\mathbf{u}, \tag{1}$$

where $M_t$ is the matrix that converts the control vector to an incremental state on timestep $t$.

## 4.2 Objective Function

Now that the system is able to influence the simulation, it must be able to measure how well the user's goals are met. To do this, we create an objective function $\varphi$ which both measures how closely the simulation meets the keyframes, and also penalizes the system for using too much control.

The user specifies a set of keyframes $\{\mathbf{q}_t^*\}$ and corresponding weight matrices $\{W_t\}$ which describe the region of interest for each timestep $t$. For example, $W_t$ could be set only to match velocities, to weight certain regions of space higher than others, or to ignore the keyframe entirely. Additionally, the user specifies a "smoothness" term $\alpha$, which weighs how much to penalize the system for excessive use of the controls. This allows the user to specify how much the animation should conform to the underlying dynamics.

These two goals, fidelity to the keyframes and physical plausibility, respectively comprise the two terms of the objective function:

$$\varphi(\mathbf{u}) = \frac{1}{2}\sum_{t=0}^{n}\left(||W_t(\gamma(\mathbf{q}_t) - \gamma(\mathbf{q}_t^*))||^2 + \alpha||M_t\mathbf{u}||^2\right). \tag{2}$$

Here, $\gamma$ is a preprocessing function we apply to the state before computing the objective function. For smoke, no preprocessing is necessary, and $\gamma$ is the identity. For water, however, directly comparing signed-distance functions turns out to be a poor metric. Instead, we would like the penalty to be proportional to the *volume* of space in which the state and keyframe do not agree. We can approximate this volume comparison by setting $\gamma(x) = 2\arctan(x)/\pi$. This differentiable, S-shaped function moves positive regions of the grid to $+1$, and negative regions to $-1$; therefore, the objective function will compare not level set distances, but discrepancies between the positive and negative regions in the two grids.

## 4.3 Implementation

Both the control parameters and objective function can be smoothly integrated into our framework by adding two new operations: at the beginning of each step, an APPLYCONTROL operation updates the state according to the control vector; at the end of the step, the MATCHKEYFRAME operation increments the objective function according to how well the simulation approximates the user's constraints. MATCHKEYFRAME leaves the state itself untouched.

## 4.4 Optimization

These two special operations, APPLYCONTROL and MATCHKEYFRAME, allow us to evaluate how well the control vector influences the simulation. Thus, the problem is reduced to a continuous function minimization:

$$\underset{\mathbf{u}}{\operatorname{argmin}}\, \varphi(\mathbf{q}_0, \mathbf{u}).$$

There are many standard numerical methods for minimizing a continuous function. As in Treuille *et al.* [2003], we use a limited memory quasi-Newton optimization technique [Zhu et al. 1994]. Since this is a derivative-based optimization, we must not only evaluate $\varphi$, but also compute its gradient, $d\varphi/d\mathbf{u}$.

In computer graphics, this gradient computation has traditionally been the bottleneck for control because each parameter required its own derivative computation. One of the the main contributions of our paper is to show how the adjoint method, long used in the optimal control community, can be adapted to this and other derivative-based problems in computer graphics.

# 5 The Adjoint Method

We now take a step back to consider gradient computation abstractly. We shall begin by showing how the adjoint method can be viewed as a special case of linear duality. Then we will apply these ideas specifically to the problem of fluid control.

## 5.1 Duality

At the heart of the adjoint method is a substitution of variables that allows us to compute the gradient of a function quickly. This substitution can be viewed in terms of linear duality [Giles and Pierce 2000]. Suppose that the matrix $A$ and the vectors $\mathbf{g}$ and $\mathbf{c}$ are known, and that we would like to compute the vector product

$$\mathbf{g}^T\mathbf{b} \quad \text{such that} \quad A\mathbf{b} = \mathbf{c}$$

in terms of the unknown vector $\mathbf{b}$. A straightforward approach would be to first solve for $\mathbf{b}$ and then compute the vector product. An alternative would be to introduce a vector $\mathbf{s}$ and compute:

$$\mathbf{s}^T\mathbf{c} \quad \text{such that} \quad A^T\mathbf{s} = \mathbf{g}.$$

This is known as the *dual* of the problem. The equivalence can be shown through substitution:

$$\mathbf{s}^T\mathbf{c} = \mathbf{s}^TA\mathbf{b} = (A^T\mathbf{s})^T\mathbf{b} = \mathbf{g}^T\mathbf{b}.$$

Of course, this new linear system is not necessarily any easier to solve. However, consider a new case where the unknown vector $\mathbf{b}$ and the known vector $\mathbf{c}$ are actually *matrices* $B$ and $C$. By the same logic, the vector-matrix product

$$\mathbf{g}^TB \quad \text{such that} \quad AB = C \tag{3}$$

is equivalent to

$$\mathbf{s}^TC \quad \text{such that} \quad A^T\mathbf{s} = \mathbf{g}. \tag{4}$$

Now these two linear systems look quite different! The former involves solving for the entire matrix $B$, while the latter is the same single linear system as in our original example. Clearly, in this case, huge benefits can be reaped by solving the dual formulation.

As it turns out, the control problem we would like to solve involves calculating a vector-matrix product of this form. The adjoint method exploits this powerful aspect of duality to drastically improve the efficiency of this computation.

## 5.2 Gradient Calculation

Let us now delve more specifically into a particular type of optimization problem often seen in graphics, of which physical simulation is one example.

Suppose we have a *fixed* initial state $\mathbf{q}_0$, which we evolve into $n$ subsequent states $\mathbf{q}_1, \ldots, \mathbf{q}_n$ according to the update rule

$$\mathbf{q}_{i+1} = \mathbf{f}_i(\mathbf{q}_i, \mathbf{u}), \tag{5}$$

where each $\mathbf{f}_i$ is an arbitrary differentiable function parameterized by a control vector $\mathbf{u}$. We aggregate these into one long vector:

$$\mathbf{Q} = \left[\mathbf{q}_1^T, \ldots, \mathbf{q}_n^T\right]^T,$$

and one long vector function:

$$\mathbf{F}(\mathbf{Q}, \mathbf{u}) = \left[\mathbf{f}_0(\mathbf{q}_0, \mathbf{u})^T, \ldots, \mathbf{f}_{n-1}(\mathbf{q}_{n-1}, \mathbf{u})^T\right]^T.$$

This allows us to write equation (5) as

$$\mathbf{Q} = \mathbf{F}(\mathbf{Q}, \mathbf{u}). \tag{6}$$

This equation is essentially a constraint on the optimization; for $\mathbf{Q}$ to represent a valid simulation generated by the sequence $\mathbf{f}_i$ of functions, equation (6) must hold.

Finally assume that we have a differentiable objective function $\varphi(\mathbf{Q},\mathbf{u})$ and we would like to compute its derivative with respect to the control vector:

$$\frac{d\varphi}{d\mathbf{u}} = \frac{\partial\varphi}{\partial\mathbf{Q}}\frac{d\mathbf{Q}}{d\mathbf{u}} + \frac{\partial\varphi}{\partial\mathbf{u}}. \tag{7}$$

Computing this directly is extremely costly, as the matrix $d\mathbf{Q}/d\mathbf{u}$ consists of an entire state sequence for each control. As we shall see, the adjoint method provides a way of side-stepping this computation while still arriving at exact derivatives of $\varphi$.

Differentiating the constraint equation (6) gives us a linear constraint on the derivative matrix $d\mathbf{Q}/d\mathbf{u}$. Thus, the first term of equation (7) calls for calculating

$$\frac{\partial\varphi}{\partial\mathbf{Q}}\frac{d\mathbf{Q}}{d\mathbf{u}} \quad \text{such that} \quad \left(I - \frac{\partial\mathbf{F}}{\partial\mathbf{Q}}\right)\frac{d\mathbf{Q}}{d\mathbf{u}} = \frac{\partial\mathbf{F}}{\partial\mathbf{u}}. \tag{8}$$

Notice that this is the exact situation described in equation (3), implying that the vector-matrix product might be much more efficiently calculated using the dual! Using the same substitution as equation (4), we introduce a vector $\mathbf{R}$ (equivalent to $\mathbf{s}$ above), and the product in equation (8) can instead be computed as

$$\mathbf{R}^T\frac{\partial\mathbf{F}}{\partial\mathbf{u}} \quad \text{such that} \quad \left(I - \frac{\partial\mathbf{F}}{\partial\mathbf{Q}}\right)^T\mathbf{R} = \frac{\partial\varphi}{\partial\mathbf{Q}}^T. \tag{9}$$

We call this new vector the *adjoint vector*. If we can calculate this adjoint $\mathbf{R}$, the overall gradient can now be computed simply:

$$\frac{d\varphi}{d\mathbf{u}} = \mathbf{R}^T\frac{\partial\mathbf{F}}{\partial\mathbf{u}} + \frac{\partial\varphi}{\partial\mathbf{u}}. \tag{10}$$

Now we must describe how to calculate $\mathbf{R}$ itself. First, rewrite the constraint in equation (9) as

$$\mathbf{R} = \left(\frac{\partial\mathbf{F}}{\partial\mathbf{Q}}\right)^T\mathbf{R} + \frac{\partial\varphi}{\partial\mathbf{Q}}^T. \tag{11}$$

The key to solving this lies in the sparse structure of $\partial\mathbf{F}/\partial\mathbf{Q}$, with its off-diagonal blocks representing each $\partial\mathbf{f}_i/\partial\mathbf{q}_i$. Much the same way that $\mathbf{Q}$ is an aggregate of a sequence of forward states $\mathbf{q}_1,\ldots,\mathbf{q}_n$, we may view $\mathbf{R}$ as an aggregate of a sequence of *adjoint* states $\mathbf{r}_1,\ldots,\mathbf{r}_n$. Equation (11) implies that $\mathbf{r}_n = (\partial\varphi/\partial\mathbf{q}_n)^T$ and

$$\mathbf{r}_i = \left(\frac{\partial\mathbf{f}_i}{\partial\mathbf{q}_i}\right)^T\mathbf{r}_{i+1} + \left(\frac{\partial\varphi}{\partial\mathbf{q}_i}\right)^T. \tag{12}$$

Note that each adjoint state depends on the *subsequent* state; therefore, whereas the regular simulation states are computed forward in time, these adjoint states must be computed in *reverse*.

## 5.3 Implementation

To make this more concrete, we now explain the specific steps involved in the adjoint computation. First, the forward states $\mathbf{q}_1,\ldots,\mathbf{q}_n$ are calculated in order, as in an ordinary simulation, except that *each simulation state must be stored*. Then, the adjoint states $\mathbf{r}_1,\ldots,\mathbf{r}_n$ are calculated, proceeding backwards through the sequence, according to equation (12). As each adjoint state is computed, the objective function is incremented according to equation (10). When $\mathbf{r}_1$ has finally been calculated, both $\varphi$ and $d\varphi/d\mathbf{u}$ are known, at essentially the cost of two computations of $\varphi$.

We store the forward states $\mathbf{q}_i$ because equation (12) depends on them for the adjoint calculation. As a result, the algorithm has memory requirements *linear* in the number of timesteps. This is the main drawback of the adjoint method.

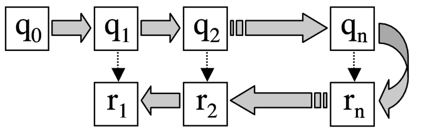

Figure 2: States during the forward pass are stored to be used later for computing the reverse pass.

To address this problem, the central issue is granularity: at what level should the functions $\mathbf{f}_i$ be defined? At the finest level, it is possible to consider each machine instruction to be its own function; in fact, this forms the basis of the "reverse mode" automatic differentiation [Griewank 2000]. Unfortunately, a naive implementation of this approach would require a tremendous amount of memory and is infeasible for all but the smallest problems.

We opt instead to view the problem at a coarser-level of granularity, matching the functions $\mathbf{f}_i$ to the fluid simulation operations. Each operation is responsible for storing data for the adjoint calculation: one operation might store all the relevant information, while another might store only some data and recompute the rest on the fly. This flexible framework, sometimes called checkpointing, allows the system designer to trade computation time for memory usage, depending on the hardware constraints.

If the memory requirements are still unmanageable, the user must break the simulation into smaller subproblems. For example, layered multiple shooting [Treuille et al. 2003] was designed for fluid control applications to help avoid local minima, but it also significantly reduces memory consumption by considering only short sequences at a time. In practice, because of the granularity at which we chose to implement the adjoint method, we did not find these techniques necessary.

## 6 Adjoint Fluid Control

Having described the adjoint method in theory, we now demonstrate how to apply it to our system. As mentioned in Section 3, each function $\mathbf{f}_i$ is one of the operations used to simulate fluids. In other words, for smoke:

$\mathbf{f}_0 =$ APPLYCONTROL,
$\mathbf{f}_1 =$ ADVECT,
$\mathbf{f}_2 =$ DIFFUSE,
$\mathbf{f}_3 =$ HEAT,
$\mathbf{f}_4 =$ PROJECT,
$\mathbf{f}_5 =$ MATCHKEYFRAME,
$\mathbf{f}_6 =$ APPLYCONTROL, *etc*...

and similarly for water. In both cases, we simulate the fluid forward in time by applying each operation in turn. Then, the adjoint state is initialized and passed through each operation's adjoint step in reverse order. Therefore, to calculate the gradient, we must know what this "adjoint step" means for each operation $\mathbf{f}$ (for the remainder of this section, we omit the subscripts to avoid confusion).

We first consider the standard fluid operations, which do not directly affect the objective function $\varphi$. In this case, when updating the adjoint state according to equation (12), the second term vanishes, and the adjoint step just becomes a multiplication by the transpose of the derivative

$$\left(\frac{\partial\mathbf{f}}{\partial\mathbf{q}}\right)^T.$$

The following subsections will demonstrate how to compute this transpose derivative for each of the standard operations. Finally, we will consider the special cases of APPLYCONTROL and MATCHKEYFRAME.

## 6.1 Heat

In smoke simulations, the HEAT operation adds upward velocities proportional to the amount of smoke density in each grid cell:

$$v'_y = v_y + h\rho.$$

Here, $v'_y$ is the y-component of the cell's velocity after heat is applied, and $h$ is a user-specified heat constant. Therefore, letting $\mathbf{f}$ denote the HEAT operation: $\mathbf{f}(\mathbf{q}) = H\mathbf{q}$, where the matrix $H$ adds the scaled densities to the y-component of the velocities. This adjoint of the linear step is merely the transpose of the matrix:

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{q}}\right)^T = H^T.$$

In other words, we scale the adjoint state's *y-velocities*, adding them to its *density* grid.

## 6.2 Gravity

Even simpler, the GRAVITY operation for water applies a *constant* downward force to the interior of the liquid. Since this force does not depend continuously on the state, its derivative is the identity. That is, the adjoint of this step leaves the state untouched.

## 6.3 Projection and Diffusion

Another class of operations (PROJECT and DIFFUSE) involves solving a linear system. In general, the adjoint involves solving the transpose linear system. However, these operations are symmetric, so the adjoint and forward computations are identical: we project and diffuse the adjoint state in the same way as the original state.

Note that this requires an accurate linear solver. Another approach would be to take the adjoint derivatives through the linear solver itself. However, we did not find this necessary.

## 6.4 Advection

Unlike the previous operations, ADVECT is nonlinear. To demonstrate the adjoint operation, we describe advecting $\rho$ through $\mathbf{v}$ (advecting the velocities is analogous).

In semi-Lagrangian advection, paths originating from each voxel are backtraced through the velocity grid. Each grid cell is updated by linearly interpolating the value at the end of its path. Let $\eta(\rho, \mathbf{x})$ be the interpolation function that takes an input grid $\rho$ and a corresponding grid of $\mathbf{x}$, and returns $\rho$ resampled at positions $\mathbf{x}$. Both $d\eta/d\rho$ and $d\eta/d\mathbf{x}$ can be easily derived from the interpolation equations.

For simplicity we present an Euler-step backtrace (these ideas also extend to more complex integrators). Thus, we can write the ADVECT operation $\mathbf{f}$ as

$$\mathbf{f}(\mathbf{q}) = \eta(\rho, \mathbf{x}_0 - \mathbf{v}),$$

where $\mathbf{x}_0$ are the positions at the voxel centers and $\rho$ and $\mathbf{v}$ are the components of $\mathbf{q}$.

We must compute the adjoint matrix $(\partial \mathbf{f}/\partial \mathbf{q})^T$, in other words $(\partial \mathbf{f}/\partial \rho)^T$ and $(\partial \mathbf{f}/\partial \mathbf{v})^T$. Both can be computed in terms of the known derivatives of linear interpolation:

$$\left(\frac{\partial \mathbf{f}}{\partial \rho}\right)^T = \left(\frac{\partial \eta}{\partial \rho}\right)^T \quad , \quad \left(\frac{\partial \mathbf{f}}{\partial \mathbf{v}}\right)^T = \left(\frac{d\eta}{d\mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{v}}\right)^T = -\mathbf{v}^T\left(\frac{d\eta}{d\mathbf{x}}\right)^T.$$

## 6.5 Level Set Redistancing

The REDISTANCE operation is considerably more complex than those described above. Given the highly discrete nature of this operation, relying on the heapsort algorithm, one might expect it not to lend itself to derivative calculation. One of the contributions of this paper is to show that reasonable derivatives *can* be computed by observing that the operation is locally smooth.

Our redistancing operation is based on the fast marching algorithm as described by Adalsteinsson and Sethian [1998]. We now sketch the algorithm so that we may subsequently describe its adjoint. In general, the REDISTANCE operation reinitializes the level set to a signed-distance function to avoid a slow degradation of the surface.

The fast marching algorithm first considers all voxels neighboring the interface and estimates the signed distance by linearly approximating the surface location. The redistanced grid value $\rho'_l$ at grid cell $l$ is the solution to a quadratic equation

$$a\rho_l'^2 + b\rho'_l + c = 0 \tag{13}$$

where the coefficients $a$, $b$, and $c$ are functions of $\rho$ at $l$ and at adjacent voxels opposite the interface. Voxels away from the interface are computed in order of increasing distance from the surface. As above, the grid value $\rho'_l$ is computed as a solution to a quadratic equation. However, in this case, the coefficients $a, b, c$ of (13) are functions of neighboring voxels that have already been redistanced.

Thus, redistancing involves solving a series of quadratic equations starting at the front and moving progressively outwards. A min-heap data structure of unprocessed candidate voxels efficiently manages the traversal order. Figure 3 shows the traversal order, and the arrows indicate the flow of information during the algorithm. At the interface, information flows across the front; elsewhere, it always flows outwards.

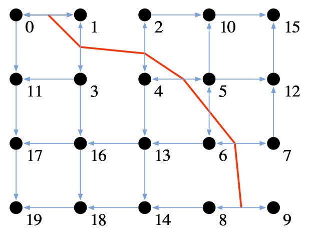

Figure 3: Voxel traversal order during fast marching.

In deriving the adjoint of this operation, the main difficulty is adapting differentiability to the discontinuous heapsort-based traversal order. The key insight is that sufficiently small perturbations of the water surface do not in general change the traversal order. Therefore, when linearizing this operation, we consider the order fixed and view each redistanced voxel as a small differentiable function of its neighbors, ignoring the min-heap.

Since information travels outwards from the interface, the adjoint information travels in the *opposite* direction. Consequently, when computing the adjoint we start with the voxel traversed last, and work our way inwards towards the interface, against the direction of the arrows in Figure 3.

When updating each voxel, we ignore the specific method used to solve the quadratic equation and instead take derivatives of the (13) itself:

$$\frac{\partial \rho'_l}{\partial \rho'_m} = \frac{\frac{\partial a}{\partial \rho'_m}\rho_l'^2 + \frac{\partial b}{\partial \rho'_m}\rho'_l + \frac{\partial c}{\partial \rho'_m}}{2a + b},$$

where $m$ is a grid cell on which $a$, $b$, or $c$ depends.

When the adjoint is complete, the entire adjoint level set is zero, except for grid points neighboring the interface. This is not surprising: redistancing the level set depends *only* on the values of $\rho$ adjacent to the surface; therefore, the operation has zero derivative with respect to all other values.

We note that while small perturbations of the level set do not *in general* change the traversal order, in some cases they do, an extreme example being topological changes. In practice, however, we found that our approach to fast marching yielded derivatives no less accurate than those for the other operations.

### 6.6 Keyframe Matching

Having considered the standard operations, we now move to the special control operations. Whereas all of the previous steps simulated the fluid without affecting the objective function, MATCHKEYFRAME does the reverse: it leaves the state $\mathbf{q}$ untouched while incrementing $\varphi$ according to equation (2).

To compute the adjoint of MATCHKEYFRAME, we note that this is the *only* operation where $\varphi$ depends directly on the state $\mathbf{q}$. By equation (12), the reverse operation involves incrementing the adjoint state by $(\partial\varphi/\partial\mathbf{q})^T$.

### 6.7 Applying Control

The APPLYCONTROL operation adds control to the state according to equation (1). Letting $\mathbf{f}$ again denote this operation, we see that $\partial\mathbf{f}/\partial\mathbf{q} = I$. Therefore, in the reverse direction, this operation leaves the adjoint state untouched.

However, it still has very important role to play! Because $\partial\mathbf{f}/\partial\mathbf{u}$ is nonzero, equation (10) tells us that the APPLYCONTROL operation must increment the objective function gradient. Indeed, this is the *only* operation that affects the gradient, forming the bridge between the adjoint states and the gradient computation. Substituting derivatives of equations (1) and (2) into (10), we see that the operation increments the gradient as follows:

$$\frac{d\varphi}{d\mathbf{u}} \leftarrow \frac{d\varphi}{d\mathbf{u}} + \mathbf{r}^T M_t + \alpha \mathbf{u} M_t^T M_t.$$

## 7 Results

We have used our system to generate a number of controlled smoke and water simulations, and have demonstrated that it runs orders of magnitude faster than the system described in [Treuille et al. 2003]. In 2D, this means that we can specify virtually unlimited control, such as pairing *every* simulation variable with a control variable. In 3D, we can produce sophisticated animations completely infeasible using earlier systems.

Figure 4 shows three examples of our system controlling a viscous fluid. In these optimizations, a ball of clay-like material is dropped onto a flat surface. A keyframe controls the shape of the clay's final resting position. In all of these examples, we found that the system had a much easier time controlling the simulation if controls were placed only after ground contact. We believe that this is because impact creates a highly nonlinear shock, through which linearizing the equations provides a poor approximation.

We have also used our system to control smoke simulations, as can be seen in Figure 5. In this case, we converted meshes of the Stanford bunny and armadillo models to density keyframes. Our system then solved for simulations in which a ball of smoke rises to form these shapes. With over $100,000$ control parameters each, these simulations would have been computationally intractable using previous control algorithms. Our system produced very close matches to the keyframes. As can be seen in the accompanying video, the animations faithfully reproduce fine scale detail such as the tail and horns of the armadillo.

Our system also can be used to touch up existing simulations. For example, Figure 6(b) shows a simulation in which a large drop of water falls into a pool. To make the resulting splash more dramatic, the system follows an animator's sketch of a droplet of water rising out of the splash and crashing back down. Figure 6(c) illustrates an even more dramatic, though less realistic, effect: three droplets emerge symmetrically from the splash.

In figure 6(a), we show a result that was created with a keyframe at every timestep from mesh data of a man punching. In this case, our system interpolates the keyframes in a fluid-like manner by choosing forces that direct the smoke through the keyframes. The wisps of smoke trailing the man's arm illustrate how the keyframes can be hit while retaining the dynamic qualities of real fluids.

Our last examples, shown at the beginning of this paper in Figure 1, show fluid simulations of a fully articulated motion-captured human. To show the versatility of our system, we optimized this sequence both for smoke and water. While the geometry and motion are preserved across both animations, the dynamics remain faithful to the underlying fluid: the smoke man billows vapor, while the water man has drop-like globules of water flowing across his body. These optimizations each used 1.5 million control variables and 600MB of memory over $45 \times 50 \times 36$ grid cells and 46 timesteps. In each case, the first half of the animation was solved, and then the second half solved using the solution to the first half as initial conditions. We split the simulation this way to speed convergence.

These results took between two hours (for the bunny) and two days (for the water man) to compute. These runtimes bounds are not "tight" in the sense that we ran simulations with overly stringent convergence criteria to avoid halting before an adequate solution was reached. In the case of the clay simulations, for example, the system had already found good simulations well before the optimization converged.

## 8 Discussion and Future Work

In this work, we have introduced a framework for controlling fluid simulations using a derivative calculation that runs orders of magnitude faster than previous methods. We do so by adapting the adjoint method, used in optimal control theory, to compute exact derivatives of the coarse, inexact solvers most often used in graphics. Without having to run additional derivative simulations for each parameter, our framework can be given nearly unlimited control over a simulation, enabling–for the first time–the fine-grained control of large fluid animations.

Additionally, this paper introduces the first work fully controlling a level set surface through the Navier-Stokes equations. By taking adjoint derivatives through the fast marching step, we have shown that derivative-based optimizations can be used to control water simulations, even in the presence of highly discontinuous operations such as the heapsort algorithm.

Nevertheless, discontinuities in the water simulator did make control more difficult. In addition to the redistancing operation, we note that, for water, the projection and diffusion operations are also discontinuous. This is because the linear systems are only solved on the *interior* of the fluid. If the surface moves slightly, the set of interior cells can change, and we must solve a different (albeit similar) linear system. These discontinuities rendered the derivatives less accurate than for smoke, making control harder. With smoke simulations, the optimizations could be tuned to an almost arbitrary precision, while water simulations often could only reach a certain level of detail. Even with this loss of accuracy, we were able to control complex water simulations through a variety of constraints. We are interested in further investigating the types of operations that can or cannot be successfully controlled.

For example, we would like to extend this adjoint control model to more sophisticated water simulation techniques, such as the hybrid-particle method. While adding and subtracting particles to

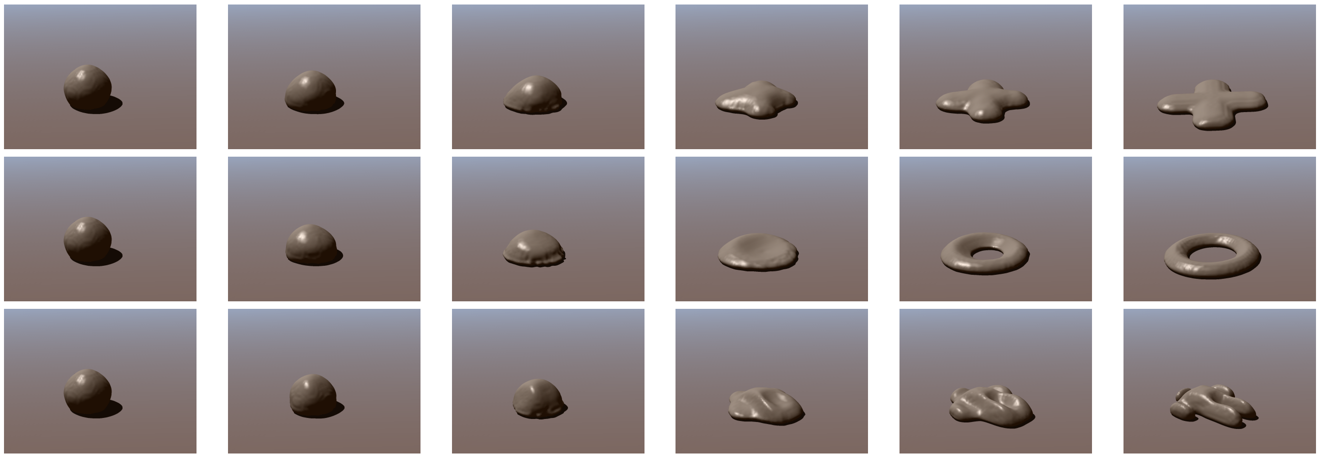

Figure 4: From identical initial conditions, clay falls into various shapes: a cross (top), a torus (middle), and a man (bottom). These were each optimized on grids of dimension $50 \times 23 \times 50$ for 10 timesteps. With over 300,000 control parameters, the system used about 200MB of memory.

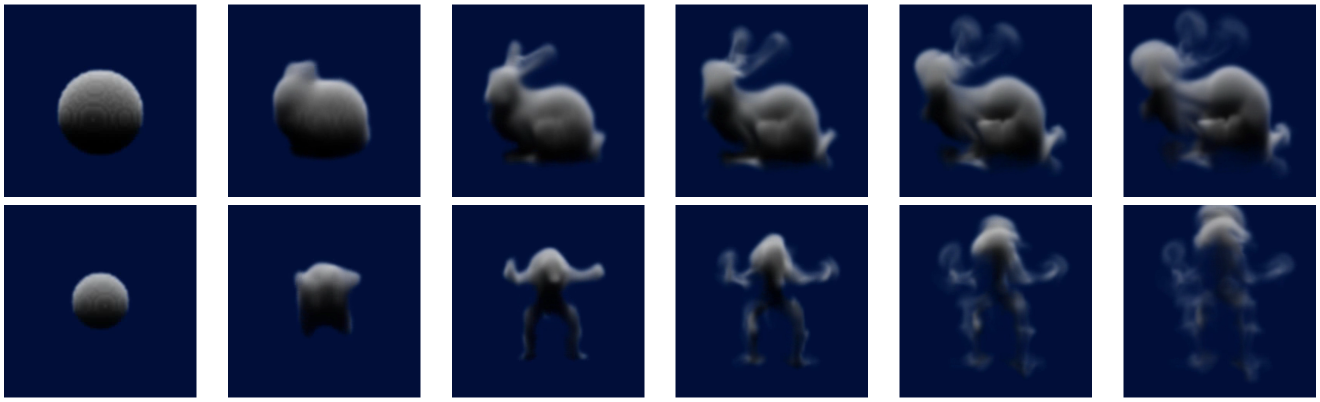

Figure 5: (top) The Stanford bunny. (bottom) The Stanford armadillo. Both were simulated on a $50 \times 50 \times 50$ grid with a single keyframe (the third image in the sequence). 100,000 controls; about 600MB memory.

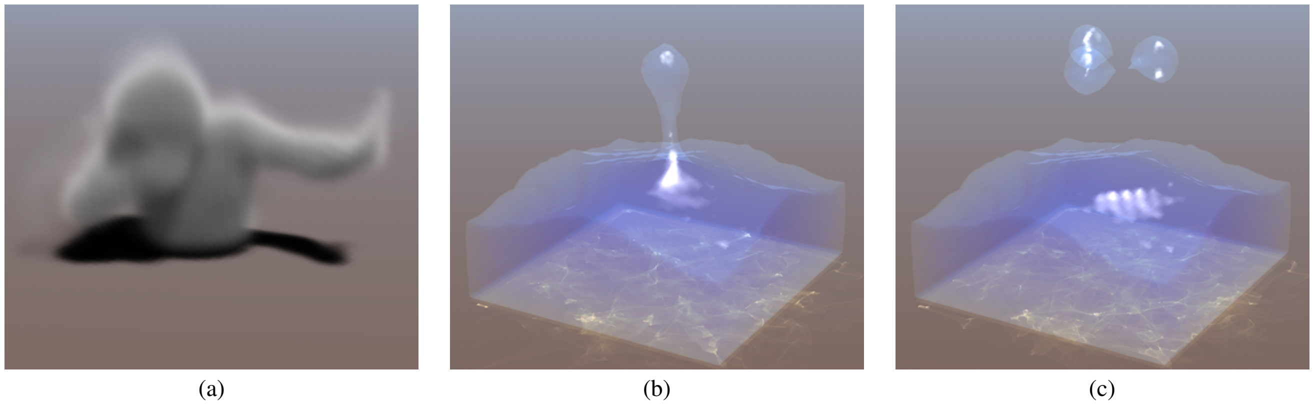

Figure 6: (a) A smoke punch in a $90,000$ voxel volume with over $600,000$ Gaussian wind force parameters distributed over the 20 timesteps, and requiring about 550MB of memory. (b) A controlled droplet emerges from a water splash. (c) Three droplets rise from the same splash. These water optimizations used about 95MB of memory in 10 timesteps over $30 \times 30 \times 30$ grid cells.

the system might introduce discontinuities, our experience with the discontinuities in fast marching suggests that the framework could cope.

Because the adjoint method can easily handle huge control vectors, the bottleneck in our framework has moved from the evaluation of the objective function to the optimizer itself. When the system has trouble matching keyframes, it is often because it is given excessive control and cannot navigate the complex search space. This suggests that intelligent model reduction on the controls may reap considerable benefits.

We would also like to consider other control paradigms. While keyframing is extremely powerful, there are many times when we might want to describe some other property of the fluid motion. For example, we might want a wave to break at a specific time, without specifying the exact shape of the surface. If these metrics could be encoded into a differentiable objective function, they could be added to our system.

Most importantly, we are excited by the many possible applications of the adjoint method in computer graphics. This approach can certainly extend directly to controlling other types of physics-based simulations, but its applicability is not limited to this domain. We hope that this work inspires others to explore how this powerful technique might be applied to their own research.

**Acknowledgments.** The authors would like to thank Jessica Kristan Miller for providing the video voiceover. In addition, the authors are particularly grateful to John Anderson, Peter Schmid, and Donna Calhoun for their valuable insights on the adjoint method. This work was supported by the UW Animation Research Labs, NSF grants CCR-0092970, ITR grant IIS-0113007, Alfred P. Sloan Fellowship, NSF Graduate Research Fellowship, Electronic Arts, Sony and Microsoft Research.

## References

ADALSTEINSSON, D., AND SETHIAN, J. 1998. The Fast Construction of Extension Velocities in Level Set Methods. *Journal of Computational Physics 148*, 2–22.

BARZEL, R., HUGHES, J. F., AND WOOD, D. 1996. Plausible motion simulation for computer animation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*.

BEWLEY, T. R., MOIN, P., AND TEMAM, R. 2001. Dns-based predictive control of turbulence: an optimal benchmark for feedback algorithms. *Journal of Fluid Mechanics 447*, 179–225.

BEWLEY, T. R. 2002. The emerging roles of model-based control theory in fluid mechanics. In *Advances in Turbulence IX. Proceedings of the Ninth European Turbulence Conference*.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling Plausible Solutions to Multi-body Constraint Problems. In *Computer Graphics (SIGGRAPH 2000)*, ACM, 219–228.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and Rendering of Complex Water Surfaces. In *Computer Graphics (SIGGRAPH 2002)*, ACM, 736–744.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. *ACM Transactions on Graphics 23*, 3 (Aug.).

FEDKIW, R., STAM, J., AND JENSEN, H. 2001. Visual Simulation of Smoke. In *Computer Graphics (SIGGRAPH 2001)*, ACM, 15–22.

FOSTER, N., AND FEDKIW, R. 2001. Practical Animation of Liquids. In *Computer Graphics (SIGGRAPH 2001)*, ACM, 23–30.

FOSTER, N., AND METAXAS, D. 1996. Realistic Animation of Liquids. *Graphical Models and Image Processing 58*, 5, 471–483.

FOSTER, N., AND METAXAS, D. 1997. Controlling fluid animation. *Computer Graphics International*, 178–188.

FOSTER, N., AND METAXAS, D. 1997. Modeling the Motion of a Hot, Turbulent Gas. In *Computer Graphics (SIGGRAPH 97)*, ACM, 181–188.

GHIL, M., IDE, K., BENNETT, A. F., COURTIER, P., KIMOTO, M., AND (EDS.), N. S. 1997. *Data Assimilation in Meteorology and Oceanography: Theory and Practice,.* Meteorological Society of Japan and Universal Academy Press.

GILES, M. B., AND PIERCE, N. A. 2000. An introduction to the adjoint approach to design. *Flow, Turbulence and Combustion 65*, 393–415.

GRIEWANK, A. 2000. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. SIAM.

KAJIYA, J. T., AND VON HERZEN, B. P. 1984. Ray Tracing Volume Densities. *Computer Graphics (SIGGRAPH 84) 18*, 3 (July), 165–174.

KASS, M., AND MILLER, G. 1990. Rapid, Stable Fluid Dynamics for Computer Graphics. *ACM Computer Graphics (SIGGRAPH '90) 24*, 4 (August), 49–57.

KUNIMATSU, A., WATANABE, Y., FUJII, H., SAITO, T., AND HIWADA, K. 2001. Fast simulation and rendering techniques for fluid objects. *Computer Graphics Forum (Proceedings of Eurographics) 20*, 3, 357–367.

LIONS, J. P. 1971. *Optimal Control of Systems Governed by Partial Differential Equations*. Springer-Verlag, New York.

MUELLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation SCA 2003*, 154–159.

OSHER, S., AND SETHIAN, J., 1988. Fronts Propagating with Curvature Dependant Speed: Alogorithms Based on Hamilton-Jacobi Formulations.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. 2000. Interactive Manipulation of Rigid Body Simulations. In *Computer Graphics (SIGGRAPH 2000)*, ACM, 209–218.

POPOVIĆ, J. 2001. *Interactive Design of Rigid-Body Simulatons for Computer Animation*. PhD thesis, Carnegie Mellon University.

PREMOZE, S., TASDIZEN, T., BIGLER, J., LEFOHN, A., AND WHITAKER, R. 2003. Particle Based Simlation of Fluids. *Computer Graphics Forum (Proceedings of Eurographics) 22*, 3, 401–410.

STAM, J. 1999. Stable Fluids. In *Computer Graphics (SIGGRAPH 99)*, ACM, 121–128.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Transactions on Graphics 22*, 3 (July), 716–723.

ZHU, C., BYRD, R., LU, P., AND NOCEDAL, J., 1994. Lbfgs-b: Fortran subroutines for large-scale bound constrained optimization.