*Published in ACM Transactions on Graphics (SIGGRAPH 2003)*

# Keyframe Control of Smoke Simulations

Adrien Treuille, University of Washington
Antoine McNamara, University of Washington
Zoran Popović, University of Washington
Jos Stam, Alias|Wavefront

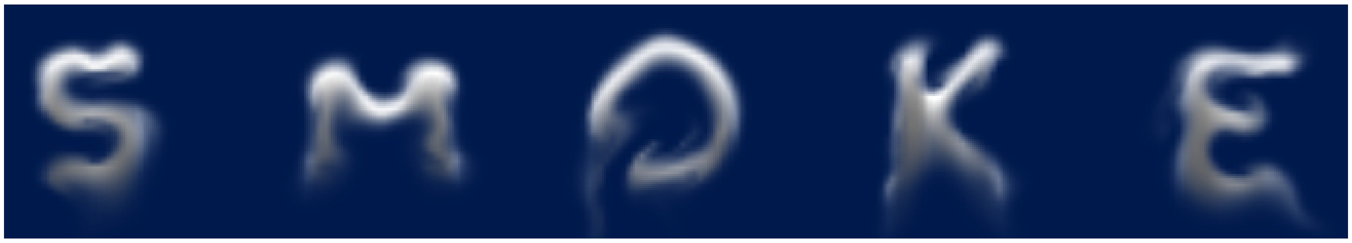

Figure 1: Single frame of a physically-based fluid animation spelling out letters.

## Abstract

We describe a method for controlling smoke simulations through user-specified keyframes. To achieve the desired behavior, a continuous quasi-Newton optimization solves for appropriate “wind” forces to be applied to the underlying velocity field throughout the simulation. The cornerstone of our approach is a method to efficiently compute exact derivatives through the steps of a fluid simulation. We formulate an objective function corresponding to how well a simulation matches the user’s keyframes, and use the derivatives to solve for force parameters that minimize this function. For animations with several keyframes, we present a novel multiple-shooting approach. By splitting large problems into smaller overlapping subproblems, we greatly speed up the optimization process while avoiding certain local minima.

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

**Keywords:** Fluid Simulation, Inverse Control, Optimization

## 1 Introduction

In recent years, computer graphics researchers have made great strides towards realistic simulation of complex fluid phenomena [Fedkiw et al. 2001; Nguyen et al. 2002; Enright et al. 2002; Foster and Fedkiw 2001]. We can now produce animations of curling smoke and splashing water with striking visual realism. These techniques are particularly appealing since fluid animations are extremely difficult to create by hand. However, just as with any other simulation-based method, the user cannot freely design the behavior of the animation. One may manipulate the initial specifications of the simulation, such as viscosity, temperature, location and quantity of smoke, but, unfortunately, these changes often alter the animation in unpredictable ways, making it virtually impossible to produce fluid simulations that achieve a specific goal. Yet, for many applications, control of fluids would be tremendously useful. In animation production, where creative control is of the utmost importance, technical directors often require specific behavior from animations of physical phenomena such as smoke, fire and water.

Ideally, in the domain of smoke simulation, animators could specify a set of suggestive keyframes describing the desired behavior. Instead of manually adding wind forces to the airfield, a system would automatically solve for the control parameters to best meet the specified goals. The resulting simulation would follow the laws of fluid dynamics, not necessarily satisfying the constraints exactly, instead producing a “smoke-like” interpretation of the keyframes.

Unfortunately, controlling complex PDEs is very hard. This is particularly true for the Navier-Stokes equations that describe the dynamics of fluids. In fact, the complexity and non-linearity of these equations makes fluid simulations considerably harder to control than other dynamic phenomena such as rigid-body simulations, elastic deformations or character dynamics. Fluid simulations are also highly chaotic — slight changes in the simulation parameters may produce drastically different animations.

This paper presents a first step towards the control of fluid simulations. Specifically, we introduce a novel algorithm for controlling smoke, extending the widely used framework introduced in [Stam 1999]. The animator controls the simulation by specifying smoke-density and velocity keyframes and an optimization process determines the appropriate wind forces needed to satisfy the constraints.

In developing this framework, we present a method for computing exact derivatives through the fluid simulation (Sections 4 and 5). In addition, our keyframe paradigm leads to a formulation of a smooth objective function for comparing discrete grids of smoke densities based on blurring simulation states (Section 6). Lastly, we contribute a “layered” variation of multiple shooting adapted specifically for this domain (Section 8).

## 2 Related Work

Modeling the motion and appearance of fluid flows has long captivated the imagination of researchers in computer graphics. Early models that relied on simple flow primitives or animating texture maps achieved convincing effects typically by a long process of trial and error. Methods based on the physics of fluids on the other hand automatically generate convincing flows. Kajiya and Von Herzen [1984] were the first in computer graphics to use physics-based methods. However, the hardware at the time only allowed them to use very small grids. No real progress was made in this area until the works of Foster and Metaxas on modeling water [1996] and gases [1997b]. They showed that impressive animations could be created even on relatively coarse grids, though they require a strict bound on the time step for their simulations to remain stable, resulting in larger computation times. Related models were proposed in [Chen et al. 1997] and [Witting 1999]. Shortly thereafter, Stam [1999] introduced the Stable Fluids algorithm to address these limitations, using a semi-Lagrangian treatment of advection combined with an implicit solver for viscosity. Fluid solvers have been coupled with level-set methods to create impressive simulations of water [Foster and Fedkiw 2001; Enright et al. 2002] and fire [Nguyen et al. 2002]. For smoke, to combat the numerical dissipation inherent in the stable solution of the equations, Fedkiw *et al.* [2001] proposed to reinject the lost energy in the small vortices of the flow using a confinement force. This smoke model is currently the state of the art, and hence forms the basis of our work.

Much less attention has been devoted to the control of fluid flows. Foster and Metaxas [1997a] propose alternate ways for the user to control the forces by hiding the low level details. Later Foster and Fedkiw [2001] propose to control the motion of the flow by setting the velocity values at specific grid cells. The dynamics are therefore ignored at those cells, except that they still force the flow to be incompressible for additional realism. None of these works, however, allow the user to specify keyframes.

The solution of this problem is also important in other areas such as data assimilation and computational fluid control, where, for example, a shape's parameters are optimized to reduce air drag. Therefore, it is not surprising that methods similar to ours have been developed in the applied sciences [Bewley 2001]. Bewley *et al.* [2001; 2002] develop a general theory to calculate the derivatives of the fluid solver required by the optimizer. The idea behind these approaches is to compute derivatives by first integrating forward in time keeping the entire trace of the fluid state, and then integrating the adjoint equations backward in time. Similar adjoint methods were used for data assimilation in [Ghil et al. 1997]. This formulation theory applies to many different solvers, but, since the derivatives are exact only for a perfect integration of the fluid equations, it isn't suitable for use with the inaccurate solvers used in graphics. We stress that while the precision of the solver is not crucial, the accuracy of the derivatives is of utmost importance. For this reason, we compute exact derivatives of the actual algorithm, not those given by the more general continuous theory.

Our control formulation draws from work on interactive control of rigid-body simulations [Popović et al. 2000; Popović 2001]. As the animator specifies constraints at arbitrary points in time, the simulation parameters that satisfy the constraints are computed in realtime. These interactive speeds are achieved by rapid computation of derivatives through the rigid-body simulation process. Control of simulations can also be achieved by probabilistic sampling methods [Chenney and Forsyth 2000]. As the space of control parameters is sampled, this algorithm constructs a probability distribution of control parameters that achieve a specific goal. This approach is particularly useful when the cost function dependency on control parameters is not smooth. In our framework, evaluating the objective function is too costly to use this approach.

Multiple shooting methods have been used successfully to control complex dynamic system [Ascher et al. 1988; Stoer and Bulirsch 1993] and recently in computer graphics for controlling rigid-body simulations [Popović 2001]. This approach typically consists of two iterating steps. The first step splits the problem into smaller subproblems which are first solved separately. The second step tries to interpolate boundary constraints for each subproblem so that there is no discontinuity across the subproblems boundaries. We use a novel variant of the multiple shooting technique that does not require us to interpolate the boundary constraints.

## 3 Overview

Our system is based on the algorithm presented in [Stam 1999]. In this framework, a state $\mathbf{q}$ in a smoke simulation consists of a grid $\rho$ of densities and a grid $\mathbf{v}$ of velocity vectors. The simulation is computed from an initial state $\mathbf{q}_0$ by repeatedly applying a step function $S$ which advances the simulation by one unit of time. That is, the state at time $t$ is computed recursively:

$$\mathbf{q}_t = S(\mathbf{q}_{t-1}).$$

A smoke simulation of $n$ steps starts with the initial state $\mathbf{q}_0$ at time $t = 0$ and repeatedly calculates the subsequent states $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$. The simulation $\mathscr{S}$ is then just the sequence of states:

$$\mathscr{S}(\mathbf{q}_0) = (\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n).$$

### 3.1 Control

Often, animators desire to "control" a simulation, meaning that there exist some properties beyond physical plausibility that would be desirable. Perhaps one would like the smoke to follow a certain path or form a certain shape at a specific time. To achieve this by hand, an animator might try to control the flow by applying external forces throughout the simulation, gently blowing the smoke towards its goal. While this approach could generate realistic looking smoke, it is incredibly difficult: the non-linear nature of fluid movement makes tiny changes to forces in the beginning of the simulation capable of large and unexpected changes in simulation states further in time.

We propose a method to automate this task. In our system, the animator specifies a set of keyframes that the smoke should achieve. In this domain, a keyframe consists of a grid of smoke densities $\rho_t^*$ which the simulation's density grid $\rho_t$ should match as closely as possible at time $t$, thus enabling the animator to "sketch" the desired smoke movement. In addition, we allow velocity keyframes $\mathbf{v}_t^*$ which specify constraints on $\mathbf{v}_t$, allowing the animator to express goals such as "at time $t$, the velocity field should be still." Together, these types of keyframes can be used to describe a motion path for the smoke or a shape that it should try to achieve.

To influence the simulation, our system also needs a set of parameterized forces. For example, one force might apply a local gust of wind at a specific time, and the parameters could include the amount/direction of the wind force to be applied, and the location of the wind. The forces may come from a generic template or be hand-specified by the animator, but the control parameters are not filled in explicitly. Instead, these form a vector $\mathbf{u}$ that controls these aspects of the simulation. Any given value of $\mathbf{u}$ defines some unique set of forces applied to the field, and hence a unique simulation $\mathscr{S}(\mathbf{q}_0, \mathbf{u})$. Our algorithm uses an optimization process to solve for the control vector $\mathbf{u}$ which produces the simulation that best matches the animator's goal.

### 3.2 Matching Keyframes

In order to solve for $\mathbf{u}$, we need a way to assess how well the simulation matches the desired behavior. If we can express these desired properties in an objective function $\varphi$ which evaluates how well a simulation meets our goals, we can rephrase the control problem as a minimization of $\varphi$ over $\mathbf{u}$.

In designing the objective function, we have two goals. First, we would like the smoke simulation to match the keyframes as closely as possible. We express this as a function $\varphi_k(\mathscr{S}(\mathbf{u}, \mathbf{q}_0))$ that takes the simulation, and measures the "difference" between the user-specified keyframes and the corresponding simulation states. Second, we would like the system to use as little force as possible to achieve these keyframes. We therefore add a term $\varphi_s(\mathbf{u})$ that measures the amount of force added to the system. Our objective function $\varphi$ becomes a linear combination of these two terms:

$$\varphi = \varphi_k + \varphi_s.$$

The goal is now to solve for the control vector $\mathbf{u}$ that minimizes $\varphi$:

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

We use a gradient-based technique to solve this problem. The derivative $d\varphi/d\mathbf{u}$ is dependent on the pair of derivatives $d\varphi_k/d\mathbf{u}$ and $d\varphi_s/d\mathbf{u}$. The latter is straightforward, the former less so. Because $\varphi_k$ depends on $\mathscr{S}$, its derivative depends on $d\mathscr{S}/d\mathbf{u}$: how the simulation states are affected by changing each control parameter.

The derivative $d\mathscr{S}/d\mathbf{u}$ is crucial to our approach. We therefore devote Sections 4 and 5 to its analytic formulation.

### 3.3 Computing Exact Derivatives

There are several approaches one could take to calculate $d\mathscr{S}/d\mathbf{u}$. An approximation using finite differences is easy to implement, but is inaccurate and inefficient. Instead, one could start from the derivatives of the continuous Navier-Stokes equations. Unfortunately, while the semi-Lagrangian fluid algorithm produces physically plausible movement, it can vary tremendously from the actual analytic solution, and so even perfect derivatives of the continuous model would not be an accurate indicator of $d\mathscr{S}/d\mathbf{u}$. Instead, we propose a method of calculating exact derivatives of $\mathscr{S}$ by simulating the entire process in a space consisting not only of a density and velocity field, but also of their derivatives. This technique was motivated by a similar approach taken in controlling rigid-body simulations [Popović et al. 2000; Popović 2001].

To calculate the simulation derivatives, we augment the state $\mathbf{q}$ with its derivatives with respect to each control:

$$\hat{\mathbf{q}}_t = \left(\mathbf{q}_t, \frac{d\mathbf{q}_t}{du_1}, \ldots, \frac{d\mathbf{q}_t}{du_c}\right).$$

Similar to standard fluid simulation, these augmented states are computed recursively using a step function $\hat{S}$:

$$\hat{\mathbf{q}}_t = \hat{S}(\hat{\mathbf{q}}_{t-1}),$$
$$\hat{\mathbf{q}}_0 = (\mathbf{q}_0, \mathbf{0}, \ldots, \mathbf{0}).$$

To define $\hat{S}$ we must look more closely at the fluid simulation process. Recall that $S$ advances the simulation state by one timestep. In standard fluid solvers, $S$ is composed of a series of smaller operations, each performing a specific transformation of the state. In particular, we use:

$$S = M \circ A_\rho \circ P \circ D \circ A_\mathbf{v} \circ F.$$

$F$ applies forces to the velocity field, either from the simulation (e.g., applying forces upwards in areas of high density to make hot smoke rise) or user-specified control forces. $A_\mathbf{v}$ advects the velocity using a semi-Lagrangian method, $D$ performs diffusion on the velocity field using a simple implicit solver and finally $P$ projects the resulting field to be divergence-free. $A_\rho$ advects the smoke densities through this newly updated velocity field. Finally, $M$ is a mass preservation step to combat dissipation.

To calculate derivatives through this process, each of these operations induces a corresponding operation in our system taking state to state and derivative to derivative. For example, the mass-preservation step $M$ becomes

$$\hat{M} = \left(M, \frac{dM}{du_1}, \ldots, \frac{dM}{du_c}\right)$$

and similarly for $\hat{A}_\rho$, $\hat{P}$, $\hat{D}$, $\hat{A}_\mathbf{v}$, and $\hat{F}$. Therefore, a step $\hat{S}$ in our framework becomes:

$$\hat{S} = \hat{M} \circ \hat{A}_\rho \circ \hat{P} \circ \hat{D} \circ \hat{A}_\mathbf{v} \circ \hat{F}.$$

By simultaneously calculating the states, $\mathbf{q}_t$, and their derivatives, $d\mathbf{q}_t/d\mathbf{u}$, we can evaluate both $\varphi$ and $d\varphi/d\mathbf{u}$ in a single process. While the derivative calculation does substantially increase the simulation time, it is remarkably easy to implement: for most of the steps, the derivative calculation is very similar, if not identical, to the normal state calculation.

## 4 Derivatives

As described above, our technique requires the derivative $d\mathscr{S}/d\mathbf{u}$ at each timestep. We compute these in parallel with $\mathscr{S}$: every step of the fluid simulation has a corresponding step in the derivative calculation. We now present these derivatives, both as a guide to the implementor and to emphasize the role of the control vector $\mathbf{u}$ in our formulation, but we reassure the reader that the specifics of each derivative are not essential to understanding our approach.

### 4.1 Projection and Diffusion

The projection step $P$ forces the velocity field to be divergence-free. Diffusion $D$ accounts for the effects of viscosity. Both of these are linear operators. Therefore, their derivatives with respect to an arbitrary control parameter $u_k$ are:

$$\frac{dP}{du_k}(\mathbf{v}) = P\left(\frac{d\mathbf{v}}{du_k}\right) \quad \text{and} \quad \frac{dD}{du_k}(\mathbf{v}) = D\left(\frac{d\mathbf{v}}{du_k}\right).$$

In other words, the derivative of the projection step is the projection of the derivative, and likewise for diffusion. This means that we can use the identical algorithm in the derivative computation step as in the fluid simulation. This result holds regardless of the boundary condition on the fluid.

### 4.2 Advection

To advect a scalar field $\sigma$ through a velocity field $\mathbf{v}$, we backtrace through $\mathbf{v}$ and then update $\sigma$ with the value at the backtraced point. We present the derivation below using first-order Euler steps, though these equations could be generalized to higher-order schemes. If $I(\sigma, \mathbf{p})$ is the evaluation of a field $\sigma$ at position $\mathbf{p}$ using linear interpolation, the advection step becomes:

$$\begin{aligned} A(\mathbf{v}, \sigma, \mathbf{p}_0) &= I(\sigma, \mathbf{p}_s) \\ \mathbf{p}_i &= \mathbf{p}_{i-1} - \Delta t I(\mathbf{v}, \mathbf{p}_{i-1}). \end{aligned}$$

The backtrace for each grid cell starts at a fixed point $\mathbf{p}_0$ (implying $d\mathbf{p}_0/du_k = 0$) and generates a sequence of $s$ new positions by repeatedly evaluating $\mathbf{v}$ and taking steps in those directions scaled by the step size $\Delta t$. The advection is then an interpolation of $\sigma$ at the final generated position $\mathbf{p}_s$. The derivative of this step is straightforward, assuming we can calculate the derivative of linear interpolation, $dI/du_k$:

$$\begin{aligned} \frac{dA}{du_k}(\mathbf{v}, \sigma, \mathbf{p}_0) &= \frac{dI}{du_k}(\sigma, \mathbf{p}_s) \\ \frac{d\mathbf{p}_i}{du_k} &= \frac{d\mathbf{p}_{i-1}}{du_k} - \Delta t \frac{dI}{du_k}(\mathbf{v}, \mathbf{p}_{i-1}). \end{aligned}$$

We must now formalize $I$ and $dI/du_k$, which we present in 2D for simplicity though the equations are equivalent in 3D. The interpolation $I$ of a field $\sigma$ at position $\mathbf{p} = [p^x, p^y]^T$ is:

$$\begin{aligned} I(\sigma, \mathbf{p}) \; = \; & (1-\alpha)(1-\beta)\sigma_{l,m} + \alpha(1-\beta)\sigma_{l+1,m} \\ & + (1-\alpha)\beta\sigma_{l,m+1} + \alpha\beta\sigma_{l+1,m+1} \end{aligned}$$

where

$$l = \lfloor p^x \rfloor, m = \lfloor p^y \rfloor, \alpha = p^x - l, \beta = p^y - m.$$

Taking the derivative of this linear interpolation function using the chain rule gives us:

$$\frac{dI}{du_k} = \frac{\partial I}{\partial \sigma}\frac{d\sigma}{du_k} + \left( \frac{\partial I}{\partial \alpha}\frac{d\alpha}{dp^x}\frac{dp^x}{du_k} + \frac{\partial I}{\partial \beta}\frac{d\beta}{dp^y}\frac{dp^y}{du_k} \right).$$

The first term is a linear interpolation of $d\sigma/du_k$ at $\mathbf{p}$. In addition, the floor function is locally constant, so $d\alpha/dp^x = d\beta/dp^y = 1$. Therefore, this derivative can be rewritten:

$$\frac{dI}{du_k} = I(\frac{d\sigma}{du_k}, \mathbf{p}) + \left( \frac{\partial I}{\partial \alpha}\frac{dp^x}{du_k} + \frac{\partial I}{\partial \beta}\frac{dp^y}{du_k} \right).$$

Note that the floor function is discontinuous at integer values, leading to derivative discontinuities at voxel faces. In practice we did not find this to be an issue.

Since the first term in this derivative is an interpolation itself, a substantial portion of the code can be reused from the standard fluid solver. The only quantities that need to be computed are the partial derivatives of $I$ with respect to the residual values $\alpha$ and $\beta$. These are straightforward; for example, the first of these derivatives is:

$$\frac{\partial I}{\partial \alpha} = -(1-\beta)\sigma_{l,m} + (1-\beta)\sigma_{l+1,m} - \beta\sigma_{l,m+1} + \beta\sigma_{l+1,m+1}$$

As mentioned above, using this formulation for $I$ and $dI/du_k$, we could derive the advection for higher order integrators, such as Runge-Kutta. In practice, however, we found that taking only a single Euler step was sufficient.

### 4.3 Mass Preservation

In theory, advecting the smoke density through a divergence free field should preserve mass; however, in practice, the smoke mass slowly dissipates over the semi-Lagrangian advection. We address this by renormalizing the density field to the correct mass.

The "mass" of the smoke is given by the sum of its density over all grid points $\sum \rho$. Suppose the field should have mass $m$. We project $\rho$ to the density field $\rho'$ with proper mass as follows:

$$\rho' = \rho \frac{m}{\sum \rho}.$$

Therefore, the derivative with respect to any parameter $u_k$ is:

$$\frac{d\rho'}{du_k} = \frac{d\rho}{du_k}\frac{m}{\sum \rho} - \rho\frac{m}{(\sum \rho)^2}\sum \frac{d\rho}{du_k}.$$

### 4.4 Forces

In the force step, $F$, we add incremental velocities to the field $\mathbf{v}$:

$$F(\mathbf{v}) = \mathbf{v} + \mathbf{f}, \quad \tfrac{dF}{du_k}(\mathbf{v}) = \tfrac{d\mathbf{v}}{du_k} + \tfrac{d\mathbf{f}}{du_k}.$$

In practice, $\mathbf{f}$ consists of various forces, including heat forces $\mathbf{f}_\tau$, vorticity confinement $\mathbf{f}_\chi$, and the set of forces $\mathbf{f}_\mathbf{u}$ parameterized by the control vector $\mathbf{u}$:

$$\mathbf{f} = \mathbf{f}_\tau + \mathbf{f}_\chi + \sum \mathbf{f}_\mathbf{u}.$$

The derivatives of each of these forces with respect to $u_k$ must be computed and added to $d\mathbf{v}/du_k$. For example, the heat force causes smoke to rise along the $y$-axis proportionally to its density:

$$f^y_\tau = k_\tau \rho, \quad \tfrac{df^y_\tau}{du_k} = k_\tau \tfrac{d\rho}{du_k}.$$

The vorticity confinement force is more complicated. We offer a novel derivation of this force and its derivative in appendix A. We discuss the control forces in the subsequent section.

## 5 Control Parameters

The above framework is independent of the actual control parameters used in the simulation, and will work so long as the derivatives can be calculated. Here we present two possible types of control forces. The force parameters make up the control vector $\mathbf{u}$.

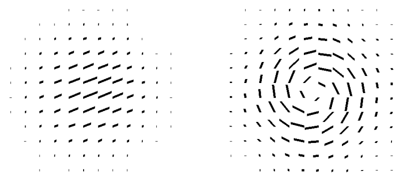
Figure 2: Control forces discretized onto a velocity grid. Wind forces (left), and vortex forces (right).

### 5.1 Wind Forces

Our basic control force $\mathbf{f}_\omega$ is a single vector applied to the grid, scaled by a Gaussian falloff function, creating a localized "wind." In 2D, if $\mathbf{w}$ is the wind direction and $\mathbf{c}$ is the center of the Gaussian:

$$(\mathbf{f}_\omega)_{i,j} = G_{i,j}\mathbf{w}$$

where $\Delta\mathbf{c} = [i, j]^T - \mathbf{c}$, $G_{i,j} = e^{-a|\Delta\mathbf{c}|^2}$, and $a$ determines the "width" of the Gaussian.

For fixed Gaussians, our system optimizes over the vector $\mathbf{w}$. That is, the components of $\mathbf{w}$ would be part of the control vector $\mathbf{u}$. We must therefore take the derivative with respect to $\mathbf{w}$:

$$\left(\frac{d\mathbf{f}_\omega}{d\mathbf{w}}\right)_{i,j} = G_{i,j}.$$

In some circumstances, we allow the system to optimize over the location of this force. In this case, the center $\mathbf{c}$ becomes a control parameter, and we need the derivative:

$$\left(\frac{d\mathbf{f}_\omega}{d\mathbf{c}}\right)_{i,j} = 2aG_{i,j}\mathbf{w}(\Delta\mathbf{c})^T.$$

These equations extend easily to the three dimensional case.

### 5.2 Vortex Forces

To create user-controlled vortices in the air field, we employ a similar Gaussian falloff approach. A single parameter $r$ controls the amount of rotational force applied:

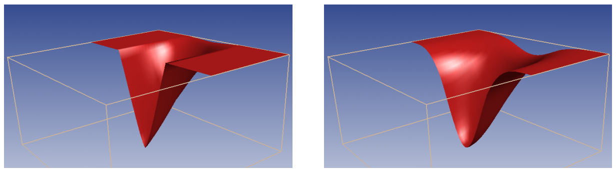
Figure 3: The effect of state blurring on an objective function for a two-dimensional control vector. Blurring (right) helps guide the optimization to a minimum by removing flat regions.

$$(\mathbf{f}_V)_{i,j} = rG_{i,j}\mathbf{R}\Delta\mathbf{c} \quad \text{where} \quad \mathbf{R} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$

is the matrix that rotates by $\pi/2$ and $G_{i,j}$ is defined as above. We take the derivatives of this vortex force with respect to the possible control parameters, the spin $r$ and the vortex center $\mathbf{c}$:

$$\left(\frac{d\mathbf{f}_V}{dr}\right)_{i,j} = G_{i,j}\mathbf{R}\Delta\mathbf{c}, \qquad \left(\frac{d\mathbf{f}_V}{d\mathbf{c}}\right)_{i,j} = rG_{i,j}\mathbf{R}(2a\Delta\mathbf{c}(\Delta\mathbf{c})^T - \mathbf{I}).$$

## 6 Objective Function

In the preceding sections, we formulated analytic derivatives of the controlled fluid simulation. This allows us to compute exactly how the objective function changes with respect to each control parameter.

Recall that the objective function $\varphi$ is composed of two terms: a smoothness term $\varphi_s$ and a keyframe-matching term $\varphi_k$. The smoothness term measures how much control force was added during the simulation. Let $\mathbf{f}_t$ be the grid of incremental velocities added to $\mathbf{v}_t$ by the control forces at time $t$. We define

$$\varphi_s = k_s \sum_{t=0}^{n} |\mathbf{f}_t|^2$$

where $k_s$ is a scaling constant, $n$ is the number of timesteps, and $|\mathbf{x}|^2 = \mathbf{x}\cdot\mathbf{x}$ is the sum of squared values taken over each grid cell. Therefore, the derivative with respect to an arbitrary control $u_k$ is:

$$\frac{d\varphi_s}{du_k} = 2k_s \sum_{t=0}^{n} \mathbf{f}_t \cdot \frac{d\mathbf{f}_t}{du_k}.$$

The other term of the objective function $\varphi_k$ measures how well the simulation matches the keyframes. That is, $\varphi_k$ measures the "error" between each keyframe and the corresponding state of the simulation. Suppose the density grid $\rho_t$ at time $t$ should match the corresponding density keyframe $\rho_t^*$. An obvious metric is given by $|\rho_t - \rho_t^*|^2$. Unfortunately, this will not work for our purposes.

The problem is that this function is extremely flat with respect to $\mathbf{u}$ unless the state happens to be close to the target. Imagine a state and a keyframe with smoke densities that do not overlap: slightly perturbing the smoke does not change this error value at all. This is problematic, since we hope to use the gradient of this function to direct us towards a solution. To resolve this issue, we blur both the state and the keyframe before evaluation (Figure 3). Each time we converge, we reduce the amount of blurring. Since blurring distributes over addition, our metric becomes $|B(\rho_t - \rho_t^*)|^2$. We also perform blurring for each velocity keyframe $\mathbf{v}_t^*$. Combining these gives us the keyframe-matching term of the objective function:

$$\varphi_k = k_d \sum_{t\in K_d} |B(\rho_t - \rho_t^*)|^2 + k_v \sum_{t\in K_v} |B(\mathbf{v}_t - \mathbf{v}_t^*)|^2.$$

Here, $K_d$ and $K_v$ are the sets of timesteps with density and velocity keyframes, respectively. The scaling terms $k_d$ and $k_v$ are described in Appendix B. The derivative of this term becomes

$$\frac{d\varphi_k}{du_k} = 2k_d \sum_{t\in K_d} B(\rho_t - \rho_t^*)\cdot B\left(\frac{d\rho_t}{du_k}\right) + 2k_v \sum_{t\in K_v} B(\mathbf{v}_t - \mathbf{v}_t^*)\cdot B\left(\frac{d\mathbf{v}_t}{du_k}\right).$$

Notice the dependency on the derivatives $d\rho_t/du_k$ and $d\mathbf{v}_t/du_k$ of the simulation. This shows explicitly why we had to compute derivatives through the entire fluid simulation.

Having computed the derivative of these two terms with respect to each control parameter $u_k$ we can combine them to form:

$$\frac{d\varphi}{d\mathbf{u}} = \frac{d\varphi_k}{d\mathbf{u}} + \frac{d\varphi_s}{d\mathbf{u}}$$

which tells us exactly how the objective function will change with respect to a change in any control parameter. This information will allow us find a smoke animation that matches the animator's desired behavior by iterating towards a minimum of the objective function.

## 7 Optimization Framework

The above definition of the objective function and its derivative provide us with all the elements needed to phrase the keyframe control of smoke as a function minimization of $\varphi$ over $\mathbf{u}$. There are numerous standard numerical methods for solving the function minimization problem. We use a limited memory quasi-Newton optimization technique [Zhu et al. 1994] which approximates the second derivative (the Hessian matrix) from a small set of most recent gradients. This technique has near-quadratic convergence with comparatively few evaluations of the function and gradient.

The quasi-Newton optimization requires us to evaluate $\varphi$ and its gradient $d\varphi/d\mathbf{u}$ for any value of the control parameters $\mathbf{u}$. In the preceding sections we derived these expressions mathematically. We now present an algorithm for computing these values. The function EVALUATE in Figure 4 takes $\mathbf{u}$ and computes both the objective and its gradient in parallel through a fluid simulation. Note that it is presented to help clarify the technique, not necessarily to describe an ideal implementation.

We include this pseudo-code to emphasize the parallels between the fluid simulation and the derivative calculation. Notice that we call identical functions for the linear steps PROJECT and DIFFUSE. Furthermore, the non-linear ADVECT and ADVECTDERIV functions can share interpolation code, as discussed in Section 4.2. Also note that at time $t$, we only calculate derivatives for the *active* controls, that is, parameters controlling forces that have already affected the simulation. All other controls have zero derivative.

The optimization process repeatedly calls EVALUATE, sampling the control space and iterating towards a local minimum. Each time this process converges, we decrease the blurring factor in the objective function and repeat, using the previous solution for $\mathbf{u}$ as the new starting point. The final optimization proceeds with no blurring at all, and we return this solution as the final control vector.

## 8 Layered Multiple Shooting

The framework described above works well for short simulations with reasonable control forces, but scales poorly when extended to lengthy problems. For one, the higher the dimensionality of the control vector, the more the optimizer needs to sample the space. Secondly, each sampling of the objective function increases in complexity with the dimension of $\mathbf{u}$. Since the derivatives for a specific control only need to be calculated from the point that control affects the simulation, more and more derivatives need to be computed

```
EVALUATE($\mathbf{u}$)
    $(\varphi, d\varphi/d\mathbf{u}) \leftarrow (0, \mathbf{0})$
    $(\rho, \mathbf{v}, d\rho/d\mathbf{u}, d\mathbf{v}/d\mathbf{u}) \leftarrow (\rho_0, \mathbf{v}_0, \mathbf{0}, \mathbf{0})$

    for $t \leftarrow 1$ to $n$
        – derivative calculation
        for each active control $u_k$
            $d\mathbf{f}/du_k \leftarrow$ FORCEDERIV($t$, $d\mathbf{v}/du_k$, $\mathbf{u}$)
            $d\mathbf{v}/du_k \leftarrow d\mathbf{v}/du_k + d\mathbf{f}/du_k$
            $d\mathbf{v}/du_k \leftarrow$ ADVECTDERIV($\mathbf{v}$, $\mathbf{v}$, $d\mathbf{v}/du_k$)
            $d\mathbf{v}/du_k \leftarrow$ DIFFUSE($d\mathbf{v}/du_k$)
            $d\mathbf{v}/du_k \leftarrow$ PROJECT($d\mathbf{v}/du_k$)
            $d\rho/du_k \leftarrow$ ADVECTDERIV($\mathbf{v}$, $\rho$, $d\rho/du_k$)
            $d\rho/du_k \leftarrow$ PRESERVEMASSDERIV($\rho$, $d\rho/du_k$)

        – fluid simulation
        $\mathbf{f} \leftarrow$ FORCE($t$, $\mathbf{u}$)
        $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{f}$
        $\mathbf{v} \leftarrow$ ADVECT($\mathbf{v}$, $\mathbf{v}$)
        $\mathbf{v} \leftarrow$ DIFFUSE($\mathbf{v}$)
        $\mathbf{v} \leftarrow$ PROJECT($\mathbf{v}$)
        $\rho \leftarrow$ ADVECT($\mathbf{v}$, $\rho$)
        $\rho \leftarrow$ PRESERVEMASS($\rho$)

        – objective function
        $\varphi \leftarrow \varphi$ + OBJECTIVE(t, $\rho$, $\mathbf{v}$, $\mathbf{f}$)
        for each active control $u_k$
            $d\varphi/du_k \leftarrow d\varphi/du_k$
                    + GRAD($t$, $\rho$, $d\rho/du_k$, $\mathbf{v}$, $d\mathbf{v}/du_k$, $\mathbf{f}$, $d\mathbf{f}/du_k$)
    return ($\varphi$, $d\varphi/d\mathbf{u}$)
```

Figure 4: Pseudo-code for evaluation of the objective and gradient.

each step. Consequently, if control forces are distributed evenly over time, doubling the length of the simulation will approximately quadruple the complexity of each derivative calculation. Lastly, solving for long simulations with many keyframes is more likely to get stuck in a local minimum of the cost function.

To resolve these issues, we use a novel form of multiple shooting adapted for this domain. The central idea behind multiple shooting is to temporally break a complex problem into a set of subproblems, each of which can be solved locally. The solutions to each of these segments are then used to inform its neighbors, propagating knowledge back and forth as we iterate towards a global solution.

Standard multiple shooting approaches interpolate the boundaries to maintain continuity. However, meaningfully interpolating smoke densities is not straightforward, and so we present a variation called "layered multiple shooting" that eliminates the need for boundary interpolation by using overlapping sets of subproblems.

## 8.1 Overlapping Schedules

To begin, we split the simulation into a sequence of segments, called the "initial schedule," where each segment boundary lies on a keyframe. In Figure 5, we label this sequence $A$, $B$, $C$, and $D$. Each of these subproblems is solved locally using the framework described above, optimizing only over the control parameters that have a direct effect during that time interval. Merging these solutions results in a poor guess for $\mathbf{u}$ that we must iteratively refine to reach our desired minimum.

This is achieved through an "alternate schedule" of subproblems ($E$, $F$, and $G$ in Figure 5), whose segments overlap those in the initial schedule. The start and end points of these segments are culled from the intermediate states of the initial segments. In other words, the state generated at the midpoint of each initial subproblem is added to the end of each alternate subproblem as a "pseudo-keyframe," shown as orange arrows. When solving these overlapping subproblems, the optimizer attempts to hit both the pseudo-keyframe at the end, and all the original keyframes along the way. This updates $\mathbf{u}$ and propagates knowledge of non-local keyframes across the boundaries of the initial schedule.

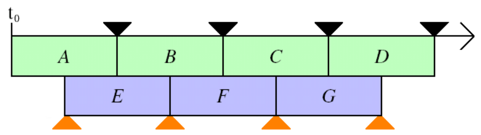

Figure 5: Multiple shooting schedules. Segments $A$, $B$, $C$, and $D$ form the initial schedule; $E$, $F$, and $G$ form the alternate schedule.

We alternate between the initial and alternate schedules, refining the control vector $\mathbf{u}$ and updating the pseudo-keyframes. The subproblems in the initial schedule always keep their original target keyframes, but these are augmented with the velocities of the overlapping segments' solutions. In this way, each pass through the initial schedule not only has a better starting value for $\mathbf{u}$, it also has an improved final target velocity for solving the later segments in the schedule.

## 8.2 Parallel vs. Sequential Schedule Processing

In the process described above, each segment starts directly from a keyframe or pseudo-keyframe. We term this the "parallel" approach, since each segment is completely independent of the other subproblems (a multi-processor implementation could compute these simultaneously). While optimizing the schedule in this fashion will smoothly improve the overall solution, the endpoints of the segments will never quite match up and error will build up when applying the composed $\mathbf{u}$ to the entire simulation.

An alternative method for processing the segments in a schedule is to begin each subproblem with the previous endpoint's solution (Figure 6). Initially, this sequential approach performs badly, since each segment must counteract bad decisions made by the previous segments. On the other hand, with a reasonable guess for $\mathbf{u}$ sequential processing converges much faster than the parallel technique. In our multiple-shooting examples, we perform two sets of parallel schedules to provide a good initial guess, followed by two sets performed sequentially to converge quickly to the minimum.

## 9 Results

We have used our optimization framework to generate several examples of keyframed smoke animations and found it robust across a broad range of constraints, in the sense that it produced "smoke-like" animations that reasonably approximated the keyframes.

Figure 7 shows a 3D simulation on a 30x30x30 grid where a ball of smoke is instructed to split in three directions. By restricting the

$$\begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \Rightarrow \begin{bmatrix} E \\ F \\ G \end{bmatrix} \Rightarrow \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} \Rightarrow \cdots$$

$$[\, A \rightarrow B \rightarrow C \rightarrow D \,] \Rightarrow [\, E \rightarrow F \rightarrow G \,] \Rightarrow \cdots$$

Figure 6: Parallel vs. Sequential Processing. In parallel processing (top), each segment is run independently, with boundaries defined by keyframe information. In sequential processing, each segment is processed in order, starting from the ending state of the previous.

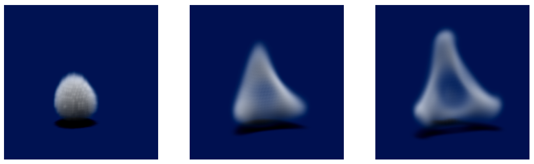
Figure 7: A ball of smoke splits in three directions.

control (20 wind forces) to the first time step of the simulation, we are essentially solving for the initial velocity grid and letting the rest of the simulation proceed unhindered. The optimization took 2 hours to run on a 2Ghz Pentium 4, though we have not made significant efforts at efficiency and believe all the examples could be achieved substantially more quickly, for example, by leveraging the parallelism described in Section 8.2.

Figure 9 demonstrates complicated keyframe shapes that can only be achieved with more fine-grain control (9 wind forces and 4 moving vortices, every 3 time steps). This example, though it only had two keyframes, benefited from our multiple shooting strategy: the 35 step simulation has 408 control parameters, far too many to optimize at once. With multiple shooting, it ran for just under 24 hours and produced an extremely close match to both keyframes.

The letters in Figure 8 were achieved by running 5 separate simulations from identical initial states. The solution to each of these 50x50 simulations took 2-5 hours, at which point their resulting states were merged into a single grid and further simulated to dissipate into one another. Occasionally, the system would get stuck in an unsatisfactory state (e.g., producing an "E" missing the center bar). Inserting a suggestive intermediate keyframe usually guided the system out of these local minima. This form of intuitive user interaction greatly expanded the set of shapes the system could achieve, without needing to increase the number of controls.

In other cases, the resulting animation matched the keyframes, but felt "too controlled." We refer the reader to our video for an example of a ball of smoke splitting and orbiting itself; in this case, the keyframes were meant more as a sketch of the overall animation than as strict shapes to hit. After increasing the smoothness term in the objective function and decreasing the controls to 9 wind forces every 4 frames, the results became more physically plausible and "smoke-like." To keep the animation realistic, we often refined the keyframes or adjusted their timing, resulting in a process which felt remarkably similar to keyframing in traditional animation.

## 10 Discussion and Future Work

In this paper, we have presented a novel technique for controlling smoke simulations through keyframing, allowing the user considerable control over the animation while retaining the realism and nuance of a fluid simulation. This idiom represents a significant advance over previous methods of controlling fluids through direct manipulation of simulation parameters.

In achieving these goals, we have developed an approach for computing exact derivatives through the fluid simulation process most widely used in graphics today. We presented a keyframe-matching objective function with meaningful gradients for this domain. Lastly, we introduced a variant on multiple shooting which doesn't require interpolating boundary states.

In addition, we believe our approach generalizes to a wider range of problems. Our formulation of derivatives is directly applicable to the many applications already using this fluid solver. Moreover, the approach to calculating exact derivatives could be applied to any system of PDEs that can be solved as a composition of differentiable functions. Finally, our staggered schedule approach to multiple-shooting would be useful in many optimization processes to avoid interpolating states.

Our current approach suffers from several drawbacks. It works well for problems of similar size to the examples presented, but becomes computationally prohibitive for large problems with fine-grained control. In addition, the quasi-Newton optimization is efficient, but can get caught in local minima, especially if keyframes are spaced far apart or if controls compete against one another. One possible direction to explore is multi-resolution force frameworks, solving the system repeatedly for increasingly fine control. This could not only significantly improve computation time, but might help avoid poor minima due to competing fine-grain control.

Another interesting avenue to pursue is the application of our technique to paradigms other than keyframing. One such approach would be to match fluid simulations to videos of real smoke. This would be useful in effects production, where real pyrotechnic elements could be digitized to interact with a 3D environment.

In conclusion, we hope that the initial steps presented in this paper and future work in controlling fluids will make simulation-based techniques more useful and help bridge the gap in animation between physical realism and artistic expression.

**Acknowledgments.** The authors would like to thank Siobhan Quinn for providing voice-over for the accompanying video. This work was supported by the UW Animation Research Labs, NSF grant CCR-0092970, ITR grant IIS-0113007, Alias|Wavefront, Microsoft Research, Intel, Electronic Arts, and Sony.

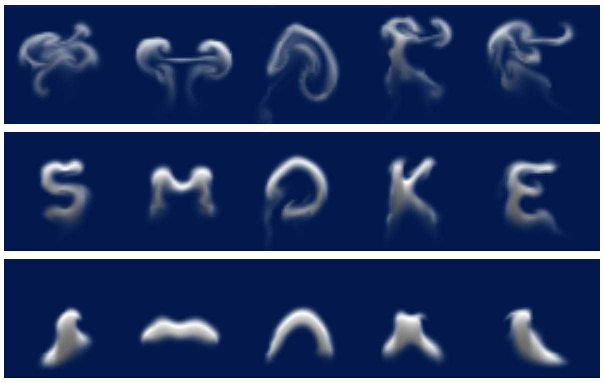
Figure 8: Smoke rising to form the letters "SMOKE."

## A Vorticity Confinement Forces

To counteract the numerical dampening of the semi-Lagrangian advection step, we add a "vorticity confinement" force $\mathbf{f}_\chi$. This force prevents spinning vortices from dissipating too quickly.

Our computation of $\mathbf{f}_\chi$ is a variation of that found in [Fedkiw et al. 2001]. Traditionally, vorticity confinement is defined in terms of the *norm* function $|\mathbf{x}| = \sqrt{\mathbf{x}\cdot\mathbf{x}}$ whose derivative has a singularity at $\mathbf{x} = \mathbf{0}$. This is problematic because the differential field $d\mathbf{v}/du_k$ often vanishes in practice. We address this by replacing the norm with the *pseudo-norm*:

$$\wr\mathbf{x}\wr = \sqrt{\mathbf{x}\cdot\mathbf{x}+e_0} \qquad \frac{d\wr\mathbf{x}\wr}{du_k} = \left(\mathbf{x}\cdot\frac{d\mathbf{x}}{du_k}\right)/\wr\mathbf{x}\wr.$$

Setting $e_0$ to a small positive constant makes the pseudo-norm's derivative everywhere well defined, while preserving the qualitative effects of vorticity confinement. The vorticity confinement equations therefore become:

$$\omega = \nabla\times\mathbf{v}, \qquad \eta = \nabla\wr\omega\wr,$$
$$N = \eta/\wr\eta\wr, \qquad \mathbf{f}_\chi = \varepsilon(N\times\omega).$$

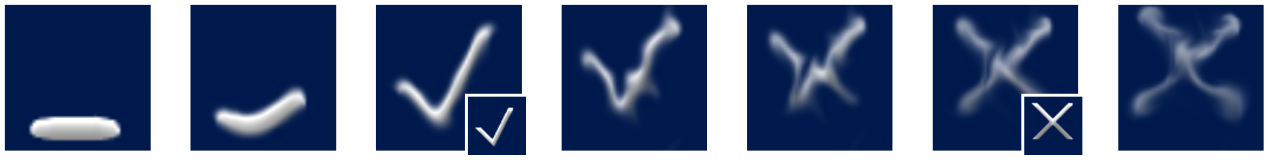

Figure 9: An animation of smoke forming "check" then "x." The keyframes for this animation are inset.

This implies the following set of derivatives with respect to $u_k$:

$$\begin{aligned}
\frac{d\omega}{du_k} &= \nabla \times \frac{d\mathbf{v}}{du_k} \\
\frac{d\eta}{du_k} &= \nabla \frac{d \wr \omega \wr}{du_k} \\
\frac{dN}{du_k} &= \left( \frac{d\eta}{du_k} \wr \eta \wr - \eta \frac{d \wr \eta \wr}{du_k} \right) / \wr \eta \wr^2 \\
\frac{d\mathbf{f}_\chi}{du_k} &= \varepsilon \left( \frac{dN}{du_k} \times \omega + N \times \frac{d\omega}{du_k} \right).
\end{aligned}$$

## B Objective Function Scaling Terms

In the objective function, we would like the constants $k_d$, $k_v$ and $k_s$ to scale appropriately with factors like the grid dimension and the amount of mass. They can be defined as follows:

$$k_d = \frac{k'_d}{n_{kd} m}, \quad k_v = \frac{k'_v}{4 v_M^2 n_{kv} d}, \quad k_s = \frac{k'_s}{f_M^2 n_f d}$$

where $m$ is the mass of smoke (we re-normalize the mass each step so this remains constant). $v_M$ is the desired upper bound on the length of velocity vectors and $f_M$ is corresponding desired bound on the vector added in the force step. $n_{kd}$, $n_{kv}$, $n_f$ are the number of density key frames, velocity key frames, and frames with applied forces, respectively. The constant $d$ is the product of the grid dimensions, that is, the number of voxels in the simulation. This way, $k'_d$, $k'_v$, and $k'_s$ can be adjusted to favor any one term, but they all should be around 1.0 and should scale to different simulations.

## References

ASCHER, U. M., MATTHEIJ, R. M. M., AND RUSELL, R. D. 1988. *Numerical Solution of Boundary Value Problems for Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, New Jersey.

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. 2001. Flow control: new challenges for a new renaissance. *Progress in Aerospace Sciences 37*, 21–58.

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*.

CHEN, J. X., DA VITTORIA LOBO, N., HUGHES, C. E., AND MOSHELL, J. M. 1997. Real-Time Fluid Simulation in a Dynamic Virtual Environment. *IEEE Computer Graphics and Applications* (May-June), 52–61.

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.

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.

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

NGUYEN, D., FEDKIW, R., AND JENSEN, H. 2002. Physically Based Modeling and Animation of Fire. In *Computer Graphics (SIGGRAPH 2002)*, ACM, 736–744.

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.

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

STOER, J., AND BULIRSCH, R. 1993. *Introduction to Numerical Analysis*, 2nd ed. Springer.

WITTING, P. 1999. Computational Fluid Dynamics in a Traditional Animation Environment. In *Computer Graphics (SIGGRAPH 99)*, ACM, 129–136.

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