# Subspace Fluid Re-Simulation

Theodore Kim*
Media Arts and Technology Program
University of California, Santa Barbara

John Delaney†
Media Arts and Technology Program
University of California, Santa Barbara

**Figure 1:** *An efficient subspace re-simulation of novel fluid dynamics. This scene was generated an* **order of magnitude** *faster than the original. The solver itself, without velocity reconstruction (§5), runs* **three orders of magnitude** *faster.*

## Abstract

We present a new subspace integration method that is capable of efficiently adding and subtracting dynamics from an existing high-resolution fluid simulation. We show how to analyze the results of an existing high-resolution simulation, discover an efficient reduced approximation, and use it to quickly "re-simulate" novel variations of the original dynamics. Prior subspace methods have had difficulty re-simulating the original input dynamics because they lack efficient means of handling semi-Lagrangian advection methods. We show that multi-dimensional *cubature* schemes can be applied to this and other advection methods, such as MacCormack advection. The remaining pressure and diffusion stages can be written as a single matrix-vector multiply, so as with previous subspace methods, no matrix inversion is needed at runtime. We additionally propose a novel importance sampling-based fitting algorithm that asymptotically accelerates the precomputation stage, and show that the Iterated Orthogonal Projection method can be used to elegantly incorporate moving internal boundaries into a subspace simulation. In addition to efficiently producing variations of the original input, our method can produce novel, abstract fluid motions that we have not seen from any other solver.

**CR Categories:** I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically based modeling

**Keywords:** fluid simulation, subspace integration, cubature
**Links:** DL PDF

*kim@mat.ucsb.edu
†delaneyj@mat.ucsb.edu

## 1 Introduction

Fluid simulation methods have made great recent progress, but working with high-resolution fluids can still be a time-consuming process. Once a large-scale simulation has completed, the results are usually considered static; obtaining new results involves launching another long-running simulation. However, having already paid the cost of simulating a sequence, can we somehow analyze its dynamics and use the results to efficiently *re-simulate* sequences that are similar to the existing one? Such a method would afford users considerable freedom when tuning parameters, as each tweak would not trigger hours of simulation. A single simulation could also be used to quickly generate high-quality effects libraries where many similar versions of the same element are needed, such as a collection of steam elements for a kitchen scene [Shah 2007].

We use *subspace integration* to construct an efficient fluid re-simulator, because it has been known to yield large simulation accelerations. This acceleration is realized by analyzing the results of previous $N$-dimensional simulations, extracting an $r$-dimensional basis, and simulating within this rank-$r$ reduced basis. In general, $r \ll N$, so large speedups can be realized. These methods are also

variously known as *model reduction*, *reduced order*, or *proper orthogonal decomposition* methods. They have a long history in fluid simulation [Lumley 1967; Berkooz et al. 1993], in part because turbulence analysis was one of their initial applications. Effective subspace fluid simulation methods have also been developed for computer graphics [Treuille et al. 2006; Wicke et al. 2009]. However, these works do not support re-simulation, as they are not (and do not claim to be) *consistent* integrators [Carlberg et al. 2011]. They leverage existing simulation data, but *cannot* reproduce the dynamics of that original data.

The inconsistency arises because these methods do not use subspace analogs of the standard integration methods used in computer graphics [Stam 1999]. In place of standard semi-Lagrangian and implicit schemes, they use finite difference and exponential schemes. The likely reason for this departure is that it has been unclear how to apply the *projected tensor* technique that usually enables efficient subspace integration [Treuille et al. 2006; Barbič and James 2005; Pentland and Williams 1989] to semi-Lagrangian methods. In this work, we present a subspace fluid simulator that performs efficient ($N$-independent) semi-Lagrangian advection and enables fast re-simulation. Our technical contributions are:

- A *cubature* approach [An et al. 2008] for efficiently computing semi-Lagrangian advection entirely within a reduced subspace. The approach generalizes to other non-linear techniques, such as MacCormack advection [Selle et al. 2008].
- A scalable, asymptotically faster method of computing the cubature points of a subspace. Precomputation time for one test case dropped from almost six days to less than half an hour.
- Efficient re-simulation with modified parameters, such as buoyancy, vorticity confinement, and total number of time steps in the simulation.
- A subspace formulation of a standard fluid solver [Stam 1999] that combines the entire pressure and diffusion stages into a *single matrix-vector multiply*.
- A subspace method for efficiently handling both static and moving obstacles based on the *Iterated Orthogonal Projection* [Molemaker et al. 2008] method.

Finally, when parameters values deviate extremely from their original values, we have found that the subspace solver does not diverge, but instead stably generates abstract dynamics that we have not seen from any other solver.

## 2 Previous Works

Since the pioneering work of Foster and Metaxas [1997] and Stam [1999], many methods have been proposed for accelerating fluid simulations. These include spatially adaptive [Losasso et al. 2004; Klingner et al. 2006], synthetic turbulence [Kim et al. 2008; Narain et al. 2008; Schechter and Bridson 2008], low viscosity [Molemaker et al. 2008; Mullen et al. 2009], coarse grid [Lentine et al. 2010], and vortex [Brochu et al. 2012; Pfaff et al. 2012] methods. We present a subspace method that is largely orthogonal to these approaches, and could be combined with any of them to produce additional accelerations.

Subspace methods were first introduced to the graphics community by Pentland and Williams [1989] and have been highly successful at efficiently simulating solids. While the initial work was in linear materials, it has since been extended to non-linear St. Venant-Kirchhoff [Barbič and James 2005], Arruda-Boyce, and Mooney-Rivlin [An et al. 2008] materials. An excellent recent tutorial [Sifakis and Barbič 2012] is available on the topic. Treuille et al. [2006] pioneered subspace methods for fluids in computer graphics, and subsequent work showed how to generalize the approach to modular tiles [Wicke et al. 2009]. Most recently, this work was generalized from reduced polynomials to general reduced algebraic functions [Stanton et al. 2013]. The eigenfunction work of DeWitt et al. [2012] could also be viewed as a subspace method, albeit one where basis functions are obtained via static analysis, not from the SVD of existing simulation data. Our method is an alternative that can be used in all of the applications areas of the previous work, but additionally enables efficient re-simulation.

As mentioned in §1, subspace fluid simulation has a long history in engineering [Lumley 1967; Berkooz et al. 1993]. Recent work has developed control methods [Bergmann et al. 2005], improved error bounds [Deparis and Rozza 2009], and improved the stability [Amsallem and Farhat 2012; Serre et al. 2012] of subspace integrators. Subspace *re-simulation* has been studied extensively, particularly in aerospace engineering [LeGresley and Alonso 2001; Anttonen et al. 2003; Bourguet et al. 2011]. However, no subspace method has been developed that supports semi-Lagrangian and similar backtrace-based advection schemes. Efficient re-simulation of scenarios that employ them has therefore not been feasible.

We design a method of efficient supporting these types of schemes within a subspace framework by employing the notion of *cubature*, i.e. multi-dimensional quadrature. This approach has been successfully applied to subspace material non-linearities in solid and shell mechanics [An et al. 2008; Kim and James 2009; Chadwick et al. 2009; Kim and James 2011]. Similar point-based approaches have also been successful, such as Monte-Carlo sampling [Baraff and Witkin 1992], and Key Point Subspace Acceleration (KPSA) [Meyer and Anderson 2007]. All these works deal with solids and shells; we will show that the approach generalizes to fluids.

## 3 Subspace Navier-Stokes

**Notation:** We will denote scalars using unbolded lower case, e.g. $w$, vectors using bold lower-case, e.g. $\mathbf{x}$, and matrices and tensors using bold upper-case, e.g. $\mathbf{A}$. Arbitrary non-linear functions will be denoted in script, e.g. $\mathcal{N}(\mathbf{x})$. When using higher-order tensors, we will follow the convention of Vasilesu and Terzopoulos [2004] and use $\times_i$ to denote a product with respect to the $i$th mode. For example, the matrix product $\mathbf{L}^T\mathbf{S}\mathbf{R}$ would instead be written as $\mathbf{S} \times_1 \mathbf{L} \times_2 \mathbf{R}$. For a third-order tensor $\mathbf{T} \in \mathbb{R}^{N\times N\times N}$, a product with respect with the third mode would be written as $\mathbf{T} \times_3 \mathbf{R}$. *Reduced* subspace variables will be denoted with a tilde, e.g. $\widetilde{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$ denotes the projection of vector $\mathbf{x}$ into the subspace spanned by $\mathbf{U}$. When referring to full rank quantities, we will use the variable $N$, and when referring to reduced rank quantities, the variable $r$. We will assume that $r \ll N$. In all of the following, $\mathbf{U} \in \mathbb{R}^{3N\times r}$, as each of the $N$ grid cells usually store a 3D vector.

### 3.1 The Projected Tensor Approach

To motivate our approach, we will summarize the subspace approach of Treuille et al. [2006], which we characterize as a *projected tensor* approach. The Navier-Stokes equations are:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\nabla^2\mathbf{u} + \nabla p + \mathbf{f}_e \quad (1)$$

$$\nabla\cdot\mathbf{u} = 0, \quad (2)$$

where $\mathbf{u}$ is the fluid velocity, $\nu$ is a viscosity constant, $p$ is pressure, and $\mathbf{f}_e$ is a vector of external forces. For a discrete simulation grid containing $N$ cells, $\mathbf{u} \in \mathbb{R}^{3N}$ and $\mathbf{f}_e \in \mathbb{R}^{3N}$.

In Treuille et al. [2006], a divergence-free basis $\mathbf{U}$ is used which eliminates the need to handle Eqn. 2 and the $\nabla p$ term in Eqn. 1.

The diffusion operator $\nu\nabla^2\mathbf{u}$ is purely linear, so if the diffusion discretization is written as a matrix $\mathbf{V}$, it can be projected to $\widetilde{\mathbf{V}} = \mathbf{U}^T\mathbf{V}\mathbf{U}$. An explicit subspace diffusion step can then be performed as $\widetilde{\mathbf{u}}^* = \nu\widetilde{\mathbf{V}}\widetilde{\mathbf{u}}$, where $\mathbf{u}^*$ denotes an intermediate variable. In lieu of a simple evaluation, Treuille et al. [2006] instead perform a full integration by employing an exponential, $\widetilde{\mathbf{u}}^* = \mathbf{e}^{\nu\widetilde{\mathbf{V}}}\widetilde{\mathbf{u}}$, where $\mathbf{e}$ denotes a matrix exponential [Moler and Van Loan 2003].

The more difficult non-linear advection term, $-(\mathbf{u}\cdot\nabla)\mathbf{u}$, must also be addressed. Unlike the diffusion term, it cannot be written as a static matrix $\mathbf{V}$, projected to a smaller $\widetilde{\mathbf{V}}$, and then used repeatedly at runtime. If a *finite difference* discretization is used, the $-(\mathbf{u}\cdot\nabla)$ term can be written as as matrix $\mathbf{A}_\mathbf{u} \in \mathbb{R}^{3N\times 3N}$, and the full advection can then be written as $\mathbf{u}^* = \mathbf{A}_\mathbf{u}\mathbf{u}$. However, $\mathbf{u}$ changes every timestep, which correspondingly changes $\mathbf{A}_\mathbf{u}$, and frustrates any attempt to precompute a useful static matrix.

Treuille et al. [2006] observed that a static 3rd order tensor, $\mathbf{A} \in \mathbb{R}^{3N\times 3N\times 3N}$ can be constructed, yielding $(\mathbf{A}\times_3\mathbf{u})\mathbf{u} = \mathbf{A}_\mathbf{u}\mathbf{u} = \mathbf{u}^*$. This static tensor *can* be projected, $\mathbf{A}\times_1\mathbf{U}\times_2\mathbf{U}\times_3\mathbf{U} = \widetilde{\mathbf{A}} \in \mathbb{R}^{r\times r\times r}$, and reused at runtime: $(\widetilde{\mathbf{A}}\times_3\widetilde{\mathbf{u}})\widetilde{\mathbf{u}} = \widetilde{\mathbf{A}}_{\widetilde{\mathbf{u}}}\widetilde{\mathbf{u}} = \widetilde{\mathbf{u}}^*$. A stabler exponential scheme is again used: $\widetilde{\mathbf{u}}^* = \mathbf{e}^{\widetilde{\mathbf{A}}_{\widetilde{\mathbf{u}}}}\widetilde{\mathbf{u}}$. The complete integration from time $t$ to $t+1$ can now be written:

$$\widetilde{\mathbf{u}}^{t+1} = \left(\mathbf{e}^{\widetilde{\mathbf{A}}\times_3\widetilde{\mathbf{u}}^t + \nu\widetilde{\mathbf{D}}}\right)\widetilde{\mathbf{u}}^t. \tag{3}$$

**Discussion:** This approach is successful at efficiently computing fluid dynamics within the subspace spanned by $\mathbf{U}$. However, it is not (and does not claim to be) a *consistent* integration method [Carlberg et al. 2011]. The columns of $\mathbf{U}$ are obtained by performing an SVD on simulation data produced by a standard solver [Stam 1999; Fedkiw et al. 2001]. However, if the *exact same* initial conditions and forcings used to generate this data are simulated using Eqn. 3, they will produce significantly different dynamics (Figure 2). This is to be expected, as Eqn. 3 uses totally different spatial and temporal discretizations: finite difference and exponential schemes instead of semi-Lagrangian and implicit schemes. This also makes Eqn. 3 difficult to apply to re-simulation, as it does not naturally capture the dynamics of the original input simulation.

In order to obtain a consistent subspace integration scheme, the integration methods used by the original solver must be matched. Unfortunately, it is unlikely that the approach of projecting a static advection tensor $\mathbf{A}$ can be applied to semi-Lagrangian schemes. In the finite difference case, the entries of the advection matrix $\mathbf{A}_\mathbf{u}$ change every timestep, but the *spatial locations* of its stencils remain static. In a semi-Lagrangian scheme, the backtrace rays select *new* stencil locations every timestep, making it impossible to build a single static 3rd order tensor. Promoting the tensor to 4th or 5th order may seem promising, but these would only capture higher order polynomial effects on static stencils; it does not address the fact that the locations change. While generalizations to algebraic functions are presented in [Stanton et al. 2013], the underlying static stencil problem remains. Clearly, a different approach is needed.

### 3.2 The Cubature Approach

An et al. [2008] proposed a *cubature* approach to computing subspace internal force response in non-linear solid mechanics. We believe that this approach can be made quite general, so we will describe it generically here. Assume we have a vector of $N$ points, $\mathbf{x} \in \mathbb{R}^{3N}$, and some non-linear function $\mathcal{F}$. We can evaluate $\mathcal{F}$ at any arbitrary point $p$ by extracting the appropriate rows from $\mathbf{x}$ to construct $\mathbf{x}_p \in \mathbb{R}^3$, and computing $\mathcal{F}_p$, a point-sampled version of $\mathcal{F}$, as $\mathbf{f}_p = \mathcal{F}_p(\mathbf{x}_p) \in \mathbb{R}^3$. If this is done for all $N$ points, we obtain

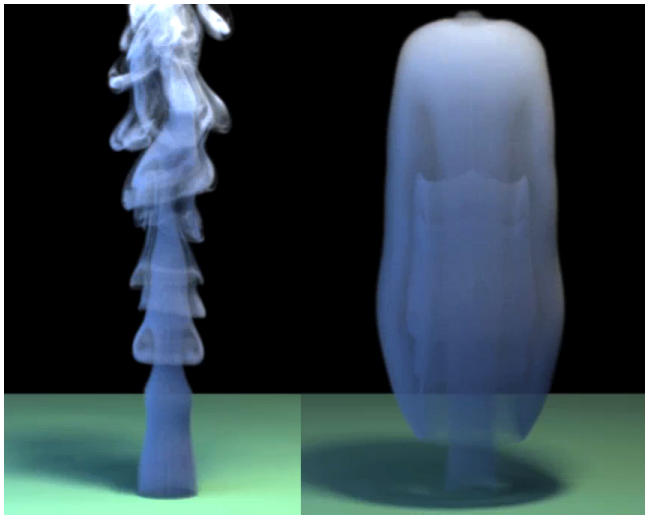

**Figure 2: Left:** *Frame from a standard [Stam 1999] fluid simulation.* **Right:** *Results of trying to reproduce the left simulation using the projected tensor [Treuille et al. 2006] approach. Different spatial and temporal discretizations are used, so without a richer basis, only the lowest frequency modes become active.*

a vector $\mathbf{f} \in \mathbb{R}^{3N}$ that represents $\mathbf{x}$ non-linearly transformed by $\mathcal{F}$,

$$\mathbf{f} = \mathcal{F}(\mathbf{x}). \tag{4}$$

The $\mathcal{F}$ is generic, so $\mathbf{x}$ can represent a velocity field before advection, $\mathbf{f}$ the field after, and $\mathcal{F}$ any arbitrary (semi-Lagrangian, MacCormack, finite difference) advection scheme.

We need a method that takes a reduced $\tilde{\mathbf{x}} = \mathbf{U}^T\mathbf{x}$ vector and efficiently outputs a reduced vector $\tilde{\mathbf{f}}$. As seen in §3.1, $\mathcal{F}$ can sometimes be written as a tensor, and a projected version can be used to efficiently compute $\widetilde{\mathbf{f}}$. What if $\mathcal{F}$ cannot be easily written as a tensor? A slow but viable method [Krysl et al. 2001] is to compute and project the full $3N$-dimensional $\mathbf{f}$ vector:

$$\widetilde{\mathbf{f}} = \mathbf{U}^T\mathbf{f} = \mathbf{U}^T\mathcal{F}(\mathbf{x}) = \mathbf{U}^T\mathcal{F}(\mathbf{U}\tilde{\mathbf{x}}). \tag{5}$$

Unfortunately, this $N$-dependent computation largely discards the speedups that would be obtained by a "subspace-only" method. However, Eqn. 5 is in fact a multi-dimensional integral over the entire simulation domain $\Omega$, with respect to the basis functions encoded by the columns of $\mathbf{U}$:

$$\widetilde{\mathbf{f}} = \mathbf{U}^T\mathcal{F}(\mathbf{U}\tilde{\mathbf{x}}) = \int_\Omega \mathbf{U}_p^T\mathcal{F}_p(\mathbf{U}_p\tilde{\mathbf{x}}). \tag{6}$$

Above, $\mathbf{U}_p \in \mathbb{R}^{3\times r}$ are the rows in $\mathbf{U}$ that correspond to some point $p$. It is straightforward to point sample this integral if we isolate the term under the integral thusly:

$$\widetilde{\mathbf{f}}_p = \mathbf{U}_p^T\mathcal{F}_p(\mathbf{U}_p\tilde{\mathbf{x}}). \tag{7}$$

Note that due to the left-most projection, $\widetilde{\mathbf{f}}_p \in \mathbb{R}^r$, not $\mathbb{R}^3$. With this point sampling method, we can apply the notion of *cubature* [Press et al. 1992] and approximate the integral as a weighted sum of $\mathcal{F}$ evaluations at $P$ carefully chosen samples:

$$\int_\Omega \mathbf{U}_p^T\mathcal{F}_p(\mathbf{U}_p\tilde{\mathbf{x}}) \approx \sum_{p=1}^{P} w_p\mathbf{U}_p^T\mathcal{F}_p(\mathbf{U}_p\tilde{\mathbf{x}}). \tag{8}$$

The problem is now one of locating a compact set of cubature points $\mathbf{x}_p$ and computing their weights $w_p$. We will address this in §4. Intuitively, $\mathbf{U}_p^T$ spreads the action of $\mathcal{F}$ from a single point to the entire domain. Other points that would have experienced similar actions inherit the results, so computing them becomes redundant.

**Discussion:** Unlike the projected tensor approach, the cubature approach does not assume that $\mathcal{F}$ can be written as a tensor, but instead treats it as a black box to be point sampled. Semi-Lagrangian schemes fit this criteria, as they can perform the backtrace and interpolation for a single cell. Potentially, a large number of cubature points may be needed to approximate $\widetilde{\mathbf{f}}$ with reasonable accuracy. Trivially, if all of the grid cells are included in the cubature set, with weights equal to one, $\widetilde{\mathbf{f}}$ will be computed perfectly, but in $O(N)$ time. In the case of solid and shell mechanics, several works [An et al. 2008; Chadwick et al. 2009; Kim and James 2009] have found that that $P \propto r$ in practice. We will show that similar efficiencies arise in fluid mechanics.

### 3.3 Our Cubature-Based Solver

With cubature preliminaries in place, we now describe our solver. As is standard in graphical fluid simulation [Stam 1999], we split the integration into stages, with intermediate states of $\mathbf{u}$ denoted with a numbered superscript $\mathbf{u}^i$, and the beginning and final values denoted $\mathbf{u}^t$ and $\mathbf{u}^{t+1}$. The standard integration scheme for an Eulerian grid of $N$ cells (e.g. [Stam 1999]) can be written as six operations. In reading order:

$$\begin{aligned} \mathbf{u}^0 &= \mathbf{u}^t + \Delta t\, \mathbf{f}_e & \mathbf{u}^1 &= \mathcal{A}(\mathbf{u}^0) \\ \mathbf{u}^2 &= \mathbf{V}\mathbf{u}^1 & \mathbf{d} &= \mathbf{W}\mathbf{u}^2 \\ \mathbf{p} &= \mathbf{X}^{-1}\mathbf{d} & \mathbf{u}^{t+1} &= \mathbf{u}^2 + \mathbf{Y}\mathbf{p}, \end{aligned}$$

where $\mathcal{A}(\cdot)$ is an arbitrary advection scheme, $\mathbf{V} \in \mathbb{R}^{3N\times 3N}$ is a diffusion matrix, $\mathbf{W} \in \mathbb{R}^{N\times 3N}$ is a velocity-to-divergence conversion matrix, $\mathbf{X} \in \mathbb{R}^{N\times N}$ is the Poisson matrix, $\mathbf{p} \in \mathbb{R}^N$ is the pressure field, and $\mathbf{Y}$ is a pressure-to-velocity conversion matrix.

**Subspace diffusion and pressure:** Putting aside the force and advection stages for the moment, we can show that the last four stages of the integration can be performed very efficiently in reduced coordinates. Assume we have an orthonormal velocity basis, $\mathbf{U} \in \mathbb{R}^{3N\times r}$, and a pressure-divergence basis, $\mathbf{P} \in \mathbb{R}^{N\times r}$, both obtained by taking the SVD of a pre-existing simulation. The last four stages can then be projected as follows:

$$\begin{aligned} &\mathbf{U}^T\mathbf{u}^2 = \left(\mathbf{U}^T\mathbf{V}\mathbf{U}\right)\mathbf{U}^T\mathbf{u}^1 \\ &\mathbf{P}^T\mathbf{d} = \left(\mathbf{P}^T\mathbf{W}\mathbf{U}\right)\mathbf{U}^T\mathbf{u}^2 \\ &\mathbf{P}^T\mathbf{p} = \left(\mathbf{P}^T\mathbf{X}\mathbf{P}\right)^{-1}\mathbf{P}^T\mathbf{d} \\ &\mathbf{U}^T\mathbf{u}^{t+1} = \mathbf{U}^T\mathbf{u}^2 + \left(\mathbf{U}^T\mathbf{Y}\mathbf{P}\right)\mathbf{P}^T\mathbf{p} \end{aligned} \quad \Rightarrow \quad \begin{aligned} \widetilde{\mathbf{u}}^2 &= \widetilde{\mathbf{V}}\widetilde{\mathbf{u}}^1 \\ \widetilde{\mathbf{d}} &= \widetilde{\mathbf{W}}\widetilde{\mathbf{u}}^2 \\ \widetilde{\mathbf{p}} &= \widetilde{\mathbf{X}}^{-1}\widetilde{\mathbf{d}} \\ \widetilde{\mathbf{u}}^{t+1} &= \widetilde{\mathbf{u}}^2 + \widetilde{\mathbf{Y}}\widetilde{\mathbf{p}} \end{aligned}.$$

Note that the matrix inverse and the projection have been interchanged the row of equations above. This swap is discussed in detail in [Stanton et al 2013]. Further details on the implementation and computation of the necessary SVDs are available in §3.4. All of these operations are linear, and the projections have made the matrices very small, so it becomes practical to combine the diffusion and pressure stages into *a single matrix-vector multiply*:

$$\widetilde{\mathbf{u}}^{t+1} = \begin{bmatrix}\mathbf{I} & \widetilde{\mathbf{Y}}\widetilde{\mathbf{X}}^{-1}\widetilde{\mathbf{W}}\end{bmatrix}\begin{bmatrix}\mathbf{I}\\ \mathbf{I}\end{bmatrix}\widetilde{\mathbf{V}}\widetilde{\mathbf{u}}^1 = \widetilde{\mathbf{Z}}\widetilde{\mathbf{u}}^1, \tag{9}$$

where $\mathbf{I}$ denotes an $r \times r$ identity matrix. Directly computing the inverse of a Poisson matrix is often discouraged because the sparse matrix becomes dense, and a large condition number can introduce numerical error. In our case, both the matrix and its inverse are dense, so space is not a consideration, and the projection by $\mathbf{U}$ clusters the eigenvalues sufficiently that we found the results of the direct inverse and a matrix solve to match to within working precision. Thus, as with other subspace methods [Treuille et al. 2006], no matrix inversion is necessary at runtime.

**Subspace Advection:** Once a cubature scheme has been precomputed, the advection stage can be efficiently computed in reduced coordinates. We will describe the runtime here, and the precomputation in §4. Assume we have a function $\mathcal{A}(\mathbf{u}_p)$ that computes advection at a grid cell $p$, containing velocity $\mathbf{u}_p$. We can compute the reduced quantity $\widetilde{\mathbf{u}}^1$ as:

$$\widetilde{\mathbf{u}}^1 = \sum_{p=1}^{P} w_p \mathbf{U}_p^T \mathcal{A}(\mathbf{U}_p\widetilde{\mathbf{u}}^0). \tag{10}$$

As $\mathcal{A}$ is generic, it can correspond to any scheme that supports point sampling, and we have successfully used it to compute both semi-Lagrangian and MacCormack advection (Figs. 6 and 7). These schemes are already essentially point-based, so rewriting existing code to support cubature took minimal effort.

**Subspace Forces:** The cubature approach can be used to compute other non-linearities. The external force term $\mathbf{f}_e$ term can contain arbitrary forces, but one popular term is vorticity confinement, $\mathbf{f}_{\texttt{conf}}$ [Fedkiw et al. 2001]. This term is non-linear due to the square root used to normalize the vorticity location vectors $\eta$. We found it straightforward to capture using a cubature scheme, $\widetilde{\mathbf{f}}_{\texttt{conf}} = \sum_{p=1}^{P} w_p \mathbf{U}_p^T \mathcal{V}(\mathbf{U}_p\widetilde{\mathbf{u}}^t)$, where $\mathcal{V}(\cdot)$ is the point sampled vorticity confinement function. While we found it possible to efficiently compute these schemes using importance sampling, we did not use them in our final results. Instead, the entire $\mathbf{f}_e$ term was computed in full coordinates in order to maximize the novelty of forces used to perturb the subspace simulation.

### 3.4 Computing Large SVDs

**Multiple SVDs:** One important detail was temporarily glossed over in §3.3 for the sake of exposition. In order to capture the full dynamics of an existing simulation, it is insufficient for $\mathbf{U}$ to only support the velocity states at the beginning and end of a timestep. If an intermediate value, e.g. $\mathbf{u}$ post-advection but pre-diffusion, is not within basis, error will accumulate quickly. However, concatenating all of these intermediate states into one monolithic $\mathbf{S}$ would drastically increase the SVD computation time and runtime memory requirements. We instead construct four separate $\mathbf{U}$ bases, $\mathbf{U}_0$, $\mathbf{U}_1$, $\mathbf{U}_2$ and $\mathbf{U}_3$, respectively for the pre-advection, pre-diffusion, pre-projection, and final velocity states. This considerably reduces precomputation time, as an SVD solve takes $O(mn^2)$ time for an $m \times n$ matrix, and we are partitioning one large $n$ solve into four smaller $\frac{n}{4}$ solves. These multiple SVDs make this approach slower than previous approaches [Treuille et al. 2006], but only by a constant factor (four). The resulting bases are used to precompute the matrices,

$$\mathbf{U}_2^T\mathbf{V}\mathbf{U}_1 = \widetilde{\mathbf{V}} \quad \mathbf{P}^T\mathbf{W}\mathbf{U}_2 = \widetilde{\mathbf{W}} \quad \mathbf{U}_3^T\mathbf{Y}\mathbf{P} = \widetilde{\mathbf{Y}}$$

but at runtime, $\mathbf{U}_1$, $\mathbf{U}_2$, and $\mathbf{P}$ do not need to be in-core. Only the bases for the initial and final states, $\mathbf{U}_0$ and $\mathbf{U}_3$ are needed. $\mathbf{U}_0$ is the rightmost $\mathbf{U}_p$ in Eqn. 10, and $\mathbf{U}_1$ is the leftmost $\mathbf{U}_p^T$. The $w_p\mathbf{U}_P^T$ products are pre-cached to avoid needing $\mathbf{U}_1$ at runtime.

**Out-of-Core SVD:** The matrix of simulation snapshots $\mathbf{S}$ is often too large to fit into memory. We address this by performing an out-of-core SVD similar to that of James and Fatahalian

[2003], but we favor accuracy over their output-sensitivity. We first perform an out-of-core QR factorization using modified Gram-Schmidt, $\mathbf{S} = \mathbf{QR}$, and then an SVD $\mathbf{R} = \mathbf{U_R}\Sigma\mathbf{V}^T$, which is very small and fits in-core. The final $\mathbf{U}$ is computed using an out-of-core multiply $\mathbf{QU_R}$, but as singular values $\Sigma$ are available, we skip building columns below a discard threshold. The running time splits evenly between computation ($\mathbf{QR}$ and $\mathbf{QU_R}$ multiplies) and disk I/O. While other out-of-core methods are available, e.g. QR using block Householder transforms [Rabani and Toledo 2001], we expect they would only accelerate computation by a constant.

### 3.5 Dirichlet and Neumann Obstacles

A practical fluid simulation method needs a method of handling static and moving obstacles. In the case of subspace simulation, the method must be selected carefully to avoid increasing the precomputation and runtime complexities. The standard method is to clamp velocities on the interior of an obstacle, and to remove these cells from the Poisson solve. This removes rows from the $\mathbf{V}$, $\mathbf{W}$, $\mathbf{X}$, and $\mathbf{Y}$ matrices of our Eulerian solve, and clips semi-Lagrangian rays to obstacle surfaces.

This is an unattractive option for moving obstacles, because it means that a separate $\widetilde{\mathbf{V}}$, $\widetilde{\mathbf{W}}$, $\widetilde{\mathbf{X}}$, $\widetilde{\mathbf{Y}}$, and cubature scheme must be computed for *every frame* in the simulation. We found that this problem can be sidestepped using the Iterated Orthogonal Projection (IOP) approach of Molemaker et al. [2008].

**The IOP approach:** Instead of removing cells from the solve, the IOP approach uses an affine transform to clamp cell velocities prior to the Poisson solve. We first demonstrate this for Dirichlet boundaries. Eqn. 9 in full, unreduced coordinates is:

$$\mathbf{u}^{t+1} = \begin{bmatrix}\mathbf{I} & \mathbf{Y}\mathbf{X}^{-1}\mathbf{W}\end{bmatrix}\begin{bmatrix}\mathbf{I}\\ \mathbf{I}\end{bmatrix}\mathbf{V}\mathbf{u}^1,$$

where $\mathbf{I} \in \mathbb{R}^{3N\times 3N}$ is an identity matrix. Similar to the reduced case, we abbreviate this as $\mathbf{u}^{t+1} = \mathbf{Z}\mathbf{u}^1$. An $\mathbf{I}$ can be trivially inserted: $\mathbf{u}^{t+1} = \mathbf{Z}\mathbf{I}\mathbf{u}^1$. To impose Dirichlet boundaries, IOP zeros out the diagonal entries in $\mathbf{I}$ that correspond to cells on the interior of obstacles. We denote this zeroed-out $\mathbf{I}$ as $\mathbf{D}$, to obtain:

$$\mathbf{u}^{t+1} = \mathbf{Z}\mathbf{D}\mathbf{u}^1. \tag{11}$$

After the pressure solve, the Dirichlet boundaries could still be violated, so the process is repeated: $\mathbf{u}^{t+1} = (\mathbf{ZD})^2\,\mathbf{u}^1$. In the limit, $\mathbf{u}^{t+1} = (\mathbf{ZD})^\infty\,\mathbf{u}^1$, the boundaries are guaranteed to be satisfied. In practice, only a few iterations are needed, and the authors [Molemaker et al. 2008] report that a single iteration is sufficient for production work. Despite the repeated projections, we observed minimal numerical smearing in the final result. As mentioned in Molemaker et al. [2008], $(\mathbf{ZD})^\infty$ corresponds to a PCG solve that directly incorporates boundaries, not multiple pressure timesteps.

Neumann boundaries are achieved by appending a homogeneous column $\mathbf{n}$ to $\mathbf{D}$. When a diagonal entry is zeroed, the corresponding row in $\mathbf{n}$ is set to the obstacle velocity. A homogenous coordinate is then added to $\mathbf{u}^1$ to obtain the Neumann version of Eqn. 11,

$$\mathbf{u}^{t+1} = \mathbf{Z}\begin{bmatrix}\mathbf{D} & \mathbf{n}\end{bmatrix}\begin{bmatrix}\mathbf{u}^1\\ 1\end{bmatrix} = \mathbf{Z}\mathbf{N}\begin{bmatrix}\mathbf{u}^1\\ 1\end{bmatrix}.$$

where $\mathbf{N}$ denotes our Neumann boundary matrix. This process can again be repeated until a desired boundary accuracy is met.

**Subspace IOP:** Molemaker et al. [2008] note that $\mathbf{D}$ and $\mathbf{N}$ do not need to be explicitly constructed; it suffices to iterate over the grid cells and clamp their values. For our appliation, we *are* interested in constructing $\mathbf{D}$ and $\mathbf{N}$, as they provide an elegant way of incorporating boundary conditions into a subspace simulation.

We show the Dirichlet case first. If we project $\mathbf{D}$ to obtain $\widetilde{\mathbf{D}} = \mathbf{U}^T\mathbf{D}\mathbf{U}$, Dirichlet boundaries can be added to Eqn. 9 by inserting a single matrix multiply:

$$\widetilde{\mathbf{u}}^{t+1} = \widetilde{\mathbf{Z}}\widetilde{\mathbf{D}}\widetilde{\mathbf{u}}^1. \tag{12}$$

As with full-coordinate IOP, if additional boundary accuracy is needed, the process can be repeated: $\widetilde{\mathbf{u}}^{t+1} = (\widetilde{\mathbf{Z}}\widetilde{\mathbf{D}})^2\widetilde{\mathbf{u}}^1$. The Neumann case adds a homogeneous column to $\mathbf{U}$:

$$\widetilde{\mathbf{N}} = \mathbf{U}^T\mathbf{N}\begin{bmatrix}\mathbf{U} & \mathbf{z}_{3N}\\ \mathbf{z}_r^T & 1\end{bmatrix}, \tag{13}$$

where $\mathbf{z}_{3N} \in \mathbb{R}^{3N}$ and $\mathbf{z}_r \in \mathbb{R}^r$ are zero vectors. The subspace Neumann solve is then:

$$\widetilde{\mathbf{u}}^{t+1} = \widetilde{\mathbf{Z}}\widetilde{\mathbf{N}}\begin{bmatrix}\widetilde{\mathbf{u}}^1\\ 1\end{bmatrix}.$$

These boundaries were imposed without modifying $\mathbf{Z}$ or $\widetilde{\mathbf{Z}}$, so the problem of needing a per-timestep $\widetilde{\mathbf{V}}$, $\widetilde{\mathbf{W}}$, $\widetilde{\mathbf{X}}$, or $\widetilde{\mathbf{Y}}$ has been removed. The IOP authors also observe that their method allows obstacle boundaries to be ignored during advection, so the need for per-frame cubature has also been avoided.

**Efficient IOP matrix construction:** Per-frame $\widetilde{\mathbf{Z}}$s do not need to be precomputed, but we need to compute per-frame $\widetilde{\mathbf{N}}$s. The direct method is the projection $\widetilde{\mathbf{D}} = \mathbf{U}^T\mathbf{D}\mathbf{U}$, but we have found that it is more efficient to instead compute $\widetilde{\mathbf{D}} = \mathbf{U}^T\mathbf{U} - \mathbf{U}^T\mathbf{D}_c\mathbf{U}$. Here, $\mathbf{D}_c$ is a *complement* matrix that flips all the zero diagonal entries in $\mathbf{D}$ to one, and vice versa. Most scenes contain relatively few obstacle cells, so $\mathbf{D}_c$ is extremely sparse, and projects quickly. $\widetilde{\mathbf{N}}$ is then constructed by appending the additional column, $\widetilde{\mathbf{n}} = \mathbf{U}^T\mathbf{n}$.

**Discussion:** While we have found that our subspace IOP method is effective for re-simulation, it has limitations. We assume that the motion of obstacles is similar or identical to those in the original simulation. If the internal obstacles undergo very novel motions, the obstacle handling method of Treuille et al. [2006] is likely to yield superior results. Our approach also inherits the limitation of IOP that slight density leaks and pass-throughs are possible. However, like in Molemaker et al. [2008], we did not observe any significant artifacts in practice.

## 4 Cubature Precomputation

We have described how to use a cubature scheme in §3.3, but we have not described how to precompute the scheme. An et al. [2008] use a greedy search algorithm, but we have found that it does not converge in a practical amount of time for subspace fluid simulations. In this section, we present an asymptotically faster method, which we motivate by first describing the greedy algorithm.

### 4.1 The Greedy Search Algorithm

In the greedy search algorithm [An et al. 2008], a set of cubature points are computed that fit a training set to a desired error tolerance. As is assumed by other subspace fluid methods [Treuille et al. 2006; Wicke et al. 2009], a training set of $T$ simulation "snapshots" is available. Assume we have a set of $P$ promising cubature points $\mathbf{x}_{1\ldots P}$. The weights $w_{1\ldots P}$ can be computed by solving a

least-squares problem,

$$\begin{bmatrix} \widetilde{\mathbf{f}}_1^1 & \cdots & \widetilde{\mathbf{f}}_p^1 & \cdots & \widetilde{\mathbf{f}}_P^1 \\ \vdots & \ddots & \vdots & & \vdots \\ \widetilde{\mathbf{f}}_1^t & \cdots & \widetilde{\mathbf{f}}_p^t & \cdots & \widetilde{\mathbf{f}}_P^t \\ \vdots & & \vdots & \ddots & \vdots \\ \widetilde{\mathbf{f}}_1^T & \cdots & \widetilde{\mathbf{f}}_p^T & \cdots & \widetilde{\mathbf{f}}_P^T \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_P \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{f}}^1 \\ \vdots \\ \widetilde{\mathbf{f}}^t \\ \vdots \\ \widetilde{\mathbf{f}}^T \end{bmatrix}. \quad (14)$$

The superscipt indexes the $t \in [1 \dots T]$ snapshots, and the subscript $p \in [1 \dots P]$ the cubature points. On the right, each $t$th row is a projected full-rank example computed using Eqn. 5. These are the projections of the example velocity fields immediately post-advection. The left is a matrix of point samples computed using Eqn. 7, corresponding to the advection of a handful of cells.

This least squares problem yields a set of cubature weights that closely approximate the examples. We abbreviate the above system as $\mathbf{Aw} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{rT \times P}$, $\mathbf{w} \in \mathbb{R}^P$ and $\mathbf{b} \in \mathbb{R}^{rT}$. If the fit given by Eqn. 14 does not meet a user-specified error bound $\epsilon$ (i.e. if $\|\mathbf{r}\|_2 = \|\mathbf{b} - \mathbf{Aw}\|_2 > \epsilon$) then a new cubature point must be added. A new column is then added to $\mathbf{A}$, a new entry is added to $\mathbf{w}$, and the least squares fit is run again. The new point, $\mathbf{x}_{P+1}$, is selected *greedily*. Every candidate point $\mathbf{x}_p$ on the grid that is not yet in the cubature set can generate a column $\mathbf{a}_p$ to be appended to $\mathbf{A}$. The candidate point whose column has the largest projection onto the residual, $g = \mathbf{a}_p \cdot \mathbf{r}$, has the most promise for reducing the overall error, and is greedily added to the cubature set.

Computing $\mathbf{a}_c$ for all the non-cubature points would be computationally prohibitive, so the candidates are randomly sampled, and the one with the largest projection is selected. An et al. [2008] perform a non-negative least squares solve (NNLS) in lieu of the usual least squares (LS) problem, as negative weights ruin the positive definiteness of a finite element stiffness matrix. We do not use stiffness matrices, but we also invoke an NNLS solver. A negative weight would flip the direction of a semi-Lagrangian backtrace, which disagrees with physical intuition. Tests with an LS solver produced large, oscillatory weights, which further supported our intuition.

**Algorithm Complexity:** The greedy search algorithm runs in $O(rTP^4)$ time. An et al. [2008] used the standard Lawson-Hanson [1974] NNLS algorithm, which runs a series of LS solves that each take $O(mn^2)$ time for an $m \times n$ matrix [Golub and Van Loan 1996]. The LS problems grow in size from 1 to $P$ in the inner loop of the Lawson-Hanson algorithm, so a single NNLS solve for $\mathbf{A} \in \mathbb{R}^{rT \times P}$ takes $\sum_{p=1}^{P} rTp^2 = O(rTP^3)$ time. The greedy algorithm calls the NNLS solver $P$ times, once for each new cubature point, yielding a running time of $\sum_{p=1}^{P} rTp^3 = O(rTP^4)$. Note we used the identities $\sum_{p=1}^{P} p^2 = O(P^3)$ and $\sum_{p=1}^{P} p^3 = O(P^4)$. This quartic complexity matches our experiments, as we found that the algorithm did not converge to $1\%$ error, even for modestly sized fluids $(48 \times 96 \times 48)$, even given almost *six days* to compute.

**Other NNLS solvers:** One possibility is to use a more efficient NNLS solver, as alternative algorithms are an area of active research [Chen and Plemmons 2007]. We compared against gradient projection [Kim et al. 2012] and block pivoting [Portugal et al. 1994] methods, but they did not yield sufficient accuracy in a competitive amount of time. These methods assume the matrix is sparse and low-rank, whereas ours is dense and full-rank. Parallel NNLS algorithms [Luo and Duraiswami 2011] parallelize across multiple right hand $\mathbf{b}$ terms and thus do not apply to our problem. The Bro and de Jong [1997] modification of the Lawson-Hanson algorithm, also known as Fast Non-Negative Least Squares (FNNLS), yielded a significant constant speedup. While it did not solve the complexity problem, we used it to accelerate all our computations.

**Discarded Alternatives:** The existing method is a *greedy* algorithm, so it is tempting to investigate other standard algorithm types, such as dynamic programming and divide-and-conquer [Kleinberg and Tardos 2006]. In the case of dynamic programming, it is not clear that the necessary optimal sub-problem property is satisfied by our problem. Even if it is, dynamic programming also constructs a solution incrementally, so it is likely to yield another $O(rTP^4)$ algorithm. Divide-and-conquer is more promising, as the cubature points discovered by smaller, independent runs of the greedy algorithm can be combined into a higher quality set of points. Unfortunately, this approach was still too slow for our application.

### 4.2 An Importance Sampling Approach

We instead design an asymptotically faster algorithm that runs in $O(rTP^3)$ time. This algorithm is motivated by three key observations. First, much of the work of the greedy algorithm is redundant, as many $\mathbf{A}$ matrices are solved that only differ by a column. This suggests that many candidates should be added to the cubature set $\mathbf{c}$ at once to amortize the cost of fitting $\mathbf{A}$ (Algorithm 1). Algorithm 1 was only observed to be faster than the greedy algorithm by a constant factor. However, analyzing its behavior led us to our second key observation: when many redundant points are present in $\mathbf{c}$, the Lawson-Hanson NNLS solver does not evenly distribute weight among them, but instead allocates all of the weight to a single point, and clamps the rest to zero. Therefore, Line 8 in Algorithm 1 will cull $\mathbf{c}$ extremely efficiently.

**Algorithm 1**: largeSamplingCubature($C$, $\varepsilon$)

**Data**: $C$ is a user-specific number of cubature candidates to add during each pass, and $\varepsilon$ is the requested cubature accuracy.

```
1 begin
2     r = b
3     The set of cubature points c = ∅
4     while ||r||_2 > ε do
5         Add C randomly selected cubature candidates to c
6         Build Aw = b from the current c
7         Solve for non-negative w, update r = b − Aw
8         Cull points in c whose weights are zero
9 end
```

A third observation leads to our efficient algorithm: the uniform sampling in Algorithm 1 resembles Monte Carlo integration. Details differ, but the goal of estimating the integrand using a minimal number of samples (in our case, cubature points), remains the same. We should therefore leverage the intuition behind importance sampling: samples drawn from a non-uniform distribution that are similar to the integrand will rapidly reduce variance, or in our case, fitting error (Pharr and Humphreys [2010] is a good reference). Informally, we should cluster samples in regions that are likely to yield non-zero weights.

We define our importance probability distribution function at each discrete fluid cell $\mathbf{x}_p$ as:

$$\mathrm{PDF}(\mathbf{x}_p) = R \left( \frac{|\mathbf{a}_p \cdot \mathbf{r}|}{\mathbf{r} \cdot \mathbf{r}} \right), \quad (15)$$

where $\mathbf{a}_p$ is the candidate column of $\mathbf{A}$ generated by $\mathbf{x}_p$, $\mathbf{r}$ is the current NNLS residual, and $R$ is the number of points that are not yet in the cubature set. The numerator is the greedy selection criteria, $g = \mathbf{a}_p \cdot \mathbf{r}$, and the denominator is a normalization constant, $\frac{\mathbf{r} \cdot \mathbf{r}}{R}$

| | Stam Plume | | MacCormack Plume | | Dirichlet (MacCormack) | | Neumann (Semi-Lagrangian) | |
|---|---|---|---|---|---|---|---|---|
| Iteration | **$L_2$ Error** | **Cub. Points** | **$L_2$ Error** | **Cub. Points** | **$L_2$ Error** | **Cub. Points** | **$L_2$ Error** | **Cub. Points** |
| 1 | 0.0428803 | 1273 | 0.0592719 | 1479 | 0.0409149 | 2033 | 0.0330917 | 2900 |
| 2 | 0.0148716 | 2311 | 0.0184463 | 2816 | 0.0145316 | 3416 | 0.0118481 | 4698 |
| 3 | 0.0107379 | 2528 | 0.0112989 | 3392 | 0.0106133 | 4149 | 0.00650847 | 5610 |
| 4 | 0.00866083 | 2747 | 0.00865156 | 3686 | 0.00871744 | 4808 | converged | converged |
| | **Total time:** 01h 18m 07s | | **Total time:** 03h 05m 58s | | **Total time:** 09h 28m 29s | | **Total time:** 05h 29m 02s | |

**Table 1:** *Performance of our importance sampling-based cubature training algorithm. The iteration count refers to the outer* **while** *loop in Algorithm 1.* **$L_2$ error** *is the relative fitting error after each iteration, and* **Cub. Points** *is the number of non-zero cubature points found. In general, MacCormack training took longer, as the function is more non-linear.*

which scales the PDF closer to the $[0, 1]$ range. This clusters sample points in regions with large projections onto the residual, and works very well in practice. We rejection sample this distribution by replacing Line 5 of Algorithm 1 with a call to Algorithm 2.

**Algorithm 2**: importanceSampledCubature($C$)

1 **while** *$C$ points have not been added to* **c** **do**
2  Randomly select a candidate $\mathbf{x}_p$ not in **c**
3  Add $\mathbf{x}_p$ to **c** with probability $\mathrm{PDF}(\mathbf{x}_p)$

This combined algorithm computes a cubature set to within error $\varepsilon = 1\%$ extremely quickly. The greedy search that took nearly six days (141h 44m 59s) in §4.1 computed in less than half an hour (29m 55s). The fitting errors after each iteration of the outer **while** loop in Algorithm 1 are shown in Table 1. As a constant number of iterations appear to be needed to converge to a desired error bound, the main expense is the NNLS call, so we characterize the average case running time of Algorithm 2 as $O(rTP^3)$.

**Complexity of projected tensor approach:** The complexity of our precomputation compares favorably with the projected tensor approach [Treuille et al. 2006]. The main bottleneck of that approach is the projection of $\mathbf{F} \in \mathbb{R}^{3N\times 3N\times 3N}$. This tensor is very sparse, containing only $9N$ non-zero entries. This sparsity can only exploited for two projections, i.e. $\mathbf{F} \times_1 \mathbf{U} \times_3 \mathbf{U} = \widetilde{\mathbf{F}}_{13}$ can be performed in $O(Nr^2)$ time, but the third projection, i.e. $\widetilde{\mathbf{F}}_{13} \times_2 \mathbf{U}$, takes $O(Nr^3)$ time. By comparison, the $O(rTP^3)$ complexity of our importance sampling method does not depend on $N$. The $O(N)$ advection examples only need to be projected once to form the right hand side of Eqn. 14 at the very beginning of training.

## 5 Implementation and Results

All our fluid simulation data were generated using a Preconditioned Conjugate Gradient (PCG) solver with a Modified Incomplete Cholesky preconditioner [Fedkiw et al. 2001]. Faster algorithms such as multigrid [Molemaker et al. 2008] or highly tuned FFT implementations [Henderson 2012] are also available, but since our subspace solver is an unoptimized prototype and PCG is standard, we selected it for our comparisons. Our code, including the PCG solver, was implemented in C++, and our tests were run on a 12 core, 2.66 Ghz Mac Pro with 96 GB of RAM. We used a collocated velocity grid for all of our simulations, but it would be straightforward to adapt the code to a staggered grid. All of our code was parallelized where appropriate using OpenMP, including cubature evaluation and velocity reconstruction. We used Eigen [Guennebaud et al. 2010] for dense linear algebra. Eigen is a highly tuned library, but its matrix-vector multiplies are currently not multi-threaded, so more fine-grained optimizations are still available for the velocity reconstruction stage. Timings for the various stages in the precomputation are listed in Table 2.

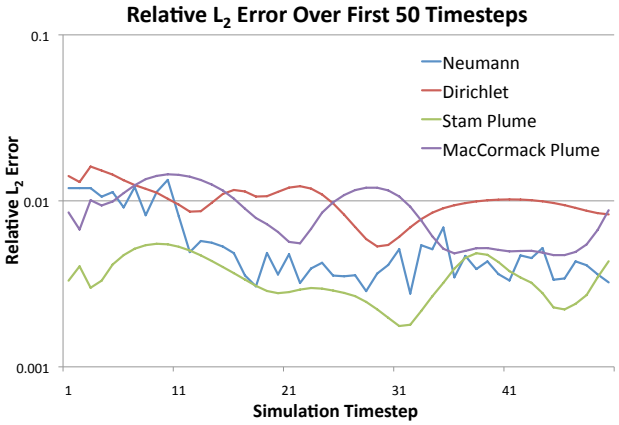


**Figure 3:** *Error of subspace simulations over the first 50 timesteps of each simulation. Interestingly, integration error does* not *appear to accumulate. Instead, it stays proportional to the 1% error of the advection cubature. Note the logarithmic scale.*

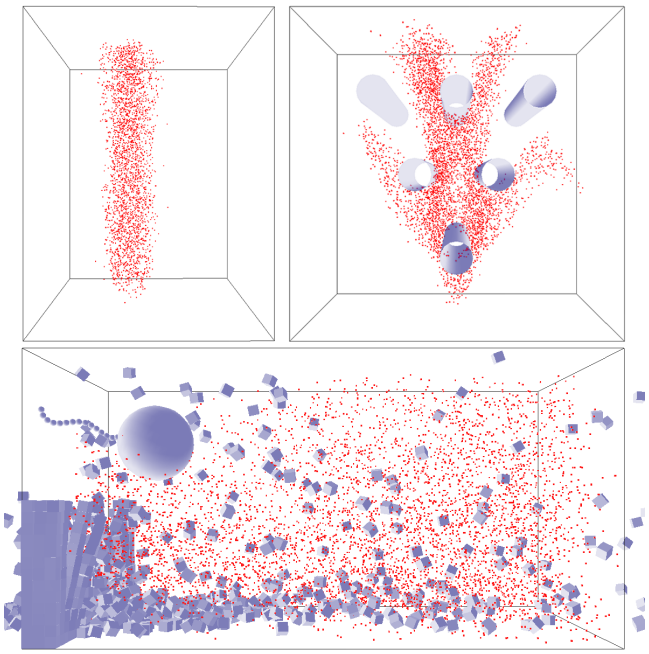

**Figure 4:** *Cubature points (in red) computed by our importance sampling method. Clockwise from upper left, the points correspond to Figs. 6, 8, and 1. The points clearly cluster in regions where interesting dynamics occur.*

| | Stam Plume | MacCormack | Dirichlet | Neumann |
|---|---|---|---|---|
| Resolution | $200 \times 266 \times 200$ | $200 \times 266 \times 200$ | $276 \times 276 \times 138$ | $175 \times 175 \times 350$ |
| Full solve time per frame | 00h 02m 47s | 00h 02m 49s | 00h 01m 26s | 00h 01m 23s |
| Full solve time total | 06h 57m 30s | 07h 02m 30s | 03h 35m 00s | 03h 27m 30s |
| Subspace solve time per frame, | | | | |
| with $\mathbf{f}_e$ and VR | 4.2s **(39x faster)** | 5.6s **(30x faster)** | 5.1s **(17x faster)** | 5.7s **(14x faster)** |
| without $\mathbf{f}_e$ and VR | 18ms **(9326x faster)** | 96ms **(1764x faster)** | 130ms **(661x faster)** | 34ms **(2435x faster)** |
| Precomputation times, | | | | |
| Multiple SVDs (§3.4) | 04h 07m 36s | 03h 14m 11s | 04h 49m 38s | 04h 18m 09s |
| Cubature construction (§4.2) | 01h 18m 07s | 03h 05m 58s | 09h 28m 09s | 05h 29m 02s |
| $\widetilde{\mathbf{V}}$, $\widetilde{\mathbf{W}}$, $\widetilde{\mathbf{X}}$, and $\widetilde{\mathbf{Y}}$ projection (§3.3) | 04h 24m 40s | 03h 04m 59s | 04h 30m 59s | 03h 54m 52s |
| $\widetilde{\mathbf{D}}$ or $\widetilde{\mathbf{N}}$ projection (§3.5) | N/A | N/A | 00h 12m 12s | 05h 11m 52s |

**Table 2: Top:** *Performance of our subspace solver compared to full-rank solves. To facilitate comparisons to previous works [Treuille et al. 2006; De Witt et al. 2012] we present timings both with and without the full-rank external force ($\mathbf{f}_e$) and velocity reconstruction (VR) steps. Sparsely reconstructing the velocity field like the previous works would yield these superior performances. All reported times are for rank $r = 150$, and all simulations were run for 150 frames. Lower accuracy $\mathbf{U}$ bases would yield additional speedups.* **Bottom:** *Running times of the steps in the precomputation stage.* Multiple SVDs *is for the construction of* all *the $\mathbf{U}$ and $\mathbf{P}$ bases. Note that the SVDs and matrix projections are highly task-parallel and could be run on different cluster nodes, but single system construction times are reported.*

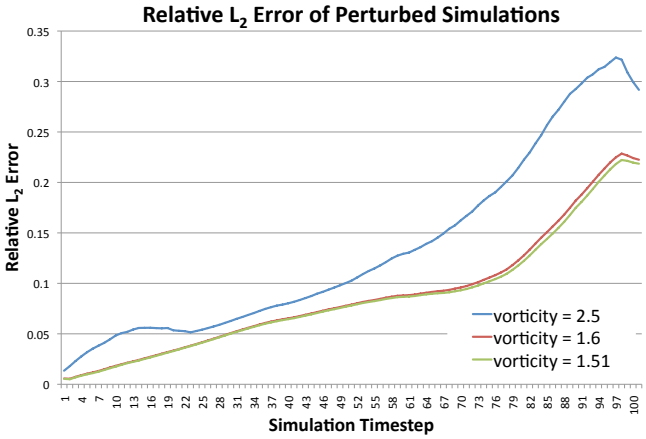


**Figure 5:** *Comparison of a perturbed subspace and full-rank ground truth simulations. The original training simulation had vorticity confinement $\epsilon = 1.5$. We perturbed it by successive orders of magnitude to $\epsilon = 1.51, 1.6$ and $2.5$. As expected, as the perturbation grows, the $L_2$ error relative to the ground truth does as well.*

All of our examples were trained to within 1% relative $L_2$ cubature error and converged after a small number of iterations (Table 1). The cubature schemes that were computed can be seen in Figure 4. To test for consistency, we compared the results of our subspace simulation to the original ground truth data (Figure 3). The error was proportional to the advection cubature approximation error, and despite the history-aware nature of the simulation, did not accumulate over time. When an $O(N)$ full-rank advection scheme was swapped in, the error dropped to within single precision. We also tested how well our subspace model generalizes in Fig. 5. As expected, $L_2$ error relative to a ground truth solution increases as the dynamics diverge from those of the training simulation. Characterizing the error of subspace simulations under arbitrary perturbations is an interesting and subtle topic [Homescu et al. 2005], so we leave a more thorough analysis as future work.

**Velocity Reconstruction:** For our subspace re-simulations, we placed the external force term $\mathbf{f}_e$ (including vorticity confinement and buoyancy) and density advection outside of the subspace, as they provide the most straightforward avenues for stimulating novel dynamics. This requires a dense $O(N)$ velocity reconstruction stage to be introduced, i.e. $\mathbf{u} = \mathbf{U}\tilde{\mathbf{u}}$, which is known to be a bottleneck in reduced order simulations [De Witt et al. 2012]. Our method also supports output-sensitive sparse velocity reconstructions, such as the leaf example from Treuille et al. [2006] and the particle-based examples from DeWitt et al. [2012]. To facilitate comparisons to these methods, we list our solver performance both with and without a full velocity reconstruction stage in Table 2. With dense velocity reconstruction, we see an **order of magnitude speedup** compared to the original full-rank solve. Without dense reconstruction, we see the dramatic **three orders of magnitude speedup** that subspace methods have been known to achieve.

**NNLS codes:** We tested several NNLS codes, including an Eigen [Guennebaud et al. 2010] implementation of FNNLS, the original Lawson-Hanson NNLS Fortran code [Lawson and Hanson 1974], and a Matlab implementation of FNNLS by one of the original authors [Bro 2001]. The Matlab code ran by far the fastest, and solved a test matrix in 38 minutes that took that Fortran code 118 minutes and the Eigen code 214 minutes. The Fortran code was faster than the Eigen code because it caches partial factorizations across LS solves. The Matlab timing included the file I/O time needed to pass the NNLS problem and solution. Most of the running time is spent in `gemm` and LS calls, so Matlab's highly optimized, multithreaded implementations explain the superior performance.

**Stam Plume Example:** Our first test was a standard semi-Lagrangian solver [Stam 1999]. Figure 6 shows that our integrator consistently reproduces the original dynamics, and that low-accuracy $\mathbf{U}$ bases produce visual results with surprising qualitative similarity. When the basis fails (Fig. 6, right) stable dynamics are still produced, and novel, abstract dynamics emerge. The semi-Lagrangian cubature appears to have inherited the stability of its full-rank counterpart. We found that the subspace solver enables *temporal continuation*: if timesteps subsequent to those from the full-rank solver are desired, plausible new frames can be efficiently generated using our subspace solver. Results can be seen in the supplemental video.

**MacCormack Plume Example:** The cubature approach generalizes to other non-linear methods, such as MacCormack advection [Selle et al. 2008]. The cubature training time was significantly longer than the Stam Plume, which is to be expected, as the MacCormack scheme contains a higher degree of non-linearity (essentially two clamped semi-Lagrangian steps). We modified the vorticity confinement constant $\epsilon$ [Fedkiw et al. 2001] in our re-simulations, and found that more laminar flow could be retrieved (Fig. 7, lower left), and that more turbulent motion could be ef-

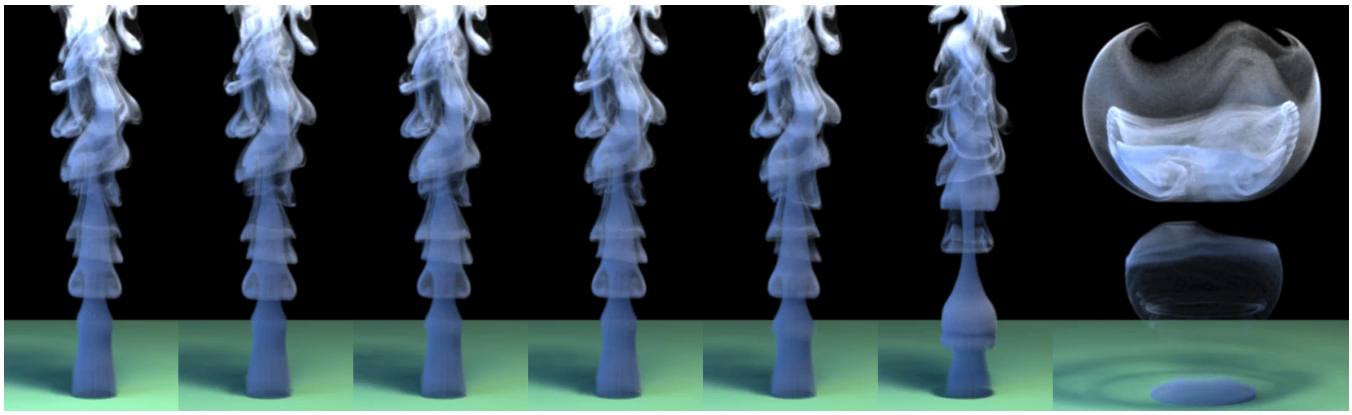

**Figure 6: Left to right:** *1. Last frame of the original semi-Lagrangian fluid simulation. 2. Last frame of our subspace simulation with SVD discard threshold* $10^{-7}$ *and rank* $r = 150$*. Note that our subspace integrator successfully reproduces the dynamics of the original simulation. 3. Discard* $10^{-6}$*,* $r = 130$*. 4. Discard* $10^{-5}$*,* $r = 105$*. 5. Discard* $10^{-4}$*,* $r = 61$*. 6. Discard* $10^{-3}$*,* $r = 28$*. 7. Discard* $10^{-2}$*,* $r = 9$*. The dynamics are surprisingly resilient to low-accuracy bases. Even when the basis fails (e.g.* $10^{-2}$*), it stably generates complex, novel dynamics.*

ficiently re-simulated (Fig. 7, lower and upper right). When the constant was set much higher, stable motion was still produced, but the dynamics became more abstract. These motions can be seen in the supplemental video.

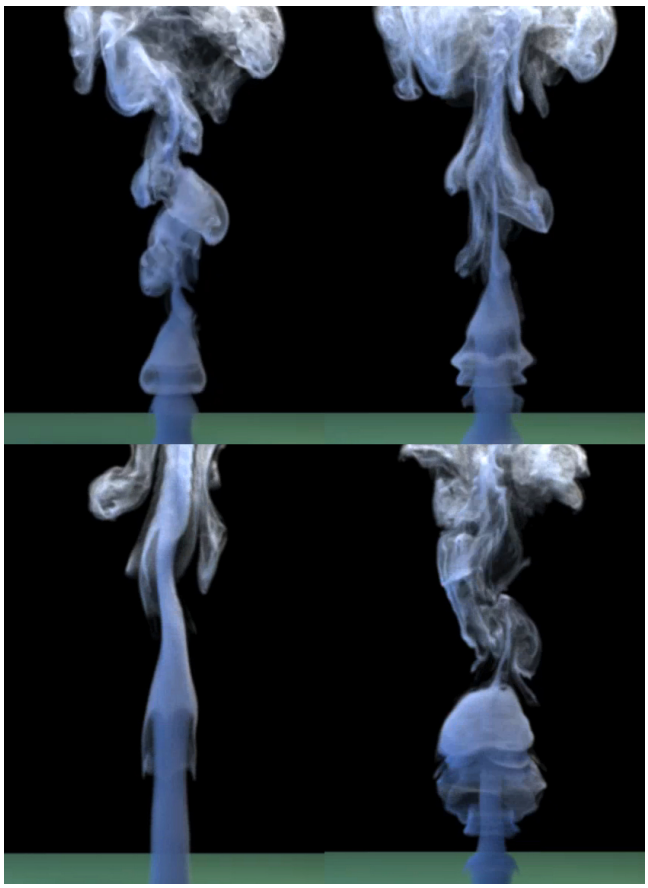

**Figure 7: Clockwise from upper left:** *1. Original MacCormack [Selle et al. 2008] simulation. 2. Novel flow efficiently generated using our subspace integrator by doubling the vorticity confinement* $\epsilon$*. 3. Novel flow generated by quadrupling the same constant. 4. Laminar flow is retrieved by setting* $\epsilon$ *to 0.*

**Dirichlet Test Example:** We tested our Dirichlet IOP approach by adding static obstacles to a plume simulation (Figure 8). The motion induced by MacCormack advection was already quite complex, so we instead set the vorticity confinement $\varepsilon$ to zero and removed turbulence from the simulation, effectively using the subspace integrator for laminar dynamics retrieval.

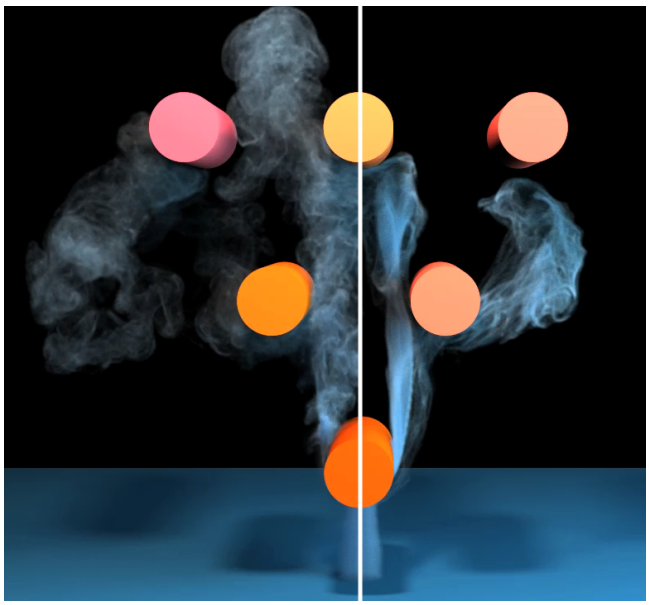

**Figure 8: Left:** *Original MacCormack simulation with Dirichlet boundaries.* **Right:** *Turbulent motion has been removed in the subspace re-simulation by setting the vorticity confinement* $\varepsilon$ *to zero.*

**Neumann Test Example:** We tested our Neumann IOP approach by introducing a one-way coupling to rigid body data from the Open Dynamics Engine (`http://www.ode.org`). For this scene, we used semi-Lagrangian advection. Buoyancy was set to zero, so all of the fluid motion is from the rigid bodies and vorticity confinement. We increased the turbulence in the re-simulation by increasing the vorticity confinement constant $\epsilon$ from 1.5 to 20. Normally this would be considered a large value for $\epsilon$, but smaller values

tended to produce less noticeable variation. We expect that this is a manifestation of the 'locking' phenomena that subspace methods are known to experience [Chadwick et al. 2009].

## 6 Discussion and Future Work

While our subspace integrator is able to efficiently re-simulate novel flows, for large parameter perturbations, the solve diverges from the results of a full-rank solve. A promising future direction is to devise enrichment methods similar to the eXtended Finite Element Method (XFEM) [Moës et al. 1999] that adds new basis functions that re-introduce the needed degrees of freedom.

We have shown that it is possible to efficiently re-simulate scenes using a subspace integrator. However, we have not attempted to address one of the known limitations of subspace methods: system memory limits the maximum rank of the velocity basis. Figure 6 suggests that even fairly drastic truncations of the basis can produce qualitatively consistent results. However, an in-depth analysis of this phenomenon is beyond the scope of the current work. Developing factorization methods that go beyond the traditional SVD [Seo et al. 2011] is a promising approach for both reducing the memory usage and accelerating the velocity reconstruction stage.

Finally, it remains to be seen if similar subspace methods can be devised for liquid simulations. In level set-based flows, the signed distance field represents a new type of non-linearity. Cubature cannot be directly applied, because algorithms such as the fast marching method [Sethian 1999] cannot be naturally point sampled. Efficient new subspace methods must be devised for this problem.

**Acknowledgements:** The authors would like to thank the reviewers for helping to improve this manuscript, particularly §3.2, Matt Wright for editing, Nils Thürey for early discussions, and Paul Weakliem for rendering support. This material is based upon work supported by a National Science Foundation CAREER award (IIS-1253948). We acknowledge rendering support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1121053), Hewlett-Packard, and NSF CNS-0960316. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

## References

AMSALLEM, D., AND FARHAT, C. 2012. Stabilization of projection-based reduced-order models. *International Journal for Numerical Methods in Engineering 91*, 4, 358–377.

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing Cubature for Efficient Integration of Subspace Deformations. *ACM Trans. on Graphics 27*, 5 (Dec.), 165.

ANTTONEN, J., KING, P., AND BERAN, P. 2003. POD-based reduced-order models with deforming grids. *Mathematical and Computer Modelling 38*, 41 – 62.

BARAFF, D., AND WITKIN, A. 1992. Dynamic simulation of non-penetrating flexible bodies. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, 303–308.

BARBIČ, J., AND JAMES, D. L. 2005. Real-Time Subspace Integration for St. Venant-Kirchhoff Deformable Models. *ACM Trans. on Graphics 24*, 3 (Aug.), 982–990.

BERGMANN, M., CORDIER, L., AND BRANCHER, J.-P. 2005. Optimal rotary control of the cylinder wake using proper orthogonal decomposition reduced-order model. *Physics of Fluids 17*, 9, 097101.

BERKOOZ, G., HOLMES, P., AND LUMLEY, J. L. 1993. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual Rev. Fluid Mech*, 539–575.

BOURGUET, R., BRAZA, M., AND DERVIEUX, A. 2011. Reduced-order modeling of transonic flows around an airfoil submitted to small deformations. *Journal of Computational Physics 230*, 1, 159 – 184.

BRO, R., AND DE JONG, S. 1997. A fast non-negativity-constrained least squares algorithm. *Journal of Chemometrics 11*, 5, 393–401.

BRO, R., 2001. The n-way toolbox. http://bit.ly/Wmq8zM.

BROCHU, T., KEELER, T., AND BRIDSON, R. 2012. Linear-time smoke animation with vortex sheet meshes. In *Proceedings of the ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, 87–95.

CARLBERG, K., BOU-MOSLEH, C., AND FARHAT, C. 2011. Efficient non-linear model reduction via a least-squares petrov-galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering 86*, 2, 155–181.

CHADWICK, J. N., AN, S. S., AND JAMES, D. L. 2009. Harmonic shells: a practical nonlinear sound model for near-rigid thin shells. *ACM Trans. Graph. 28*, 5 (Dec.), 119:1–119:10.

CHEN, D., AND PLEMMONS, R. 2007. Nonnegativity constraints in numerical analysis. In *Symposium on the Birth of Numerical Analysis*.

DE WITT, T., LESSIG, C., AND FIUME, E. 2012. Fluid simulation using laplacian eigenfunctions. *ACM Trans. Graph. 31*, 1, 10:1–10:11.

DEPARIS, S., AND ROZZA, G. 2009. Reduced basis method for multi-parameter-dependent steady navierstokes equations: Applications to natural convection in a cavity. *Journal of Computational Physics 228*, 12, 4359 – 4378.

FEDKIW, R., STAM, J., AND JENSEN, H. W. 2001. Visual simulation of smoke. In *Proceedings of SIGGRAPH*, 15–22.

FOSTER, N., AND METAXAS, D. 1997. Modeling the motion of a hot, turbulent gas. In *Proceedings of SIGGRAPH*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 181–188.

GOLUB, G., AND VAN LOAN, C. 1996. *Matrix Computations*, 3rd ed. The Johns Hopkins University Press, Baltimore.

GUENNEBAUD, G., JACOB, B., ET AL., 2010. Eigen v3. http://eigen.tuxfamily.org.

HENDERSON, R. D. 2012. Scalable fluid simulation in linear time on shared memory multiprocessors. In *Proceedings of the Digital Production Symposium*, ACM Press, 43–52.

HOMESCU, C., PETZOLD, L. R., AND SERBAN, R. 2005. Error estimation for reduced-order models of dynamical systems. *SIAM Journal on Numerical Analysis 43*, 4, 1693–1714.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics 22*, 3 (July), 879–887.

KIM, T., AND JAMES, D. L. 2009. Skipping steps in deformable simulation with online model reduction. *ACM Transactions on Graphics 28*, 5 (Dec.), 123:1–123:9.

KIM, T., AND JAMES, D. L. 2011. Physics-based character skinning using multi-domain subspace deformations. In *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, ACM, New York, NY, USA, 63–72.

KIM, T., THÜREY, N., JAMES, D., AND GROSS, M. 2008. Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27* (August), 50:1–50:6.

KIM, D., SRA, S., AND DHILLON, I. S. 2012. A non-monotonic method for large-scale non-negative least squares. *Optimization Methods and Software (OMS)* (Jan.).

KLEINBERG, J., AND TARDOS, E. 2006. *Algorithm Design*. Addison-Wesley.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. In *Proceedings of SIGGRAPH*, 820–825.

KRYSL, P., LALL, S., AND MARSDEN, J. E. 2001. Dimensional model reduction in non-linear finite element dynamics of solids and structures. *International Journal for Numerical Methods in Engineering 51*, 479–504.

LAWSON, C. L., AND HANSON, R. J. 1974. *Solving Least Square Problems*. Prentice Hall, Englewood Cliffs, NJ.

LEGRESLEY, P. A., AND ALONSO, J. J. 2001. Investigation of non-linear projection for pod based reduced order models for aerodynamics. In *AIAA Aerospace Sciences Meeting and Exhibit*.

LENTINE, M., ZHENG, W., AND FEDKIW, R. 2010. A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph. 29* (July), 114:1–114:9.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 457–462.

LUMLEY, J. 1967. The structure of inhomogeneous turbulent flows. *Atmospheric turbulence and radio wave propagation*, 166–178.

LUO, Y., AND DURAISWAMI, R. 2011. Efficient parallel nonnegative least squares on multicore architectures. *SIAM Journal on Scientific Computing 33*, 5, 2848–2863.

MEYER, M., AND ANDERSON, J. 2007. Key Point Subspace Acceleration and Soft Caching. *ACM Transactions on Graphics 26*, 3 (July), 74.

MOËS, N., DOLBOW, J., AND BELYTSCHKO, T. 1999. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering 46*, 1, 131–150.

MOLEMAKER, J., COHEN, J. M., PATEL, S., AND NOH, J. 2008. Low viscosity flow simulations for animation. In *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, 9–18.

MOLER, C., AND VAN LOAN, C. 2003. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review 45*, 1, 3–49.

MULLEN, P., CRANE, K., PAVLOV, D., TONG, Y., AND DESBRUN, M. 2009. Energy-preserving integrators for fluid animation. *ACM Trans. Graph. 28*, 3 (July), 38:1–38:8.

NARAIN, R., SEWALL, J., CARLSON, M., AND LIN, M. C. 2008. Fast animation of turbulence using energy transport and procedural synthesis. *ACM Trans. Graph. 27* (December), 166:1–166:8.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, 215–222.

PFAFF, T., THUEREY, N., AND GROSS, M. 2012. Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. 31*, 4 (July), 112:1–112:8.

PHARR, M., AND HUMPHREYS, G. 2010. *Physically-Based Rendering: From Theory to Implementation*. Morgan Kaufmann.

PORTUGAL, L. F., JÚDICE, J. J., AND VICENTE, L. N. 1994. A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables. *Mathematics of Computation 63*, 208 (Oct.), 625–643.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA.

RABANI, E., AND TOLEDO, S. 2001. Out-of-core svd and qr decompositions. In *SIAM Conference on Parallel Processing for Scientific Computing*.

SCHECHTER, H., AND BRIDSON, R. 2008. Evolving sub-grid turbulence for smoke animation. In *ACM SIGGRAPH/Eurographics Sym. on Computer Animation*, 1–7.

SELLE, A., FEDKIW, R., KIM, B., LIU, Y., AND ROSSIGNAC, J. 2008. An unconditionally stable maccormack method. *J. Sci. Comput. 35*, 2-3 (June), 350–371.

SEO, J., IRVING, G., LEWIS, J. P., AND NOH, J. 2011. Compression and direct manipulation of complex blendshape models. *ACM Trans. Graph. 30*, 6 (Dec.), 164:1–164:10.

SERRE, G., LAFON, P., GLOERFELT, X., AND BAILLY, C. 2012. Reliable reduced-order models for time-dependent linearized euler equations. *Journal of Computational Physics 231*, 15, 5176 – 5194.

SETHIAN, J. 1999. *Level set methods and fast marching methods*. Cambridge University Press.

SHAH, A. 2007. Cooking effects. In *ACM SIGGRAPH 2007 courses*, ACM, New York, NY, USA, SIGGRAPH '07, 45–58.

SIFAKIS, E., AND BARBIČ, J. 2012. Fem simulation of 3d deformable solids: a practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses*, ACM, New York, NY, USA, SIGGRAPH '12, 20:1–20:50.

STAM, J. 1999. Stable fluids. In *SIGGRAPH 1999*, 121–128.

STANTON, M., SHENG, Y., WICKE, M., PERAZZI, F., YUEN, A., AND ANDADRIEN TREUILLE, S. N. 2013. Non-polynomial galerkin projection on deforming meshes. *ACM Trans. Graph. 32* (July).

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. *ACM Transactions on Graphics 25*, 3 (July), 826–834.

VASILESCU, M. A. O., AND TERZOPOULOS, D. 2004. Tensortextures: multilinear image-based rendering. *ACM Trans. Graph. 23*, 3 (Aug.), 336–342.

WICKE, M., STANTON, M., AND TREUILLE, A. 2009. Modular bases for fluid dynamics. *ACM Trans. on Graphics 28*, 3 (Aug.), 39.