Model Reduction for Real-time Fluids

Adrien Treuille[1] Andrew Lewis[1] Zoran Popović[1,2]

[1]University of Washington [2] Electronic Arts

Abstract

We present a new model reduction approach to fluid simulation, enabling large, real-time, detailed flows with continuous user interaction. Our reduced model can also handle moving obstacles immersed in the flow. We create separate models for the velocity field and for each moving boundary, and show that the coupling forces may be reduced as well. Our results indicate that surprisingly few basis functions are needed to resolve small but visually important features such as spinning vortices.

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

Keywords: Model Reduction, Proper Orthogonal Decomposition, Flow with Boundaries

Figure 1: An interactive fluid simulation of a car driving through 4000 leaves runs at 24 frames/second on a 256×64×128 grid. (Car model courtesy of Digimation.)

1 Introduction

Computer graphics researchers have developed strikingly realistic models of fluid phenomena such as curling smoke and splashing water for cinematic special effects. A much less explored domain is real-time interactive fluid simulation, which opens new avenues for training, computer games, and interactive media. Simulation steps must occur in fraction of a second, handle fluid-object interaction, and produce visually convincing fluid flows that capture important coherent features such as turbulence. To reach such rates, simulation time should be proportional to the rendering complexity, not the size of the computational domain. Further, since interactive simulations do not have preordained length, it would be useful to have unconditional stability and controlled energy dissipation. To date, there is no known fluid model that enjoys these properties.

This work attempts to fill this gap with a model reduction approach to fluids. We begin by using an accurate off-line solver to produce a set of high-resolution fluid simulations representative of the expected user input. These velocity fields are distilled into a small basis of size proportional to the system's principle modes of variation. We then project the fluid equations onto this low-dimensional basis with techniques drawn from model reduction. After these precomputations, we can simulate the velocity field very quickly through the subspace, achieving simulation runtime costs polynomial in the number of basis states, but *independent* of the number of simulation voxels. Our algorithm also can preserve the energy in the basis, allowing for controlled energy dissipation. Finally, multiple moving objects may be immersed in the flow at runtime, and fluid-object interaction can be handled entirely in the reduced space.

Model reduction has unique properties not present in full-dimensional simulations. Complex phenomena can be observed on large grids (e.g. the wake of a speeding car) and the number of variables is essentially proportional to the underlying degrees of freedom (e.g. the position and direction of the cars). As such, each model represents a problem-specific subspace of all simulations and is inherently less accurate or general than full dynamics. However, our technique has very different time complexity as well: even if we choose twice or ten times higher spatial resolution problems, the reduced algorithm remains just as fast. In fact, in our examples our method is not bound by the size of the computational domain but by the computer memory and cost of rendering.

Depending on the rendering choice, we couple our reduced simulator with either immersed particles (textured smoke sprites, dust, leaves), or a grid-based density simulation. Immersed particles, which are used extensively in games to reduce rendering costs, are especially well suited to our technique as they need only sparsely query the velocity field. Density simulation and rendering both require a pass through the entire grid, and therefore dominate any fluid computations in the reduced space. Importantly, for both particles and densities, our algorithm is *output-sensitive:* the cost of simulation is proportional to the cost of rendering.

Contributions. We present a new algorithm for the reduced-dimensional simulation of incompressible fluids tailored for interactive graphics applications; we describe how complex moving and fixed boundaries can also be handled in the reduced space. Our algorithm allows for output-sensitive simulation of higher resolution and more complex fluid dynamics than previously possible in real-time computer graphics.

2 Related Work

In graphical fluid dynamics, an important step towards interactivity was taken by Stam [1999] with the introduction of the Stable Fluids algorithm. This algorithm is unconditionally stable, al-

lowing for long timesteps and fast simulations. Subsequent work aimed at speeding up this algorithm through hierarchical space decomposition [Losasso et al. 2004] or computation on non-uniform meshes [Feldman et al. 2005; Elcott et al. 2005] to date has not achieved real-time simulation of high-resolution fluids. Solving the fluid equations on the hierarchical spatial subdivision [Losasso et al. 2004] has yielded impressively high resolution results, but the high per-step computation cost prevents this method from being used in real-time contexts. Under very specific conditions with strictly periodic boundaries, Fourier methods [Stam 2001] can produce very fast performance. While hierarchical, non-uniform, and Fourier methods can provide substantial speedups, they still require computation proportional to the size of the domain. Pure vortex methods [Park and Kim 2005] are not restricted to a particular domain, but the effective resolution coarsens as the particles are spaced farther apart. Vortex methods can also present difficulties handling boundaries and diffusion. Hybrid grid and vortex methods have also been proposed [Selle et al. 2005] although the intent is to produce more faithful dynamics, not faster simulation. Both grid and vortex-based methods are at best linear in the velocity sampling. In our method, by contrast, the number of variables is essentially proportional to the underlying degrees of freedom, such as the number of possible input forces.

Various fluid simulation methods have been mapped to the GPU as a stream processor including a Lattice Boltzman Method [Li et al. 2003], Coupled Map Lattice [Harris et al. 2002], semi-Lagrangian solver with boundaries [Wu et al. 2005], and a Poisson solver [Krüger and Westermann 2003; Bolz et al. 2003; Goodnight et al. 2003]. These algorithms have the same time complexity as their CPU versions and do not offer resolution-independent simulation. They can however achieve large constant factor speedup over CPU algorithms, often producing real-time performance for medium-size problems. We use the GPU for our examples with density advection and rendering; however our method's major computational savings come from our reduced simulator which is completely implemented on the CPU.

Model reduction is an increasingly important technique in computer graphics. It has been used to reduce a wide range of problems, ranging from global illumination to elastostatics and dynamics. James and Fatahalian [2003] precompute nonlinear deformation responses to a finite set of user impulses and apply dimension reduction using PCA. Their approach restricts the range of possible runtime interactions to a small discrete set of pre-selected impulses. Barbič and James [2005] present a more general approach for FEM models that allows general runtime forcing within the reduced-dimensional subspace. Like us, they exploit the property that a dynamical system's polynomial degree remains unchanged under model reduction. Our work on reduced fluids was greatly influenced by this work. Multilinear model reduction [Wang et al. 2005] has also been used as a compression tool for non-interactive time-sequences. Within computer graphics, model reduction has not been previously applied to fluid simulation.

In the applied mathematics literature, as well as in computational fluid dynamics (CFD), dimensional model reduction is well studied [Lumley 1970]. These methods use Galerkin projection to yield systems of differential equations involving fewer equations and fewer unknown variables. The reduced equations can be solved much more quickly than the original problem, at some accuracy cost. These methods also appear in literature under the names of proper orthogonal decomposition (POD), Karhunen-Loéve decomposition and subspace integration. In CFD, these methods are primarily used to study organized spatial characteristics in a flow, i.e. coherent structures [Holmes et al. 1996], and have been used to design flow control systems since they provide a more controllable description of the system dynamics [Rowley et al. 2006; Ausseur et al. 2004]. Sirovich [1987] introduced snapshot POD, which essentially constructs a reduced model from a sequence of instantaneous spatial fields acquired at different times. To increase accuracy, these models can be subsequently calibrated to the observed data [Couplet et al. 2005]. After a long-time integration of dissipative flows, POD-based simulation methods exhibit drifting inaccuracies that have been handled with non-linear Galerkin projection [Marion and Temam 1989] and viscosity method correction [Sirisup and Karniadakis 2004]. Extended POD methods [Maurel et al. 2001] were developed to study correlated events in turbulent flow that are not directly related to energy.

Treatment of moving boundary conditions is not as extensively studied for model reduction. A straightforward way to treat moving boundaries is to construct a separate set of bases for each possible boundary configuration [Schmit and Glasuer 2002]. This approach leads to an explosion of the required bases unless the boundary movement is restricted to a small subspace. For flows with periodic boundaries, and with inherent symmetries within the flow dynamics, it is possible remove uniform translation modes [Rowley et al. 2003; Rowley and Marsden 2000]. In a restricted case when analyzing a single boundary in free flow, one can form the reduced basis in the frame of reference of the boundary as it is moved through various angles of flow attack [Ausseur et al. 2004]. Similarly, when a single moving boundary moves along a single dimension such as pistons within the engine cylinder, stretching and aligning of the flow basis can allow for a reduced model [Fogleman et al. 2004]. We are not aware of any references on reduced methods that deal with multiple moving boundaries, or boundaries that move in unpredictable ways.

3 Model Reduction Overview

At the heart of our fast fluid simulator is a model-reduced simulation of the incompressible Navier-Stokes equations. In this section we give a general overview of dimension reduction and discuss its connection to physical simulation.

Dimension reduction means representing a vector in high dimensional space $\mathbf{u} \in \mathbf{R}^n$ with a corresponding vector in a much lower dimensional space $\mathbf{r} \in \mathbf{R}^m$. While necessarily approximate, such a representation can be quite useful for compressing redundant information in $\mathbf{u}$. To move between the two spaces we need a *projection operator* $P : \mathbf{u} \mapsto \mathbf{r}$ and its inverse $P^{-1} : \mathbf{r} \mapsto \mathbf{u}$. The former cannot be one-to-one because $m < n$ by assumption.

For some applications, dimension reduction alone is a useful technique. In physical simulation, however we are interested not only in the state $\mathbf{u}$, but also in its time evolution, typically described by an ordinary differential equation: $\dot{\mathbf{u}} = F(\mathbf{u})$. (The dot indicates a time derivative.)

The link between dimension reduction and physics is *model reduction*. We seek an analogous evolution equation for the reduced state: $\dot{\mathbf{r}} = \hat{F}(\mathbf{r})$. Ideally, this equation should be computable in closed form without resorting to the high-dimensional space. The standard solution is to compute the *Galerkin projection* of F onto the reduced dimensional space

$$\hat{F} = P \circ F \circ P^{-1}. \tag{1}$$

We now consider an important special case: suppose the reduced vectors cover an m-dimensional *linear* subspace of $\mathbf{R}^n$. That is, there exists an $n \times m$ matrix B such that $\mathbf{u} = B\mathbf{r}$. Further, if B is orthonormal then $\mathbf{r} \approx B^T\mathbf{u}$, with equality when $\mathbf{u}$ lies in the subspace spanned by B. Armed with this linear subspace, consider the case of linear differential equation $\dot{\mathbf{u}} = M\mathbf{u}$. The Galerkin projection of M yields another linear differential equation $\dot{\mathbf{r}} = B^T MB\mathbf{r}$. Moreover, we can *precompute* the $m \times m$ matrix product $B^T MB$ to simulate without resorting to the high-dimensional space. More generally, if F is an n-dimensional polynomial, then $\hat{F}$ is an m-dimensional

polynomial of the same degree which can be computed in closed form without resorting to the high-dimensional space [Barbič and James 2005]. We will use this important fact several times.

4 Model Reduction of Fluids

We now explain how to apply model reduction specifically to create a high speed fluid simulator. Our first task is to create a low-dimensional basis B for the simulation. We assume that the domain has been spatially discretized so that all velocity fields can be represented as n-dimensional vectors. In practice we use the standard MAC-style discretization [Foster and Metaxas 1996], but our technique is general enough to handle arbitrary arrangements such as irregular tetrahedral meshes [Feldman et al. 2005; Elcott et al. 2005].

We also assume a set of example velocity fields $\{\mathbf{u}_i\}$, forming a representative basis U of the interactions expected in the system. These example states typically are snapshots drawn from a set of off-line, full-dimensional fluid simulations. We also assume that the velocity fields in U all satisfy two important properties:

1. The states are *divergence free*: $\nabla \cdot \mathbf{u}_i = 0$.[1]
2. The states satisfy *free-slip* boundary conditions: for all fixed surface points $\mathbf{x}$ with normal $\mathbf{n}$, we have $\mathbf{u}_i(\mathbf{x}) \cdot \mathbf{n} = 0$.

We seek a low-dimensional orthonormal basis $B = [\hat{\mathbf{u}}_1, \hat{\mathbf{u}}_2, \ldots, \hat{\mathbf{u}}_m]$ ($m \ll n$) that minimizes the square reconstruction error

$$||U - BB^T U||_F^2, \tag{2}$$

subject to the basis states $\hat{\mathbf{u}}_i$ preserving the two constraints above. Here $||\cdot||_F^2$ denotes the sum of squared matrix entries, known as the Frobenius norm.

Equation (2) can be minimized by setting B to the first m eigenvectors of the matrix UU^T, effectively performing a Principal Component Analysis (PCA). Further, the vectors produced by PCA satisfy the above constraints without modification. To see why, note that both constraints are linear. That is, for each constraint there exists a *constraint matrix* C whose null space is exactly the set of vectors satisfying the constraint (in particular $CU = 0$). Now suppose that $\hat{\mathbf{u}}_i$ is a vector in B. Then by definition $UU^T\hat{\mathbf{u}}_i = \lambda_i\hat{\mathbf{u}}_i$, which implies that $CUU^T\hat{\mathbf{u}}_i = \lambda_i C\hat{\mathbf{u}}_i = 0$. So the basis vector $\hat{\mathbf{u}}_i$ will satisfy the constraints as long as the eigenvalue λ_i is nonzero, as it will be in general. Figure 2 shows the basis constructed for a domain with a fixed yin-yang boundary. Notice that all basis states satisfy both the divergence and boundary constraints.

5 Simulation

Given the reduced velocity basis B, we now present an algorithm capable of simulating the incompressible fluid equations through the reduced subspace. The incompressible Navier-Stokes equations can be written as:

$$\dot{\mathbf{u}} = \underset{\text{(A)}}{-(\mathbf{u}\cdot\nabla)\mathbf{u}} - \underset{\text{(B)}}{\nu\nabla^2\mathbf{u}} + \underset{\text{(C)}}{\nabla p} + \underset{\text{(D)}}{\mathbf{f}}, \quad \text{s.t.} \quad \underset{\text{(C)}}{\nabla\cdot\mathbf{u} = 0}, \tag{3}$$

where $\mathbf{u}$ denotes the velocity field, p the pressure, ν the viscosity, and $\mathbf{f}$ the external forces. Following Stam [1999], we use a technique called operator splitting to decompose the simulation into a series of separate steps, each one tackling a different term in Equation (3). These steps are: (A) Advection, (B) Diffusion, (C) Projection, and (D) External Forces. In the remainder of this section, we explain how each of these four steps can be simulated on the subspace B, with a per-timestep time complexity *that depends only on the basis dimensionality*.

[1] Divergence-free in the discrete sense: every voxel has zero net flux.

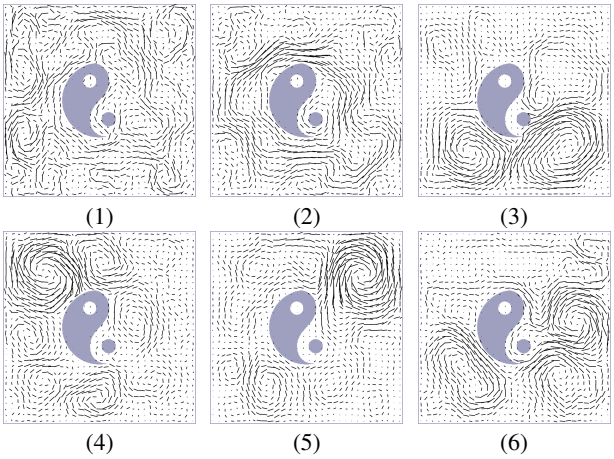

Figure 2: First 6 basis states for a 512×512 simulation. (Grids have been subsampled for display.)

5.1 Advection

We split the velocity field into its three components $\mathbf{u} = [u^x, u^y, u^z]^T$, and can write the velocity advection of each component as $\dot{u}^\star = -(\mathbf{u}\cdot\nabla)u^\star = -\nabla\cdot(\mathbf{u}u^\star)$, where $\star$ stands for x, y, or z. The right hand side of this equation is called the *conservation form* of the advection equation, and equality follows from the zero divergence of the velocity field. The conservation form can be discretized about each grid face (i,j,k) to form the partial differential equation:

$$\begin{aligned}\dot{u}^\star_{i,j,k} = \tfrac{1}{\Delta h}\big(&(u^x u^\star)_{i-1/2,j,k} - (u^x u^\star)_{i+1/2,j,k} + \\ &(u^y u^\star)_{i,j-1/2,k} - (u^y u^\star)_{i,j+1/2,k} + \\ &(u^z u^\star)_{i,j,k-1/2} - (u^z u^\star)_{i,j,k+1/2}\big),\end{aligned} \tag{4}$$

where Δh is the grid spacing. By fixing the velocity fluxes and recombining the three grids we get a linear differential equation:

$$\dot{\mathbf{u}} = A_{\mathbf{u}}\mathbf{u} \tag{5}$$

which indicates how the velocity vector $\mathbf{u}$ would change if advected through a fixed velocity field defined by the advection matrix $A_{\mathbf{u}}$.

We now project this equation onto the reduced subspace B. Let $\mathbf{r} = [r_1, \ldots, r_m]^T$ represent the reduced state so that $\mathbf{u} = B\mathbf{r}$. Then the advection matrix in Equation (5) can be expressed as a linear combination of advection matrices for each velocity basis state:

$$\dot{\mathbf{u}} = \left(r_1 A_{\hat{\mathbf{u}}_1} + \cdots + r_m A_{\hat{\mathbf{u}}_m}\right)\mathbf{u}, \tag{6}$$

effectively contracting the advection tensor. This equation can, in turn, be orthogonally projected onto the reduced dimensional basis as:

$$\dot{\mathbf{r}} = \left(r_1 B^T A_{\hat{\mathbf{u}}_1} B + \cdots + r_m B^T A_{\hat{\mathbf{u}}_m} B\right)\mathbf{r} = \hat{A}\mathbf{r}. \tag{7}$$

Hence, at each timestep we take a linear combination of the m precomputed reduced advection matrices $B^T A_{\hat{\mathbf{u}}_i} B$ to form the reduced advection matrix $\hat{A}$ as in Equation (7). The reduced state then can be advanced to the next timestep by exactly solving the linear ODE using the eigen-decomposition method:

$$\mathbf{r}(t+\Delta t) = Ee^{\Delta t\Lambda}E^{-1}\mathbf{r}, \tag{8}$$

where $E\Lambda E^{-1} = \hat{A}$ is the eigen-decomposition of $\hat{A}$. All of these operations take time $O(m^3)$ and are independent of n, the number of simulation voxels. For simulations with a very large basis, this

computation might be replaced with other symplectic or implicit integration methods in the reduced space. However, in all of our examples, the cost of this integration has been negligible.

5.1.1 Kinetic Energy Preservation

This advection algorithm naturally preserves kinetic energy, a highly useful property for which we now give a proof sketch. It can be shown that our discretization (4) always yields a zero definite advection matrix ($\forall \mathbf{x} \quad \mathbf{x}^T A_{\mathbf{u}} \mathbf{x} = 0$) given a divergence-free field $\nabla \cdot \mathbf{u} = 0$. The total kinetic energy is defined as $E = \frac{1}{2}\mathbf{u}^T\mathbf{u}$ (where $\mathbf{u}$ is the vector over all grid velocities). By the chain rule, $\dot{E} = \mathbf{u}^T\dot{\mathbf{u}} = \mathbf{r}^T B^T B\dot{\mathbf{r}} = \mathbf{r}^T\dot{\mathbf{r}}$. Substituting Equation (7) for $\dot{\mathbf{r}}$ gives $\dot{E} = \sum_i \mathbf{r}^T B^T A_{\hat{\mathbf{u}}_i} B\mathbf{r}$. Because the advection matrices are all zero-definite, the terms $\mathbf{r}^T B^T A_{\hat{\mathbf{u}}_i} B\mathbf{r}$ all vanish, implying $\dot{E} = 0$. Hence, total energy remains unchanged if we exactly solve the linear advection ODE, as in Equation (8).

We emphasize that energy is preserved *only within the reduced subspace*; velocity modes not present in the subspace have zero energy, and represent our algorithm's primary source of error. Moreover in practice, we intentionally dissipate energy through diffusion (Section 5.2) because completely inviscid flows are not seen in nature, and will gradually depart from the observed basis. Also, diffusion adds stability during user interaction. Nonetheless, energy preservation allows us to control the energy dissipation profile precisely (Figure 3).

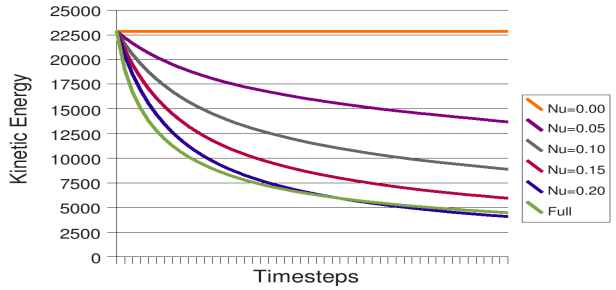

Figure 3: **Energy Dissipation** in a 128×128 simulation. Our simulator exactly preserves kinetic energy in the absence of diffusion, $\nu = 0.0$. A diffusion coefficient of $\nu = 0.2$ (blue) most accurately matches the numerical viscosity inherent in our full dimensional simulator (green).

5.2 Diffusion

The diffusion step accounts for viscosity. Simulation proceeds by discretizing the diffusion operator $\nu\nabla^2$ into a diffusion matrix νD. In an unreduced simulation, the next step would be to solve the linear ODE either explicitly or implicitly, each of which have problems. Explicit methods become unstable for high kinematic viscosities ν, while implicit solvers are stable, but require solving a linear system. Our reduced technique resolves both of these problems: we can simulate more quickly than an explicit solver and more accurately than an implicit method. The algorithm is as follows: the diffusion matrix D can be orthogonally projected to produce the reduced diffusion matrix $\hat{D} = B^T DB$. As with advection, we *exactly* solve the linear ordinary differential equation defined by the reduced diffusion matrix $\mathbf{r}(t+\Delta t) = Ee^{\Delta t\Lambda}E^{-1}\mathbf{r}$ (where $E\Lambda E^{-1} = \hat{D}$ now denotes the eigen-decomposition of the precomputed *diffusion* matrix). Unlike advection, this transition matrix is constant throughout the simulation and therefore can be precomputed. This enables stable, accurate diffusion with a single $m\times m$ matrix multiply. We note that as with advection, accuracy is only within the reduced subspace; the system cannot diffuse into configurations outside the subspace.

5.3 Projection

Projection involves adding a pressure force ∇p to enforce both the flow's incompressibility and correct fluid-object boundary conditions. As noted in Section 4, our model reduction approach preserves these constraints automatically for fixed boundaries. Thus $p = 0$, and we can completely skip the projection step. This yields substantial speed gains because projection is in general the most costly step of grid-based simulators; even with symmetric operators on hierarchical grids, this step can take 25% of each timestep [Losasso et al. 2004]. Section 7 shows how our technique may be extended to handle moving objects through a local approximation of the pressure forces around each object.

5.4 External Forces

External forces can be added by projecting the forces $\mathbf{f}$ onto the reduced subspace $\hat{\mathbf{f}} = B^T\mathbf{f}$, and adding those onto the reduced velocity vector: $\mathbf{r} += \hat{\mathbf{f}}$. Note that because the basis is global, even sparse forces $\mathbf{f}$ project to non-sparse reduced force vectors $\hat{\mathbf{f}}$.

6 Immersed Media

For the flow to be visible, pure velocity simulations must be coupled with immersed media, such as particles (including dust, leaves, and texture sprites) or volumetric smoke densities. Particles are particularly well suited to real-time reduced fluid simulation: particle rendering is fast, and simulation only requires reconstructing velocities in the neighborhood of each particle. Thus, we see particular speedups when marrying model reduction with particle simulation. To simulate volumetric densities, we perform semi-Lagrangian advection [Stam 1999] through the full velocity field $\mathbf{u}$ at each timestep. While the $O(mn)$ velocity reconstruction step introduces a performance dependency on the number of voxels n, this approach is justified because the cost of rendering is also proportional to the number of voxels, and is generally much higher than that of simulation. For both particles and densities our fluid simulator is *output-sensitive*: the costs of simulation and rendering are proportional to one another.

7 Boundary Conditions

The above algorithm simulates the complete incompressible Navier-Stokes equations on a reduced dimensional basis. This algorithm is self-contained, and generally much faster than traditional techniques. The only means of user interaction, however, is the external force term (Section 5.4). While forces can be useful, a more natural form of user interaction would be to *move* objects through the flow. We now extend the above simulator to do just that.

Boundaries are handled via the pressure term (3C), which adds the minimal force so that the fluid velocity field $\mathbf{u}$ equals the speed of the object $\mathbf{v}$ along all surface normals. That is, at surface point $\mathbf{x}$ with normal $\mathbf{n}$, we have $\mathbf{u}(\mathbf{x}) \cdot \mathbf{n} = \mathbf{v}(\mathbf{x}) \cdot \mathbf{n}$ (tangential fluid movement is allowed). This *free-slip* condition cannot be satisfied by the low-dimensional physics alone because the model does not have sufficient degrees of freedom. Therefore, our model must somehow be extended. Ideally, we would have high-frequency fidelity along the surface: for example, smoke would appear to curl around a surface rather than penetrate it. It should also be possible to handle multiple objects moving independently. Most importantly, we seek a low-dimensional (possibly approximate) model which admits all computations in the reduced-dimensional space.

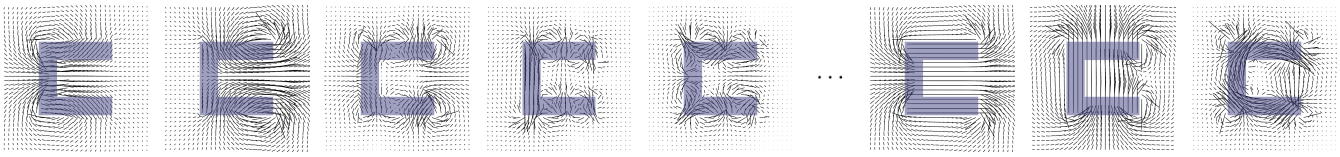

Figure 4: (1)-(9): The first 9 basis grids for a c-shape object (skipping every other). These form a basis for the expected velocities at the surface, and are divergence-free everywhere else. $(\dot{x})$, $(\dot{y})$, and $(\dot{\theta})$: The additional translational and rotational basis (Section 7.2.3).

7.1 The Boundary Basis

Our solution is to add a set of forces in the local neighborhood of each object which enforces the boundary conditions at the surface. We chose a linear representation, so that these forces consist of a boundary basis $\tilde{B} = [\tilde{\mathbf{u}}_1, \ldots, \tilde{\mathbf{u}}_p]$ combined with a reduced state vector $\tilde{\mathbf{r}} = [\tilde{r}_1, \ldots, \tilde{r}_p]^T$. The power of the boundary bases is that they track the location of their objects. For an object with rotation matrix Φ and position vector $\mathbf{t}$, let the *resampling operator* $R_{\Phi,\mathbf{t}}$ convert a velocity field from the object's frame of reference into the global coordinate system: $\left[R_{\Phi,\mathbf{t}}\mathbf{u}\right](\mathbf{x}) \equiv \Phi\mathbf{u}(\Phi^{-1}\mathbf{x} - \mathbf{t})$, as illustrated in Figure 5. We now say that the full velocity field $\mathbf{u}'$ combines both simulation basis and all boundary bases (properly oriented in the global coordinate system):

$$\mathbf{u}' = B\mathbf{r} + \sum_i R_i \tilde{B}_i \tilde{\mathbf{r}}_i, \tag{9}$$

where $R_i = R_{\Phi_i,\mathbf{t}_i}$ is shorthand for object i's resampling operator. By combining object tracking with a linear representation, boundary bases allow the quick and compact enforcement of the free-slip conditions.

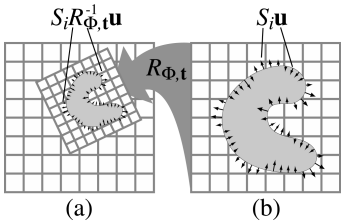

Figure 5: The resampling operator $R_{\Phi,\mathbf{t}}$ changes a velocity field's coordinate system, while the surface speed operator S_i measures surface normal velocity.

Since the free-slip boundary conditions constrain surface speed, it will be useful to define an operator which measures these velocities. To do so, we discretize object i's surface into a set of points $\{\mathbf{x}_1, \ldots, \mathbf{x}_q\}$ with corresponding surface normals $\{\mathbf{n}_1, \ldots, \mathbf{n}_q\}$. We now define the *surface speed operator* $S_i\mathbf{u} \equiv \left[\mathbf{u}(\mathbf{x}_1)\cdot\mathbf{n}_1, \ldots, \mathbf{u}(\mathbf{x}_q)\cdot\mathbf{n}_q\right]^T$. Intuitively, the indices into this vector correspond to points along the surface, and the entries are the normal speeds at these points. Graphically, these are the lengths of the spiky surface vectors in Figure 5(b). Surface speeds can be computed for other object orientations $(\Phi, \mathbf{t})$ by first resampling the velocity field $S_i R^{-1}_{\Phi,\mathbf{t}}\mathbf{u}$ as in Figure 5(a). (The inverse R^{-1} is required to convert into the *object*'s frame of reference.)

We now turn to how the boundary basis is constructed. For simplicity, we provisionally assume that each object is at rest. Hence, the boundary basis should cancel all velocity normal to the object surface: $S_i R_i^{-1}\mathbf{u} = \mathbf{0}$, meaning that the basis should match the velocity basis as closely as possible (with opposite sign). Therefore, our goal is to construct a boundary basis as representative as possible of the expected surface velocities. To do so, we perform a double loop over the velocity fields in the simulation basis $\mathbf{u} \in B$ and a representative set of orientations $(\Phi, \mathbf{t}) \in \mathcal{P}$. For every pair of velocity field and orientation, we measure the surface speed assuming object i were thus situated, and we collect these vectors into a large set $\mathcal{S}_i = \{S_i R^{-1}_{\Phi,\mathbf{t}}\mathbf{u} \mid (\Phi,\mathbf{t}) \in \mathcal{P}, \mathbf{u} \in B\}$. Running PCA gives us the most representative p-dimensional basis for $\mathcal{S}_i$. In fact, we set the boundary basis $\tilde{B}_i$ to be the set of minimum energy divergence-free velocity fields whose surface speeds are those PCA eigenvectors: $S_i\tilde{B}_i = PCA_p[\mathcal{S}_i]$. Note that because PCA always returns an orthonormal set of vectors, we have the useful property that: $\left(S_i\tilde{B}_i\right)^T S_i\tilde{B}_i = I$. Figure 4(1)-(9) shows the resulting boundary basis for a c-shaped object.

7.2 Coupling

Taking the boundary bases into account requires a modest extension to our algorithm. At each timestep, after the reduced state $\mathbf{r}$ has been simulated but *before* immersed media have been advected, we compute the reduced state vector $\tilde{\mathbf{r}}$ for each object. Immersed media can then be advected through the *complete* velocity field $\mathbf{u}'$ (9). At the end of each timestep, we compute the Galerkin projection of these boundary forces back into the reduced simulation basis. Figure 6 gives more precise pseudocode. For now, we take a closer look at how information is propagated between the simulation and boundary bases, which we call the *feedforward* $F_i : \mathbf{r} \mapsto \tilde{\mathbf{r}}_i$ and *feedback* $G_i : \tilde{\mathbf{r}}_i \mapsto \mathbf{r}$ operators, respectively.

1.	Advect the velocities.	$O(m^3)$	(5.1)
2.	Diffuse the velocities.	$O(m^2)$	(5.2)
3.	Add forces.	$O(m)$	(5.4)
4.	**for each object** i:		
	// Feedforward.		
	$\tilde{\mathbf{r}}_i \leftarrow F_i\mathbf{r}$	$O(pm)$	(7.2.1)
5.	Advect immersed media in $\mathbf{u}'$.		(6)
6.	**for each object** i:		
	// Feedback.		
	$\mathbf{r} \mathrel{+}= G_i\tilde{\mathbf{r}}_i$	$O(pm)$	(7.2.2)

Figure 6: **Per-timestep pseudocode.** We first simulate the global field (1)-(3), then account for boundaries (4)-(6).

7.2.1 Forward Coupling

For simplicity, we shall continue to assume that object i is at rest, relaxing this assumption only in Section 7.2.3. As mentioned above, this implies no velocity along surface normals: $S_i R_i^{-1}\mathbf{u} = \mathbf{0}$. Therefore, if we premultiply Equation (9) by $S_i R_i^{-1}$ and ignore all objects but i we get:

$$\mathbf{0} = S_i R_i^{-1} B\mathbf{r} + S_i\tilde{B}_i\tilde{\mathbf{r}}_i.$$

Ignoring these objects is justified because coupling effects are negligible unless two objects are very close. Using the orthonormality of $S_i\tilde{B}_i$, the above equation can be solved as $\tilde{\mathbf{r}}_i = -\left(S_i\tilde{B}_i\right)^T S_i R_i^{-1} B\mathbf{r}$. Therefore we define the feedforward operation:

$$F_i \equiv -\left(S_i\tilde{B}_i\right)^T S_i R_i^{-1} B. \tag{10}$$

Intuitively, we have taken the simulation velocity field (restricted to the object surface) $S_i R_i^{-1} B$ and projected this into the space of boundary basis fields $S_i \tilde{B}_i$ to get the reduced boundary vector $\tilde{\mathbf{r}}$ which best cancels all velocities along the surface. Computationally, this operation requires first iterating over all object faces and accumulating velocities, and then projecting the surface speeds onto the boundary basis. Performing these two steps takes time $O(qp+qm)$, where q is the number of faces, and m and p are the sizes of the simulation and boundary bases, respectively. In Section 7.3, we show how this complexity may be greatly reduced through precomputation.

7.2.2 Backward Coupling

Once the reduced boundary vectors have been computed for each object, and any immersed media have been advected through the flow, we must project the effect of the boundary forces back onto the reduced simulation basis. Equation (9) can be Galerkin-projected (premultiplied by B^T) to read:

$$\mathbf{r}' = \mathbf{r} + \sum_i B^T R_i \tilde{B}_i \tilde{\mathbf{r}}_i.$$

Therefore, the feedback operation

$$G_i \equiv B^T R_i \tilde{B}_i \tag{11}$$

requires iterating over the voxels in the boundary basis, accumulating boundary forces, and then projecting these forces onto the reduced simulation basis. This two step process takes time $O(bp+bm)$, where b is the number of voxels in the boundary basis. Like forward coupling, the feedback operator may be reduced, as described in Section 7.3.

7.2.3 Extension to Moving Objects

Until now we have assumed that the object is at rest. A simple extension handles moving objects. First, the boundary basis $\tilde{B}$ must be extended to include local forces due to translation and rotation. In 2D there are two translational forces, $\dot{\mathbf{x}}$ and $\dot{\mathbf{y}}$, and one rotational force $\dot{\theta}$, as shown in Figure 4. Therefore the basis is expanded to become $\tilde{B}' = \left[\tilde{B}, \dot{\mathbf{x}}, \dot{\mathbf{y}}, \dot{\theta}\right]$. In 3D there are three translational forces and two rotational forces. At runtime, the feedforward operator is computed as before (Section 7.2.1), except that after surface velocities have been canceled, the extra forces are added equal to the translation and rotation of the object. The feedback operation (Section 7.2.2) remains identical, except that the full boundary basis, including the extra forces, must be projected back into the reduced basis.

7.3 Precomputed Coupling

In Sections 7.2.1 and 7.2.2 we defined the feedforward and feedback operators in the full space: feedforward requires iterating over all object faces, while feedback requires iterating over all voxels in the boundary basis. Notice however that these operations (10) and (11) are *linear* in their operands (although nonlinear in the object location). Therefore, we can *precompute* the matrices F and G over a discretized set of boundary orientations $(\Phi, \mathbf{t})$, and simulate feedforward and feedback entirely in the reduced space. Both operations become virtually instantaneous matrix multiplies taking time $O(pm)$ per object, where p and m are the dimensions of the boundary and simulation bases respectively.

Precomputing the feedback and feedforward operators dramatically improves simulation time. However, it can be very costly in terms of memory, particularly if matrices are being stored along both translational and rotational axes. High-order tensor compression offers the most promising solution to this problem. We used PCA to compress the feedback matrices, storing only the coefficients at each grid cell. The technique of Wang *et al.* [2005], and related ideas might yield even greater memory savings. There are also a number of other strategies one can employ to reduce the memory burden. For example, if an object were restricted to a certain part of the domain or certain orientations, then only the corresponding matrices need be stored. A balance must also be sought between finely discretized boundary orientations and memory. Doubling the spacing between precomputed matrix samples reduces memory requirements by 75% in 2D, but also decreases simulation accuracy, a tradeoff which is explored in Figure 11. In some cases, boundary operations may not be the bottleneck, and precomputation is not necessary. However especially in particle simulations, feedforward and feedback precomputation allows us to work completely in the reduced space, yielding large time savings.

7.4 The Reduced Model's Relation to the Physics

The pressure term of the Navier-Stokes equations (3C) ensures both that the flow is divergence-free and that surface boundary conditions are met. In our model, the former constraint is implicitly inherited from the basis grids. Time-varying boundary conditions, however, must be enforced explicitly using boundary bases. In fact, the boundary basis model can be viewed as an approximation of the pressure term: both add forces to the flow to enforce boundary conditions. However, the boundary basis technique relies on a number of physical approximations that we now make explicit.

Most importantly, our algorithm does not exactly satisfy the free-slip conditions unless the number of boundary basis states equals the number of object faces. However, PCA allows us to solve for the optimal truncated set of boundary basis grids. In practice, we have found that visually accurate simulations can be achieved with 64 basis states, the number we used for all examples. Figure 11 quantifies the success of this approach. Second, we assume that the *instantaneous* effect of an object on the flow is limited to the extent of that object's basis. While the pressure equation is theoretically global, in reality the effects outside the boundary basis are vanishing small. Note that non-local time effects are handled by our feedback term. Third, we compute every object independently, ignoring the visually negligible coupling between constraints. Finally, the pressure equation requires that the field be minimally disturbed to accommodate boundaries. Because the boundary basis may not be orthogonal to the underlying simulation basis, we cannot make such an assurance. Compression of the feedback matrices introduces yet another layer of approximation as described in Section 7.3.

8 Results

We have produced a number of examples with our system (Table 1). Precomputation began by compressing a set of high-resolution fluid snapshots into the simulation basis, then compressing a representative set of surface speed vectors into the boundary basis for each object. For dimension reduction, we used the output-sensitive PCA algorithm described in a technical report by James and Fatahalian [2003]. Precomputation runtimes are summarized in Table 2. All precomputations fit in 3.5GB of main memory, except for our 2 million voxel *Leaves* example for which we used out-of-core PCA. After precomputation, we performed real-time simulation and rendering on a 3.6GHz Intel Xeon computer with an NVIDIA Quadro FX 3450 graphics card. All fluid and particle simulation was performed on the CPU, while density simulation and rendering were GPU-accelerated.

We now explain in more detail how the results were created. The *2D Boundaries* example was trained on 144 simulations of a moving c-shaped object immersed in the flow (Figure 4). We then introduced two additional objects, maintaining visually plausibility

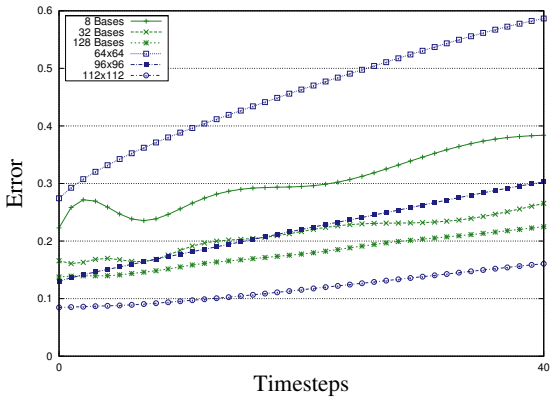

Figure 7: Simulation Error for our *2D Boundaries* example. Error is measured against a full 128×128 semi-Lagrangian simulation.

although the simulator had not trained on these objects. The *Forces* example (Figure 9) was trained with 15 examples of impulse forces which the user can interactively mix at runtime. In both of our 2D examples, the density coloration is purely aesthetic. For the *3D Boundaries* example (Figure 8), we trained the system with a flat boundary object at three locations in a turbulent rising velocity field. At runtime, the user can move both this, and a similar object containing an aperture. We cause the smoke to rise by immersing it in a uniform updraft that unfortunately produces a rather laminar flow. A more principled approach would use a model-reduced temperature field in conjunction with the velocities: much of the turbulence of rising smoke is caused by forces due to temperature gradients. Finally, our *Leaves* example (Figure 1) involves 4000 leaf particles immersed in a large fluid grid. The leaves were subject to both the velocity field and a small gravitational force. We trained these examples by adding impulse forces and by simulating a car moving rapidly through the grid following one of four precomputed paths. In real-time, the user is able to automatically add wind forces, or move the cars with the mouse. We note that the car example was completely implemented on the CPU. We encourage the reader to view the animated results in the companion video.

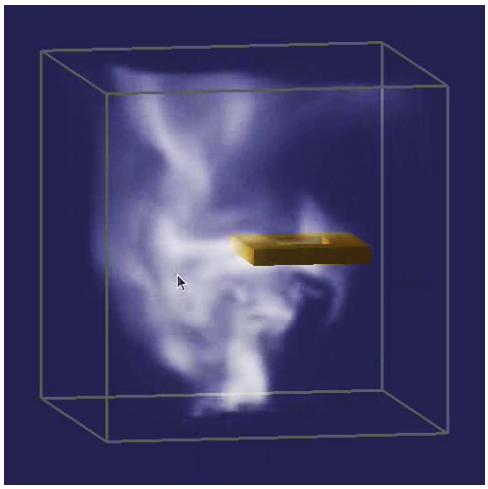

Figure 8: Real-time fluid-object interaction in a 64×64×64 cube.

Table 1 on the next page presents timing comparisons against two semi-Lagrangian fluid solvers, one on the CPU and the other on the GPU. For reference, our CPU comparison solver is faster than the the Maya 6.0 fluid solver on comparable hardware. Our GPU comparison solver performs 3D fluid simulation using 2D texture hardware, thus incurring a performance cost relative to pure 2D GPU solvers. This implementation is necessary, however, to compare against our *3D Boundaries* example. Texture size limits prevented GPU comparison against *Leaves*, although this reduced example did not use the GPU anyway. We note some subtleties in the data. In our *2D* and *3D Boundaries* examples we precomputed only the feedback matrices, calculating feedforward forces on the fly. Because volumetric density rendering is so slow ($\sim 75\%$ of each timestep in 3D), this allowed us to save memory at a negligible loss of frame rate, but increased our boundary costs. For the *Leaves* example, we precomputed all coupling forces, yielding much lower boundary costs. Note that *Leaves* is our fastest result despite the large number of voxels, and despite the completely CPU-based implementation. While much of the savings is due to using particles over densities, these timing results also underscore the voxel-independent time complexity of our fluid simulator.

Name	Snapshots	Velocities	Boundaries
2D Boundaries	8208	8	2
Forces	1024	11	0
3D Boundaries	400	10	113
Leaves	364	17	50

Table 2: **Precomputation time.** The "Velocities" column shows the time to compress the simulation basis and project the physics, while "Boundaries" shows the time to compress the boundary basis and compute reduced coupling forces for each object. All times are in hours. Note that these are *total* times which we reduced in practice by parallelizing over a 16 processor cluster.

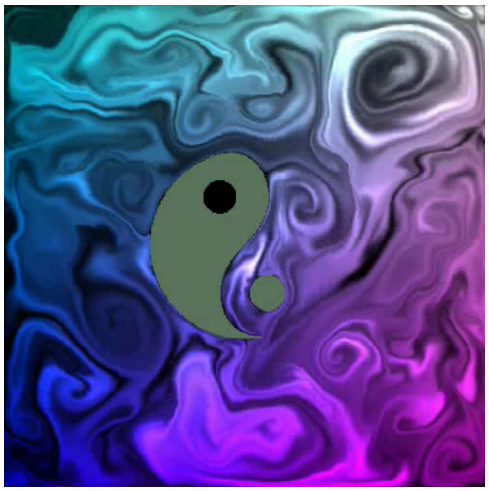

Figure 9: User-added forces on a 512×512 domain running at 41 frames/second.

Our results show that model reduction can preserve qualitatively important features such as turbulence (Figure 9). However, there are also obvious inaccuracies relative to full simulation. We now explore the sources of error. First, the *choice and number of basis functions* plays an important role. Figure 7 shows how error varies with the basis size. For comparison, we have also included three lower resolution full simulations. These results show that spatial downsampling rapidly yields error at least as high as model reduction. For example, downsampling from 128×128 to 96×96 produces roughly the same error as projecting onto only 32 basis states. Because these trials were run without external input, the reduced simulations adhered closely to the full simulation. Another important source of error, however, is *divergence from observed data*. For example, the user may present runtime inputs on which the system has not been trained. While this is difficult to measure, Figure 10 shows one such example: we trained a 128×128 simula-

Info				Full Runtime		Reduced Runtime				Speedup (Slowdown)	
Name	**Shape**	**Bases**	**Type**	**GPU**	**CPU**	**Velocities**	**Boundaries**	**Media**	**Total**	**vs. GPU**	**vs. CPU**
2D Boundaries	128×128	64	CPU/GPU	6.9	107	8.8	5.7	2.4	16.9	**(2)**	**6**
Forces	512×512	64	CPU/GPU	139	3595	8.8	0.0	14.5	23.3	**6**	**154**
3D Boundaries	64×64×64	64	CPU/GPU	521	3381	8.8	8.9	15.6	33.3	**16**	**102**
Leaves	256×64×128	32	CPU	–	26893	1.2	0.3	0.2	1.7	**–**	**15819**

Table 1: **Timing summary.** The "Type" column indicates the reduced simulation's processor architecture. The "Media" column indicates runtime for immersed media. No runtime is given for the *Leaves* GPU example, because the grids exceeded the texture size. Times are in ms.

tion with initial wind forces pointing at 0° and 90°. We then tested the reduced simulator with wind forces at other angles. As the wind forces move towards 45° the simulation remains visually plausible but numerical accuracy diminishes, as expected. Both Figures 7 and 10 measure error using the relative l^2 norm $||\mathbf{u}-\mathbf{u}^*||_2/||\mathbf{u}^*||_2$ against the "ground truth" $\mathbf{u}^*$, a full 128×128 simulation. *Approximations due to the boundary basis* represent another source of error. In this case we choose a different metric: the percentage of surface energy removed per timestep by our technique. Perfect boundary conditions (100% energy removal) would only be possible with full set of boundary bases and exact feedforward computation. In practice, we use a truncated set of bases, and precomputed feedforward matrices drawn from a finite set $\mathcal{P}$ of boundary positions. Figure 11 shows the effect of using progressively smaller sets of translational boundary positions (as measured in the distance between samples). When feedforward matrix samples are one voxel apart, we achieved $\sim 95\%$ surface energy removal per timestep for the c-shaped object in our *2D Boundaries* example. Finally, there are *global statistical measures of the performance*, such as energy preservation. As noted in Section 5.1, our system actually can preserve energy when the diffusion coefficient $\nu = 0$. Otherwise, we can precisely control the energy falloff (Figure 3).

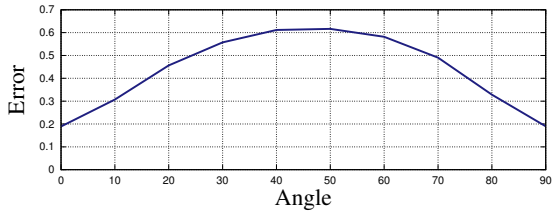

Figure 10: **Simulation Error for Unseen Inputs.** Observed force directions (0° and 90°) have lower error than unobserved force directions ($0° < \theta < 90°$).

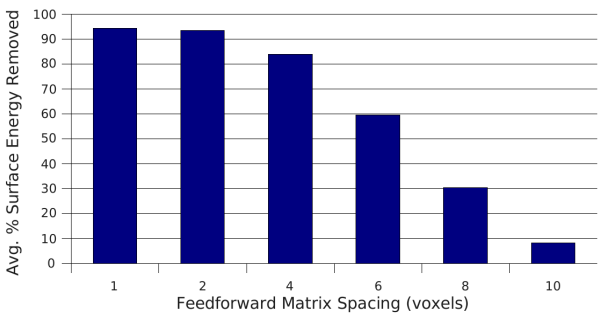

Figure 11: Using fewer precomputed feedforward matrices (spacing the samples further apart) yields lower boundary fidelity.

9 Discussion and Future Work

This paper presents a model reduction approach to real-time fluid simulation for interactive applications. We show that moving objects can be integrated into the flow by endowing them with their own reduced models. Moreover, the coupling forces between the individual models can be computed in the reduced space as well. Our treatment of moving objects overcomes one of the principle hurdles to adapting model reduction to interactive fluid simulation. In addition, our treatment of the reduced advection term allows us to precisely control the amount of energy dissipation in the simulation.

Within the gamut of fluid solvers, model reduction has a unique time complexity: the simulation runtime depends only on the basis size. Thus, complex phenomena can be observed on large grids and the number of variables is essentially proportional to the underlying degrees of freedom, such as the number of possible input forces. Grid resolution could be doubled, for example, without loss of speed. In practice, simulation and rendering are both dominated by the amount of immersed media. This makes model reduction particularly well suited to simulating immersed particles, where the velocity field need only be sparsely queried.

This speed, however, comes at the cost of accuracy, generality, and memory consumption. The system is unable to transfer into states not in the subspace, representing an inherent source of error particularly when the user presents inputs not in the training set. Moreover, because the basis states have global support, errors can have correspondingly global effect on the field. This is not to say that we are limited only to reproducing simulations in the training set. Our results demonstrate that the system is robust to general user interaction, and to new objects. However, we do not expect to see completely new flows. A related drawback is that our system must be trained before simulating. Changing the fixed boundary conditions, or creating new objects require lengthy precomputations. Once training is complete however, the system may be reused many times to produce simulation. As mentioned in the Section 7, the memory requirements of our algorithm are high. In fact, the grid size of our GPU-based 3D simulations was constrained by our GPU's 256 MB memory ceiling, and the cost of streaming grids to the GPU at each timestep prevents storing this data in CPU memory. Looking forward, however, we believe that the relative importance of model reduction will increase as memory costs decrease.

Another limitation of our technique is that the density simulation is computed in the full space. In principle, the density simulation could be reduced and the advection equation projected onto the joint reduced density/velocity space. However, issues related to density non-negativity and boundary conditions would have to be addressed, and it is questionable if a small number of modes could capture the required density configurations as well as they do the velocities. Moreover, as mentioned in Section 8, the runtime bottleneck is actually the volumetric density rendering, not the simulation. If precomputed radiance transfer (essentially, model-reduced rendering) could be applied to the density field, then density reduction would certainly become an attractive possibility. In the meantime, immersed particles have the most promising cost relationship with model reduction.

We believe there are many other exciting branches to be explored. New model reduction techniques such as nonlinear Galerkin projection and balanced truncation might offer richer simulations for a similar number of modes. We are particularly interested in exploring the coupling between separate reduced models similar to our treatment of boundaries. Such coupling

is crucial for generalizing model reduction to a wider set of phenomena, including compressible fluids, free-surface liquids, even particle systems such as hair and cloth. In general, we believe that model reduction offers an promising approach towards bringing the cinematic special effects which are a hallmark of computer graphics to the interactive space.

Acknowledgments. The authors would like to thank Jia-chi Wu for creating the video, Christopher Cameron for his insightful help on the exposition, and the anonymous reviewers for their comments. This work was supported by the UW Animation Research Labs, NSF grants EIA-0121326, CCR-0092970, IIS-0113007, an Alfred P. Sloan Fellowship, an NSF Graduate Research Fellowship, Electronic Arts, Sony, and Microsoft Research.

References

AUSSEUR, J., PINIER, J., GLAUSER, M., AND HIGUCHI, H. 2004. Predicting the Dynamics of the Flow over a NACA 4412 using POD. *APS Meeting Abstracts* (Nov.), D8.

BARBIČ, J., AND JAMES, D. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Transactions on Graphics 24*, 3 (Aug.), 982–990.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Transactions on Graphics 22*, 3 (July), 917–924.

COUPLET, M., BASDEVANT, C., AND SAGAUT, P. 2005. Calibrated reduced-order POD-Galerkin system for fluid flow modelling. *J. Comput. Phys. 207*, 1, 192–220.

ELCOTT, S., TONG, Y., KANSO, E., SCHRÖDER, P., AND DESBRUN, M. 2005. Stable, circulation-preserving, simplicial fluids. In *Discrete Differential Geometry*, Chapter 9 of Course Notes. ACM SIGGRAPH.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. *ACM Transactions on Graphics 24*, 3 (Aug.), 904–909.

FOGLEMAN, M., LUMLEY, J., REMPFER, D., AND HAWORTH, D. 2004. Application of the proper orthogonal decomposition to datasets of internal combustion engine flows. *Journal of Turbulence 5*, 23 (June).

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

GOODNIGHT, N., WOOLLEY, C., LUEBKE, D., AND HUMPHREYS, G. A. 2003. Multigrid solver for boundary value problems using programmable graphics hardware. In *Proceeding of Graphics Hardware*, 102.

HARRIS, M. J., COOMBE, G., SCHEUERMANN, T., AND LASTRA, A. 2002. Physically-based visual simulation on graphics hardware. In *Graphics Hardware 2002*, 109–118.

HOLMES, P., LUMLEY, J. L., AND BERKOOZ, G. 1996. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*. Cambridge University Press, Cambridge, MA.

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

KRÜGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics 22*, 3 (July), 908–916.

LI, W., WEI, X., AND KAUFMAN, A. 2003. Implementing lattice Boltzmann computation on graphics hardware. *The Visual Computer 19*, 7-8, 444–456.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics 23*, 3 (Aug.), 457–462.

LUMLEY, J. L. 1970. *Stochastic Tools in Turbulence*, vol. 12 of *Applied Mathematics and Mechanics*. Academic Press, New York.

MARION, M., AND TEMAM, R. 1989. Nonlinear Galerkin methods. *SIAM J. Numer. Anal. 26*, 5, 1139–1157.

MAUREL, S., BOREE, J., AND LUMLEY, J. 2001. Extended proper orthogonal decomposition: Application to jet/vortex interaction. *Flow, Turbulence and Combustion 67*, 2 (June), 125–36.

PARK, S. I., AND KIM, M. J. 2005. Vortex fluid for gaseous phenomena. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 261–270.

ROWLEY, C. W., AND MARSDEN, J. E. 2000. Reconstruction equations and the Karhunen-Loéve expansion for systems with symmetry. *Phys. D 142*, 1-2, 1–19.

ROWLEY, C. W., KEVREKIDIS, I. G., MARSDEN, J. E., AND LUST, K. 2003. Reduction and reconstruction for self-similar dynamical systems. *Nonlinearity 16* (July), 1257–1275.

ROWLEY, C., WILLIAMS, D., COLONIUS, T., MURRAY, R., AND MACMARTIN, D. 2006. Linear models for control of cavity flow oscillations. *J. Fluid Mech.* (Jan.).

SCHMIT, R., AND GLASUER, M. 2002. Low dimensional tools for flow-structure interaction problems: Application to micro air vehicles. *APS Meeting Abstracts* (Nov.), D1+.

SELLE, A., RASMUSSEN, N., AND FEDKIW, R. 2005. A vortex particle method for smoke, water and explosions. *ACM Transactions on Graphics 24*, 3 (Aug.), 910–914.

SIRISUP, S., AND KARNIADAKIS, G. E. 2004. A spectral viscosity method for correcting the long-term behavior of POD models. *J. Comput. Phys. 194*, 1, 92–116.

SIROVICH, L. 1987. Turbulence and the dynamics of coherent structures. I - Coherent structures. II - Symmetries and transformations. III - Dynamics and scaling. *Quarterly of Applied Mathematics 45* (Oct.), 561–571.

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

STAM, J. 2001. A simple fluid solver based on the fft. *Journal of graphics tools 6*, 2, 43–52.

WANG, H., WU, Q., SHI, L., YU, Y., AND AHUJA, N. 2005. Out-of-core tensor approximation of multi-dimensional matrices of visual data. *ACM Transactions on Graphics 24*, 3 (Aug.), 527–535.

WU, E., LIU, Y., AND LIU, X. 2005. An improved study of real-time fluid simulation on GPU. *Computer Animation and Virtual Worlds 15*, 3-4, 139–146.