To appear in ACM TOG 32(4).

Non-Polynomial Galerkin Projection on Deforming Meshes

Matt Stanton[1] Yu Sheng[1] Martin Wicke[2] Federico Perazzi[1] Amos Yuen[1] Srinivasa Narasimhan[1] Adrien Treuille[1]

[1]Carnegie Mellon University [2]Otherlab

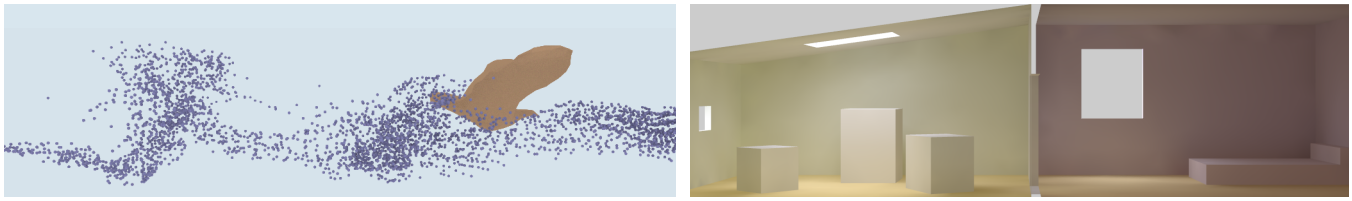


Figure 1: *Our method enables reduced simulation of fluid flow around this flying bird over 2000 times faster than the corresponding full simulation and reduced radiosity computation in this architectural scene over 113 times faster than the corresponding full radiosity.*

Abstract

This paper extends Galerkin projection to a large class of non-polynomial functions typically encountered in graphics. We demonstrate the broad applicability of our approach by applying it to two strikingly different problems: fluid simulation and radiosity rendering, both using deforming meshes. Standard Galerkin projection cannot efficiently approximate these phenomena. Our approach, by contrast, enables the compact representation and approximation of these complex non-polynomial systems, including quotients and roots of polynomials. We rely on representing each function to be model-reduced as a composition of tensor products, matrix inversions, and matrix roots. Once a function has been represented in this form, it can be easily model-reduced, and its reduced form can be evaluated with time and memory costs dependent only on the dimension of the reduced space.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation, Radiosity; I.6.8 [Simulation and Modeling]: Types of Simulation—Animation;

Keywords: reduced models, fluid simulation, solid-fluid coupling, radiosity

1 Introduction

Galerkin projection has enabled astonishing speedups in graphics. However, despite a breadth of applications – everything from global illumination [Sloan et al. 2002] to fluids [Treuille et al. 2006] – a key limitation of this approach is that the underlying phenomena must be *polynomial*, a restriction which limits its applicability within computer graphics.

This paper proposes an efficient extension of Galerkin projection to any function composed of *elementary algebraic operations* – the four operations of arithmetic plus rational roots – thus expanding the applicability of this model reduction approach across graphics. To demonstrate the broad applicability of our method, we apply it to two strikingly different problems: radiosity rendering and fluid simulation. Although both of these phenomena can be expressed in polynomial form on a fixed mesh, we show that allowing geometric deformation requires non-polynomial operations to express changes to the dynamics and appearance. Applying standard Galerkin projection to such functions is theoretically possible, but does not yield any runtime speed improvement. Our technique, by contrast, can efficiently model these complex non-polynomial systems. Similar to standard Galerkin projection, our approach not only preserves key optimality guarantees, but also generates a compact, analytic model in the reduced space.

Our method has a wide range of applications. Interactive environments, such as video games and architectural design applications, increasingly incorporate physical simulation. Our technique could add fluid effects around rigged objects including characters and animals. We could also compute real-time radiosity for scenes containing these elements, correctly accounting for their motion, deformation, and appearance. More generally, this paper is the first demonstration of real-time Galerkin projection for non-polynomial systems, potentially opening up interactive simulation to a wide range of new phenomena.

2 Related Work

Galerkin projection has been used in interactive graphics to speed up both linear and nonlinear deformation [Pentland and Williams 1989; James and Pai 2002; Hauser et al. 2003; Barbič and James 2005], sound [James et al. 2006], rendering [Sloan et al. 2002; Sloan et al. 2005], and fluids [Treuille et al. 2006; Gupta and Narasimhan 2007; Barbič and Popović 2008; Wicke et al. 2009]. A common thread running through these applications is that the governing equations are polynomial. Our method builds upon these previous efforts by extending Galerkin projection to non-polynomial functions, which cover a broader range of phenomena.

Model reduction of non-polynomial functions. In numerical analysis, Galerkin projection has primarily been used for linear and *rational functions*. Rational Krylov methods approximate a rational transfer function in the frequency domain using moment matching [Olssen 2005; Ebert and Stykel 2007; Gugercin et al. 2006] to find good bases for linear time-invariant (LTI) systems and extensions thereof. Alternatives include rational function fitting (also known as multipoint methods) [Liua et al. 2008; Gallivan et al. 1996; Grimme 1997] and balanced truncation [Li 2000; Zhou 2002; Gugercin and Antoulas 2004]. The key difference between these works and ours is that their goal is to reduce linear, time-invariant systems (LTIs) by analyzing their rational transfer functions, whereas we are interested in the reduction of non-polynomial functions for their own sake.

Because of the restrictive nature of LTIs, extensions have been proposed for time-varying systems, both linear [Phillips 1998; Sandberg and Rantzer 2004; Chahlaoui and van Dooren 2005; Hossain and Benner 2008] and multilinear [Savas and Eldén 2009]. The

work closest to ours is Farhood and Dullerud [2007], who apply rational Krylov methods to linear systems rationally dependent on time-varying parameters. However, unlike our work, theirs does not guarantee preservation of polynomial degree for arbitrary compositions of elementary algebraic operations and therefore can become computationally intractable for complex phenomena. Using an algebraic approach similar to that of Debusschere et al. [2004] for probabilistic dynamics, we alter Galerkin projection to enable order-preserving reduction for arbitrary compositions of elementary algebraic operations. To our knowledge, ours is the first work to present a general framework for Galerkin projection of arbitrary compositions of elementary algebraic operations while preserving polynomial degree, an essential property for real-time graphics, and the first work to simulate fluid flow or radiosity on deforming meshes.

Techniques other than Galerkin projection can model-reduce non-polynomial systems. An et al. [2008] demonstrated model reduction of nonlinear elastic dynamics using cubature, which approximates nonlinear functions in low dimension by sampling the original functions and fitting a model to the samples. This method was also used to reduce thin shell dynamics by Chadwick et al. [2009], and can be extended to support online updating of the basis [Kim and James 2009] and domain decomposition [Kim and James 2011]. Kim et al. [2013] also applied this technique to fluid simulation. Similar to our work, this work also model reduces inverse matrices and builds separate bases for each step of the dynamics. In general, the accuracy of cubature depends on selecting good bases and a good set of points at which to sample the simulation dynamics, unlike our work, where the accuracy depends only on the choice of bases.

Reduced-order fluid models. Previous reduced fluid models in graphics and numerical analysis suffered from highly restrictive boundary conditions. A straightforward solution is to construct separate bases for each possible boundary configuration [Schmit and Glasuer 2002]. However, this approach cannot scale to continuously deforming boundaries. For flows with periodic boundaries, and with inherent symmetries within the flow dynamics, it is possible to remove uniform translation modes [Rowley et al. 2003; Rowley and Marsden 2000]. For a single rotating object, the basis can be constructed in the object's frame of reference and then simulated at various angles [Ausseur et al. 2004]. Treuille et al. [2006] enabled the insertion of rigidly moving boundaries, and Wicke et al. [2009] allowed discrete boundary reconfiguration at runtime. Perhaps the work closest to ours is that of Fogleman et al. [2004], which enabled linear deformation along a single axis to model reduce piston and combustion simulation.

We enable continuous boundary motion by embedding the fluid in a tetrahedral mesh, similar to Elcott et al. [2007], and deforming the mesh along with the boundary. This requires a full-dimensional fluid solver that works for tetrahedral meshes. We considered several classes of solvers including finite element methods (e. g. [Feldman et al. 2005a; Feldman et al. 2005b]), methods based on discrete exterior calculus [Mullen et al. 2009; Pavlov et al. 2011], and ALE methods (such as [Klingner et al. 2006]). We chose the residual distribution scheme [Sewall et al. 2007; Dobes and Deconinck 2006; Deconinck and Ricchiuto 2007], which is akin to a finite-difference fluid approximation [Foster and Metaxas 1996], but generalized to a tetrahedral mesh, due to its amenability to our non-polynomial Galerkin projection and the fact that it can be stably integrated in the reduced space.

Reduced-order global illumination. Existing reduced-order global illumination techniques, such as Precomputed Radiance Transfer [Sloan et al. 2002], cannot accurately model the effect of general continuous large-scale scene deformations on nonlocal radiance transfer. James and Fatahalian [2003] allowed for deformable objects, but only for a certain set of physics-based deformations. Sloan et al. [2005] accounted for changes in local light transport arising from a more general set of deformations. More recent models allow discrete scenes to be composed without requiring additional precomputation [Loos et al. 2011; Loos et al. 2012], but do not allow for general continuous change in shape. In this paper, we focus on simulating low-frequency lighting effects and use radiosity [Goral et al. 1984] as the global illumination algorithm. Numerous methods [Hanrahan et al. 1991; Drettakis and Sillion 1997] have been proposed to accelerate the computational speed of radiosity for dynamic scenes. However, most previous methods are limited to rigid transformations, such as inserting, deleting, and moving objects in the scene. Realtime computation of radiosity for deformable scenes remains a challenge. (We remark that [Zatz 1993] uses the term "Galerkin" in a different sense to describe the approximation of curved geometry using flat patches; this is unrelated to our goal of accelerating standard radiosity).

3 Polynomial Galerkin Projection

Many areas of graphics require the time-consuming computation of high-dimensional functions, such as the light transport and fluid equations. At its core, model reduction is about approximating these functions with fewer dimensions.

Suppose we wish to evaluate a function $\mathbf{y} = \mathbf{f}(\mathbf{x})$ where the input $\mathbf{x} \in \mathbb{R}^n$ and output $\mathbf{y} \in \mathbb{R}^m$ are very high-dimensional. We seek a reduced approximation $\hat{\mathbf{y}} = \hat{\mathbf{q}}(\hat{\mathbf{x}})$, where the reduced input $\hat{\mathbf{x}} \in \mathbb{R}^{\hat{n}}$ and output $\hat{\mathbf{y}} \in \mathbb{R}^{\hat{m}}$ lie in much lower-dimensional spaces: $\hat{n} \ll n$ and $\hat{m} \ll m$. The first step is to linearly *dimension-reduce* the state vectors, which means finding a pair of orthonormal bases $\mathbf{B}_x, \mathbf{B}_y$ which convert reduced vectors to full vectors: $\mathbf{x} = \mathbf{B}_x\hat{\mathbf{x}}$ and $\mathbf{y} = \mathbf{B}_y\hat{\mathbf{y}}$. We can project from the full to the reduced space through multiplication by the transpose: $\hat{\mathbf{x}} = \mathbf{B}_x^T\mathbf{x}$, and $\hat{\mathbf{y}} = \mathbf{B}_y^T\mathbf{y}$. The second step is to *model-reduce* the transformation $\mathbf{f}$, which means finding an efficient reduced approximation $\hat{\mathbf{q}} : \mathbb{R}^{\hat{n}} \to \mathbb{R}^{\hat{m}}$ operating entirely in the reduced space. The standard approach is *Galerkin projection*, which works well if $\mathbf{f}$ is polynomial but can be inefficient otherwise.

The following sections present our technique to efficiently model-reduce equations of the form

$$\mathbf{y} = \mathbf{f}(\mathbf{x})$$

where $\mathbf{f}(\mathbf{x})$ can be written using only the four basic arithmetic operations and fractional powers. We begin by describing standard Galerkin projection, which allows us to model-reduce polynomials, and forms the foundation of the rest of our approach. We then describe how this technique can be combined with a small number of basic matrix operations in order to reduce a wide range of non-polynomial functions. Sections 5-9 apply this method to the simulation of radiosity and fluids.

Notation. In the remainder of the paper, scalars appear in lower case: x, vectors in bold lower case: $\mathbf{x}$, and matrices and tensors in bold upper case: $\mathbf{X}$. We write $\mathbf{Q} \otimes_a \mathbf{M}$ to denote tensor multiplication of the tensor $\mathbf{Q}$ by the matrix or vector $\mathbf{M}$ along the axis with index a, and $\mathbf{Q} \otimes_{a\ldots b} \mathbf{M}$ to denote the repeated tensor product $\mathbf{Q} \otimes_a \mathbf{M} \otimes_{a+1} \mathbf{M} \ldots \otimes_b \mathbf{M}$. We number tensor axes starting from 0. For clarity, we sometimes employ the matrix notation $\mathbf{B}^T\mathbf{Q}$ to denote multiplication along the 0th axis, $\mathbf{Q} \otimes_0 \mathbf{B}$, and the notation $\mathbf{QB}$ to denote multiplication along the 1st axis, $\mathbf{Q} \otimes_1 \mathbf{B}$. We refer to multiplication of a tensor by a vector as *tensor contraction*, an operation which reduces the tensor order by one.

A degree-d polynomial $\mathbf{q}(\mathbf{x})$ can be represented as a $d+1$th-order

tensor $\mathbf{Q}$. For example, if some polynomial $\mathbf{p}(\mathbf{x})$ has degree 3 and $\mathbf{x}$ is of length n, then we can write the components of $\mathbf{p}(\mathbf{x})$ as

$$p_i(\mathbf{x}) = \sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{\ell=1}^{n} c_{ijk\ell} x_j x_k x_\ell$$

where $c_{ijk\ell}$ are the coefficients of the polynomial $\mathbf{p}(\mathbf{x})$. The 4th-order tensor $\mathbf{P}$ simply consists of the polynomial coefficients: $P_{ijk\ell} = c_{ijk\ell}$, and we can write $\mathbf{p}(\mathbf{x}) = \mathbf{P} \otimes_{1\ldots3} \mathbf{x}$. Likewise, we can evaluate the degree-d polynomial $\mathbf{q}(\mathbf{x})$ by contracting its corresponding $d+1$th-order tensor $\mathbf{Q}$:

$$\mathbf{q}(\mathbf{x}) = \mathbf{Q} \otimes_{1\ldots d} \mathbf{x}.$$

To compute the Galerkin projection of the polynomial

$$\mathbf{y} = \mathbf{Q} \otimes_{1\ldots d} \mathbf{x}$$

we start by substituting the reduced variables:

$$\mathbf{B}_y \hat{\mathbf{y}} \approx \mathbf{Q} \otimes_{1\ldots d} \mathbf{B}_x \mathbf{x}$$

(We write $\approx$ to remind ourselves that $\hat{\mathbf{y}}$ has too few degrees of freedom to ensure that this equation has an exact solution.) We then multiply by $\mathbf{B}_y^T$:

$$\hat{\mathbf{y}} = \mathbf{B}_y^T \left(\mathbf{Q} \otimes_{1\ldots d} \mathbf{B}_x \mathbf{x}\right). \tag{1}$$

We can compute the value of $\hat{\mathbf{y}}$ at runtime quickly as $\hat{\mathbf{y}} = \hat{\mathbf{Q}} \otimes_{1\ldots d} \hat{\mathbf{x}}$, where

$$\hat{\mathbf{Q}} = \mathbf{B}_y^T \mathbf{Q} \otimes_{1\ldots d} \mathbf{B}_x, \tag{2}$$

and the product $\mathbf{B}_y^T \mathbf{Q}$ denotes the product of $\mathbf{Q}$ with $\mathbf{B}_y$ along the tensor axis corresponding to the result vector $\hat{\mathbf{y}}$. $\hat{\mathbf{Q}}$ is called the *Galerkin projection* of $\mathbf{Q}$. Intuitively, $\hat{\mathbf{Q}}$ transforms the reduced input $\hat{\mathbf{x}}$ into the full space using $\mathbf{B}_x$, applies $\mathbf{Q}$, then projects back into the reduced space using $\mathbf{B}_y^T$. The final projection incurs some accuracy cost, but it unambiguously specifies $\hat{\mathbf{y}}$, so we can write Eq. 2 with an equals sign. The projection is fast: evaluating $\hat{\mathbf{Q}}$ takes the same number of contractions as evaluating $\mathbf{Q}$, although each contraction is now with a vector of length $\hat{n}$ instead of length n.

4 Non-Polynomial Galerkin Projection

It is more difficult to see how we can efficiently apply Galerkin projection to non-polynomial functions. For example, consider the rational function $\mathbf{y} = \mathbf{f}(\mathbf{x})$, where $\mathbf{x} = [x_1, x_2]^T$, $\mathbf{y} = [y_1, y_2]^T$, and:

$$y_1 = \frac{x_1 x_2 + x_2^2}{x_1^2} \qquad y_2 = \frac{x_1^2}{x_2^2}. \tag{3}$$

We could compute the Galerkin projection of this function by transforming reduced vectors into the full space, applying the full space equations, and projecting the results back to the reduced space, but this would yield no speed advantage. Instead, we use two tensors to express $\mathbf{f}(\mathbf{x})$, rewriting it as a matrix-vector product:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1^2 & 0 \\ 0 & x_2^2 \end{bmatrix}^{-1} \begin{bmatrix} x_1 x_2 + x_2^2 \\ x_1^2 \end{bmatrix}$$

Both the matrix and the vector are polynomial (specifically, quadratic) in $\mathbf{x}$. Therefore we can evaluate the matrix using a 4th-order tensor $\mathbf{Q}_1$, and the vector using an 3rd-order tensor $\mathbf{Q}_2$:

$$\begin{bmatrix} x_1^2 & 0 \\ 0 & x_2^2 \end{bmatrix} = \mathbf{Q}_1 \otimes_2 \mathbf{x} \otimes_3 \mathbf{x}$$

$$\begin{bmatrix} x_1 x_2 + x_2^2 \\ x_1^2 \end{bmatrix} = \mathbf{Q}_2 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x}$$

where $\mathbf{Q}_1$ and $\mathbf{Q}_2$ are (labeling each tensor slice by its associated polynomial term):

$$\mathbf{Q}_1 \otimes_2 \mathbf{x} \otimes_3 \mathbf{x} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} x_1^2 + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} x_1 x_2 + \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} x_2 x_1 + \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} x_2^2$$

and

$$\mathbf{Q}_2 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} x_1^2 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_1 x_2 + \begin{bmatrix} 0 \\ 0 \end{bmatrix} x_2 x_1 + \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_2^2$$

To evaluate $\mathbf{f}(\mathbf{x})$, we contract $\mathbf{Q}_1$ and $\mathbf{Q}_2$ as shown above, then invert the matrix $\mathbf{Q}_1 \otimes_2 \mathbf{x} \otimes_3 \mathbf{x}$, and finally compute the matrix-vector product. In the form of a single equation:

$$\mathbf{y} = \left(\mathbf{Q}_1 \otimes_2 \mathbf{x} \otimes_3 \mathbf{x}\right)^{-1} \otimes_1 \left(\mathbf{Q}_2 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x}\right) \tag{4}$$

More generally, we reduce a non-polynomial function $\mathbf{f}(\mathbf{x})$ by writing $\mathbf{f}(\mathbf{x})$ in terms of a collection of tensors $\mathbf{Q}_1 \ldots \mathbf{Q}_k$, to which we apply a series of tensor contractions and matrix operations. We can reduce functions expressible using only the following operations:

	Name	Full Form
(i)	Tensor Product	$\mathbf{Q} \otimes \mathbf{X}$
(ii)	Matrix Inverse	$\mathbf{Q}^{-1}$
(iii)	Matrix Root	$\mathbf{Q}^{\frac{1}{n}}$

where $\mathbf{Q}$ is a matrix in (ii) and (iii). In (iii), $\mathbf{Q}$ is additionally symmetric and positive semidefinite, and $\mathbf{Q}^{\frac{1}{n}}$ is uniquely identified as the positive semidefinite matrix such that $(\mathbf{Q}^{\frac{1}{n}})^n = \mathbf{Q}$.

As we saw in §3, polynomial functions require only one tensor and repeated applications of operation (i) for evaluation. However, as we showed in Eq. 3, there are many cases in which it may be necessary to use multiple tensors and operations beyond (i) to evaluate $\mathbf{f}(\mathbf{x})$.

Once we have expressed a function $\mathbf{f}(\mathbf{x})$ in terms of tensors $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ and our allowed matrix operations, we can reduce it. This requires that we first find bases for every axis of every tensor $\mathbf{Q}_i$. For example, for a polynomial $\mathbf{y} = \mathbf{q}_i(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ we require *separate* bases for $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_d$. Once we have found these bases, we can *pre-multiply* each tensor by its associated bases, as we did in Eq. 2. Computing $\hat{\mathbf{f}}(\hat{\mathbf{x}})$ then follows *exactly* the sequence of operations used to compute $\mathbf{f}(\mathbf{x})$, except that we replace each tensor $\mathbf{Q}_i$ with its reduced counterpart $\hat{\mathbf{Q}}_i$. That is, each operation transforms as follows:

	Full Form	Reduced Form
(i)	$\mathbf{Q} \otimes_k \mathbf{X}$	$\hat{\mathbf{Q}} \otimes_k \hat{\mathbf{X}} = \mathbf{B}_q^T \mathbf{Q} \otimes_k \mathbf{B}_x \hat{\mathbf{X}}$
(ii)	$\mathbf{Q}^{-1}$	$\hat{\mathbf{Q}}^{-1} = (\mathbf{B}_q^T \mathbf{Q} \mathbf{B}_q)^{-1}$
(iii)	$\mathbf{Q}^{\frac{1}{n}}$	$\hat{\mathbf{Q}}^{\frac{1}{n}} = (\mathbf{B}_q^T \mathbf{Q} \mathbf{B}_q)^{\frac{1}{n}}$

While each reduction is straightforward, these reductions do have implications for basis selection, as shown in the last column of the table. Since we need to reduce every tensor along all of its axes, we need a basis for each axis of each tensor. Operations (ii) and (iii) require that $\mathbf{Q}$ be reduced using the same basis along both axes. §4.2 explains this restriction. $\hat{\mathbf{Q}}$ is typically small enough that we can efficiently compute $\hat{\mathbf{Q}}^{\frac{1}{n}}$ using eigen-decomposition.

4.1 Reduction Example

To reduce our example function (Eq. 3) from the previous section, we inspect the tensor form (Eq. 4) to see which bases we will need. Reading from right to left, it appears that we require a basis $\mathbf{B}_x$ for $\mathbf{x}$, a basis $\mathbf{B}_n$ for $\mathbf{Q}_2 \otimes_1 \mathbf{x} \otimes_2 \mathbf{x}$, and a basis $\mathbf{B}_y$ for $\mathbf{y}$. Due to the restriction that an inverted matrix must be reduced by the same basis along both axes, $\mathbf{B}_y = \mathbf{B}_n$, and we will refer to both as $\mathbf{B}_y$. The reduction of $\mathbf{Q}_1$ and $\mathbf{Q}_2$ is then:

$$\begin{aligned}\hat{\mathbf{Q}}_1 &= \mathbf{B}_y^T \mathbf{Q}_1 \otimes_1 \mathbf{B}_y \otimes_2 \mathbf{B}_x \otimes_3 \mathbf{B}_x \\ \hat{\mathbf{Q}}_2 &= \mathbf{B}_y^T \mathbf{Q}_2 \otimes_1 \mathbf{B}_x \otimes_2 \mathbf{B}_x\end{aligned}$$

and to evaluate $\hat{\mathbf{f}}(\hat{\mathbf{x}})$ at runtime we perform:

$$\hat{\mathbf{y}} = (\hat{\mathbf{Q}}_1 \otimes_2 \hat{\mathbf{x}} \otimes_3 \hat{\mathbf{x}})^{-1} \otimes_1 (\hat{\mathbf{Q}}_2 \otimes_1 \hat{\mathbf{x}} \otimes_2 \hat{\mathbf{x}})$$

The only data we require to perform this computation are the two reduced tensors $\hat{\mathbf{Q}}_1$ and $\hat{\mathbf{Q}}_2$. If we wish to display the full space result of the computation at runtime, then we will we will also need $\mathbf{B}_y$ to compute $\mathbf{y} = \mathbf{B}_y \hat{\mathbf{y}}$.

4.2 Properties

Our choice of reduction is not the only one we could have used for non-polynomial functions. We could have chosen to replace each full-space operation with something more complex than the same operation performed on reduced tensors. Our method does, however, have three principal advantages. First, it is *simple*: once a function has been expressed in terms of tensors, the reduction consists only in replacing full-dimensional tensors with corresponding reduced-dimensional tensors. Second, it is *efficient*: tensor order is preserved by the reduction, which is important because both tensor storage costs and evaluation time complexity are exponential in the tensor order. Third, it is *optimal*: each reduction rule is constructed to minimize some measure of error (similar to [Carlberg et al. 2011]), given a particular decomposition into tensors of the target function. We discuss the optimality properties of each of our operations in turn.

Tensor product. If we have a single tensor $\mathbf{Q}$ of order $d+1$ which we contract d times by a fixed vector $\mathbf{x}$: $\mathbf{y} = \mathbf{Q} \otimes_{1\ldots d} \mathbf{x}$, then $\min_{\hat{\mathbf{y}}} ||\mathbf{B}_y \hat{\mathbf{y}} - \mathbf{Q} \otimes_{1\ldots d} \mathbf{B}_x \hat{\mathbf{x}}||$ is minimized at $\hat{\mathbf{y}} = \mathbf{B}_y^T \mathbf{Q} \otimes_{1\ldots d} \mathbf{x}$, exactly as in Eq. 1.

Matrix inverse. For the matrix inverse, operation (ii), let $\mathbf{y} = \mathbf{Q}^{-1}\mathbf{x}$. $\min_{\hat{\mathbf{y}}} ||\mathbf{B}_x \hat{\mathbf{y}} - \mathbf{Q}^{-1}\mathbf{B}_x \hat{\mathbf{x}}||_{\mathbf{Q}}$ is minimized at $\hat{\mathbf{y}} = (\mathbf{B}_x^T \mathbf{Q} \mathbf{B}_y)^{-1}$. Note that this is not the same as the ordinary Galerkin projection of $\mathbf{Q}^{-1}$, which would be $\mathbf{B}_x^T \mathbf{Q}^{-1} \mathbf{B}_x$. We could also allow the bases for $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ to differ, in which case the error would be minimized at $\hat{\mathbf{y}} = (\mathbf{B}_x^T \mathbf{Q} \mathbf{B}_y)^{-1} \mathbf{B}_y^T \mathbf{B}_x$. However, we have found that setting $\mathbf{B}_x = \mathbf{B}_y$ has significant benefits in practice, such as energy conservation in our fluids application (§A.2).

Matrix root. Matrix roots, operation (iii), approximate $\mathbf{Q}^{\frac{1}{n}}$ by first finding the reduced matrix $\hat{\mathbf{Q}}$ that best approximates $\left(\mathbf{Q}^{\frac{1}{n}}\right)^n = \mathbf{Q}$, computing $\min_{\hat{\mathbf{Q}}} ||\mathbf{B}_x \hat{\mathbf{Q}} \mathbf{x} - \mathbf{Q} \mathbf{B}_x \mathbf{x}||$. This reduced matrix is $\hat{\mathbf{Q}} = \mathbf{B}_x^T \mathbf{Q} \mathbf{B}_x$. We then use its nth root, $\hat{\mathbf{Q}}^{\frac{1}{n}}$, to approximate $\mathbf{Q}^{\frac{1}{n}}$. Multiplying $\mathbf{Q}$ by the same basis along both axes ensures that $\hat{\mathbf{Q}}$ is also symmetric and positive semidefinite. Again, our reduction differs from the ordinary Galerkin projection of $\mathbf{Q}^{\frac{1}{n}}$, which is $\mathbf{B}_x^T \mathbf{Q}^{\frac{1}{n}} \mathbf{B}_x$.

Speed-optimality tradeoff. In our optimality discussion for operation (i), we restricted ourselves to the case where we represent the function $\mathbf{f}(\mathbf{x})$ using a single tensor. However, in many cases we can choose to represent a polynomial using multiple tensors. This allows us to exchange optimality for speed by composing polynomials. Suppose $\mathbf{q}(\mathbf{x}) = \mathbf{q}_1(\mathbf{q}_2(\mathbf{x}))$, where $\mathbf{q}$ has degree $d = ab$, $\mathbf{q}_1$ has degree a, and $\mathbf{q}_2$ has degree b. We can choose to express $\mathbf{q}(\mathbf{x})$ as either $\mathbf{Q} \otimes_{1\ldots d} \mathbf{x}$ or $\mathbf{Q}_1 \otimes_{1\ldots a} (\mathbf{Q}_2 \otimes_{1\ldots b} \mathbf{x})$. In the full space, these expressions are identical. When reduced, however, these expressions become $(\mathbf{B}_y^T \mathbf{Q} \otimes_{1\ldots d} \mathbf{B}_x) \otimes_{1\ldots d} \hat{\mathbf{x}}$ and $(\mathbf{B}_y^T \mathbf{Q}_1 \otimes_{1\ldots a} \mathbf{B}_z) \otimes_{1\ldots a} ((\mathbf{B}_z^T \mathbf{Q}_2 \otimes_{1\ldots b} \hat{\mathbf{x}})$. We can also write the second case as:

$$\mathbf{B}_y^T \mathbf{Q}_1 \otimes_{1\ldots e} (\mathbf{B}_z \mathbf{B}_z^T \mathbf{Q}_2 \otimes_{1\ldots g} \mathbf{B}_x \hat{\mathbf{x}})$$

which is equivalent to:

$$\mathbf{B}_x^T \mathbf{q}_1(\mathbf{B}_z \mathbf{B}_z^T \mathbf{q}_2(\mathbf{B}_x \hat{\mathbf{x}})).$$

Notice that the composition introduces an extra projection $\mathbf{B}_z \mathbf{B}_z^T$, which reduces the accuracy of the reduced result. On the other hand, the reduced composition is faster to compute than the reduced original polynomial: the composition replaces one reduced tensor containing $\hat{n}^{eg+1}$ elements with two much smaller reduced tensors, one containing $\hat{n}^{e+1}$ elements, and the other containing $\hat{n}^{g+1}$.

4.3 Summary

In this section, we have demonstrated that any function constructable from tensor contraction, matrix inversion, and matrix roots can be easily model-reduced using our non-polynomial Galerkin projection method. To reduce such a function $\mathbf{f}(\mathbf{x})$, we construct it using a collection of tensors $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$, to which we apply our tensor and matrix operations. The reduced counterpart $\hat{\mathbf{f}}(\hat{\mathbf{x}})$ can then be computed by simply replacing the tensors $\mathbf{Q}_1, \ldots, \mathbf{Q}_k$ with the tensors $\hat{\mathbf{Q}}_1, \ldots, \hat{\mathbf{Q}}_k$, and maintaining exactly the same sequence of operations. Our method is simple, efficient, and optimal in the sense described in §4.2.

In the remainder of the paper, we demonstrate applying this technique to two different problems: fluid flow around deforming objects, and global illumination in the presence of deforming objects.

5 Fluids

As a first example of our non-polynomial Galerkin projection method, we describe the reduction of fluid flow on a deforming tetrahedral mesh. In this system, movement of the mesh exerts force on the fluid, but forces from fluid motion do not cause the mesh to move. The simulation state consists of fluid velocities $\mathbf{u}$ and fluid momenta $\mathbf{p}$, located at the centroid of each element, and of fluid fluxes $\mathbf{f}$ through each face. (Ordinarily, one would use only one of these descriptions of fluid flow. As we shall see however, non-polynomial Galerkin projection demands that we treat these quantities separately.) We require that the mesh topology remains constant, but allow continuous deformation of the positions of the mesh vertices denoted $\mathbf{g}$ (Fig. 3).

We begin from the incompressible Navier-Stokes momentum equation:

$$\dot{\mathbf{u}} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \nu \nabla^2 \mathbf{u} + \nabla p + \mathbf{e}, \tag{5}$$

where p denotes pressure, ν viscosity, and $\mathbf{e}$ external forces. We discretize this equation using the residual distribution scheme of Dobes, Deconinck, and Ricchiuto [2006; 2007], which is essentially a finite-differencing method applied to tetrahedral meshes. We use operator splitting [Stam 1999] to divide the velocity update

	begin fullSimStep($\mathbf{u}_t$, $\mathbf{f}_t$, $\mathbf{g}_t$, $\mathbf{g}_{t+1}$):	**begin** reducedSimStep($\hat{\mathbf{u}}_t$, $\hat{\mathbf{f}}_t$, $\hat{\mathbf{g}}_t$, $\hat{\mathbf{g}}_{t+1}$)
flux combination	$\dot{\mathbf{g}} \leftarrow \mathbf{g}_{t+1} - \mathbf{g}_t$ $\mathbf{f}' \leftarrow \mathbf{f}_t - \mathbf{h}(\dot{\mathbf{g}})$	$\hat{\mathbf{f}}' \leftarrow \hat{\mathbf{f}}_t - \mathbf{B}_f^T \mathbf{B}_h \hat{\mathbf{h}}_t$
advection (§5.1) *diffusion (§5.2)*	$\mathbf{u}' \leftarrow \mathbf{u}_t + \int_t^{t+1} [\mathbf{V}^{-1}(\mathbf{g}_t)(\mathbf{A} \otimes_2 \mathbf{f}' - \mu\Delta)]\mathbf{u}_t$	$\hat{\mathbf{u}}' \leftarrow \exp[\Delta t \hat{\mathbf{V}}^{-1}(\hat{\mathbf{g}}_t)(\hat{\mathbf{A}} \otimes_2 \hat{\mathbf{f}}' - \mu\hat{\Delta})]\hat{\mathbf{u}}_t$
pressure projection (§5.3)	$\mathbf{f}_{t+1} \leftarrow \min \|\mathbf{u}' - \mathbf{V}^{-1}(\mathbf{g}_{t+1})(\mathbf{P} \otimes_2 \mathbf{g}_{t+1})\mathbf{f}\|_{\mathbf{V}(\mathbf{g}_{t+1})}$ s.t. $\mathbf{Df} - \mathbf{Dh}(\dot{\mathbf{g}}) = 0$ and $\nabla \cdot \mathbf{f} = 0$	$\hat{\mathbf{f}}_{t+1} \leftarrow \min_{\hat{\mathbf{f}}} \|\hat{\mathbf{u}}' - \hat{\mathbf{V}}^{-1}(\hat{\mathbf{g}}_{t+1})(\hat{\mathbf{P}} \otimes_2 \hat{\mathbf{g}}_{t+1})\hat{\mathbf{f}}\|_{\hat{\mathbf{V}}(\hat{\mathbf{g}}_{t+1})}$ s.t. $\hat{\mathbf{D}}\hat{\mathbf{f}} - \hat{\mathbf{D}}_h \hat{\mathbf{h}}_t = 0$
convert to velocity	$\mathbf{u}_{t+1} \leftarrow \mathbf{V}^{-1}(\mathbf{g}_{t+1})(\mathbf{P} \otimes_2 \mathbf{g}_{t+1})\mathbf{f}_{t+1}$	$\hat{\mathbf{u}}_{t+1} \leftarrow \hat{\mathbf{V}}^{-1}(\hat{\mathbf{g}}_{t+1})(\hat{\mathbf{P}} \otimes_2 \hat{\mathbf{g}}_{t+1})\hat{\mathbf{f}}_{t+1}$
	end	**end**

Figure 2: *Algorithmic summary of the full-dimensional and reduced time steps. Note the close correspondence.*

into advection, diffusion, and pressure projection steps. The algorithm is summarized in Fig. 2, where we integrate the full-space equations using a 4th order Runge-Kutta integrator. We now describe each step in detail.

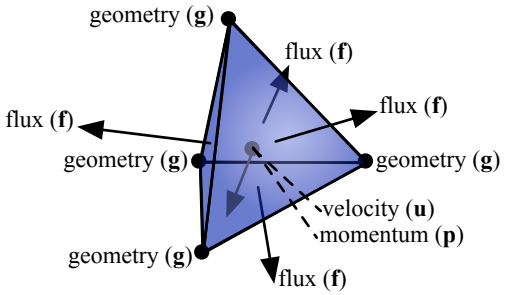


Figure 3: *Geometric layout of the simulation variables on a tetrahedral element.*

5.1 Advection

The advection step transports quantities through the mesh. We treat each of the x, y, and z components of velocity separately, and transport them between mesh elements according to the flux $\mathbf{f}$. This transport is described by the advection tensor $\mathbf{A}$, which interpolates velocities onto the mesh faces, multiplies the interpolated velocities by $\mathbf{f}$ to find the rate of momentum transport over each face, and finally sums the momentum transported over a cell's faces to find the momentum derivative at that cell as $\dot{\mathbf{p}} = (\mathbf{A} \otimes_2 \mathbf{f})\mathbf{u}$. For a cell i, $\mathbf{A}$ computes:

$$\dot{\mathbf{p}}_i = \sum_e \mathbf{f}_{f_{ie}} \frac{1}{2}(\mathbf{u}_e + \mathbf{u}_i), \tag{6}$$

where the sum index e runs over the four face-adjacent cells to i, and f_{ie} denotes the index of the (oriented) face between i and e.

However, we are not primarily interested in momentum $\dot{\mathbf{p}}$, but in velocity $\dot{\mathbf{u}}$. We assume a constant density (incompressible) fluid[1], so $\mathbf{u}$ and $\mathbf{p}$ are related by the 5th-order *volume tensor* $\mathbf{V}$, which we use to compute a matrix-valued cubic polynomial in the vertex locations:

$$\mathbf{V} \otimes_{2\ldots4} \mathbf{g} = \begin{bmatrix} v_1(\mathbf{g}) & & \\ & \ddots & \\ & & v_n(\mathbf{g}) \end{bmatrix} \tag{7}$$

where $v_k(\mathbf{g})$ is the volume of cell k as a function of the vertex positions. For clarity, we will write $\mathbf{V}(\mathbf{g})$ in place of $\mathbf{V} \otimes_{2\ldots4} \mathbf{g}$.

[1] Specifically, we assume the density is 1.

Given $\mathbf{V}$, we can compute momentum: $\mathbf{p} = \mathbf{V}(\mathbf{g})\mathbf{u}$. This gives us the advection equation:

$$\dot{\mathbf{u}} = \mathbf{V}^{-1}(\mathbf{g})(\mathbf{A} \otimes_2 \mathbf{f})\mathbf{u}. \tag{8}$$

Note that this equation depends on the geometry $\mathbf{g}$. If $\mathbf{g}$ were constant, $\mathbf{V}$ would simply be a constant matrix, and we could precompute $\mathbf{V}^{-1}$ and absorb it into $\mathbf{A}$. Because we want to simulate the fluid behavior in the presence of changing geometry, we must explicitly represent $\mathbf{V}$ as a tensor which we can contract to a matrix and invert. This requires our non-polynomial Galerkin projection technique.

5.2 Diffusion

We discretize viscosity as follows:

$$\dot{\mathbf{u}} = -\mu \mathbf{V}^{-1}(\mathbf{g}) \triangle \mathbf{u}, \tag{9}$$

where μ is a viscosity coefficient, $\triangle$ is the graph Laplacian $(\triangle \mathbf{u})_i = -|\mathcal{N}_i|\mathbf{u}_i + \sum_{j \in \mathcal{N}_i} \mathbf{u}_j$, $\mathcal{N}_i$ are the (usually four) tetrahedra neighboring tetrahedron i, and $\mathbf{V}$ is the volume tensor. We have found that this simple, geometric approximation to diffusion is sufficient in both the full and reduced spaces. After computing the contribution of both advection and diffusion to $\dot{\mathbf{u}}$, we use an explicit time integration scheme to update the velocity.

5.3 Projection

Given a velocity $\mathbf{u}$, the projection step first generates a flux $\mathbf{f}$ that is close to $\mathbf{u}$ and satisfies the incompressibility constraint $\nabla \cdot \mathbf{f} = 0$. We can then convert $\mathbf{f}$ back to a velocity that corresponds exactly to the incompressible flux.

To perform the conversion from flux to velocity, we introduce the tensor $\mathbf{P}$, which sums the volume-weighted directed fluxes of a cell to obtain the momentum of the cell: $\mathbf{p} = (\mathbf{P} \otimes_2 \mathbf{g})\mathbf{f}$. Thus, the velocity corresponding to $\mathbf{f}$ is given by $\mathbf{u} = \mathbf{V}^{-1}(\mathbf{g})(\mathbf{P} \otimes_2 \mathbf{g})\mathbf{f}$. Note that this relation only holds if the fluxes $\mathbf{f}$ are divergence-free.

Using the flux-to-velocity conversion and a velocity field $\mathbf{u}$, we can find the flux field $\mathbf{f}$ that fulfills our incompressibility constraint $\nabla \cdot \mathbf{f} = 0$, while minimizing the energy of the difference between its corresponding velocity $\mathbf{V}^{-1}(\mathbf{g})(\mathbf{P} \otimes_2 \mathbf{g})\mathbf{f}$ and $\mathbf{u}$. The energy to be minimized is

$$\frac{1}{2}||\mathbf{u} - \mathbf{V}^{-1}(\mathbf{g})(\mathbf{P} \otimes_2 \mathbf{g})\mathbf{f}||^2_{\mathbf{V}(\mathbf{g})}, \tag{10}$$

where $\|\mathbf{x}\|^2_{\mathbf{M}} = \mathbf{x}^T\mathbf{M}\mathbf{x}$. We solve the minimization using Uzawa's method as described in [Benzi et al. 2005].

Again, since we need to perform flux-to-velocity conversions, we have to perform divisions by cell volumes $\mathbf{V}(\mathbf{g})$. In this case, these divisions appear in the objective of our optimization.

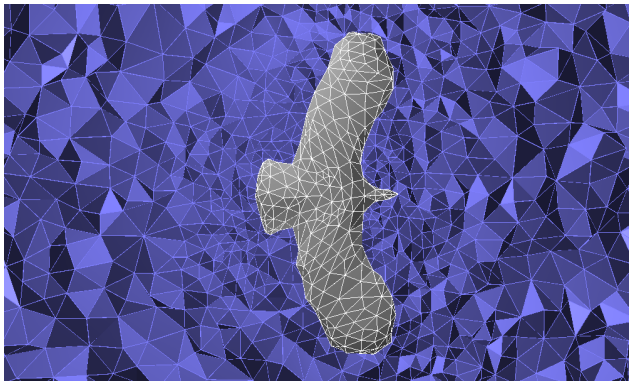


Figure 4: *A cutaway view of a tetrahedral mesh used in our fluid simulation application.*

5.4 Fluid-Geometry Coupling

To allow deforming objects to exert forces on the fluid around them, we modify both the advection and projection steps of our simulation. To model the effect of the moving mesh on advection, we compute and then subtract the flow $\mathbf{h}$ induced by the motion of the mesh. This ensures that velocities do not translate simply because their discretization element moves through space. We also modify the projection step to ensure that we never advect fluid across a moving domain boundary.

The movement of face i induces a flux $h_i(\dot{\mathbf{g}})$ through that face equal to

$$h_i = A_i \dot{\mathbf{c}}_i \cdot \mathbf{n}_i, \tag{11}$$

where A_i is the area of the face, $\mathbf{n}_i$ its normal, and $\dot{\mathbf{c}}_i$ the velocity of its centroid. To compensate for mesh movement, we can simply subtract $\mathbf{h}(\dot{\mathbf{g}})$ wherever we use $\mathbf{f}$.

In particular, in the advection step we subtract the effect of advection due to induced fluxes from the effect of advection due to fluid fluxes: $\dot{\mathbf{p}} = \mathbf{A} \otimes_2 \mathbf{f} - \mathbf{A} \otimes_2 \mathbf{h}$. In the projection step, we enforce that there is no flow across the boundary by adding constraints $\mathbf{Df} + \mathbf{Dh} = 0$ to the projection, where $\mathbf{D}$ is an operator which selects the boundary faces. These modifications ensure that the flow inside of the domain is independent of the movement of the mesh, and that there is no flow across (possibly moving) domain boundaries.

6 Reduced Fluids

We construct the reduced simulation by applying our non-polynomial reduction rules (§4) to the fluid simulation method from the previous section.

In order to reduce the governing equations, we have to reduce the tensors $\mathbf{V}$, $\mathbf{P}$, $\mathbf{A}$, $\triangle$, and $\mathbf{D}$. Because we require bases for each axis of each of these tensors (last paragraph before §4.1), we will need a flux basis $\mathbf{B}_f$, velocity basis $\mathbf{B}_u$, and momentum basis $\mathbf{B}_p$. Since $\mathbf{V}$ and $\mathbf{P}$ depend on the geometry, we need a geometry basis $\mathbf{B}_g$. We also need a basis $\mathbf{B}_h$ for the fluxes induced by mesh motion and a basis $\mathbf{B}_d$ for boundary fluxes.

6.1 Basis Construction

The quality of the runtime simulation depends significantly on our choice of bases. We build $\mathbf{B}_f$, $\mathbf{B}_h$, $\mathbf{B}_u$, and $\mathbf{B}_p$ using a method similar to [Treuille et al. 2006]. We run a set of full-dimensional simulations and collect snapshots of simulation quantities into large matrices, where each column represents a simulation frame. We create bases by running out-of-core Singular Value Decomposition (SVD) on these matrices using the method of James and Fatahalian [2003]. After computing the momentum and velocity bases independently, we concatenate these bases and re-orthonomalize. This process ensures that $\mathbf{B}_u = \mathbf{B}_p$, which ensures energy preservation in the reduced space (§A.2) and complies with the requirements of matrix inversion reduction (§4).

Creating the geometry basis $\mathbf{B}_g$ is more difficult. We begin with a sequence of triangle meshes animating changes to the boundary conditions, such as the flapping wings of a bird. We select an intermediate pose for the boundary mesh and construct a tetrahedral *base mesh* discretizing the simulation domain. For each deformed state of the surface model, we then use Laplacian deformation transfer [Sorkine 2006] to deform the base mesh so that the embedded surface matches the deformed surface model. Unfortunately, this step can lead to inverted elements, which would be fatal to the simulation. To fix inversions, we interleave the following two steps. First, we increase the weights around inverted elements to increase rigidity during Laplacian deformation and thus avoid inversion. Second, we improve the quality of the mesh by running Stellar [Klingner and Shewchuk 2007], which we modified to leave surface vertices unchanged. Unfortunately, the latter step can lead to discontinuities in the animation sequence, where the configuration of internal vertices rapidly changes between simulation frames. This popping artifact, which can be seen in our example video, is not fatal, but removing it would likely improve simulation quality and is an open question for future research. Once we have an inversion-free tetrahedral mesh animation, we run SVD to create $\mathbf{B}_g$.

6.2 Tensor Reduction

Using these bases, we follow the procedure described in §4, turning the full space tensors into their reduced equivalents:

	Tensor	Galerkin Projection
Advection	$\mathbf{A}$	$\hat{\mathbf{A}} = \mathbf{B}_p^T \mathbf{A} \otimes_1 \mathbf{B}_u \otimes_2 \mathbf{B}_f$
Induced Advection	$\mathbf{A}_h$	$\hat{\mathbf{A}}_h = \mathbf{B}_p^T \mathbf{A} \otimes_1 \mathbf{B}_u \otimes_2 \mathbf{B}_h$
Diffusion	$\triangle$	$\hat{\triangle} = \mathbf{B}_u^T \triangle \mathbf{B}_u$
Flux to momentum	$\mathbf{P}$	$\hat{\mathbf{P}} = \mathbf{B}_p^T \mathbf{P} \otimes_1 \mathbf{B}_f \otimes_2 \mathbf{B}_g$
Volume	$\mathbf{V}$	$\hat{\mathbf{V}} = \mathbf{B}_p^T \mathbf{V} \otimes_1 \mathbf{B}_u \otimes_{2\ldots4} \mathbf{B}_g$
Boundary	$\mathbf{D}$	$\hat{\mathbf{D}} = \mathbf{B}_d^T \mathbf{D} \mathbf{B}_f$
Induced Boundary	$\mathbf{D}_h$	$\hat{\mathbf{D}}_h = \mathbf{B}_d^T \mathbf{D}_h \mathbf{B}_h$

The reduced space simulation procedure is nearly identical to the full space one (Fig. 2), although we do make several small changes. First, we store values of $\hat{\mathbf{h}}$ along deformation trajectories that we will use at runtime and replay these trajectories to find $\hat{\mathbf{h}}$, rather than computing $\mathbf{h}$ from $\hat{\mathbf{g}}$. Second, we integrate advection and diffusion by matrix exponentiation, instead of explicitly as we do in the full space [Treuille et al. 2006]. In the full space, integration by exponentiation would make sure that the simulation is stable for any time step (§A.1). The fact that $\mathbf{B}_u = \mathbf{B}_p$ ensures that this stability result can be carried over into the reduced space (§A.2).

6.3 Constraints

Fluid simulation requires that we maintain full-dimensional constraints exactly in the reduced-dimensional simulation. In particular, we must maintain a hard incompressibility constraint in the reduced simulation. As in [Treuille et al. 2006], all basis vectors of $\mathbf{B}_f$ are divergence-free by construction. They remain divergence-free under geometric deformation, since the units of flux are *volume*

per unit time, which are independent of the geometry. However, unlike earlier work, we have multiple bases whose relationships may change with the mesh geometry. In particular, the velocity and momentum bases are not necessarily divergence-free. A reduced projection step is therefore necessary. Reducing the full-dimensional projection (Eq. 10), we obtain:

$$\frac{1}{2}||\hat{\mathbf{u}} - (\hat{\mathbf{V}}^{-1}(\hat{\mathbf{g}})(\hat{\mathbf{P}} \otimes_2 \hat{\mathbf{g}})\hat{\mathbf{f}}||_{\hat{\mathbf{V}}(\hat{\mathbf{g}})}. \tag{12}$$

Since the flux basis guarantees $\nabla \cdot \mathbf{B}_f\hat{\mathbf{f}} = \mathbf{0}^T$, we can omit the constraint in the reduced projection.

6.4 Fluid-Geometry Coupling

To make the flow independent of the changes in the geometry, we proceed as in the full space. We subtract the induced fluxes from the fluxes when computing advection, which, since $\mathbf{B}_f$ and $\mathbf{B}_h$ are different, requires us to generate two reduced advection tensors $\hat{\mathbf{A}}$ and $\hat{\mathbf{A}}_h$ (§6.2); boundary constraints $\hat{\mathbf{D}}\hat{\mathbf{f}} + \hat{\mathbf{D}}_h\hat{\mathbf{h}} = 0$ are added to the projection. While the constraints are in the reduced space, we apply constraint reduction [Wicke et al. 2009] to ensure that fulfilling the reduced constraints entails fulfilling the corresponding full space constraints exactly; this process also gives us our boundary flux basis $\mathbf{B}_d$.

6.5 Runtime Visualization

To visualize the flow field, we advect massless marker particles with the flow. Each particle can be advected separately. Because the velocity state is not necessarily divergence free, we reconstruct advection velocities from the flux field. We assume that the velocity is constant in each cell. Whenever we need to evaluate a velocity in a cell i, we locally compute

$$\mathbf{u}_i = \left(\mathbf{V}^{-1}(\mathbf{B}_g\hat{\mathbf{g}})(\mathbf{P} \otimes_2 \mathbf{B}_g\hat{\mathbf{g}})\mathbf{B}_f\hat{\mathbf{f}}\right)_i. \tag{13}$$

Using this technique, we never need to expand the full representation of the geometry or flux. Instead, for each particle we remember the cell i that currently contains it. We then evaluate only those parts of the geometry that are necessary to compute $\mathbf{u}_i$. We only have to compute the positions of vertices incident to the current element and its neighbors, as well as the fluxes across the faces of the current element. We can then advect the particle with the computed velocity. A particle may leave its current cell, either because the particle is moved by advection, or because the mesh deforms. In that case, we walk across the mesh starting at the particle's last known position, and moving in the direction of the particle's new position. We evaluate the mesh geometry only locally, and continue the walk until we have found an element that contains the particle. For advection, we use explicit Euler integration, with ten substeps per frame for the examples shown in the accompanying video.

7 Fluid Results

We evaluate our reduced fluid simulation in two ways: evaluating its numerical error in a 2D wind tunnel domain containing a simple obstacle, and demonstrating its qualitative behavior in two different 3D domains with complex boundary motion.

7.1 Numerical Error Analysis

To assess the accuracy of our method, we measure deviation from a full-dimensional ground truth simulation for a simple two-dimensional example. We ran full simulations in a domain consist-

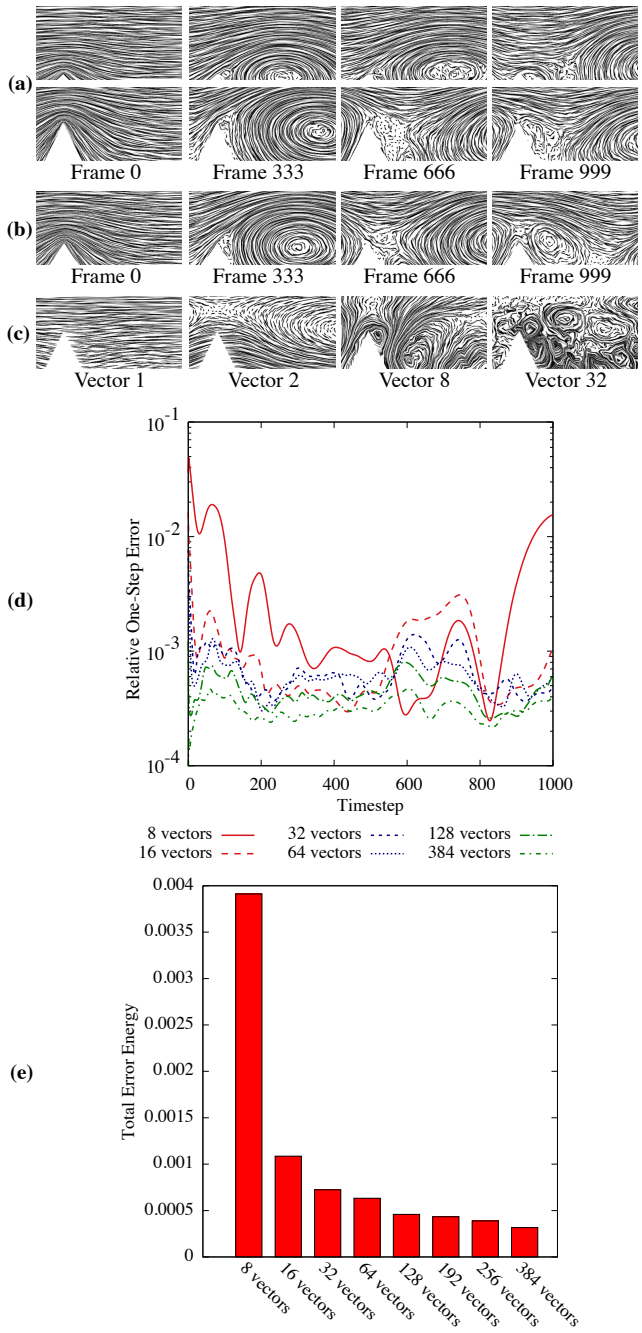


Figure 5: *The results of our fluid application error analysis.* ***(a)*** *Frames from the two training simulations: one with a large obstacle, and one with a small obstacle.* ***(b)*** *The corresponding frames from a reduced test simulation in a domain with a medium-sized obstacle.* ***(c)*** *Selected vectors for the velocity basis trained from part (a) and used to run part (b).* ***(d)*** *Relative one-step error over the course of 1000 frames for reduced models with different size bases.* ***(e)*** *Integrated one-step error for models with different size bases. The time-averaged integrated error monotonically decreases with basis size.*

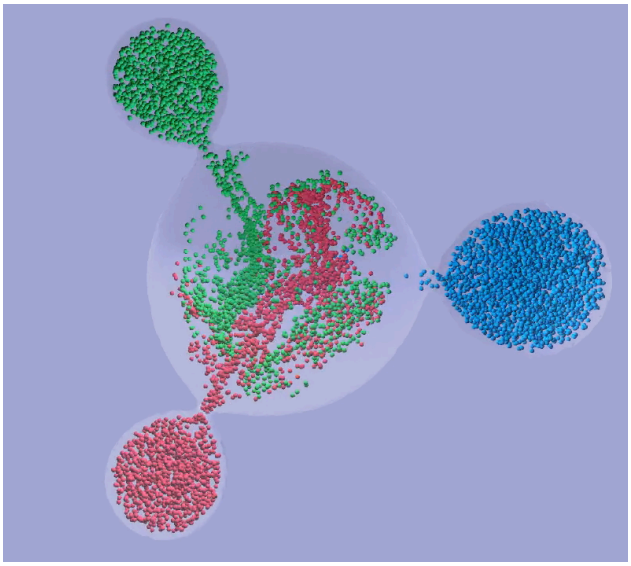


Figure 6: *Our method simulates this deformable mixing chamber at 70 frames per second (3 frames per second with rendering).*

ing of a 2D periodic tunnel-shaped domain containing a single triangular obstacle, and used the simulation frames to compute bases of different sizes. We show some example frames and basis vectors in Fig. 5(a-c).

To give a clear picture of how accurately a reduced model built with velocity basis $\mathbf{B}_u$ captures the simulation dynamics over a range of states, we use a *one-step* error measurement. To find the one-step error, we begin with a sequence of velocity snapshots $\mathbf{u}_1, \ldots, \mathbf{u}_T$ from a full-dimensional simulation and project them into $\mathbf{B}_u$ using an energy-conserving projection[2] to find reduced coordinates $\hat{\mathbf{r}}_1, \ldots, \hat{\mathbf{r}}_T$. From each reduced coordinate, we take a full timestep to obtain $\mathbf{u}_{\text{ground}}$ and we take a reduced timestep to obtain $\mathbf{u}_{\text{test}}$. Let the energy $E(\mathbf{u})$ of a velocity field $\mathbf{u}$ be $E(\mathbf{u}) = \frac{1}{2}\mathbf{u}^T\mathbf{V}(\mathbf{g})\mathbf{u}$. The one-step error is then $E(\mathbf{u}_{\text{ground}} - \mathbf{u}_{\text{test}})/E(\mathbf{u}_{\text{ground}})$.

Fig. 5(d) plots the one-step error over the evolution of a simulation for a range of basis sizes. Fig. 5(e) plots this same error integrated over the entire simulation. In both Fig. 5(d) and Fig. 5(e), we see that the error tends to decrease as we add more basis vectors. This reassures us that our method converges to the ground truth as the number of basis vectors grows to the dimension of the full simulation. While the decline in error with basis size is monotonic when the error is averaged over time (e), it is not necessarily so instantaneously (d). Our one-step error measure ensures that each simulation begins each step from as close an approximation to the same full state as possible, however, simulations with different basis sizes will start from slightly different initial states due to differences in basis expressivity. Therefore, some deviation from a monotonic decrease in error at some timesteps should be expected.

7.2 3D Results

In these results, we used a 110-node cluster with 2.2 GHz SMT quad-core AMD Opteron processors for precomputation, and a 2.6 GHz 8-core Intel Xeon processor with 24GB of memory for real-time simulation, rendering and timing comparison. Our precomputation wall-clock timing results include both PCA and tensor premultiplication time. We used a serial PCA implementation to compute each basis, so only the tensor premultiplication was fully parallelized across the cluster. Our runtime timing results (Table 1) include both simulation and particle advection runtime.

Deforming cavity. The *cavity* model (Fig. 6) consists of a large central cavity and three adjacent smaller ones, connected with thin tunnels. Each of the small cavities can be individually compressed, causing the fluid inside them to flow into the central cavity, which changes its volume accordingly. We designed the cavity using simple level set primitives, and then tetrahedralized the interior of the isosurface using CGAL [CGAL].[3] We generated a sequence of example deformations by analytically compressing the outer cavities while renormalizing the volume and used these deformations to generate 14 full-space simulations, each capturing the compression and re-inflation of one or two small cavities. Not all deformations seen in the video are part of the original training set (for instance, the training set contains no examples where one cavity expands while another remains compressed). Precomputation for this example took 12 hours of wall-clock time, of which approximately the last half hour consisted of tensor premultiplication. At runtime, we can simulate at 70 frames per second, however, due to the dense coverage of particles in the simulation, advecting and rendering particles decreases the frame rate to 3 frames per second. Various forms of optimization which we have not pursued, such as parallelization, could certainly increase the frame rate further.

Eagle. Our eagle example (Fig. 1, Fig. 4) shows a different use case. Here we started with a 601 frame triangle mesh animation of a flying eagle flapping its wings and soaring left and right. We built a deformable tetrahedral mesh as described in §6.1, generating the base mesh by using Tetgen [Si 2007] to tetrahedralize the space around the eagle with fully extended wings. We computed a single 266 frame full-dimensional simulation to use for snapshots. Precomputation for this example took approximately 4 hours of wall-clock time, of which again approximately the last half hour consisted of tensor premultiplication.

Note that the specific deformation sequence need not be preordained; in principle, this method could be extended to model the deformations in an entire motion graph, with the actual character motion determined at runtime. The accompanying video presents these simulations, which were simulated interactively but rendered off-line with pbrt [Pharr and Humphreys 2004], along with comparisons to a full simulation.

8 Reduced Radiosity

We now apply our model reduction technique to real time computation of global illumination (radiosity) on diffuse deformable objects under varying lighting. As before, we first reformulate the radiosity equation in terms of tensor contractions and matrix operations, and then apply the projection rules in §4 to obtain a reduced model. We use this reduced model to interactively explore illumination design in architectural environments.

Following Goral et al. [1984], we divide the scene into discrete patches, and compute radiosity as:

$$\mathbf{b} = (\mathbf{I} - \rho\mathbf{F})^{-1}\,\mathbf{e} \tag{14}$$

where $\mathbf{b}$ is a vector of total face radiosity, ρ is a diagonal matrix of albedos, $\mathbf{e}$ is a vector of incident lighting intensities, and $\mathbf{F}$ is a matrix of *form factors*: $\mathbf{F}_{ij}$ describes the fraction of light incident on face j that is reflected to face i. Notice that, unlike fluid simulation,

[2] The projection is the minimization $\min_{\hat{\mathbf{r}}_t} ||\mathbf{u}_t - \mathbf{B}_u\hat{\mathbf{r}}_t||_{\mathbf{V}(\mathbf{g}_t)}$.

[3] Due to the regular shape of the domain, we were able to use a simpler method than the one described in §6.1 for generating the tetrahedral mesh.

	full simulation						reduced simulation							
	dimensions			runtime			basis dimensions			#particles	memory	runtime		speedup
	#cells	#faces	#vertices	t_d	t_a	t_p	m_f	m_u	m_g			t_s	t_{pa}	
Cavity	88671	13339	17823	2.17s	18.4s	220s	64	128	5	10000	29.7MB	0.0145s	0.308s	16565×
Eagle	300217	149903	28613	2.92s	18.9s	82.0s	70	193	7	~ 9500	131MB	0.0469s	0.0381s	2252×

Table 1: *Runtimes and statistics for our examples: The table shows timings for full dimensional deformation t_d, advection t_a and projection t_p, as well as reduced simulation t_s and particle advection t_{pa}. It also contains the number of cells, faces and edges determining the dimension of the full simulation, and the reduced basis dimensions of the flux, velocity, and geometry bases, m_f, m_u, and m_g, respectively.*

radiosity requires modeling dense interactions between scene components, which increases the computational complexity in a scene with n faces to $O(n^2)$.

8.1 Tensor Formulation of Radiosity

Ordinarily, Eq. 14 describes a straightforward linear relationship between direct lighting and radiosity. However, if objects in the scene move or deform, this equation becomes highly nonlinear. The crucial term in Eq. 14 is the form factor matrix $\mathbf{F}$, which we sample at the centroid of each face to obtain

$$F_{ij} = \frac{(\mathbf{n}_i \cdot (\mathbf{c}_j - \mathbf{c}_i))\,(\mathbf{n}_j \cdot (\mathbf{c}_i - \mathbf{c}_j))\,\mathrm{Vis}\,(i,j)}{\pi\,(\mathbf{c}_i - \mathbf{c}_j)^4\,A_i}, \qquad (15)$$

where $\mathrm{Vis}(i,j)$ is 1 if faces i and j are visible to each other and 0 otherwise, A_i is the area of face i, $\mathbf{c}_i$ is the centroid of triangle i, and $\mathbf{n}_i$ is computed as the vector cross product of two of i's edges.

In order to model reduce this equation using our non-polynomial Galerkin projection method, we need to represent it as a composition of tensor products, matrix inverses, and matrix roots. This composition requires more tensors than our fluids application, however, the use of each tensor tends to be simpler. Our tensor representation requires that we be able to unroll $\mathbf{F} \in \mathbb{R}^{n\times n}$ into a vector $\mathbf{p} \in \mathbb{R}^{n^2}$ by re-indexing: $p_k = F_{ij}$, where $k = ni + j$ and n is the number of faces in the mesh. We also need the transpose index $k^T = nj + i$, so that $p_{k^T} = F_{ji}$. We summarize the necessary bases and tensors in Table 2; interested readers may refer to §B in the supplemental material for a full derivation.

We can then evaluate $\mathbf{b}$ by evaluating each row of Table 2 in order. The tensor form of the radiosity equation (Eq. 14) is given in the final row of Table 2:

$$\mathbf{b} = (\mathbf{I} - \mathbf{E} \otimes_2 \mathbf{p})^{-1}\mathbf{e}, \qquad (16)$$

where $\mathbf{E}$ is the 3rd-order tensor that transforms the unrolled form factor vector $\mathbf{p}$ back into the form factor matrix $\mathbf{F}$ and left-multiplies $\mathbf{F}$ by the face albedos ρ.

8.2 Reducing the Radiosity Equation

We begin with a set of mesh deformations $\mathbf{G} = \{\mathbf{g}_1, \ldots, \mathbf{g}_m\}$ and for each deformation compute the intermediate vector quantities described in columns 2 and 3 of Table 2. We run PCA on each of these vector quantities to obtain the corresponding 10 bases (column 4). Using these bases, we apply our non-polynomial Galerkin projection technique §4 to compute reduced form factors $\hat{\mathbf{p}}$. This reduction consists of replacing every tensor in column 3 of Table 2 with its reduced equivalent in column 5.

8.3 Reduced Visibility

It is not clear how to write a binary discontinuous function like visibility using our tensor formulation, so we handle it separately. Note that computing the visibility of a new deformation at runtime is too slow and would defeat the purpose of model reduction. Instead, we use the following strategy: first, we compute the visibilities $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_m\}$ corresponding to the deformation set $\mathbf{G}$, and run PCA on them to form the visibility basis $\mathbf{B}_v$. Then, we determine the reduced visibility $\hat{\mathbf{v}}_{\mathrm{test}}$ of a new deformation $\hat{\mathbf{g}}_{\mathrm{test}}$ as a convex combination of the visibilities in the training set. We find the convex combination $\mathbf{x}$ of training deformations that best predicts $\hat{\mathbf{g}}_{\mathrm{test}}$:

$$\arg\min_{\mathbf{x}} ||\hat{\mathbf{g}}_{\mathrm{test}} - \mathbf{B}_g^T\mathbf{G}\mathbf{x}|| \qquad (17)$$

$\hat{\mathbf{v}}_{\mathrm{test}} = \mathbf{B}_v^T\mathbf{V}\mathbf{x}$ is the convex combination of reduced visibility vectors with the same coefficients. Note that $\mathbf{B}_g^T\mathbf{G}$ and $\mathbf{B}_v^T\mathbf{V}$ can be precomputed and that their sizes do not depend on the number of vertices in the mesh. This is a reasonable strategy, since two deformations with similar reduced geometry will have similar full-space geometry, and therefore will also have similar visibilities and we are primarily concerned with low-frequency interreflections under area lighting.

9 Radiosity Results

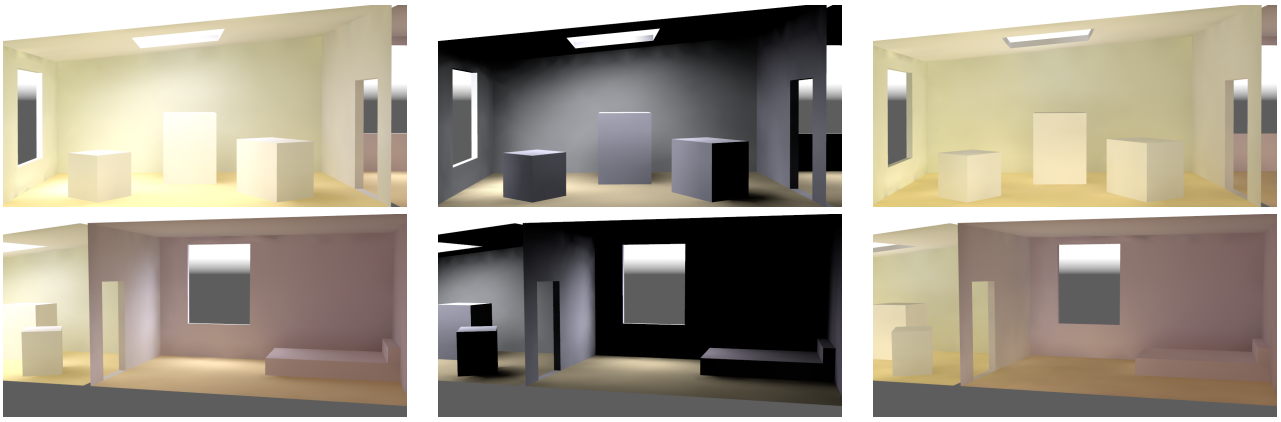
We demonstrate our reduced radiosity method by demonstrating an interactive method for exploring a space of architectural designs in order to achieve certain desired lighting conditions, similar to [Dorsey et al. 1991]. Creating a pleasingly-illuminated space requires careful selection of room arrangements, room sizes, window placements, orientation of the space relative to natural lighting sources, and so forth.

We modeled a scene with a living room and bedroom, consisting of 5012 faces. The living room is brightly-lit with light colored walls, and the bedroom is more dimly-lit with darker walls. In the scene, the sizes of the skylight, windows, and doors can be changed interactively, and the living room ceiling can be tilted.

Our training set for this scene consists of 540 samples, including extremal positions along the axes of the configuration space and samples drawn from the space's interior. We ran PCA on the examples to select basis vectors capturing 99.9% of the variance of each intermediate quantity (Table 2, column 4), up to a maximum of 60 vectors. The visibility basis $\mathbf{B}_v$, the area-free form factor basis $\mathbf{B}_d$, and the form factor basis $\mathbf{B}_p$ reached the 60 vector cap; all of the others were able to capture 99.9% of the variance using fewer than 60 vectors. As with fluid simulation, we merged and orthogonalized pairs of bases to comply with our matrix inverse and matrix root basis requirements. The basis pairs we merged are $\mathbf{B}_s$ and $\mathbf{B}_h$, and $\mathbf{B}_d$ and $\mathbf{B}_p$. We used identical 120-vector bases for $\mathbf{B}_e$ and $\mathbf{B}_b$. We ran the basis selection and precomputation on a 8-node Amazon EC2 cluster composed of 8-core 2.67GHz Intel processors

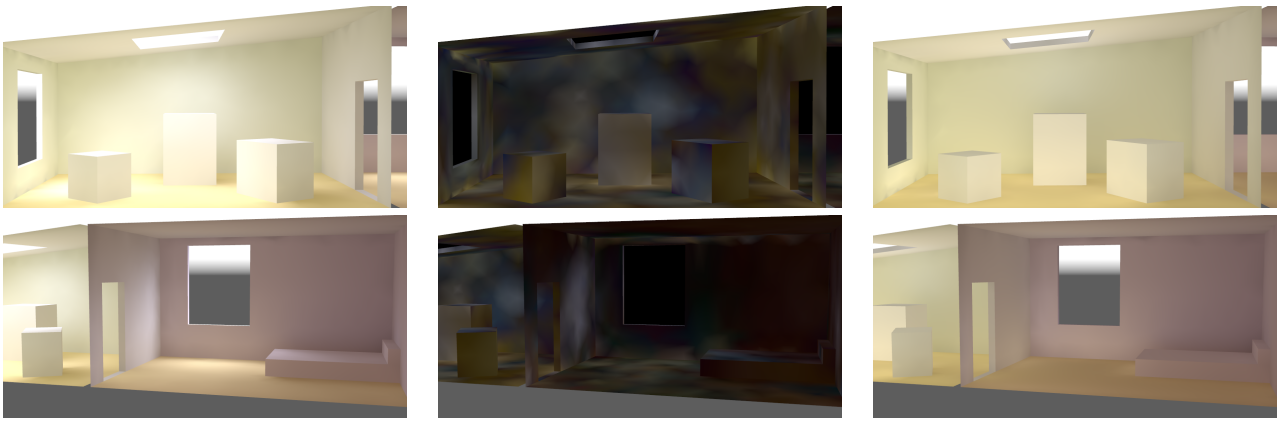
Intermediate Component	Full Space Element Notation	Full Space Tensor Notation	Result Basis	Galerkin Projection
Geometry (vertex positions)		$\mathbf{g}$	$\mathbf{B}_g$	
Normals	$\mathbf{n}_i = (\mathbf{g}_{i,1} - \mathbf{g}_{i,0}) \times (\mathbf{g}_{i,2} - \mathbf{g}_{i,0})$	$\mathbf{n} = \mathbf{N} \otimes_{1\ldots2} \mathbf{g}$	$\mathbf{B}_n$	$\hat{\mathbf{N}} = \mathbf{B}_n^T \mathbf{N} \otimes_{1\ldots2} \mathbf{B}_g$
Scaled Cosines	$s_k = \mathbf{n}_i \cdot (\mathbf{c}_j - \mathbf{c}_i)$	$\mathbf{s} = \mathbf{S} \otimes_1 \mathbf{n} \otimes_2 \mathbf{g}$	$\mathbf{B}_s$	$\hat{\mathbf{S}} = \mathbf{B}_s^T \mathbf{S} \otimes_1 \mathbf{B}_n \otimes_2 \mathbf{B}_g$
Squared Distances	$r_k = (\mathbf{c}_i - \mathbf{c}_j)^T (\mathbf{c}_i - \mathbf{c}_j)$	$\mathbf{r} = \mathbf{R} \otimes_{1\ldots2} \mathbf{g}$	$\mathbf{B}_r$	$\hat{\mathbf{R}} = \mathbf{B}_r^T \mathbf{R} \otimes_{1\ldots2} \mathbf{B}_g$
Half Form Factors	$h_k = s_k / r_k$	$\mathbf{h} = (\mathbf{H} \otimes_2 \mathbf{r})^{-1} \mathbf{s}$	$\mathbf{B}_h$	$\hat{\mathbf{H}} = \mathbf{B}_s^T \mathbf{H} \otimes_1 \mathbf{B}_h \otimes_2 \mathbf{B}_r$
Visibility-Free Form Factors	$c_k = h_k h_k{}^T$	$\mathbf{c} = \mathbf{C} \otimes_{1\ldots2} \mathbf{h}$	$\mathbf{B}_c$	$\hat{\mathbf{C}} = \mathbf{B}_c^T \mathbf{C} \otimes_{1\ldots2} \mathbf{B}_h$
Visibility	$v_k = \mathrm{Vis}(i, j)$	$\mathbf{v} = \mathbf{v}(\mathbf{g})$	$\mathbf{B}_v$	Learned model: see §8.3.
Area-Free Form Factors	$d_k = d_k v_k$	$\mathbf{d} = \mathbf{D} \otimes_1 \mathbf{c} \otimes_2 \mathbf{v}$	$\mathbf{B}_d$	$\hat{\mathbf{D}} = \mathbf{B}_d^T \mathbf{D} \otimes_1 \mathbf{B}_c \otimes_2 \mathbf{B}_v$
Squared Areas	$a_i = \mathbf{n}_i \cdot \mathbf{n}_i$	$\mathbf{a} = \mathbf{A} \otimes_{1\ldots2} \mathbf{n}$	$\mathbf{B}_a$	$\hat{\mathbf{A}} = \mathbf{B}_a^T \mathbf{A} \otimes_{1\ldots2} \mathbf{B}_n$
Form Factor Vector	$p_k = d_k / \sqrt{a_i}$	$\mathbf{p} = (\mathbf{P} \otimes_2 \mathbf{a})^{-\frac{1}{2}} \mathbf{d}$	$\mathbf{B}_p$	$\hat{\mathbf{P}} = \mathbf{B}_d^T \mathbf{P} \otimes_1 \mathbf{B}_p \otimes_2 \mathbf{B}_a$
Incident Illumination		$\mathbf{e}$	$\mathbf{B}_e$	
Radiosity	$b_i = \sum_{j=0}^{n} (\mathbf{I} - \rho \mathbf{F})_{ij}^{-1} e_j$	$\mathbf{b} = (\mathbf{I} - \mathbf{E} \otimes_2 \mathbf{p})^{-1} \mathbf{e}$	$\mathbf{B}_b$	$\hat{\mathbf{E}} = \mathbf{B}_e^T \mathbf{F} \otimes_1 \mathbf{B}_b \otimes_2 \mathbf{B}_p$

Table 2: *Bases, operators, and Galerkin projections used in our reduced form factor implementation. Radiosity rendering consists of computing this table from top to bottom, either in the full space (columns 2 and 3) or the reduced space (column 5). Note that while the fluids application required only 4 bases, computing the radiosity form factors requires 9 (there are 12 bases listed here, but we constrain $\mathbf{B}_s = \mathbf{B}_h$, $\mathbf{B}_d = \mathbf{B}_p$, and $\mathbf{B}_b = \mathbf{B}_e$). For a detailed derivation and an explanation of the notation, please see §B in the supplementary material.*



Radiosity | Direct Illumination | Indirect Illumination

(a) Model Reduction



Radiosity | Difference Image | Indirect Illumination

(b) Ground Truth

Figure 7: *Results from our architectural rendering example. The scene consists of two rooms, a brightly-lit living room with light colored walls linked by a door to a dimmer bedroom with darker-colored walls. The scene is illuminated by an overcast sky through two windows and a skylight. The scene mesh consists of 5012 faces, and the sizes of the skylights, windows, and doors can be changed interactively. In addition, the living room ceiling can be tilted. (a) Results generated by our model reduced radiosity implementation. (b) Results generated by a full space radiosity implementation. Notice the qualitative similarity between the ground truth and model reduced renderings. Notice the bright spots near the cubes and the foot of the bed, as well as below the windows. There are artifacts in both the reduced and full space renderings due to the coarse meshing near the edges.*

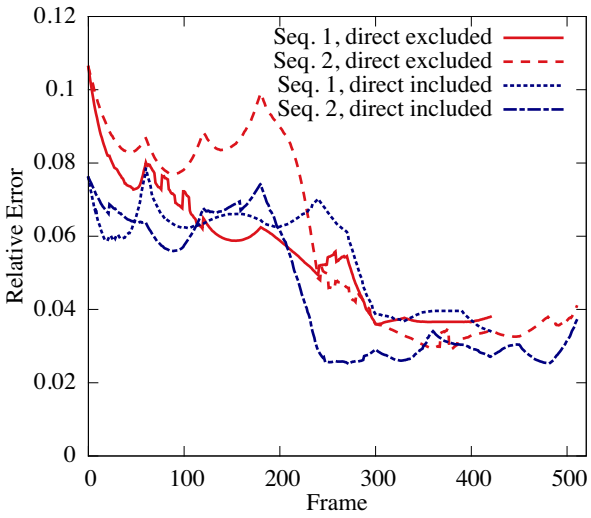


Figure 8: *Relative error over the course of the two radiosity animation sequences for direct illumination included and excluded from the model reduction process. Our radiosity basis computation sampled brighter scenes more heavily than dimly lit scenes, so brighter scenes are represented more accurately in the final results.*

with 67.5GB RAM[4], however, both stages were primarily limited by cluster I/O bandwidth. Precomputation took 5 hours 1 minute wall-clock time for PCA, and 1 hour 36 minutes wall-clock time for tensor premultiplication. We rendered two simulation sequences, of 420 and 500 frames, using our reduced model, and compared them with full-space renderings (Fig. 7). We rendered each sequence twice, using two different methods for computing the direct illumination. The first method was to compute the direct illumination outside of our reduced radiosity at runtime using physically-based sky model; the second was to compute direct illumination inside of our model by placing area lights in the windows. Fig. 8 shows the relative error of these results, which varies from 3% to 11% over the course of the test animations for both methods.

A comprehensive analysis of the space and time requirements for full-space and reduced methods for radiosity depends on the particular algorithm and implementation used. We only discuss relative benefits with respect to classical radiosity for an interactive application. Our full-space radiosity implementation requires no precomputation time, runs at 0.2 frames per second and requires 200 MB of memory. Our reduced radiosity implementation requires 6 hours and 37 minutes of precomputation time, runs at 22.7 frames per second, and requires only 50 MB of memory. In this setting, our method achieves a runtime speedup of 113 times. If we precompute the form factors, we can load them from disk at 0.4 frames per second, reducing the speedup to 57 times, but at the cost of substantial precomputation time and required storage which, unlike for our reduced method, will grow with the number of frames viewed. Much of the execution time (around 80%) of our method is devoted to computing visibility, which we do not perform using our non-polynomial Galerkin projection technique. We expect that improved methods for computing visibility could dramatically improve our reduced radiosity implementation's performance.

[4]Instance type `m2.4xlarge`.

10 Limitations

Like all model reduction techniques, the success of non-polynomial Galerkin projection is limited by the representational power of the precomputed subspaces. If these subspaces do not capture the underlying dynamics well, then the reduced results will likely diverge from the full space results. Both of our applications are particularly susceptible to this form of error, since evaluating our model reduced system at each timestep involves multiple matrix inverses (Fig. 2, Table 2), each of which minimizes error using a scaled norm that can vary over time and may or may not be well suited to the application at hand (§4.2). On the other hand, non-polynomial Galerkin projection is more widely applicable than previous methods, and our results demonstrate that our technique captures complex flow structures and lighting effects in real time. Moreover, error can always be decreased by increasing the size of the basis, at a corresponding polynomial runtime cost.

Our fluid simulation method can compensate for changes in the geometry of the underlying discretization, and therefore can compute flow through deforming meshes. However, the topology of the discretization must remain fixed. This requires finding a single mesh topology which can accommodate all deformations experienced by the contained geometry. In our experience, using mesh improvement [Klingner and Shewchuk 2007] to create a good base mesh, and applying optimization-based smoothing to obtain good meshes after deformation, works well. For extreme deformations, constructing a usable mesh becomes difficult. In particular, we have noticed that at particular points in the eagle animation sequence, large numbers of tetrahedra will suddenly "pop" into new configurations. While our mesh construction methods usually avoid inverting any tetrahedra during this popping, the existence of this behavior suggests that it may not be always be possible to represent every desired deformation using a single mesh topology. An exciting avenue for future work would be to break up the deformation into shorter deformation sequences and use meshes with different topology for each sequence. This might allow us to capture even more drastic deformation, but would require reduced resampling operators to convert the fluid from one geometry basis to another at runtime. Meshing is much easier in the radiosity case, since that application only requires triangle meshes.

We select bases using PCA because it provides good representational fidelity and reasonable results in practice. Unfortunately, PCA bases do not allow us to provide any explicit guarantees about long-term simulation error generated by Galerkin projection, nor error arising from polynomial composition. In the fluids case, using modal bases derived from the simulation operators, as in [de Witt et al. 2012], which uses the modes of the Laplacian as a simulation basis, could allow us to more specifically describe the characteristics of the error.

11 Conclusion

We have shown that Galerkin projection can be extended from polynomials to the much broader class of compositions of *elementary algebraic operations*, enabling the analytic approximation of functions containing addition, subtraction, multiplication, division, and roots, all in closed form. Unlike standard Galerkin projection, our *non-polynomial Galerkin projection* is guaranteed to preserve polynomial degree, essentially bounding the computational complexity of the approximation. Because of the widespread use of dimension reduction in graphics, we present non-polynomial Galerkin projection in general mathematical terms without specific reference to physical simulation. We believe that our approach can be broadly applied to Galerkin-project functions which previously could not be efficiently approximated.

We showed two different examples of such non-polynomial systems: global illumination of deformable objects, and fluid flow on deforming meshes. Standard Galerkin projection cannot be applied to these phenomena. We demonstrated that non-polynomial Galerkin projection can be applied to both of these phenomena. We also showed that non-polynomial Galerkin projection enables, for the first time, interactive simulations of high-resolution fluid flow around deforming geometry, such as a flying bird, as well as interactive radiosity for lighting design. We believe that this technique could also find use in design and engineering to give interactive feedback about the dynamic effects of geometric changes.

We are also excited to apply non-polynomial Galerkin projection to entire classes of new phenomena which cannot be modeled with previous approaches. For example, n-body gravitational systems and atomic-scale Lennard-Jones interactions can now be model reduced. The existence of non-polynomial Galerkin projection also encourages us to search for further methods to analytically capture, simulate and render the many complex, high-dimensional phenomena in the world around us.

Acknowledgements. This work was supported by an NSF Graduate Research Fellowship, an NSF Career Award (IIS-0953985), an NSF AIR Award (IIP 1127777), NSF Grant IIS-0964562, ONR Grant N00014-11-1-0295, and by generous gifts from Google, Qualcomm, Adobe, Intel, and the Okawa Foundation. We would like to thank Yiling Tay for her help in generating fluid simulation meshes and the anonymous reviewers for their valuable comments.

References

AN, S. S., KIM, T., AND JAMES, D. L. 2008. Optimizing cubature for efficient integration of subspace deformations. *ACM Transactions on Graphics 27*, 5 (Dec.), 165:1–165:10.

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*, D8.

BARBIČ, J., AND JAMES, D. 2005. Real-time subspace integration for St. Venant-Kirchhoff deformable models. In *Proc. SIGGRAPH '05*.

BARBIČ, J., AND POPOVIĆ, J. 2008. Real-time control of physically based simulations using gentle forces. *ACM Transactions on Graphics 27*, 5.

BENZI, M., GOLUB, G. H., AND LIESEN, J. 2005. Numerical solution of saddle point problems. *Acta Numerica 14*, 1–137.

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.

CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

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

CHAHLAOUI, Y., AND VAN DOOREN, P. 2005. Model reduction of time-varying systems. In *Dimension Reduction of Large-Scale Systems*. Springer, 131–148.

DE WITT, T., LESSIG, C., AND FIUME, E. 2012. Fluid simulation using Laplacian eigenfunctions. *ACM Transactions on Graphics 31*, 1 (Jan.).

DEBUSSCHERE, B. J., NAJM, H. N., PEBAY, P. P., KNIO, O. M., GHANEM, R. G., AND MAITRE, O. P. L. 2004. Numerical challenges in the use of polynomial chaos representations for stochastic processes. *SIAM J. Sci. Comp. 26*, 2, 698–719.

DECONINCK, H., AND RICCHIUTO, M. 2007. Residual distribution schemes : foundation and analysis. In *Encyclopedia of Computational Mechanics*, E. Stein, E. de Borst, and T. Hughes, Eds., vol. 3. John Wiley and Sons, Ltd.

DOBES, J., AND DECONINCK, H. 2006. An ALE formulation of the multidimensional residual distribution scheme for computations on moving meshes. In *Proc. Int. Conf. CFD*.

DORSEY, J., SILLION, F., AND GREENBERG, D. 1991. Design and simulation of opera lighting and projection effects. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, 41–50.

DRETTAKIS, G., AND SILLION, F. 1997. Interactive update of global illumination using a line-space hierarchy. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 57–64.

EBERT, F., AND STYKEL, T. 2007. Rational interpolation, minimal realization and model reduction. Tech. rep., DFG Research Center Matheon.

ELCOTT, S., TONG, Y., KANSO, E., SCHRÖDER, P., AND DESBRUN, M. 2007. Stable, circulation-preserving, simplicial fluids. *ACM Transactions on Graphics 26*, 1 (Jan.).

FARHOOD, M., AND DULLERUD, G. E. 2007. Model reduction of nonstationary LPV systems. *IEEE Transactions on Automatic Control 52*, 2, 181–196.

FELDMAN, B. E., O'BRIEN, J. F., AND KLINGNER, B. M. 2005. Animating gases with hybrid meshes. In *Proc. SIGGRAPH '05*.

FELDMAN, B. E., O'BRIEN, J. F., KLINGNER, B. M., AND GOKTEKIN, T. G. 2005. Fluids in deforming meshes. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 255–260.

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.

GALLIVAN, K., GRIMME, E., AND DOOREN, P. V. 1996. A rational Lanczos algorithm for model reduction. *Numerical Algorithms 12*, 33–63.

GORAL, C. M., TORRANCE, K. E., GREENBERG, D. P., AND BATTAILE, B. 1984. Modeling the interaction of light between diffuse surfaces. *Computer Graphics 18*, 3 (July), 213–222.

GRIMME, E. J. 1997. *Krylov Projection Methods For Model Reduction*. PhD thesis, Ohio State University.

GUGERCIN, S., AND ANTOULAS, A. 2004. A survey of model reduction by balanced truncation and some new results. *International Journal of Control 77*, 8, 748–766.

GUGERCIN, S., ANTOULAS, A., AND BEATTIE, C. A. 2006. A rational Krylov iteration for optimal H2 model reduction. In *Intl. Symposium on Mathematical Theory of Networks and Systems*.

GUPTA, M., AND NARASIMHAN, S. G. 2007. Legendre fluids: A unified framework for analytic reduced space modeling and rendering of participating media. In *2007 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 17–26.

HANRAHAN, P., SALZMAN, D., AND AUPPERLE, L. 1991. A rapid hierarchical radiosity algorithm. In *Proc of SIGGRAPH*.

HAUSER, K. K., SHEN, C., AND O'BRIEN, J. F. 2003. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, CIPS, Canadian Human-Computer Commnication Society, 247–256.

HOSSAIN, M.-S., AND BENNER, P. 2008. Projection-based model reduction for time-varying descriptor systems using recycled Krylov subspaces. *Proceedings in Applied Mathematics and Mechanics 8*, 1, 10081–10084.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. In *Proc. SIGGRAPH '03*.

JAMES, D. L., AND PAI, D. K. 2002. DyRT: Dynamic response textures for real time deformation simulation with graphics hardware. *ACM Transactions on Graphics 21*, 3 (July), 582–585.

JAMES, D. L., BARBIC, J., AND PAI, D. K. 2006. Precomputed acoustic transfer: output-sensitive, accurate sound generation for geometrically complex vibration sources. *ACM Transactions on Graphics 25*, 3 (July), 987–995.

KIM, T., AND DELANEY, J. 2013. Subspace fluid re-simulation. *ACM Transactions on Graphics 32*, 4.

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 *Proc. SCA '11*.

KLINGNER, B. M., AND SHEWCHUK, J. R. 2007. Agressive tetrahedral mesh improvement. In *Proceedings of the 16th International Meshing Roundtable*, 3–23.

KLINGNER, B. M., FELDMAN, B. E., CHENTANEZ, N., AND O'BRIEN, J. F. 2006. Fluid animation with dynamic meshes. *ACM Transactions on Graphics 25*, 3 (July), 820–825.

LI, J.-R. 2000. *Model Reduction of Large Linear Systems*. PhD thesis, Massachusetts Institute of Techology.

LIUA, C., YUAN, X., MULLAGURU, A., AND FAN, J. 2008. Model order reduction via rational transfer function fitting and eigenmode analysis. In *International Conference on Modeling, Identification and Control*.

LOOS, B. J., ANTANI, L., MITCHELL, K., NOWROUZEZAHRAI, D., JAROSZ, W., AND SLOAN, P.-P. 2011. Modular radiance transfer. *ACM Transactions on Graphics 30*, 6 (Dec.).

LOOS, B. J., NOWROUZEZAHRAI, D., JAROSZ, W., AND SLOAN, P.-P. 2012. Delta radiance transfer. In *Proceedings of the 2012 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM, New York, NY, USA, I3D 2012.

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

OLSSEN, K. H. A. 2005. *Model Order Reduction with Rational Krylov Methods*. PhD thesis, KTH Stockholm.

PAVLOV, D., MULLEN, P., TONG, Y., KANSO, E., MARSDEN, J., AND DESBRUN, M. 2011. Structure-preserving discretization of incompressible fluids. *Physica D: Nonlinear Phenomena 240*, 6, 443 – 458.

PENTLAND, A., AND WILLIAMS, J. 1989. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics (SIGGRAPH 89) 23*, 3 (July), 215–222. Held in Boston, Massachusetts.

PHARR, M., AND HUMPHREYS, G. 2004. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

PHILLIPS, J. R. 1998. Model reduction of time-varying linear systems using approximate multipoint Krylov-subspace projectors. *Computer Aided Design*, 96–102.

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.

SANDBERG, H., AND RANTZER, A. 2004. Balanced truncation of linear time-varying systems. *IEEE Transactions on Automatic Control 49*, 2, 217–229.

SAVAS, B., AND ELDÉN, L. 2009. Krylov subspace methods for tensor computations. Tech. Rep. LITH-MAT-R-2009-02-SE, Department of Mathematics, Linköpings Universitet.

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

SEWALL, J., MECKLENBURG, P., MITRAN, S., AND LIN, M. 2007. Fast fluid simulation using residual distribution schemes. In *Proc. Eurographics Workshop on Natural Phenomena*.

SI, H. 2007. A quality tetrahedral mesh generator and 3-dimensional Delaunay triangulator. *http://tetgen.berlios.de/*.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *Proc. SIGGRAPH '02*.

SLOAN, P.-P., LUNA, B., AND SNYDER, J. 2005. Local, deformable precomputed radiance transfer. *ACM Transactions on Graphics 24*, 3 (Aug.), 1216–1224.

SORKINE, O. 2006. Differential representations for mesh processing. *Computer Graphics Forum 25*, 4, 789–807.

STAM, J. 1999. Stable fluids. In *Computer Graphics (SIGGRAPH 99)*.

TREUILLE, A., LEWIS, A., AND POPOVIĆ, Z. 2006. Model reduction for real-time fluids. In *Proc. SIGGRAPH '06*.

WICKE, M., STANTON, M., AND TREUILLE, A. 2009. Modular bases for fluid dynamics. *ACM Transactions on Graphics 28*, 3.

ZATZ, H. R. 1993. Galerkin radiosity: a higher order solution method for global illumination. In *Proc. SIGGRAPH '93*, ACM.

ZHOU, Y. 2002. *Numerical Methods for Large Scale Matrix Equations with Applications in LTI System Model Reduction*. PhD thesis, Rice University.

A Fluid Stability

In this section, we provide a detailed discussion of the stability of our fluid simulation model in both the full and reduced spaces.

A.1 Full Space

Our discretization of the fluid equations in §5 is *stable*, meaning that the discrete versions of the partial differential equation do not inherently gain energy. To see why, let us consider the advection, diffusion, and projection steps separately. Energy is given by $\mathrm{E}(\mathbf{u}) = \frac{1}{2}||\mathbf{u}||^2_{\mathbf{V}(\mathbf{g})}$ and its time derivative is $\dot{\mathrm{E}} = \mathbf{u}^T\mathbf{V}(\mathbf{g})\dot{\mathbf{u}}$. Noting that we use oriented fluxes in (6), it is easy to see that the advection matrix $\mathbf{A} \otimes_2 \mathbf{f}$ is antisymmetric. Substituting for $\dot{\mathbf{u}}$ according to (8), we can see that the advection step exactly conserves energy.

$$\dot{\mathrm{E}} = \mathbf{u}^T(\mathbf{A} \otimes_2 \mathbf{f})\mathbf{u} = 0 \tag{A.1}$$

The graph Laplacian $\triangle$ has only positive eigenvalues, so substituting (9) for $\dot{\mathbf{u}}$ gives us

$$\dot{\mathrm{E}} = \mathbf{u}^T\Big(-\mu\mathbf{V}^{-1}(\mathbf{g})\triangle\Big)\mathbf{u} < 0. \tag{A.2}$$

Finally, in the projection step, we minimize the energy difference between the current velocities and the velocities corresponding to new, divergence free fluxes. Let $\mathbf{u}$ be the velocities before projection, $\mathbf{u}' = \mathbf{V}^{-1}(\mathbf{g})(\mathbf{P} \otimes_2 \mathbf{g})\mathbf{f}$ be the divergence-free velocities after projection, and $\mathbf{u}_\perp$ be their difference: $\mathbf{u} = \mathbf{u}' + \mathbf{u}_\perp$. (This is known as the Helmholtz-Hodge decomposition [Stam 1999].) We use $\|\cdot\|_{\mathbf{V}(\mathbf{g})}$ in our objective function (10), meaning $\mathbf{u}'$ and $\mathbf{u}_\perp$ are orthogonal in *energy space*. So the triangle inequality is tight: $\mathrm{E}(\mathbf{u}) = \mathrm{E}(\mathbf{u}') + \mathrm{E}(\mathbf{u}_\perp)$, which implies $\mathrm{E}(\mathbf{u}) \geq \mathrm{E}(\mathbf{u}')$, ensuring that the projection never gains energy.

Note that this does not mean that the method is unconditionally stable independent of the time integration method and time step chosen. However, we now use the above arguments to show that for our choice of integrator, the reduced simulation is in fact unconditionally stable.

A.2 Reduced Space

The definition of energy in the full space $\mathrm{E}(\mathbf{u}) = ||\mathbf{u}||^2_{\mathbf{V}(\mathbf{g})}$ leads naturally to a definition of reduced energy $\hat{\mathrm{E}}(\hat{\mathbf{u}}) = ||\hat{\mathbf{u}}||^2_{\hat{\mathbf{V}}(\hat{\mathbf{g}})}$, and all our stability arguments from §A.1 carry over directly with one exception: advection. In order to ensure energy-preserving advection, we must be careful in basis selection. Constructing the reduced equivalent of Eq. A.1, we see that the derivative in energy due to reduced-space advection is given by

$$\dot{\mathrm{E}} = \hat{\mathbf{u}}^T\mathbf{B}_p^T\Big((\mathbf{A} \otimes_2 \mathbf{B}_f) \otimes_2 \hat{\mathbf{f}}\Big)\mathbf{B}_u\hat{\mathbf{u}}. \tag{A.3}$$

If we set $\mathbf{B}_u = \mathbf{B}_p$, then $\hat{\mathbf{u}}^T\mathbf{B}_p^T = (\mathbf{B}_u\hat{\mathbf{u}})^T$, and since the full space matrix $(\mathbf{A}\otimes_2\mathbf{B}_f)\otimes_2\hat{\mathbf{f}}$ itself is antisymmetric, we once again are in possession of an energy-conserving discretization. Combined with analytic integration using matrix exponentiation, this basis choice results in an unconditionally stable system. To achieve $\mathbf{B}_u = \mathbf{B}_p$, we first compute the momentum and velocity bases independently. We then concatenate them and re-orthogonalize the result. We use this combined basis for both $\mathbf{B}_u$ and $\mathbf{B}_p$ in our fluid simulations, which guarantees that the simulations will preserve energy.

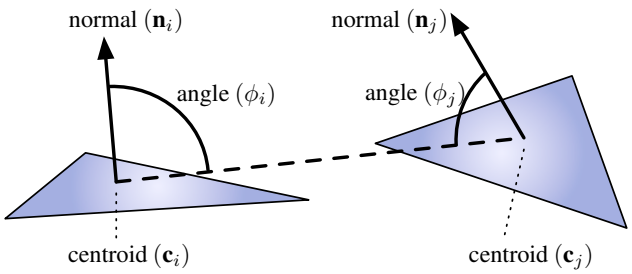


Figure B.1: *Geometric layout of the illumination variables on two triangles.*

B Radiosity Derivation

They key step in the model reduction of radiosity using our technique is the reduction of the form factor equation (Eq. 15). To model reduce this equation, we follow the process described in §4: we represent it as a collection of tensors, composed using tensor products, matrix inversion, and matrix roots. In this section we provide a detailed derivation of the tensors and sequence of operations shown in Table 2. We arrive at this sequence of operations by decomposing the radiosity equation (Eq. 14), so we will be describing the tensors and operations in Table 2 from bottom to top.

To evaluate the radiosity equation, we need to compute the matrix $\mathbf{I}-\rho\mathbf{F}$. As described in §8.1, we begin by unrolling $\mathbf{F} \in \mathbb{R}^{n\times n}$ into a vector $\mathbf{p} \in \mathbb{R}^{n^2}$ by re-indexing: $p_k = \mathbf{F}_{i,j}$, where $k = ni+j$ and n is the number of faces in the mesh. We also need the transpose index $k^T = nj + i$, so that $p_{k^T} = \mathbf{F}_{j,i}$. The tensor form of the radiosity equation (Eq. 14) is then:

$$\mathbf{b} = (\mathbf{I} - \mathbf{E} \otimes_2 \mathbf{p})^{-1}\mathbf{e}, \tag{B.1}$$

where $\mathbf{E}$ is the 3rd-order tensor that transforms the unrolled form factor vector $\mathbf{p}$ back into the form factor matrix $\mathbf{F}$ and left-multiplies $\mathbf{F}$ by the face albedos ρ.

Our task is now to compute $\mathbf{p}$, the vector of form factors. At this stage, we choose to separate the factor of area in the denominator of Eq. 15, making this stage a division of *area-free* form factors, denoted by $\mathbf{d}$, by the square roots of squared areas, where we denote squared areas by $\mathbf{a}$:

$$p_k = d_k/\sqrt{a_i}. \tag{B.2}$$

We implement this division using a tensor $\mathbf{P}$:

$$\mathbf{p} = (\mathbf{P} \otimes_2 \mathbf{a})^{-\frac{1}{2}}\mathbf{d}. \tag{B.3}$$

$\mathbf{a}$ is a simple function of the normals $\mathbf{n}$: $a_k = \mathbf{n}_i \cdot \mathbf{n}_i$. (Recall that $\mathbf{n}_i$ is the normal of the ith face, making it a 3-vector.) We define a tensor $\mathbf{N}$ to implement these dot products:

$$\mathbf{a} = \mathbf{N} \otimes_{1\ldots2} \mathbf{n}. \tag{B.4}$$

Returning to $\mathbf{d}$, the next factor we separate is visibility, which gives us the expression

$$d_k = c_k\mathrm{Vis}(i,j), \tag{B.5}$$

where $\mathbf{c}$ are *visibility-free* form factors. We define the visibility vector $\mathbf{v}$ such that $v_k = \mathrm{Vis}(i,j)$. We treat $\mathbf{v}(\mathbf{g})$ as a function of geometry directly, and reduce the computation of $\mathrm{Vis}(i,j)$ using a non-Galerkin method which we describe in more detail in §8.3.

Now we can write $\mathbf{c}$ as an element-wise product of two vectors:

$$c_k = \frac{1}{\pi} h_k h_{k^T}, \tag{B.6}$$

where

$$h_k = \frac{\mathbf{n}_i \cdot (\mathbf{c}_j - \mathbf{c}_i)}{||\mathbf{c}_i - \mathbf{c}_j||^2}. \tag{B.7}$$

We call $\mathbf{h}$ *half form-factors*, and define a tensor $\mathbf{C}$ implementing Eq. B.6:

$$\mathbf{c} = \mathbf{C} \otimes_{1\ldots2} \mathbf{h}. \tag{B.8}$$

Next, we rewrite both parts of the quotient in Eq. B.7:

$$h_k = \frac{s_k}{r_k}, \tag{B.9}$$

where

$$r_k = ||\mathbf{c}_i - \mathbf{c}_j||^2 \tag{B.10}$$

and

$$s_k = \mathbf{n}_i \cdot (\mathbf{c}_j - \mathbf{c}_i). \tag{B.11}$$

$\mathbf{r}$ is a vector of squared distances between face centroids. $\mathbf{s}$ is a vector of *scaled cosines*: $\mathbf{s}_k$ is the cosine of the angle between $\mathbf{n}_i$ and $\mathbf{c}_j - \mathbf{c}_i$, multiplied by face area and the distance between face centroids. We use a tensor $\mathbf{H}$ to construct a diagonal matrix, with $\mathbf{r}$ as its entries, which we invert to find $\mathbf{h}$:

$$\mathbf{h} = (\mathbf{H} \otimes_2 \mathbf{r})^{-1} \mathbf{s}. \tag{B.12}$$

While $\mathbf{s}$ is polynomial in $\mathbf{g}$, and so it would be possible to compute it directly as a from $\mathbf{g}$ by contracting a single tensor, we can reduce the polynomial degree of this system from 3 to 2 (and the maximum tensor order from 4 to 3) by decomposing $\mathbf{s}$ one step further. Note that Eq. B.11 is bilinear in $\mathbf{n}$ and $\mathbf{c}$, the latter of which is linear in $\mathbf{g}$. Therefore, we can define a tensor $\mathbf{S}$ such that

$$\mathbf{s} = \mathbf{S} \otimes_1 \mathbf{n} \otimes_2 \mathbf{g}. \tag{B.13}$$

This leaves us with only $\mathbf{n}$ to compute. The normals are given by

$$\mathbf{n}_i = (\mathbf{g}_{i,2} - \mathbf{g}_{i,0}) \times (\mathbf{g}_{i,1} - \mathbf{g}_{i,0}), \tag{B.14}$$

where $\mathbf{g}_{i,\ell}$ is the ℓth vertex of face i and $\times$ is the vector cross product. This expression is a low-degree (quadratic, in fact) polynomial in geometry, and so we can complete the decomposition with a final tensor $\mathbf{N}$ such that

$$\mathbf{n} = \mathbf{N} \otimes_{1\ldots2} \mathbf{g} \tag{B.15}$$

With the tensors defined above, we can now compute radiosity as described in Table 2 given only the scene geometry $\mathbf{g}$ and the incident illumination $\mathbf{e}$.