Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2016)
Ladislav Kavan and Chris Wojtan (Editors)

Compressing Fluid Subspaces

Aaron Demby Jones[1] Pradeep Sen[1] and Theodore Kim[1,2]

[1]University of California, Santa Barbara
[2]Pixar Animation Studios

Abstract

Subspace fluid simulations, also known as reduced-order simulations, can be extremely fast, but also require basis matrices that consume an enormous amount of memory. Motivated by the extreme sparsity of Laplacian eigenfunctions in the frequency domain, we design a frequency-space codec that is capable of compressing basis matrices by up to an order of magnitude. However, if computed naïvely, decompression can be highly inefficient and dominate the running time, effectively negating the advantage of the subspace approach. We show how to significantly accelerate the decompressor by performing the key matrix-vector product in the sparse frequency domain. Subsequently, our codec only adds a factor of three or four to the overall runtime. The compression preserves the overall quality of the simulation, which we show in a variety of examples.

Categories and Subject Descriptors (according to ACM CCS): I.6.8 [Computer Graphics]: Simulation and Modeling—Types of Simulation–Animation, Computer Graphics

1. Introduction

Subspace methods, also known as model reduction or reduced order methods, have experienced recent success at accelerating fluid simulations in computer graphics. They have been applied to problems as varied as real-time interaction [TLP06], real-time control [BP08], and efficient re-simulation [KD13]. However, one of the primary drawbacks of this approach is that the speed increase comes at the cost of much larger memory requirements. Subspace simulations easily consume tens of gigabytes of memory when dealing with high-resolution scenes.

Memory consumption is a known challenge with subspace methods, and compression strategies have been developed for other applications that benefit from the approach, such as deformable models [SILN11] and sound synthesis [LAJJ14]. However, no practical subspace compression method has been developed for fluid simulations in computer graphics. The closest analog is a method for domain decomposition [WST09], which is a complementary approach that exploits very low-frequency regularities. Ideally, both domain decomposition and compression could be applied simultaneously to alleviate memory issues.

The large memory footprint occurs because a basis matrix $\mathbf{U} \in \mathbb{R}^{3N \times r}$ is needed at runtime. Here, N is the number of grid cells in the fluid simulation, and r is the size of some lower-dimensional subspace. In general, $r \ll N$, and the simulation is efficient because time integration can be performed in the r-dimensional subspace instead of the much larger N-dimensional space. However, each column of $\mathbf{U}$ is essentially a copy of the entire simulation grid, so even if r is small, $\mathbf{U}$ can be very large. For example, if $N = 256^3$ and $r = 50$, $\mathbf{U}$ already consumes over 10 GB of memory.

In this paper, we present a compression method that reduces the size of $\mathbf{U}$ by up to an order of magnitude. Motivated by the fact that Laplacian Eigenfunctions [DWLF12] exhibit extreme sparsity in the Fourier domain, we apply a frequency transform to $\mathbf{U}$. We then design an adaptive quantization strategy to arrive at a JPEG-like 3D compression scheme that encodes our final, sparse representation.

Once compressed, the basis matrix $\mathbf{U}$ must be decompressed at runtime. We show that our approach enables efficient decompression by performing the matrix-vector product sparsely in the frequency domain. The final running time is within a factor of three or four of the original, uncompressed subspace simulation.

Our contributions are as follows:

- A novel frequency domain compressor for subspace matrices that uses adaptive quantization to reduce memory usage by up to an order of magnitude.
- A decompressor that allows matrix-vector multiplies to be performed efficiently in the frequency domain.
- Visual and numerical evidence that the proposed codec results in minimal artifacts and preserves the generality of the subspace.

2. Previous Work

Efficiently simulating visually complex fluids is a continuing challenge in computer graphics. For an overview of recent techniques, we refer the reader to the book by Bridson [Bri15]. Many different approaches have been taken for improving the performance of these

simulations, such as spatial coarsening [LGF04, ATW13], vortex sheet [BKB12, PTG12], position-based [MMCK14], and turbulence methods [TKP13]. In this work, we specifically focus on the subspace approach. This acceleration technique has been applied successfully to a variety of physical phenomena, such as hyperelastic solid deformation [BJ05,AKJ08,vTSSH13,WMW15,YLX*15], sound synthesis [OSG02, LAJJ14, LFZ15], and cloth simulation [HTC*14,XUC*14].

It has also had some success in fluid simulation for computer graphics, starting with the work of Treuille et al. [TLP06]. From there, the approach has been applied to control [BP08] and resimulation [KD13] problems, and extended to handle arbitrary tetrahedral meshes [LMH*15] and deforming domains [SSW*13]. Many of these works feature 2D results most prominently [BP08, GKSB15], which we expect is in part due to the memory limitations we address here. The work of Wicke et al. [WST09] also aimed to address this limitation by providing a modular basis that can be tiled throughout a domain. Our approach is complementary; the memory footprint of the modular tiles could be further reduced by applying the compression techniques from this paper.

As mentioned in §1, the problematic need for large subspace matrices has been addressed in sound [LAJJ14] and deformation [SILN11] applications, but we are unaware of any approaches that address this problem for fluids. Inspired by the success of Fourier-based turbulence visualization techniques [YL95,TBR*12] and their surprising efficacy when compressing Laplacian eigenfunctions [DWLF12] (see §4.1), we present a 3D, transform-based scheme [Say12] that is able to discover sparse, memory-efficient approximations to arbitrary subspace velocity matrices. Our transform-based technique uses the visualization-oriented approaches as a starting point, and adds critical features such as fast matrix-vector products and adaptive quantization to make it practical for subspace simulation.

3. Subspace Preliminaries

We will begin with a brief description of the subspace approach in order to identify the source of its large memory requirements. Subsequently, we will construct a set of requirements that a compression algorithm should fulfill.

While many subspace fluid simulation algorithms have been developed for computer graphics, they all use the Galerkin projection approach [Bat96]. A large-dimensional velocity field that contains $3N$ degrees of freedom over N grid cells is projected into a smaller, r-dimensional velocity subspace using a static linear basis. We denote this change-of-basis matrix as $\mathbf{U} \in \mathbb{R}^{3N \times r}$, where each column of $\mathbf{U}$ is itself a velocity field. Specifically, we can project a vector $\mathbf{u} \in \mathbb{R}^{3N}$ down to its reduced-order counterpart $\mathbf{q} \in \mathbb{R}^r$ by computing the product $\mathbf{U}^T\mathbf{u} = \mathbf{q}$. In a well-designed subspace simulation, the velocity fields spanned by the columns of $\mathbf{U}$ will be sufficient to capture the space of fields that interest the user. This projection often occurs at the beginning of each timestep, as a force vector $\mathbf{f}$ that contains important information such as user inputs, buoyancy, and vorticity confinement forces must be projected into the subspace.

Once this projection has been performed, the equations of motion can be integrated over this reduced number of variables very quickly. This can be done using exponential integration [TLP06], Stable Fluids-like semi-Lagrangian advection coupled with Chorin projection [KD13], variational integration [LMH*15], or even simple explicit Euler integration [DWLF12]. Since $\mathbf{U}$ is agnostic to the underlying integrator, our compression scheme can be applied to any of these methods.

Once time integration has been performed, the change-of-basis must be reversed so that the results can be displayed. This is known as the "velocity reconstruction" stage [TLP06, DWLF12], and several options are available. The most direct method is a *full* reconstruction where the $3N$-dimensional velocity field is reconstructed directly, i.e. $\mathbf{u} = \mathbf{U}\mathbf{q}$. This method is preferred if a dense particulate is present in most of the simulation domain. Velocity values are then needed essentially everywhere to advect the particulate forward in time. Alternatively, if immersed media such as leaves are being simulated, a *sparse* reconstruction is possible because velocities are only needed at the grid cells containing leaves. These velocities $\mathbf{v} \in \mathbb{R}^3$ can be obtained by extracting the three rows of $\mathbf{U}$ that correspond to each cell of interest, $\mathbf{U}_\mathbf{v} \in \mathbb{R}^{3 \times r}$, and computing the product $\mathbf{v} = \mathbf{U}_\mathbf{v}\mathbf{q}$. If the number of leaves is much smaller than N, this approach can be highly efficient.

The main drawback of the subspace approach is that the matrix $\mathbf{U}$ can be very large, as each of its columns is itself a velocity field. A compression method that reduces the size of $\mathbf{U}$ is most useful if it efficiently supports at least three different operations:

- **Projection:** The time needed to compute the full $\mathbf{U}^T\mathbf{u} = \mathbf{q}$ matrix-vector product should not prohibitively increase due to the presence of a decompressor.
- **Dense reconstruction:** Conversely, $\mathbf{U}\mathbf{q} = \mathbf{u}$ must also be fast.
- **Batched random access:** In order to support sparse reconstruction, it should be possible to query the velocity field at a set of random points on the simulation grid. In addition to efficient sparse reconstruction, this feature is also needed to support certain types of time integration, e.g. cubature-based semi-Lagrangian schemes [KD13].

With these requirements in mind, we can design our codec, which we present in the next section.

4. A Subspace Compression Scheme

4.1. Compression Basis Selection

In order to design a compression scheme for the velocity fields that comprise the columns of $\mathbf{U}$, we must first select a transform basis that would ideally result in extremely sparse fields. Both discrete cosine transform (DCT) [YL95] and wavelet [GWGS02, TBR*12] bases have been successfully used in the past to compress scalar volumes, so they are promising candidates for velocity fields. We choose to use DCT because we observe that in the special case of Laplacian Eigenfunctions [DWLF12], they actually yield ideal compression. The eigenfunctions inside a closed 3D box take the general form:

$$\begin{aligned} \mathbf{u}_x(k_1,k_2,k_3) &= \kappa_x \sin(k_1 x)\cos(k_2 y)\cos(k_3 z) \\ \mathbf{u}_y(k_1,k_2,k_3) &= \kappa_y \cos(k_1 x)\sin(k_2 y)\cos(k_3 z) \\ \mathbf{u}_z(k_1,k_2,k_3) &= \kappa_z \cos(k_1 x)\cos(k_2 y)\sin(k_3 z), \end{aligned} \tag{1}$$

where $\mathbf{u}_x, \mathbf{u}_y$, and $\mathbf{u}_z$ respectively represent the x, y, and z components of a velocity field. The k_i coefficients determine the frequency content of the field, and the three κ terms are scaling coefficients that are derived from k_i.

We make the straightforward observation that the spatially varying components of each of these functions are purely trigonometric functions, so by applying appropriately interleaved DCTs and discrete sine transforms (DSTs) to these fields, they can be reduced to delta functions, regardless of their spatial frequency. In the notation of Long and Reinhard [LR09], if we use $\mathcal{F}_{\text{SCC}}$ to denote a DST in the x direction and DCTs in the y and z directions, we obtain,

$$\mathcal{F}_{\text{SCC}}\left[\kappa_x \sin(k_1 x)\cos(k_2 y)\cos(k_3 z)\right] = \kappa_x \delta(k_1, k_2, k_3), \quad (2)$$

where $\delta(k_1, k_2, k_3)$ is a delta function in SCC frequency space located at the (k_1, k_2, k_3) grid cell. Similar transforms, e.g. $\mathcal{F}_{\text{CSC}}$ and $\mathcal{F}_{\text{CCS}}$ can be used to generate delta functions for $\mathbf{u}_y$ and $\mathbf{u}_z$. Thus, any eigenfunction can be compressed down to three integers (the k_is) and three floats (the κ terms).

Asymptotically, this mixed DCT/DST losslessly transforms an $O(N)$ eigenfunction down to $O(1)$; no sparser representation is possible. This result only applies to the ideal case of analytic divergence-free velocity fields defined on the interior of a box. However, the high compression it achieves is encouraging, and motivates our further use of DCT to compress more general divergence-free fields.

4.2. DCT-Based Compression

Following from the previous discussion, we design a DCT-based, JPEG-like compression scheme. Each column of $\mathbf{U}$ represents a vector field, and the columns are usually constructed using an SVD. Since this SVD has already minimized the amount of redundant information between columns, we compress them separately. Each column contain x, y and z velocity components, and we extract each of these components and compress them separately. Thus, if we describe the encoding procedure for a single scalar field, it can be applied to each velocity component of each column of $\mathbf{U}$.

Analogously to JPEG, given a 3D scalar field, we decompose it into small blocks of size $b \times b \times b$, adding continuous extra padding in the case that one or more resolutions are not evenly divisible by b. We then perform a 3D DCT on each block. In anticipation of quantization, the result is then normalized so that the largest frequency-domain value maps to the largest signed value for a 32-bit integer.

Adaptive Quantization: After transforming the signal to the frequency domain and normalizing the coefficients, JPEG then performs an element-wise division of the coefficients using a 2D quantization matrix in order to increase the likelihood that they will quantize to zero. In the JPEG standard, this matrix is adjusted depending on the quality setting; higher quality settings will have lower values to preserve more detail after dividing by the matrix, while lower quality settings will have higher values in the matrix to suppress more coefficients. For example, the following is the JPEG matrix that corresponds to 50% quality:

$$\mathbf{Q}_{\text{2D}} = \begin{bmatrix} 16 & 11 & 10 & 16 & \dots \\ 12 & 12 & 14 & 19 & \dots \\ 14 & 13 & 16 & 24 & \dots \\ 14 & 17 & 22 & 29 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \in \mathbb{R}^{b \times b}. \quad (3)$$

We index the matrix entries $\mathbf{Q}_{\text{2D}}(u, v)$ such that the upper-left corner, $\mathbf{Q}_{\text{2D}}(0,0) = 16$, corresponds to the DC component. The entries of $\mathbf{Q}_{\text{2D}}$ were obtained from perceptual data [Say12], and in general, higher frequency components have larger entries in order to suppress these coefficients which tend to carry less information than those in the lower frequencies.

Since we are not working with 2D color data but rather with 3D velocity fields, we need to construct a 3D version of this matrix, $\mathbf{Q}_{\text{3D}} \in \mathbb{R}^{b \times b \times b}$. A close inspection of all of the complete matrix corresponding to Eqn. 3 suggests that $\mathbf{Q}_{\text{2D}}(u, v) \propto u + v$, so a straightforward first attempt is:

$$\mathbf{Q}_{\text{3D}}(u, v, w) = 1 + u + v + w. \quad (4)$$

Other applications [YL95] have used similar reasoning to arrive at similar matrices. However, while the 2D case has a suite of $\mathbf{Q}_{\text{2D}}$ matrices at its disposal that correspond to different levels of perceived visual quality, this data does not generalize to non-chromatic, 3D velocity data.

We instead propose to automatically generate a variety of different $\mathbf{Q}_{\text{3D}}$ matrices during the compression stage. For each $b \times b \times b$ block, the energy is likely to reside in different frequencies, so we generate a custom matrix,

$$\mathbf{Q}_{\text{3D}}(u, v, w) = (1 + u + v + w)^{\gamma}, \quad (5)$$

where γ is a parameter that is adjusted per block. Linear models are also possible, e.g. $\mathbf{Q}_{\text{3D}}(u, v, w) = \gamma\,(1 + u + v + w)$, but preliminary experiments found that this model was too simple to yield useful compressions. Analogous to the 2D case, the user specifies a quality parameter p. In 3D, we interpret p as the percentage of the original energy that should be preserved. Each block then performs a bisection search over the range $\gamma \in [0, n]$, where $n = 32$ is the number of bits that were used for normalization prior to quantization. Higher values of n are essentially meaningless, as they damp everything except the DC component to zero. In practice, we found that this bisection search terminates within a very small tolerance of the desired energy preservation after at most 8 iterations.

This approach provides a custom quantization matrix for each block while maintaining an approximately constant energy loss per block. Important high-frequency components are preserved when they are present, while smoother, low-frequency blocks are still aggressively compressed. This block-varying value γ must then be computed by the encoder and provided to the decoder in the encoded bytestream. For an $8 \times 8 \times 8$ block, the memory footprint of a single additional scalar γ per block is negligible. We compare this strategy to a uniform non-adaptive quantization approach in §5.

Flattening and Encoding: After quantization, we convert the 3D array into a 1D array and perform run-length encoding [YL95, Say12]. No novel strategy needs to be devised for this component.

The 3D to 1D conversion is performed in a zigzag pattern that is a straightforward 3D extension of the usual 2D JPEG ordering, which tries to group coefficients with similar sizes together in the bytestream. In our case, the entries of $\mathbf{Q}_{3D}(u,v,w)$ are arranged in increasing order of their sum $u+v+w$, which effectively clusters components with approximately the same frequency. The results are then run-length encoded in order to discover long runs of zeros.

4.3. Subspace Decompression

Batched Random Access: The block-wise compression scheme we have described supports the batched random access requirements from §3 at runtime. For a single random access, the block containing the cell of interest is decompressed, which means the other $b \times b \times b - 1$ entries are potentially decoded needlessly. However, the coherency of the underlying incompressible flow tends to cluster cells of interest in the same blocks, so we did not find that this extra overhead created a major bottleneck.

The decompression proceeds in two stages, where an initial pass assembles the batch of requested cells and determines which blocks need to be decompressed. A second pass then decompresses the actual blocks. By consolidating the cell requests, it is guaranteed that no block is ever redundantly decompressed twice.

Projection and Reconstruction: The fast projection and reconstruction requirements from §3 are not as straightforward. A naïve strategy is to decompress the entire matrix $\mathbf{U}$ for each projection and reconstruction. For a given block size $B = b \times b \times b$, this results in $\frac{3N \times r}{B}$ DCTs and IDCTs at every timestep. These transforms dominate the running time (Table 1), and largely negate the performance gains of the subspace approach. While memory savings are achieved, the speed-memory tradeoff is unacceptable.

However, we observe that both of the matrix-vector products $\mathbf{Uq} = \mathbf{u}$ and $\mathbf{U}^T\mathbf{u} = \mathbf{q}$ can be performed *sparsely in the frequency domain*. The projection operator then only performs a DCT on $\mathbf{u}$, not all r columns of $\mathbf{U}$. This operation is permissible because the DCT is a unitary transform, and therefore preserves inner products. Specifically, if $\mathbf{x}$ and $\mathbf{y}$ are vectors in the spatial domain and $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ are their counterparts in the frequency domain, $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \hat{\mathbf{x}}, \hat{\mathbf{y}} \rangle$, where $\langle \cdot, \cdot \rangle$ denotes the usual Euclidean inner product.

We define the following notation to describe the advantages of this approach. The frequency domain version of a quantity is denoted with a hat, e.g., $\mathbf{U}$ with DCT applied to each column is $\widehat{\mathbf{U}}$. The *lossy, compressed* version of $\widehat{\mathbf{U}}$, where near-zero values have been quantized to zero, is denoted $\widehat{\mathbf{C}}$. The spatial domain version of $\widehat{\mathbf{C}}$ is correspondingly $\mathbf{C}$. In essence, our compression scheme has introduced the approximations $\mathbf{U} \approx \mathbf{C}$ and $\widehat{\mathbf{U}} \approx \widehat{\mathbf{C}}$.

Using the unitary property, we can see that if we use DCT to transform $\mathbf{u}$ to $\hat{\mathbf{u}}$, then $\mathbf{U}^T\mathbf{u} = \mathbf{q}$ is equivalent to $\widehat{\mathbf{U}}^T\hat{\mathbf{u}} = \mathbf{q}$. Using the compressed versions will yield a result slightly different from $\mathbf{q}$, but the same relation holds: $\mathbf{C}^T\mathbf{u} = \widehat{\mathbf{C}}^T\hat{\mathbf{u}} \approx \mathbf{q}$. The naïve approach spends most of its time transforming $\widehat{\mathbf{C}}$ to $\mathbf{C}$, but by constructing $\hat{\mathbf{u}}$, we can avoid this stage altogether. Replacing the IDCT over all r columns of $\widehat{\mathbf{C}}$ with a single DCT of $\mathbf{u}$ is significant, because even for a modest $r = 10$, the number of transforms is reduced by an order of magnitude. Additionally, the $\widehat{\mathbf{C}}^T\hat{\mathbf{u}}$ product can now exploit the sparsity of $\widehat{\mathbf{C}}$ that was discovered by the compression stage. Since $\widehat{\mathbf{C}}$ is static over the lifetime of a simulation, the location of all the zero entries can be cached, and these multiplies can be skipped.

A fast reconstruction strategy then follows: $\widehat{\mathbf{C}}\mathbf{q} \approx \hat{\mathbf{u}}$. The sparsity of $\widehat{\mathbf{C}}$ can again be exploited here, as each column $\widehat{\mathbf{C}}$ is scaled by an entry of $\mathbf{q}$, and all the multiplies with respect to zeroes can again be skipped. Once $\hat{\mathbf{u}}$ is known, an IDCT can be performed on it once, and IDCTs over all r columns of $\widehat{\mathbf{C}}$ is again avoided.

For a $\mathbf{C}$ matrix containing $3N \times r$ non-zero entries, after taking the complexity of the DCTs and IDCTs into account, naïve projection and reconstruction each take $O\left(3N \times r + \frac{3N \times r}{B} B \log B\right) = O\left((3N \times r) \log B\right)$ time. Using our approach, a $\widehat{\mathbf{C}}$ containing S non-zero entries instead takes $O\left(S + \frac{3N}{B} B \log B\right) = O(S + 3N \log B)$ time. The r factor has been removed as a multiplier of $\log B$, and replaced with the additive S term. As seen in Table 1, this results in speedups that approach an order of magnitude.

4.4. Discussion

We have described one possible scheme for compressing subspace fluid basis matrices. Several initially promising possibilities were also investigated, but ultimately discarded.

The motivating example from §4.1 uses mixed DST and DCT to achieve ideal compression, whereas we only use DCT. The use of DST was investigated as well, but was not found to give superior results. Unless the velocity values along a block border are all exactly zero, i.e. the block has Dirichlet boundaries along all its walls, the DCT consistently yields superior results.

The $\mathbf{U}$ matrix is usually constructed using an SVD [TLP06, KD13] or an eigenanalysis [DWLF12, LMH*15]. Therefore, corresponding singular values or eigenvalues are usually available for each column of $\mathbf{U}$. These values could be used to guide the compressor, e.g., by allowing columns with unimportant singular values σ_i to be compressed more aggressively. However, we found that the relationship between σ_i and visual quality is not straightforward. Especially during re-simulation, columns that had unimportant σ_i during the initial analysis can obtain large coefficients in $\mathbf{q}$. In such cases, the aggressive compression can become visible.

Prior to compression, an additional SVD could be run on each $b \times b \times b$ block in $\mathbf{U}$ to determine if there is a superior coordinate system for compression other than the canonical x, y, and z axes. However, the per-block rotation that this introduces breaks the fast matrix-vector multiply described in §4.3. Thus, we put this aside in favor of the fast multiplies, but a method that supports both operations is an interesting direction for future work.

5. Results

We tested our compression scheme on several subspace fluid re-simulation scenarios generated by the open-source package Zephyr [KD13]. The fluid simulation data were generated using a Preconditioned Conjugate Gradient (PCG) solver with a Modified Incomplete Cholesky preconditioner [Bri15]. The codebase was implemented in C++ and tests were run on a 12-core, 2.66 GHz Mac Pro

	Naïve, e.g., $\mathbf{Cq}$	Sparse, e.g., $\widehat{\mathbf{C}}\widehat{\mathbf{u}}$	Speedup
Plume	72.0s	8.7s	8.3X
Sphere	74.9s	7.6s	9.9X
Fan	72.6s	12.9s	5.6X

Table 1: Timings of naïve projections vs. sparse projections. The sparse projection is significantly faster, and dramatically reduces the overhead of using the compressed representation of **U**. These timings represent the average time to perform *both* the projection and reconstruction stages in a single timestep.

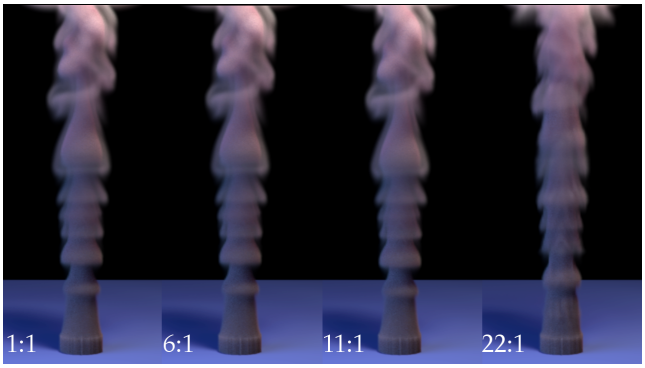

Figure 1: **Plume scene:** The overall motion and visual quality of the plume is preserved until the compression ratio is increased to approximately 22 : 1. The 1 : 1 corresponds to using the original **U** matrix.

with 96 GB RAM. For the DCT and IDCT, we used FFTW [FJ05], and Eigen [GJ*10] was used for other linear algebra operations.

In all of our simulations, we set $b = 8$, so $8 \times 8 \times 8$ blocks were used. Block sizes of $b = 4$ and $b = 16$ were also tested, but we found that smaller blocks redundantly captured the same low frequency information, while larger blocks lessened the likelihood of finding smooth regions that could be compressed aggressively.

In all of our subspace simulations, we used the matrix-vector product strategy from §4.3, which accelerated the computation significantly (Table 1). Without this acceleration, the subspace simulations ran almost at parity with the original full-space simulations, invalidating any speed advantages of the subspace approach. On average, our sparse product ran roughly 3–4 times as slow as the uncompressed matrix vector product. Asymptotically, our sparse product can have a superior running time, as it does not need to touch all $3N \times r$ entries in the matrix. Our scenes did not achieve sufficient sparsity to demonstrate this superiority, but we expect that as compression methods improve, multiplying against $\widehat{\mathbf{C}}$ will eventually become faster than multiplying with **C**.

Plume scene: We simulated the buoyant flow of a plume in a scene containing no obstacles, as shown in Figure 1. The original simulation resolution is $200 \times 266 \times 200$, was run for 150 frames, and took 09h 48m 49s (3.92 minutes per frame).

The SVD to construct the subspace from this data took 07h 05m 40s, and the compressing the subspace took at most 02h 15m 59s (Table 2). Constructing and compressing the subspace is therefore roughly at parity with running the entire simulation a second time. However, once the subspace has been constructed once, we can run new simulations very quickly.

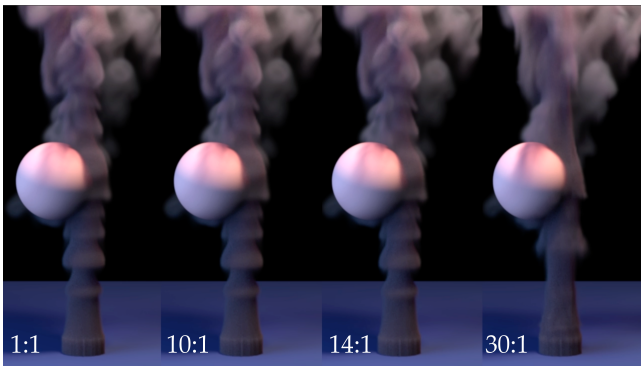

Figure 2: **Sphere scene:** The motion and visual quality remains high until approximately 14 : 1 compression. At 30 : 1, the differences are very significant.

We found that the subspace could be compressed by roughly an order of magnitude (11 : 1) before the visual quality began to degrade. However, we also noted that the compression scheme appears to degrade relatively gracefully. For higher compression rates, the motion gradually deviates from the uncompressed motion, and JPEG-like block artifacts begin to appear.

Sphere scene: Next, we simulated the same plume in the presence of a sphere obstacle. The original simulation resolution is $200 \times 266 \times 200$, was run for 150 frames, and took 10h 37m 50s (4.25 minutes per frame). The time to construct and compress the subspace, respectively 09h 17m 19s and a maximum of 02h 34m 30s, was again found to be roughly at parity with running the simulation a second time.

Surprisingly, we found that this subspace compressed slightly better than the plume scene (14 : 1) before the visual quality began to degrade. The expectation was that the sphere boundary would create a discontinuity in the velocity field that the compressor would have trouble capturing. Instead, the interior of the static obstacle created a region of constant (zero) velocity, and also induced the formation of smooth, near-zero regions in its vicinity. Rather than create a discontinuity containing many high frequencies, these constant and smooth regions mostly contained low-frequencies that the compressor could easily leverage. No-slip boundaries were used along the surface of the obstacle; if free-slip were used instead, the anticipated discontinuities may still appear.

For this scene, we also compared our adaptive quantization strategy to a uniform, non-adaptive approach. There is no canonical 3D version of Eqn. 3, so we instead selected a uniform γ that produced an equivalent energy loss in the highest frequency component, i.e., we set γ such that it matched the 32-bit equivalent of the lower-right hand entry of Eqn. 3. This strategy still produced a 7 : 1 compression, but our adaptive strategy was able to achieve a higher compression of 14 : 1.

Out-of-core Comparison: We compared our performance to an

Plume	Uncompressed	6 : 1	8 : 1	**11 : 1**	13 : 1	22 : 1
Time per frame	4.5s	19.9s	17.9s	16.1s	15.2s	12.8s
Compression preprocess	N/A	02h 07m 55s	01h 53m 48s	02h 07m 55s	02h 12m 17s	02h 15m 59s
Sphere	Uncompressed	10 : 1	**14 : 1**	16 : 1	23 : 1	30 : 1
Time per frame	5.4s	17.1	15.1s	14.4s	13.0s	12.2s
Compression preprocess	N/A	02h 01m 38s	02h 20m 14s	02h 07m 59s	02h 34m 30s	02h 27m 59s
Fan	Uncompressed	5 : 1	**6 : 1**	8 : 1	11 : 1	29 : 1
Time per frame	5.7s	24.4s	23.2s	19.8s	18.3s	14.8s
Compression preprocess	N/A	01h 09m 37s	01h 08m 25s	01h 26m 57s	02h 18m 45s	02h 11m 51s

Table 2: Compression performance for each of the three scenes. The "sweet spot" for each scene that achieves a good balance between compression and visual quality is shown in gray.

uncompressed simulation that does not fit in core by running the 14 : 1 version of the sphere scene on a 2-core, 1.8 GHz Macbook Air with 8 GB RAM. On this system, the full space simulation ran at 286.5s per frame, while our compressed subspace simulation ran at 71.7s, yielding a speedup of roughly 4.0×.

The uncompressed subspace simulation requires over 70 GB of memory, and immediately started to swap on the 8 GB system. It ran for over 17 hours without completing a single frame, at which time it had more than doubled the running time of the original full space simulation, and was terminated.

Fan scene: Finally, we simulated smoke being stirred by a fan. The original simulation resolution is $266 \times 200 \times 200$, was run for 150 frames, and took 12h 18m 05s (4.92 minutes per frame). The subspace construction and compression times were 08h 13m 22s and at most 02h 18m 45s.

The moving obstacle scene achieved a compression ratio of 6 : 1 before the quality began to degrade. Here, the reduced compression we expected to see in the sphere scene appeared. Instead of having a smoothing effect on the velocity field, the obstacle creates new, high-frequency velocities that are more difficult to compress. However, we did not choose to leverage any knowledge of the obstacle motion during compression, which could be a promising avenue for finding sparser representations.

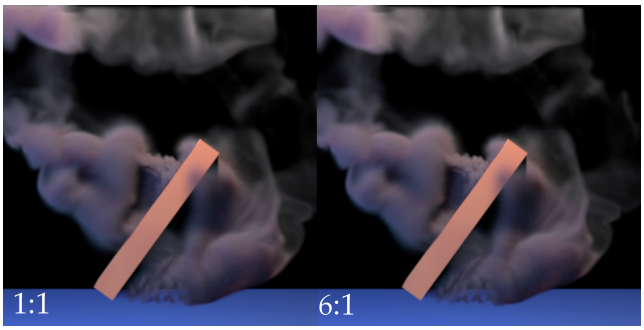

Figure 4: **Fan Re-Simulation:** With vorticity confinement increased by a factor of ten, the novel turbulent detail that is introduced remains intact, even after basis compression.

6. Discussion and Conclusions

In order to determine whether the compression compromised the generality of the subspace, we also ran re-simulations [KD13] on each of our scenes using a variety of different simulation settings. In the plume scene, we both reduced the vorticity confinement constant to zero and doubled the number of total timesteps (see video), for the sphere scene we halved the buoyancy constant (see video) and in the fan scene, we increased the vorticity confinement constant by a factor of ten (Fig. 4). The re-simulations were all run on the "sweet-spot" compression ratios that are highlighted in Table 2. In all cases, the overall motion was preserved, and we did not observe any significant visual artifacts compared to the uncompressed subspace re-simulation.

In order to better understand the error introduced by the compression, we plotted the relative L_2 error between the $\mathbf{q}$ vectors obtained by the uncompressed and compressed simulations. While it is difficult to tell whether a given compressed $\mathbf{U}$ is too aggressive *a priori*, an overaggressively-compressed $\mathbf{U}$ quantitatively corresponds to a 10^{-1} relative error appearing early in the simulation. In this case, the compression error exceeds that of the cubature integration scheme [KD13], and begins to visually dominate.

Conclusions and Future Work: We have presented a compression method for subspace fluid simulations that is able to reduce the size of the $\mathbf{U}$ matrix by up to an order of magnitude. The most immediate direction for future work is the development of algorithms that are able to reduce this matrix size even further. As seen in Table 2, as the sparsity of the compressed representation increases, the speed of the matrix-vector product improves. According to our asymptotic analysis, it should eventually surpass the performance of the direct $\mathbf{Uq}$ product. Therefore, finding new bases that possess the same unitary property as the DCT while increasing sparsity should yield algorithms that are efficient in both memory and time.

Lossless compression schemes such as Huffman coding could also be investigated, but these would not directly increase the sparsity of the representation, so the tradeoff between memory savings and decompression speed would need to be balanced. Finally, the method we have presented assumes that the velocity field is defined on a regular grid. An approach that can be applied to unstructured, tetrahedral meshes [LMH*15, SSW*13] would also be welcome.

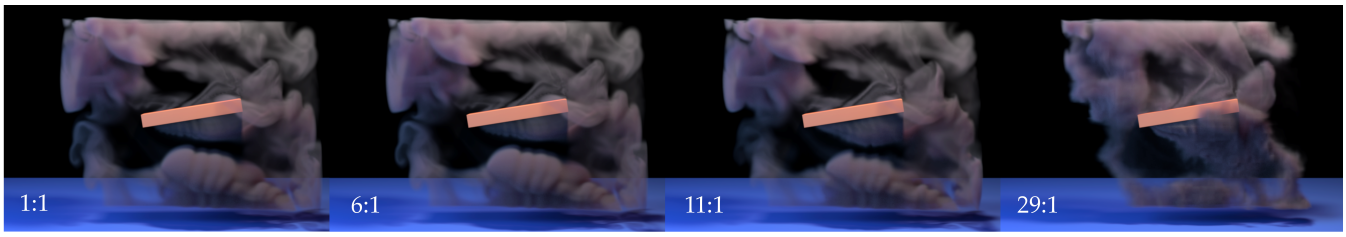

Figure 3: **Fan scene:** The quality is preserved at 6 : 1, but at 11 : 1, the motion begins to change, and at 29 : 1, artifacts begin to appear.

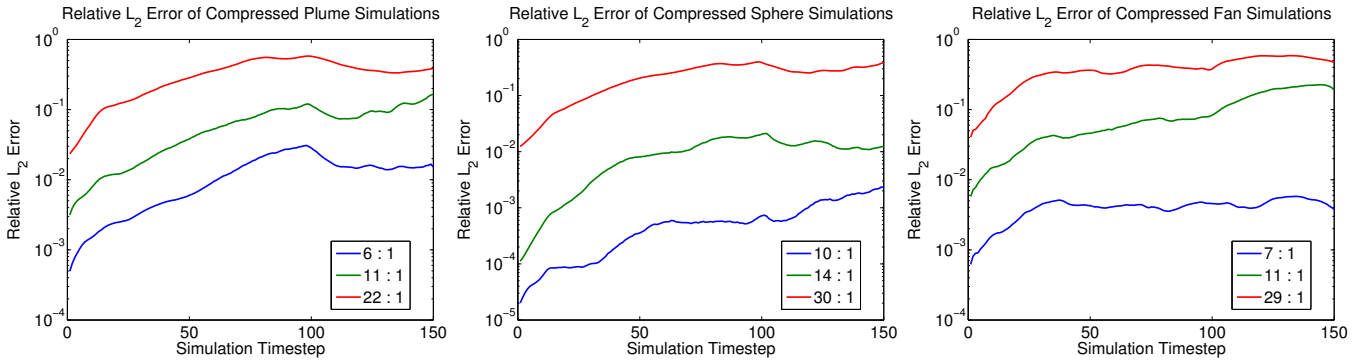

Figure 5: Relative L_2 error introduced by the compressed matrix $\mathbf{C}$ compared to an uncompressed $\mathbf{U}$ over the course of a simulation. The transition between visually undetectable and visible error corresponds to roughly an order of magnitude jump in the relative error.

Acknowledgements

This work was supported by a National Science Foundation CAREER award (IIS-1253948). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation. We acknowledge support from the Center for Scientific Computing from the CNSI, MRL: an NSF MRSEC (DMR-1121053) and Hewlett Packard.

References

[AKJ08] AN S. S., KIM T., JAMES D. L.: Optimizing cubature for efficient integration of subspace deformations. *ACM Trans. on Graphics 27*, 5 (Dec. 2008), 165. 1

[ATW13] ANDO R., THÜREY N., WOJTAN C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (July 2013), 103:1–103:10. 1

[Bat96] BATHE K.-J.: *Finite Element Procedures*, second ed. Prentice Hall, 1996. 2

[BJ05] BARBIČ J., JAMES D. L.: Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. on Graphics 24*, 3 (Aug. 2005), 982–990. 1

[BKB12] BROCHU T., KEELER T., BRIDSON R.: Linear-time smoke animation with vortex sheet meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2012), pp. 87–95. 1

[BP08] BARBIČ J., POPOVIĆ J.: Real-time control of physically based simulations using gentle forces. *ACM Trans. Graph. 27*, 5 (2008), 163:1–163:10. 1, 2

[Bri15] BRIDSON R.: *Fluid Simulation for Computer Graphics*. CRC Press, 2015. 1, 4

[DWLF12] DE WITT T., LESSIG C., FIUME E.: Fluid simulation using Laplacian eigenfunctions. *ACM Trans. Graph. 31*, 1 (2012), 10:1–10:11. 1, 2, 4

[FJ05] FRIGO M., JOHNSON S. G.: The design and implementation of FFTW3. *Proceedings of the IEEE 93*, 2 (2005), 216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation". 4

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010. 4

[GKSB15] GERSZEWSKI D., KAVAN L., SLOAN P.-P., BARGTEIL A. W.: Basis enrichment and solid-fluid coupling for model-reduced fluid simulation. *Comput. Animat. Virtual Worlds 26*, 2 (Mar. 2015), 109–117. 2

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *IEEE Visualization* (2002), pp. 53–60. 2

[HTC*14] HAHN F., THOMASZEWSKI B., COROS S., SUMNER R. W., COLE F., MEYER M., DEROSE T., GROSS M.: Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. 33*, 4 (July 2014), 105:1–105:9. 1

[KD13] KIM T., DELANEY J.: Subspace fluid re-simulation. *ACM Trans. Graph. 32*, 4 (July 2013), 62:1–62:9. 1, 2, 4, 6

[LAJJ14] LANGLOIS T. R., AN S. S., JIN K. K., JAMES D. L.: Eigenmode compression for modal sound models. *ACM Trans. Graph. 33*, 4 (July 2014), 40:1–40:9. 1, 2

[LFZ15] LI D., FEI Y., ZHENG C.: Interactive acoustic transfer approximation for modal sound. *ACM Trans. Graph. 35*, 1 (Dec. 2015), 2:1–2:16. 1

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 457–462. 1

[LMH*15] LIU B., MASON G., HODGSON J., TONG Y., DESBRUN M.: Model-reduced variational fluid simulation. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 244:1–244:12. 2, 4, 7

[LR09] LONG B., REINHARD E.: Real-time fluid simulation using discrete sine/cosine transforms. In *Symposium on Interactive 3D graphics and games* (2009), ACM, pp. 99–106. 3

[MMCK14] MACKLIN M., MÜLLER M., CHENTANEZ N., KIM T.-Y.: Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 104. 1

[OSG02] O'BRIEN J. F., SHEN C., GATCHALIAN C. M.: Synthesizing sounds from rigid-body simulations. In *ACM SIGGRAPH Symposium on Computer Animation* (July 2002), ACM Press, pp. 175–181. 1

[PTG12] PFAFF T., THUEREY N., GROSS M.: Lagrangian vortex sheets for animating fluids. *ACM Trans. Graph. – SIGGAPH 2012 Papers 31*, 4 (July 2012), 112:1–112:8. 1

[Say12] SAYOOD K.: *Introduction to Data Compression*, 4 ed. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2012. 2, 3

[SILN11] SEO J., IRVING G., LEWIS J. P., NOH J.: Compression and direct manipulation of complex blendshape models. *ACM Trans. Graph. 30*, 6 (Dec. 2011), 164:1–164:10. 1, 2

[SSW*13] STANTON M., SHENG Y., WICKE M., PERAZZI F., YUEN A., NARASIMHAN S., TREUILLE A.: Non-polynomial Galerkin projection on deforming meshes. *ACM Trans. Graph. 32*, 4 (July 2013), 86:1–86:14. 2, 7

[TBR*12] TREIB M., BÜRGER K., REICHL F., MENEVEAU C., SZALAY A., WESTERMANN R.: Turbulence visualization at the terascale on desktop PCs. *IEEE Transactions on Visualization and Computer Graphics 18*, 12 (December 2012), 2169–2177. 2

[TKP13] THUEREY N., KIM T., PFAFF T.: Turbulent fluids. In *SIGGRAPH Courses* (2013). 1

[TLP06] TREUILLE A., LEWIS A., POPOVIĆ Z.: Model reduction for real-time fluids. *ACM Transactions on Graphics 25*, 3 (July 2006), 826–834. 1, 2, 4

[vTSSH13] VON TYCOWICZ C., SCHULZ C., SEIDEL H.-P., HILDEBRANDT K.: An efficient construction of reduced deformable objects. *ACM Trans. Graph. 32*, 6 (Nov. 2013), 213:1–213:10. 1

[WMW15] WU X., MUKHERJEE R., WANG H.: A unified approach for subspace simulation of deformable bodies in multiple domains. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 241:1–241:9. 1

[WST09] WICKE M., STANTON M., TREUILLE A.: Modular bases for fluid dynamics. *ACM Trans. on Graphics 28*, 3 (Aug. 2009), 39. 1, 2

[XUC*14] XU W., UMENTANI N., CHAO Q., MAO J., JIN X., TONG X.: Sensitivity-optimized rigging for example-based real-time clothing synthesis. *ACM Trans. Graph. 33*, 4 (July 2014), 107:1–107:11. 1

[YL95] YEO B.-L., LIU B.: Volume rendering of DCT-based compressed 3d scalar data. *IEEE Transactions on Visualization and Computer Graphics 1*, 1 (Mar. 1995), 29–43. 2, 3

[YLX*15] YANG Y., LI D., XU W., TIAN Y., ZHENG C.: Expediting precomputation for reduced deformable simulation. *ACM Trans. Graph. 34*, 6 (Oct. 2015), 243:1–243:13. 1