Compact Poisson Filters for Fast Fluid Simulation

Amir Hossein Rabbani
amir.hossein-rabbani@ubisoft.com
Ubisoft Montreal
Canada

Jean-Philippe Guertin
jean-philippe.guertin@ubisoft.com
Ubisoft Montreal
Canada

Damien Rioux-Lavoie
damien.rioux-lavoie@mail.mcgill.ca
Ubisoft Montreal
McGill University
Canada

Arnaud Schoentgen
arnaud.schoentgen@ubisoft.com
Ubisoft Montreal
Canada

Kaitai Tong
kaitai@alumni.ubc.ca
Ubisoft Montreal
University of British Columbia
Canada

Alexandre Sirois-Vigneux
alexandre.sirois-vigneux@mail.mcgill.ca
Ubisoft Montreal
McGill University
Canada

Derek Nowrouzezahrai
derek@cim.mcgill.ca
McGill University
Canada

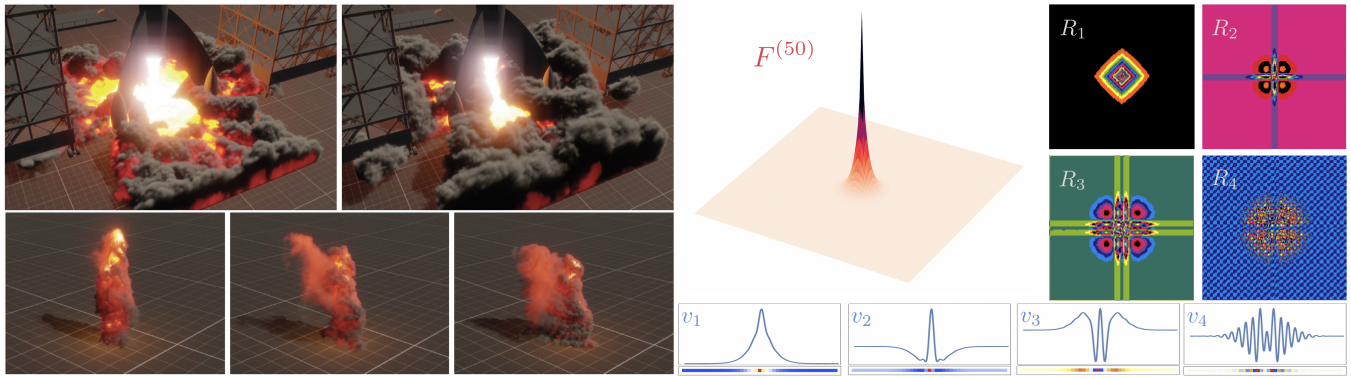


Figure 1: ROCKET (top left) illustrates large scale smoke and fire and BURNINGMAN (bottom left) showcases a complex user-controlled dynamic scene: our *Poisson filter solver* enforces incompressibility at interactive rates. *Right:* 50^{th}-order inverse Poisson kernel, its first four rank terms and associated convolution filters.

ABSTRACT

Poisson equations appear in many graphics settings including, but not limited to, physics-based fluid simulation. Numerical solvers for such problems strike context-specific memory, performance, stability and accuracy trade-offs. We propose a new *Poisson filter-based* solver that balances between the strengths of spectral and iterative methods. We derive *universal Poisson kernels* for forward and inverse Poisson problems, leveraging careful adaptive filter truncation to localize their extent, all while maintaining stability and accuracy. Iterative composition of our compact filters improves solver iteration time by orders-of-magnitude compared to optimized linear methods. While motivated by spectral formulations, we overcome important limitations of spectral methods while retaining many of their desirable properties. We focus on the application of our method to high-performance and high-fidelity fluid simulation, but we also demonstrate its broader applicability. We release our source code at https://github.com/Ubisoft-LaForge/CompactPoissonFilters .

SIGGRAPH '22 Conference Proceedings, August 7–11, 2022, Vancouver, BC, Canada

ACM ISBN 978-1-4503-9337-9/22/08…$15.00
https://doi.org/10.1145/3528233.3530737

CCS CONCEPTS

• **Computing methodologies** → **Physical simulation**.

KEYWORDS

iterative methods, reduced modeling

ACM Reference Format:
Amir Hossein Rabbani, Jean-Philippe Guertin, Damien Rioux-Lavoie, Arnaud Schoentgen, Kaitai Tong, Alexandre Sirois-Vigneux, and Derek Nowrouzezahrai. 2022. Compact Poisson Filters for Fast Fluid Simulation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings (SIGGRAPH '22 Conference Proceedings), August 7–11, 2022, Vancouver, BC, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3528233.3530737

1 INTRODUCTION

Many graphics applications require solving Poisson equations, e.g., the pressure projection stage in Eulerian fluid simulations [Stam 1999] and many instances of diffusion or heat equations on manifolds [Crane et al. 2017]. Iterative solvers are commonly employed, however spectral and data-driven methods may also be better suited in certain contexts [Kettunen et al. 2019; Ummenhofer et al. 2020].

We propose an efficient alternative to these solvers and focus on demonstrating its effectiveness for fluid simulation. Specifically, we develop a novel, analytically-separable Poisson filtering formulation that is easily paralellizable while admitting controllable approximation error through adaptive filter truncation. We analyze stability and accuracy-performance trade-offs, evidencing our solver's advantages here compared to strong baselines in the fluid setting.

The separability of our Poisson filters allows our solver to scale *sublinearly with resolution* as we increase the dimensionality of the problem setting. When compared at equal accuracy, the relative performance gains of our method *increases* as a function of the total iteration count needed by typical iterative solvers. In practice, we observe roughly 10× performance improvement in 3D at equal accuracy and with reasonable parameter settings.

For interactive graphics, ours is a drop-in replacement for iterative solvers on grids, surpassing the efficiency of Jacobi, red-black Gauss Seidel and preconditioned Conjugate Gradient solvers. Our solver shares favourable properties of spectral methods albeit with fewer limitations, e.g., our localized filters admit efficient, scalable implementation. Our contributions are:

- a theory of localized and factorized Poisson filter kernels,
- analytic derivations for compact realizations of these kernels,
- efficient parallel implementation of forward and inverse problems,
- scalability and accuracy comparisons against iterative solvers,
- applications to fast fluid simulation and heat diffusion on meshes.

Our solver precludes the need for careful preconditioning, scales favorably with the size of the problem, handles Neumann boundary conditions, and is a drop-in replacement with controllable error tolerance for existing solvers in high-performance settings.

2 RELATED WORK

Our work is motivated by spectral and data-driven methods, although focusing on interactive, limited-convergence settings and addressing important limitations: e.g., supporting non-periodic domains, admitting simple and efficient parallel implementation, and treating (Neumann) boundary conditions. We focus our review to those most relevant works across a diversity of areas.

Subspace and Spectral Methods. Dual-space methods, e.g., Fourier-based synthesis, have a long history in simulation [Orszag 1969]. Windowing and clustering can be used to localize global basis support, at the cost of added complications from basis discontinuities. Boyd [2001] and Trefethen [2000] offer comprehensive surveys.

In graphics, the Fast Fourier (FFT) [Stam 2001] and Discrete Cosine Transforms (DCT) have been used for fast dual-space pressure solves in fluids, first with periodic – and later, Dirichlet – domain boundaries [Long and Reinhard 2009a]. These methods also admit favorable optimization on multi-core platforms [Henderson 2012].

In the inviscid setting, Laplacian eigenfunctions form a divergence-free *EigenFluid* basis with global support, admitting efficient dual-space advection without dissipation [De Witt et al. 2012]. The basis has an analytic form for simple domain geometries, and follow-up work apply DCTs to reduce their memory footprint and to treat uniform Neumann and Dirichlet boundaries [Cui et al. 2018]. We categorize these as spectral methods as they draw on spectral modes and eigenfunctions of the Laplacian, however they may also be considered *decomposition-based*, e.g., with data-oriented extensions discussed later in our review [Mercier and Nowrouzezahrai 2020].

In contrast, we instead rely on spectral modes to accelerate *primal domain* simulation performance, e.g., in mass conservation and diffusion processes. We show that combining Poisson filters with iterative (i.e., Jacobi) methods in the primal domain leads to a class of solver with trade-offs ideally suited to the interactive setting. Efficient methods for computing the spectral representation of an N^d-field for $d \in \{2, 3\}$ has computational cost $O(N^d \log N)$, whereas separable convolution with k-wide isotropic filters ($k \ll \log N$) scale as $O(k\, N^d)$ [Fialka and Cadik 2006; Guillet and Teyssier 2011].

While efficient GPU-based FFT methods can incorporate approximate boundaries (e.g. [Henderson 2012]), they do not admit a simple treatment for internal boundaries, with axis-aligned mirroring only treating wall boundaries. Existing solutions for internal boundaries rely, e.g., on DCT functions and a priori knowledge of collider geometry with precomputed collision masks. As with iterative solvers, our method precludes any such precomputation and can treat dynamic colliders, an important feature in real-time applications.

Advantages of our approach include its ability to handle Neumann boundary conditions during advection and to support vorticity confinement, both of which are limitations in spectral based methods. Our memory requirements are also orders of magnitude smaller, and scale sub-linearly in the size of the domain geometry.

Operator Factorizations. After discretizing our Poisson kernels, we factorize their matrix (in 2D) and tensor (in 3D) forms, relying on low rank approximations during simulation.

Matrix and tensor factorization have been used in compressing high-dimensional appearance data, such as bidirectional reflectance distribution functions from real-world captures [Matusik 2003], and for reconstruction and rendering directly in reduced spaces [McCool et al. 2001; Mcgraw 2015]. N-mode singular value decomposition (SVD) is commonly used for textured appearance compression [Vasilescu and Terzopoulos 2004], whereas – similarly to spectral methods – clustered principal components analysis (PCA) and moving basis decomposition can be used to localize the

data-driven basis for, e.g., precomputed lighting compression [Silvennoinen and Sloan 2021; Sloan et al. 2003].

Similar techniques applied to *EigenFluids* yield hierarchical data-driven flow bases [Mercier and Nowrouzezahrai 2020]. Unlike the analytic basis [De Witt et al. 2012], these sparse flow operators are *precomputed* and tabulated, instead of computed on-the-fly.

Using polyadic (CANDECOMP/PARAFAC; CP) decompositions [Kindermann and Navasca 2011] we generalize this construction to 3D. Note that we use this decomposition instead of, HOSVD or Tucker decompositions, as they yield compact kernels with rank-1 separable modes. We discuss existence, computability and rank implications when using CP, in Section 4.2.

Efficient Solvers. Linear solvers specialized to graphics target unique performance-convergence profiles. Amador and Gomes propose GPU-accelerated linear solvers for fluid simulation [Amador and Gomes 2010a,b, 2012] and, benchmarking GPU-based Jacobi, Gauss-Seidel (GS) and Conjugate Gradient (CG) solvers. Their latter work treats static and moving boundaries, motivating our own benchmarks: our GPU-accelerated Poisson filter solver compares favorably to GPU-accelerated Jacobi, preconditioned CG, and red-black GS at equal convergence. We build on the recursive nature of Jacobi to build compact and efficient separable convolution kernels.

Multigrid. Multigrid solvers are powerful methods [Chentanez and Müller 2011; Glimberg et al. 2009; McAdams et al. 2010; Molemaker et al. 2008] that admit interesting design space trade-offs. Our work instead focuses on the core Poisson formulation and its realization in a simple high-performance solver. Adapting Poisson filters however to support different resolutions, and hence replace the iterative solvers at each stage of the multigrid, is an exciting avenue of future work. Such adaptation would require some care when, i.e., enforcing boundary conditions across resolutions.

Learning Surrogates. Recent work learns MLP-based localized projection methods to replace preconditioned Conjugate Gradient solves using supervision from ground truth simulated pressure field data pairs [Yang et al. 2016]. Discrepancies between runtime and training time simulation initialization, and the lack autoregression can lead to errors over long time horizons, but this axis remains exciting and promising. An alternative unsupervised data-driven variant instead learns projection through fluid field inference, yielding 4× speed ups compared to Jacobi solvers [Tompson et al. 2016].

3 PRELIMINARIES AND APPLICATION SETTINGS

Preliminaries. Poisson equations $\nabla^2\varphi = f$, with Laplace operator ∇^2 and real- or complex-valued functions f and φ, are the simplest nontrivial example of elliptic partial differential equations (PDE). They appear in many physical settings, e.g., in electrostatics, Newtonian gravity, diffusion and fluids simulation.

As with other elliptic equations, Poisson equations are suited to describing equilibrium states, i.e., with smoothed discontinuities: for instance, the transient heat diffusion equation

$$\partial u/\partial t = \kappa\nabla^2 u = \kappa\left[\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2 + \partial^2 u/\partial z^2\right], \quad (1)$$

for a function $u(x, y, z, t)$ of spatial (x, y, z) and temporal (t) variables, and with diffusivity coefficient κ. Equation (1) describes a *forward* Poisson problem where $\varphi\ (= u)$ is given and $f\ (= \partial u/\partial t)$ is sought.

When f is given and φ is sought we have an *inverse* Poisson problem. An important inverse problem for graphics appears when solving the so-called *Poisson-pressure* equations for incompressible fluids simulation. Consider the convective form of incompressible Navier–Stokes equations

$$\partial u/\partial t = -(u\cdot\nabla)u - (1/\rho)\nabla p + v\nabla^2 u + F \quad \text{with} \quad \nabla\cdot u = 0\,, \quad (2)$$

where u is the flow velocity, ρ and v the density and kinematic viscosity, F the external forces, and $\nabla\cdot u = 0$ the incompressibility condition with divergence operator $\nabla\cdot$. We henceforth assume the same parameterization as Equation (1) and omit variables in equations for brevity.

The *Helmholtz–Hodge* decomposition is often employed to enforce incompressibility in Equation (2), yielding a divergence-free field by *projection*, $u = w - \nabla p$, where w is the divergent field and ∇ the gradient operator. Applying the divergence operator to both sides and solving for pressure presents the inverse Poisson problem: $p = \nabla^{-2}(\nabla\cdot w)$. Note that applying the diffusive term of the Navier-Stokes equation can be recast with operator splitting as an implicit forward Poisson problem [Stam 1999]; here, both a forward and inverse Poisson solve are required.

Application Settings. Poisson equations do not generally admit closed form solutions. Many numerical methods have been proposed with various performance versus stability trade-offs [Golub et al. 1992; Nocedal and Wright 2006].

Explicit methods are faster but can suffer from numerical instability. Implicit methods are more stable but iterative, requiring many iterations to reach an acceptable convergence threshold, resulting in poor execution times. This limits their application in interactive graphics settings, such as real-time Eulerian fluid simulation in games. Our method targets the high-performance setting, improving on the performance-accuracy profile of existing solutions.

We benchmark primarily against solvers that balance between performance and numerical quality, and that admit parallel (i.e., GPU friendly) implementation. Finite difference schemes coupled with a sparse linear solvers for either strictly diagonally dominant or symmetric positive definite systems provide good numerical stability: e.g., Jacobi, Gauss-Seidel (GS) and preconditioned Conjugate Gradient (PCG). Jacobi is commonly used in real-time applications due to its parallelizability and reasonable convergence profiles. Gauss-Seidel, on other hand, has improved convergence but is more difficult to parallelize. Various methods improve upon GS, such as GPU-based Red-Black GS [Amador and Gomes 2012] and Jacobi Embedded GS [Ahmadi et al. 2021]. PCG has superior convergence compared to Jacobi and GS, but remains more expensive than other two thus, not suited for real-time applications. Therefore, within an operating range of convergences, Jacobi remains the workhorse method for high-performance applications.

For fluid simulation, solving the projection Poisson equation remains the simulation performance bottleneck for Jacobi solvers (despite their excellent GPU performance among linear solvers). This is due to their multipass iterative nature. Our method replaces these passes with a single-step solve, improving total performance

whilst maintaining a convergence operating region similar to Jacobi's. Since we base our derivations on Jacobi and GS – both of which converge in the limit – we can match our output accuracy to a target Jacobi/GS iteration count, all with better performance.

Discrete Poisson equation. Given a finite difference approximation stencil K of the Laplacian, solving the Poisson equation amounts to solving a linear system $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ and $\mathbf{b}$ are linearizations of the spatial simulation domain and source term, and A is the coefficient matrix a stencil K with appropriate boundary conditions. For simplicity, we begin by assuming a periodic domain where A is symmetric positive definite circulant matrix. The circulant form allows us to recast the system as a convolution equation $K * X = B$, where $*$ is the convolution operator, and X and B are matrix forms of $\mathbf{x}$ and $\mathbf{b}$. This form motivates our Poisson kernel construction.

Implicit solve with Jacobi. The Jacobi method solves for $\mathbf{x}$ using iterative relaxation of the decomposed system $A = D + L + U$ with D, L and U as the diagonal, lower and upper triangular portions of A. The approximate solution at iterations k is

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L + U)\,\mathbf{x}^{(k)}). \tag{3}$$

To facilitate parallel GPU implementation, we return to the convolution expression of the system. In 2D, with a central finite difference stencil, we obtain

$$X_{i,j}^{(k+1)} = \left(X_{i-1,j}^{(k)} + X_{i+1,j}^{(k)} + X_{i,j-1}^{(k)} + X_{i,j+1}^{(k)} + \alpha B_{i,j}\right) \Big/ \beta, \tag{4}$$

where (i, j) are 2D spatial indices and α and β depend on the dimension, stencil and "direction" of the solve (see Table 1).

Equation (4) generalizes to 3D with two more difference terms. Setting α and β for the inverse and forward Poisson equations respectively lead to a unified update kernel (see Table 1).

Table 1: Forward and inverse parameters for dimension $d \in \{2, 3\}$, diffusivity κ and diffusion transient time Δt. For simplicity, we assume square/cubic cell sizes, but one can express the update for $\Delta x \neq \Delta y \neq \Delta z$.

	Inverse Solve	Forward Solve
α	$-(\Delta x)^2$	$(\Delta x)^2/(\kappa \Delta t)$
β	$2d$	$2d + \alpha$

The kernel-based update step is commonly found in high-performance GPU Jacobi implementations and we will capitalize on the amenability of Equation (4) for fast implementation when deriving our filters, below. Our formulations also permit unified solvers for inverse and forward Poisson equations, detailed in Equation (4) of our supplement. After deriving our compact filters under the simplifying assumption of periodic domains, we will elaborate a method to treat Neumann boundary conditions, as in the fluid projection step. As such, our general formulation will support non-periodic domains.

4 METHOD

We now present the core of our method, building a compact *Poisson kernel* $F^{(k)}$ that – in a single application – yields an equivalent result as k Jacobi iterations, where k is the *order* of the kernel F. One goal is to target GPU-friendly operations whilst avoiding the need for many iteration; despite this, since the effective size of the kernel will increase linearly in the number of its repeated applications, applying it can quickly become impractical after even a handful of iterations. To address this, we use a spectral decomposition of the filter into separable and sparse filters, drastically reducing the amount of computation and memory needed during convolution.

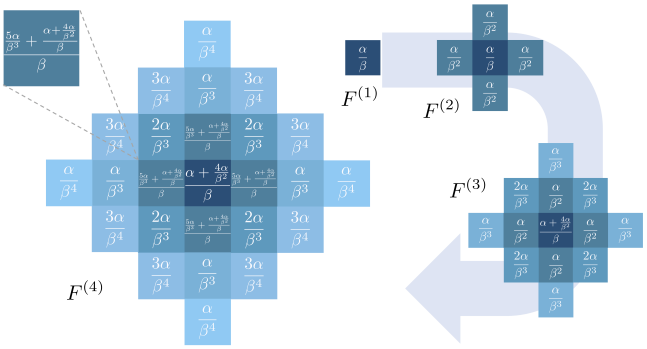


Figure 2: Parametric 2D kernel growth for 4 Jacobi iterations using recursive backward iteration. Starting from the 4th Jacobi iteration ($F^{(1)}$), we iterate backwards to obtain the final 4th-order kernel $F^{(4)}$. We evaluated this kernel *directly*, serving as a one-shot convolution kernel on the GPU, yielding the same result as 4 Jacobi iterations.

4.1 Unified Kernel

We construct $F^{(k)}$ by recursively computing the value of $X^{(k)}$ based on all its previous values for the last k steps, i.e., computing $X^{(k)}$ backwards to $X^{(1)}$. Figure 2 illustrates the kernel growth process: updating a single cell value after the n^{th} iteration with a 3×3 base Jacobi kernel at each iteration, is equivalent to applying a much larger kernel only once at iteration 1. This short-circuiting is foundational to our method, transforming an iterative solver to a direct one. Absorbing $\text{sign}(\alpha)$ into the source term B allows the same filter to apply in both the forward or inverse settings. Note, the kernel size grows as $(2 \times k + 1)^d$ for dimension $d \in \{2, 3\}$.

4.2 Compact Filter Computation

Since our Poisson kernel grows polynomially with iterations, we devise a compact approximation to reduce its computational overhead. First, we decompose the kernel into a spectral expansion consisting of a sum of separable 1D kernels. This makes it possible to obtain 1D filters by satisfying the rank-1 separability condition. We further sparsify each 1D filter by leveraging their symmetry and rapid falloff. We fix the number of iterations k in $F^{(k)}$ below and omit the superscript.

2D. Poisson kernels are symmetric and so their eigen-decomposition is $F = Q\Lambda Q^\intercal$, where $Q = [\mathbf{u}_1, \cdots, \mathbf{u}_n]$ is the orthonormal matrix with eigenvector columns and $\Lambda = \text{Diag}(\lambda_1, \cdots, \lambda_n)$ the diagonal eigenvalue matrix. Expanding into a spectral form as

$$F = \textstyle\sum_{i=0}^{r} \lambda_i \mathbf{u}_i \mathbf{u}_i^\intercal = \sum_{i=0}^{r} \mathbf{v}_i \mathbf{v}_i^\intercal, \tag{5}$$

yields a summation of rank-1 matrices, where r is the rank of F and $\mathbf{v}_i$ is a scaled eigenvector $\mathbf{v}_i = \sqrt{\lambda_i}\mathbf{u}_i$. Note that the bi-symmetry of our filter implies that our eigenvectors are symmetric, meaning

that $\mathbf{v}_{i,j} = \mathbf{v}_{i,n-j}$, and so we can rewrite the decomposition as a sum of self-convolutions, $F = \sum_{i=0}^{r} \mathbf{v}_i * \mathbf{v}_i^\mathsf{T}$. This means that, when convolving our decomposed filter with a matrix X, we can express each term as two sequential 1D convolutions, $*_r$ and $*_c$, applied on X's rows and then its columns,

$$F * X = \sum_{i=0}^{r} \mathbf{v}_i *_c (\mathbf{v}_i *_r X). \tag{6}$$

3D. While the rank of a tensor is ill-defined, we seek a similar decomposition in 3D. Since our filter remains symmetric in each dimension, a symmetric canonical polyadic decomposition in 3D yields

$$F = \sum_{i=0}^{r} \lambda_i \mathbf{u}_i \otimes \mathbf{u}_i \otimes \mathbf{u}_i = \sum_{i=0}^{r} \mathbf{v}_i \otimes \mathbf{v}_i \otimes \mathbf{v}_i, \tag{7}$$

with $\mathbf{v}_i = \sqrt[3]{\lambda_i}\mathbf{u}_i$ and $\otimes$ is a tensor product. Treating the tensor product as a generalization of the outer product $\mathbf{v} \otimes \mathbf{v} := \mathbf{v}\mathbf{v}^\mathsf{T}$, we can similarly generalize equations 5 and 6 to 3D. The filter tri-symmetry leads to a triple convolution form for each of the decomposition terms, acting across the tensor dimensions: when convolving our Poisson filter with a 3D tensor X, we can use a series of 1D convolutions applied on the fibers, rows and columns of X,

$$F * X = \sum_{i=0}^{r} \mathbf{v}_i *_c (\mathbf{v}_i *_r (\mathbf{v}_i *_f X)). \tag{8}$$

Sparsity. The aforementioned sparsity leads us to only need to store half of the eigenvectors. Moreover, eigenvector component values decrease quickly in magnitude as move farther off the center of each filter (Figure 1), so we also only retain those elements above threshold – as we will see, this does not add much error. In practice, we set these thresholds experimentally to yield an acceptable trade-off between accuracy and performance. To do so, we truncate filters below a fixed threshold to prune small values with insignificant impact in the convolution. This results in different filter sizes for each term of the decomposition. We call this method *adaptive truncation* as it computes the truncation percentile adaptively for each filter, as opposed to a constant percentile across them. We provide an additional memory footprint analysis in our supplement.

4.3 Boundary Condition

As with spectral methods, dealing with non-periodic domains is a challenge. When only wall boundaries are present, we can employ mirroring methods similar to those in [Long and Reinhard 2009b]. Otherwise, we extend our method to deal with complex object boundaries inside the domain: we approximately enforce *pure Neumann* boundary conditions, i.e., $\nabla\varphi \cdot \mathbf{n} = 0$ on the boundary where $\mathbf{n}$ is the boundary normal, required by the pressure-projection step in the inverse Poisson setting. This yields zero velocity gradient along boundary edges. Specifically, we use a *mirror marching* strategy illustrated in Figure 3: starting from each filter center O, we first march to the rightmost solid index (A) before mirroring the direction and marching to the leftmost solid index (B). If the total marched steps are less than the original required index n_{iR}, we continue flipping and marching until we arrive at the point S whose index satisfies the total required marching steps. We apply the same strategy when marching to the left, as well as when convolving vertical and fiber (depth) filters. We include pseudo-code in our supplement and provide a quantitative analysis in Figure 4.

5 IMPLEMENTATION DETAILS

Our implementation almost entirely runs on the GPU through our in-house, multiplatform prototyping engine. We precompute filter values at the desired ranks using TensorLy [Kossaifi et al. 2019] for different kernel orders and parameters, adaptively truncating the output using the technique described in the previous sections. The resulting truncated 1D arrays are merged into 4-wide vectors at the final step of preprocessing and stored directly as shader code for execution in compute shaders. This makes it possible to apply up to four kernels simultaneously during convolution at no additional cost thanks to 128-bit SIMD (see Section 4.2). Each volume is evaluated sequentially and intermediate data is stored in pairs of textures for efficient read-modify-write cycles. We discretize the Navier-Stokes equations using a standard staggered Marker-and-Cell (MAC) grid [Harlow and Welch 1965]. Our implementation is based on a semi-Lagrangian advection [Stam 1999] with tri-cubic density interpolation and vorticity confinement [Fedkiw et al. 2001].

6 RESULTS

We benchmark our method using 4 ranks for threshold values $\delta = 10^{-3}$ against Jacobi, red-black Gauss–Seidel [Amador and Gomes 2012] and PCG using incomplete Poisson preconditioners [Ament et al. 2010] as these are the most GPU-friendly solver to our knowledge. We chose $\delta = 10^{-3}$ since it strikes a nice trade-off between quality and speed experimentally. We generate the source term $\mathbf{b}$ randomly and assume square Neumann boundary conditions.

When validating our approach against other methods using pure Neumann boundary conditions, further considerations must be accounted for. First, we enforce the compatibility constraint on our system for it to be well defined; please refer to our supplemental material for more information. In practice, this can be achieved by subtracting from the source $\mathbf{b}$ its average making it integrate to zero. This has the effect of constraining our solution to a fix level set and making it unique. Since we are only interested in using its gradient, the level set can be anything. While this is enough for our method, Jacobi and Gauss–Seidel, in the PCG case it is more challenging as its preconditioner tries to approximate the inverse of A, which is singular. We further discuss this topic in our supplemental material.

Figure 4 illustrates the relative error of our Neumann boundary marching implementation when compared to Jacobi. While wall boundaries demonstrate no artifacts, minor vertical artifacts appear around the object due the the 1D filter ordering, but they are local

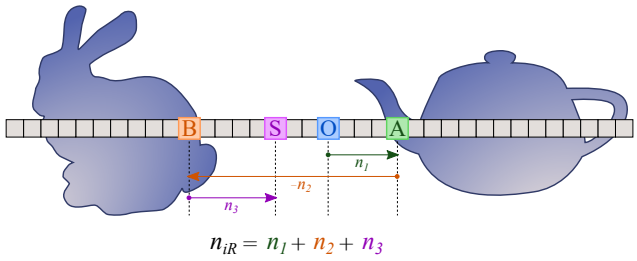


Figure 3: Mirror marching example to fetch a convolution index that falls within a solid: n_{iR} is the i-th index to fetch towards the right direction.

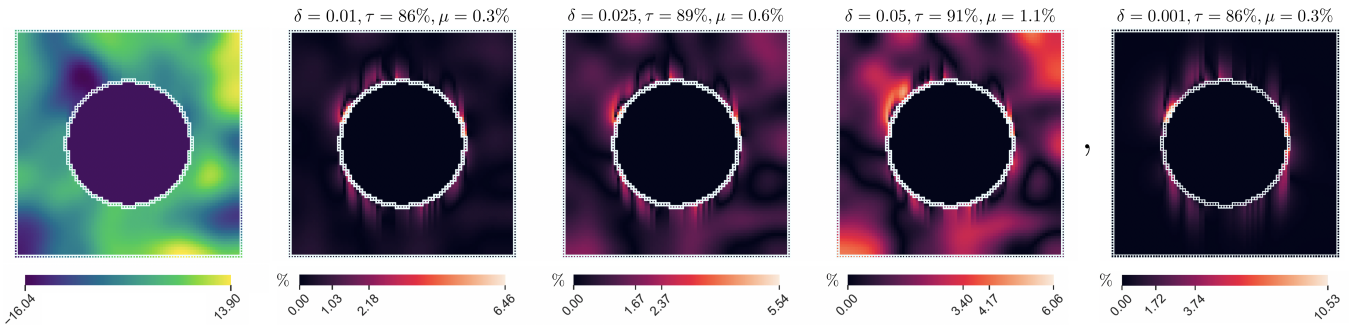


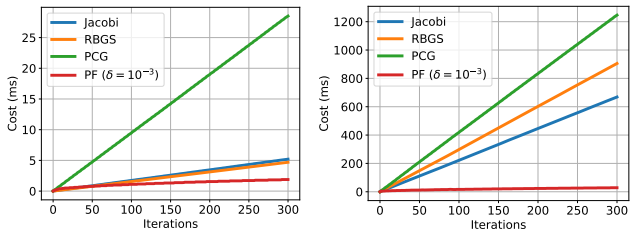
Figure 4: Pressure solves with Neumann boundaries (solid $\nabla P = 0$ circle), with rank-4 100^{th}-order filters, random inputs in ± 1 and 81^2 domain size. *Left:* solution. *Middle 3:* percentage error compared to 100 Jacobi iterations with truncation thresholds of mild ($\delta = 0.01$) to aggressive ($\delta = 0.05$), resulting in filter shrinkage τ and mean percentage error μ. *Right:* percentage error for 200^{th}-order filters. Color bars indicate min/max $95^{\text{th}}/99^{\text{th}}$ percentiles. Note: removing 86% of the 100^{th}-order filters ($\delta = 0.01$) yields $\leq 2\%$ error for 99% of the domain and $\leq 4\%$ for 200^{th}-order filters at equal τ (both with equal mean percentage error).

and have small relative error magnitude, which we consider to fall within an acceptable margin in our interactive setting.

In Figure 5 we present the overall cost of our Poisson filter for different iteration counts against other methods. As expected, PCG is extremely expensive as its GPU implementation is more complex and Jacobi / Gauss–Seidel operate roughly in the same range. Our Poisson filter is faster than all of these at a matching order. A more exhaustive cost table is available in the supplemental material.

In Figure 7, we summarize the relative efficiency of our method compared to others. Since our method is a direct (non-iterative) solver, our convergence *speed* analysis uses actual GPU cost for a target iteration (i.e., matching filter order) across solvers, instead of using the number of iterations. Our method converges as Jacobi since Poisson filters are generated from a Jacobi kernel. Jacobi convergence bounds Poisson filter performance, meaning – without adaptive filter truncation – their residuals match.

The non-monotonic behaviour of PCG is due to using Neumann boundary conditions and the incomplete Poisson preconditioner, as discussed in the supplementary material. While PCG gives a better residual for high enough iterations, its cost is way above what one would consider within real-time budgets. Next, we observe that while residual between us and Jacobi are mostly the same,



(a) 2D resolution 256^2 **(b) 3D resolution 256^3**

Figure 5: Comparing total solver cost vs. iteration count (or filter order, for Poisson filters (*PF*)). Here, we use a rank-4 PF (in red) with maximum threshold $\delta = 10^{-3}$.

our approach is an order of magnitude faster. Finally, while Gauss–Seidel outperforms Jacobi in term of efficiency, we can always find a matching Poisson kernel that is faster with the same residual. We use a user-determined target iteration to generate the filters. As such, picking a target residual is not straightforward as residuals depend on the RHS input. We leave residual-based targeting to future work.

Applications. We applied our method to efficiently solve the Poisson equation for pressure in a context of real-time fluid simulation. We generated a wide range of 2D and 3D animations (e.g., Figure 1). In these examples, the use of Poisson filters allows us to significantly reduce the computational time of the pressure projection step which remains the bottleneck of most Eulerian fluid solvers. In 3D, the average per-frame solver and simulation time using our method are $6.81ms$ and $16.57ms$. Jacobi requires $68.33ms$ and $78.11ms$, corresponding to a speed-up of $5-10\times$. These performance numbers are representative, however absolute performance varies with the number of iterations and resolution; e.g., at higher iteration counts, we can achieve speed-ups of up to **20×**. Note that most of the animations in our video demonstrate interactions between simulated fluid and walls, static and/or moving objects. Here, our *mirror marching* strategy is able to enforce approximate Neumann boundary conditions at the solid/fluid interface during the pressure solve step. To further highlight the real-time user-interaction our method supports, we showcase two examples in which a character controlled by a user is either emitting fire or colliding with smoke.

In the Torches scene, we showcase 60 independent $2D$, camera-facing billboard fire simulation instances. Here, we exceptionally leverage temporal-reprojection post-processing to account for the impact of camera motion on the 2D billboard renderings, details of which are provided in our supplemental material.

In addition to our inverse pressure-projection Poisson problem in fluid simulation, we demonstrate our method on two additional *forward* Poisson problems: first we showcase our implicit forward approach (derivation included in our supplemental material) in the context of density diffusion (Figure 6); second, we solve the heat equation on a surface mesh (derivation and an application to

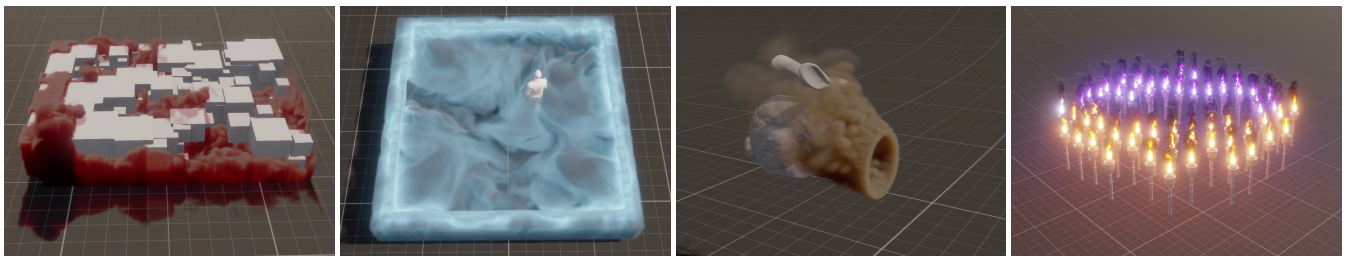


Figure 6: Interactive fire and smoke results generated with our method. Left to right: a stream of smoke flowing and colliding across a CubeField, the user-controlled ManSmoke character moving into a pool of smoke, DiffusiveSmoke interacting with the bunny using *forward* Poisson filters, and 60 independently-simulated 2D billboard Torches at interactive rates. Please refer to our video for more examples, captured live from our interactive simulator.

solving the heat equation on surface meshes are included in our supplemental material).

Note that we do not treat mesh curvature in the latter example, so we are *not* solving with the Laplace-Beltrami operator for curved surfaces, but rather assume a regular grid mesh with equal size cells and no curvature. Despite this approximation, it remains insightful to highlight how Poisson filters can be applied to solve non-fluid problems. Extensions to other domains remains an avenue for future work.

Table 3 gives statistics about all the simulations showcased in the accompanying video and in this paper. All our results were run on a desktop with 64 GB of memory, NVIDIA GeForce RTX 2070 SUPER and a 6-core Intel(R) Xeon(R) CPU E5-1650 v4 running at 3.6 GHz.

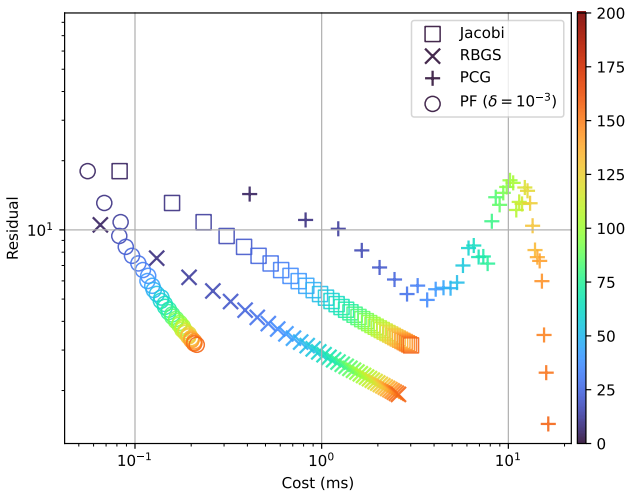


Figure 7: Comparing 2D simulation efficiency across solvers, and with different iteration counts (colors), at resolution 127^2. We plot residual error (y-axis) vs. total cost (GPU + CPU; x-axis). Rank-4 Poisson filters (circles) use a maximum threshold of $\delta = 10^{-3}$. Color bars indicate target iterations, and circle colors the filter order that match the other solvers' iteration counts.

Multigrid. We compare Poisson filters to the V-cycle multigrid method with a Jacobi smoother, $log(N)$ coarsening levels for resolution N. Table 2 demonstrates results for 128^3 and 512^3 resolutions, with smoothing iteration counts of 2 and 3 for each resolution respectively (also same count for pre- and post-smoothing steps). We use the multigrid residual at each iteration to find the best matching Poisson filter order and its δ. Poisson filtering demonstrates a similar cost behaviour to that of the multigrid. Note that our multigrid implementation is not fully optimized, and there are many choices of multigrid methods with different trade-offs between speed and the rate of convergence. We also expect inferior long term convergence behaviour with Poisson filtering compared to multigrid, as the convergence of our method is bounded to Jacobi while multigrid cycles typically reduce all error components by a fixed amount.

7 CONCLUSION AND DISCUSSION

We presented an analytic formulation of Poisson filters that, when combined with a spectral decomposition scheme, admits scalable and high-performance solvers for forward and inverse Poisson problems. We demonstrate their applicability primarily on high-fidelity interactive fluid simulation. We develop a treatment for Neumann boundary conditions and evidence a unique and favourable performance-accuracy profile in the interactive, limited-convergence regime, outperforming iterative methods. Our work opens avenues of discussion and future work, some of which we discuss, below.

Table 2: Multigrid vs. Poisson filters. We compare residual and performances for 128^3 and 512^3, with random inputs in range ± 1. Multigrid residuals are used to find the best matching filter orders with similar convergence values. Filter δ is empirically set to achieve $\approx$ 80% truncation for all filters, where $\delta \in \{10^{-4}, 10^{-1}\}$.

	Resolution (128^3/512^3)		Cost in ms (128^3/512^3)	
MG Itr	MG Residual	PF Matching Order	MG	PF
1	24.5 / 88.6	39 / 51	2.1 / 70	1.3/ 77.5
2	18.8 / 64.7	78 / 102	2.7 / 102	2.4/ 149
3	14.3 / 50.7	117 / 153	3.3 / 139	3.4/ 211
4	11.1 / 41.2	156 / 204	3.8 / 170	4.5/ 286
5	8.9 / 34.4	195 / 255	4.4 / 196	6/ 352

Table 3: Scene statistics for fluid simulations using Poisson filters (PF) of different orders matching target Jacobi iterations. Those simulations omitted from figures appear in our video.

			Cost (solver / simulation in ms)		
Scene	Resolution	Target Iteration	Jacobi	PF	Speed up
Baseline	$192 \times 192 \times 256$	200	216.5 / 226.3	10.8 / 23.2	20.0 / 9.8
ScriptedBall	$192 \times 192 \times 256$	60	65.3 / 78.5	7.8 / 19.8	8.4 / 4.0
BunnySimple	$192 \times 192 \times 256$	60	69.5 / 83	7.6 / 20	9.1 / 4.2
Rabbit	$160 \times 160 \times 256$	60	44.3 / 52.3	5.2 / 13.9	8.5 / 3.8
BunnyContainer	$256 \times 256 \times 256$	60	50.1 / 62.0	4.8 / 16.6	10.4 / 3.7
Rocket	$256 \times 256 \times 96$	60	42.6 / 51.6	5.1 / 13.7	8.4 / 3.8
CubeField	$184 \times 256 \times 88$	60	25.8 / 31.0	2.9 / 8.2	8.9 / 3.8
SmokeRing	$192 \times 192 \times 256$	100	94.2 / 106.5	14.8 / 26.8	6.4 / 4
ManSmoke	$192 \times 192 \times 32$	60	9.3 / 11.6	1.32 / 3.4	7.0 / 3.4
BurningMan	$224 \times 224 \times 192$	60	65.7 / 78.3	7.8 / 20.1	8.4 / 3.9
Torches	$(60\times)72 \times 144$	70	24 / 37.7	1.86 / 12.3	12.9 / 3.1

Improved Boundary Condition Treatment. While *mirror marching* yields reasonable results, some use cases (e.g., non-interactive CFD) require more accurate boundary condition enforcement. We discuss two possible extensions to our method towards this goal.

First, consider hierarchical kernels whose sizes decrease as they approach a boundary. Here, we could treat the internal domain with high-iteration kernels in a first pass before propagating these kernels to boundary borders using ever shrinking kernels; this process resembles Gauss-Seidel iteration. For Neumann boundary conditions, where flow at the boundary (ideally) smoothly matches the behavior of dynamics outside the boundary, such a hierarchical approach may *simultaneously* improve convergence and accuracy.

One could alternatively refine boundary treatment with a semi-iterative strategy using many smaller filters applied sequentially around complex boundaries. Similarly to Jacobi, explicit boundary treatment could be enforced between these iterations.

Relationship to Green's function. Recall that the Green's function constructs solutions to the Poisson equation through repeated convolution with a source term [Evans 1998]. Consider the Poisson problem $\nabla y = f$ on an infinite domain, where the true solution is the convolution $G * f$ for $\nabla G = \delta$, G is the Green's function and δ the Dirac delta. We argue that, since Jacobi converges to the true solution, our Poisson kernel F converges to G. One can regard Poisson filtering as an approximation of the Green's function for the Poisson problem on an infinite domain. This points to an alternative Poisson filter construction *beginning from* the Green's function, instead of from Jacobi iterations. Pursuing this avenue is left to future work and we suspect it may facilitate theoretical convergence analyses and derivation of closed form filters. Moreover, the Green's function could be applied to different domain geometries and boundary conditions, generalizing to a *family* of Poisson filters.

Warm Starting. Our method allows for warm starting in the forward setting. In the inverse setting, we can use b as a warm start (instead of 0) while skipping the first iteration of the kernel generator. Taking traditional perspective, the inability to warm start may seem like a drawback, however warm starting does not improve asymptotic convergence and only serves performance. Given the added context of our method's convergence and performance behavior, and considering that we always only apply our filters once, the traditional gains of warm starting do not hold much weight, here; e.g., we can choose filter orders that match the application of many Jacobi iterations. In our problem settings, warm starting does not affect the solution; however, for more general linear system solves, i.e., in other applications of the Poisson equation (e.g., seamless cloning in image processing), warm starting can be of interest when starting from arbitrary initial guesses.

Limitations and Future Work. As discussed above, supporting a diversity of boundary conditions (e.g., Dirichlet) would broaden the applications that can benefit from Poisson filters. Extensions to irregular grids, such as a multi-resolution generalization of Poisson filters compatible with multigrids [Goodnight et al. 2005], are also worth pursuing. Applying Poisson filters to fluid viscosity and heat diffusion on curved meshes using the Laplace-Beltrami operator are also interesting future directions. Finally, exploring the many structural and conceptual relationships between our iterative filtering process, recurrent neural architectures such as Neural ODEs [Chen et al. 2018], recent works that combine Fourier operators with neural architectures to solve parametric families of PDEs [Li et al. 2020] may lead to interesting hybrid architectures capable of more data-efficient learning of complex dynamics.

REFERENCES

A. Ahmadi, F. Manganiello, A. Khademi, and M. C. Smith. 2021. A Parallel Jacobi-Embedded Gauss-Seidel Method. *IEEE Transactions on Parallel and Distributed Systems* 32, 06 (2021), 1452–1464.

Gonçalo N. P. Amador and Abel João Padrão Gomes. 2010a. A CUDA-Based Implementation of Stable Fluids in 3D with Internal and Moving Boundaries. *2010 International Conference on Computational Science and Its Applications* (2010), 118–128.

Gonçalo N. P. Amador and Abel João Padrão Gomes. 2010b. CUDA-Based Linear Solvers for Stable Fluids. *2010 International Conference on Information Science and Applications* (2010), 1–8.

Gonçalo N. P. Amador and Abel João Padrão Gomes. 2012. Linear Solvers for Stable Fluids: GPU vs CPU.

Marco Ament, Gunter Knittel, Daniel Weiskopf, and Wolfgang Strasser. 2010. A Parallel Preconditioned Conjugate Gradient Solver for the Poisson Problem on a Multi-GPU Platform. In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. 583–592.

John P. Boyd. 2001. *Chebyshev and Fourier Spectral Methods* (second ed.). Dover Publications, Mineola, NY.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. Neural Ordinary Differential Equations. *CoRR* abs/1806.07366 (2018). arXiv:1806.07366

Nuttapong Chentanez and Matthias Müller. 2011. A Fluid Pressure Solver Handling Separating Solid Boundary Conditions. 83–90.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. 2017. The Heat Method for Distance Computation. *Commun. ACM* 60, 11 (2017), 90–99.

Qiaodong Cui, Pradeep Sen, and Theodore Kim. 2018. Scalable Laplacian Eigenfluids. *ACM Trans. Graph.* 37, 4, Article 87 (2018), 12 pages.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid Simulation Using Laplacian Eigenfunctions. *ACM Trans. Graph.* 31, 1, Article 10 (2012), 11 pages.

Lawrence C Evans. 1998. Partial differential equations. *Graduate studies in mathematics* 19, 4 (1998), 7.

Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. 2001. Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 15–22.

O. Fialka and Martin Cadik. 2006. FFT and Convolution Performance in Image Filtering on GPU. In *Tenth International Conference on Information Visualisation (IV'06)*. 609 – 614.

Stefan L. Glimberg, Kenny Erleben, and Jens C. Bennetsen. 2009. Smoke Simulation for Fire Engineering using a Multigrid Method on Graphics Hardware. In *Workshop in Virtual Reality Interactions and Physical Simulation "VRIPHYS" (2009)*, Hartmut Prautzsch, Alfred Schmitt, Jan Bender, and Matthias Teschner (Eds.). The Eurographics Association. https://doi.org/10.2312/PE/vriphys/vriphys09/011-020

Gene H Golub, James M Ortega, et al. 1992. *Scientific computing and differential equations: an introduction to numerical methods*. Academic press.

Nolan Goodnight, Cliff Woolley, Gregory Lewin, David Luebke, and Greg Humphreys. 2005. A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware. In *ACM SIGGRAPH 2005 Courses* (Los Angeles, California) *(SIGGRAPH '05)*. Association for Computing Machinery, New York, NY, USA, 193–es.

Thomas Guillet and Romain Teyssier. 2011. A simple multigrid scheme for solving the Poisson equation with arbitrary domain boundaries. *J. Comput. Phys.* 230, 12 (jun 2011), 4756–4771. https://doi.org/10.1016/j.jcp.2011.02.044

Francis H Harlow and J Eddie Welch. 1965. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The physics of fluids* 8, 12 (1965), 2182–2189.

Ronald D. Henderson. 2012. Scalable Fluid Simulation in Linear Time on Shared Memory Multiprocessors *(DigiPro '12)*. Association for Computing Machinery, New York, NY, USA, 43–52.

Markus Kettunen, Erik Härkönen, and Jaakko Lehtinen. 2019. Deep Convolutional Reconstruction for Gradient-Domain Rendering. *ACM Trans. Graph.* 38, 4, Article 126 (jul 2019), 12 pages. https://doi.org/10.1145/3306346.3323038

Stefan Kindermann and Carmeliza Navasca. 2011. Analysis and Approximation of the Canonical Polyadic Tensor Decomposition. arXiv:1109.3832 [math.NA]

Jean Kossaifi, Yannis Panagakis, Anima Anandkumar, and Maja Pantic. 2019. TensorLy: Tensor Learning in Python. *Journal of Machine Learning Research* 20, 26 (2019), 1–6.

Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. 2020. Fourier Neural Operator for Parametric Partial Differential Equations. *CoRR* abs/2010.08895 (2020). arXiv:2010.08895

Benjamin Long and Erik Reinhard. 2009a. Real-time fluid simulation using discrete sine/cosine transforms. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics*. ACM, 99–106.

Benjamin Long and Erik Reinhard. 2009b. Real-time fluid simulation using discrete sine/cosine transforms. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. 99–106.

Wojciech Matusik. 2003. *A data-driven reflectance model*. Ph.D. Dissertation. Massachusetts Institute of Technology.

Aleka McAdams, Eftychios Sifakis, and Joseph Teran. 2010. A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids. 65–73. https://doi.org/10.2312/SCA/SCA10/065-073

Michael D McCool, Jason Ang, and Anis Ahmad. 2001. Homomorphic factorization of BRDFs for high-performance rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. 171–178.

Tim Mcgraw. 2015. Fast Bokeh Effects Using Low-Rank Linear Filters. *Vis. Comput.* 31, 5 (2015), 601–611.

Olivier Mercier and Derek Nowrouzezahrai. 2020. Local Bases for Model-reduced Smoke Simulations. *Comput. Graph. Forum* 39, 2 (2020), 9–22.

Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, and Jonyong Noh. 2008. Low viscosity flow simulations for animation. In *SCA '08*.

Jorge Nocedal and Stephen Wright. 2006. *Numerical optimization*. Springer Science & Business Media.

Steven A. Orszag. 1969. Numerical Methods for the Simulation of Turbulence. *The Physics of Fluids* 12, 12 (1969), II–250–II–257. arXiv:https://aip.scitation.org/doi/pdf/10.1063/1.1692445

Ari Silvennoinen and Peter-Pike Sloan. 2021. Moving Basis Decomposition for Precomputed Light Transport. *Comput. Graph. Forum* 40, 4 (2021), 127–137.

Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. 2003. Clustered Principal Components for Precomputed Radiance Transfer. *ACM Trans. Graph.* 22, 3 (jul 2003), 382–391. https://doi.org/10.1145/882262.882281

Jos Stam. 1999. Stable Fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999, Los Angeles, CA, USA, August 8-13, 1999*, Warren N. Waggenspack (Ed.). ACM, 121–128.

Jos Stam. 2001. A Simple Fluid Solver based on the FFT. *Journal of Graphics Tools* 6 (2001), 43–52. https://doi.org/10.1080/10867651.2001.10487540

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *CoRR* abs/1607.03597 (2016). arXiv:1607.03597

Lloyd N. Trefethen. 2000. *Spectral Methods in MatLab*. Society for Industrial and Applied Mathematics, USA.

Benjamin Ummenhofer, Lukas Prantl, Nils Thuerey, and Vladlen Koltun. 2020. Lagrangian Fluid Simulation with Continuous Convolutions. In *International Conference on Learning Representations*.

M. Alex O. Vasilescu and Demetri Terzopoulos. 2004. TensorTextures: Multilinear Image-Based Rendering. *ACM Trans. Graph.* 23, 3 (2004), 336–342.

Cheng Yang, Xubo Yang, and Xiangyun Xiao. 2016. Data-driven projection method in fluid simulation: Data-driven projection method in fluid simulation. *Computer Animation and Virtual Worlds* 27 (08 2016). https://doi.org/10.1002/cav.1695