ACM SIGGRAPH / Eurographics Symposium on Computer Animation (2020)
J. Bender and T. Popa (Guest Editors)

Fast Eulerian Fluid Simulation In Games Using Poisson Filters

A. H. Rabbani[1] and S. Khiat[2]

[1]Ubisoft La Forge
[2]Ubisoft La Forge (Former Employee)

Abstract

We present separable Poisson filters to accelerate the projection step in Eulerian fluid simulation. These filters are analytically computed offline and are easy to integrate into any fluid algorithm with a Poisson pressure computation step. We take advantage of the recursive structure of the Jacobi method to construct and then reduce a kernel that is used to solve the Poisson pressure entirely on GPU. Our method demonstrates promising speedups that scale well with both the grid resolution and the target Jacobi iteration.

CCS Concepts

• ***Computing methodologies*** → ***Physical simulation;***

1. Introduction

The efficient simulation of Navier-Stokes equations has always been a longstanding challenge in applied mathematics. The seminal work of Jos Stam [Sta99] and numerous follow-up methods have paved the way for Eulerian fluid simulation algorithms that provide both accuracy and speed for real-time applications. The most demanding part of the Eulerian method is a step known as projection, which ensures the incompressibility condition of the fluid. This step is particularly computationally expensive because contrary to most parts of a stable fluid algorithm that runs efficiently on GPU, the projection step remains heavily dependent on CPU. Traditionally, the bottleneck is caused by solving the discrete Poisson equation by GPU friendly relaxation methods, like the Jacobi algorithm, which still requires additional iterations on the CPU side.

We propose separable Poisson filters to speedup projection by replacing the iterative Jacobi solver with analytically computed separable filters that directly run on GPU in two convolutional passes. Such filters are computed offline and are compatible with any stable fluid algorithm as long as there is a projection step. We introduce a truncation factor to minimize the size of our filters with a negligible impact on the projection equality, further reducing the number of arithmetic operations. For the convenience of the user, the tradeoff between the cost and the quality of the convolution step is controlled in real-time in the form of GUI sliders.

The presented work is a 2D system that is capable of interacting with objects in a 3D scene through depth buffer analysis. We also showcase a set of rendering techniques that create the illusion of a 3D simulation. This is a research work in progress to extend Poisson filters to 3D. We are likewise interested in the application of our filters to speed up the viscous-diffusion step, which is even more expensive than the projection step.

2. Related Work

A number of recent studies propose promising data-driven solutions to accelerate the projection steps (e.g. [TSSP16]). While successful in reducing the dependency on CPU, these methods often suffer from reproducibility of the results due to dependency on the training step. Moreover, dealing with Neumann boundary conditions forcibly result in the additional complexity of the trained model in the form of added hidden layers in a convolutional neural network, for instance, where the added layers increase the arithmetic computational cost as well. In contrast, our proposed filters are easy to implement, do not need retraining for different pipelines, and they deal with the Neumann boundaries the same way that the traditional solvers do.

Perhaps the closets work in nature to our method is the family of studies that uses spectral analysis. These algorithms simulate the fluid partially or entirely in a spectral domain. Some of these methods (e.g. [Sta01] and its followup works) use Fast Fourier Transform (FFT) and only work in periodic domains with no possibility for complex boundary considerations. For comparison, our separable Poisson filters work in both periodic or non-periodic spatial domains and exhibit better performance in the projection step, specifically for high-resolution grids and small filter sizes. The computational complexity of filtering a $N \times N$ field with separable convolution is $O(2k.N^2)$ where k denotes the size of the kernel, whereas with FFT the time increases to $O(4N^2.\log_2(N))$ [FC06]. More recent techniques overcome the limitations of the original FFT approaches in addressing the Neumann boundaries (e.g. [CSK18]) and present interesting subtleties in the fluid behavior. We argue that for realtime applications like a smoke effect in a game where the visuals are often sacrificed by highly constrained computational budgets, our method is more preferable because it provides a satis-

factory visual quality with a reliable baseline for both accuracy and speed.

3. Poisson Filters For The Projection Step

Our Eulerian fluid dynamics behaviour is governed by the Navier-Stokes equations on a Cartesian grid, with a velocity field u and a pressure field p

$$\frac{\partial u}{\partial t} = -(u.\nabla)u - \frac{1}{\rho}\nabla p + \upsilon\nabla^2 u + F, \quad \nabla.u = 0 \tag{1}$$

where ρ and υ are the density and kinematic viscosity, F is the external force, and $\nabla.u = 0$ is the incompressibility condition. To ensure $\nabla.u = 0$ we perform a projection step in two steps. First computing the pressure by solving $\nabla^2 p = \nabla.w$ where w is the divergent velocity field to be corrected, then subtracting the pressure gradient from w to make the velocity divergence-free. The latter step works due to the Helmholtz-Hodge Decomposition. The reader is referred to [Sta99] for more details.

Equations of the form $\nabla^2\mathbf{x} = \mathbf{b}$ are known as Poisson equations, where the special case of $\mathbf{b} = 0$ is known as Laplace's equation. While in what follows we mainly focus on Laplace's equation to make the fluid incompressible during the projection step, we prefer to call our method Poisson Filters because the same principles in our speedup technique are applicable to the general case of $\mathbf{b} \neq 0$. An example is the viscous-diffusion step that is not discussed in this work but it is a part of our ongoing research work.

3.1. Jacobi-Poisson Kernel Computation

We seek to solve for the unknown pressure p in $\nabla^2 p = \nabla.w$, which requires setting up a linear system of the form $A\mathbf{x} = \mathbf{b}$. Instead of explicitly computing ∇^2, iterative relaxation methods like Jabobi are often used to implicitly approximate A. This is done by the decomposition $A = D + L + U$ where D, L and U are the diagonal, lower and upper triangular parts. This also makes it possible to avoid inverting A explicitly. The solution after k steps is approximated by

$$\mathbf{x}^{(k+1)} = D^{-1}(\mathbf{b} - (L+U)\mathbf{x}^{(k)}). \tag{2}$$

The Jacobi method is GPU friendly, meaning the update step can be executed in parallel for all elements of $\mathbf{x}$ on GPU. The number of iterations (k) determines the CPU-side call cycles, which is substantially more expensive than the GPU overhead. In order to minimize the CPU workload, we can reformulate the problem to have a full convolutional kernel matrix C that is executed only once and on GPU. Such kernel is equivalent to the k_{th} step of the Jacobi method

$$\mathbf{x}^{(k)} = C^{(k)} * \mathbf{b}. \tag{3}$$

We take advantage of the recursive structure of the Jacobi method to compute C. The element wise version of the Jacobi method for computing the pressure is

$$x_{i,j}^{k+1} = \frac{x_{i-1,j}^{k} + x_{i+1,j}^{k} + x_{i,j-1}^{k} + x_{i,j+1}^{k} + \alpha b_{i,j}}{\beta}, \tag{4}$$

where $x_{i,j}$ is pressure, $b_{i,j}$ is divergence ($\nabla.w$), $\alpha = -(dx)^2$ for a given unit cell size of dx and $\beta = 4$. Knowing α, β and $b_{i,j}$ we construct C through recursively computing the value of $x_{i,j}^{k}$ based on all its previous values for the last k steps, i.e. computing $x_{i,j}^{k}$ backwards to $x_{i,j}^{0}$. Note that the size of the kernel grows linearly by $2 \times k + 1$ for k iterations. This means C might become very large because many Jacobi iterations are often necessary during the projection step to achieve accurate pressure values, making it computational very expensive even on GPU. To remedy this problem we look into linearizing and reducing the Kernel into horizontal and vertical filters.

3.2. Filter Computation

We aim to replace the large kernel C with separable filters that best approximate its functionality. Doing so reduces the computational cost as we have $O(2c)$ for separable filters computed from a $c \times c$ kernel size as opposed to $O(c^2)$ for the full kernel. The performance of the filters greatly dominates both the full kernel and the Jacobi method specially when high Jacobi iterations are of interest. To that end, we perform Singular Value Decomposition (SVD) on C but from a different point of view. We can interpret SVD of C as a weighted sum of separable matrices C_is

$$C = \sum_i C_i = \sum_i \sigma_i U_i \otimes V_i, \tag{5}$$

with U_i and V_i being the i_{th} columns of the SVD matrices with corresponding singular values σ_i, where $\otimes$ is the outer product. This helps us converting each separable kernel C into an outer product of two vectors. We compute a pair of rank 1 separable filters, v and h, by choosing $i = 1$ while replacing $\mathbf{x}$ and $\mathbf{b}$ in equation 3 with the pressure (p) and divergence ($\nabla.w$) to get our final equation

$$p = (\sigma_1 U_1 \otimes V_1) * \nabla.w = (v \otimes h) * \nabla.w \tag{6}$$

where v and h are the vertical and the horizontal separable filters, each scaled by $\sqrt{\sigma_1}$. We dropped (k) from equation 3 for simplicity without loss of generality. In practice, we use v and h in two convolutional passes over the divergence field. The order of convolution is a horizontal pass followed by a vertical pass. In order to avoid shrinkage of the computed field due to the convolutional passes, we use a dynamic zero padding, which involves adding adequate zeros to the boundaries based on the filter size. In addition, we observed about 80% of the computed filters are significantly small and have little effect on the convolution operation. By further truncating the filters we substantially reduce the computational cost. An example of Poisson filters for a Jacobi iteration of 10 is shown in Figure 1. Another reassuring aspect of our work is the fact that the variance explained of the reduced kernel never drops below 83%, which occurs only for very high number of target Jacobi iterations. This ensures an acceptable loss of quality when performing the projection step. Note that while reducing the dimensionality of C to 1 produces satisfactory results, we can likewise opt for the inclusion of higher rank filters for higher quality pressure computations. Still our method will result in a small added cost because we can convolve these filters in parallel on GPU. This is something we will present in our upcoming research work.

4. Results

We implemented our version of the stable fluid algorithm in our in-house Ubisoft La Forge prototyping platform RenderKitt. The GPU components are implemented in Direct3D 12 shader code. All performance numbers presented in this document or in the accompanying video were measured on PC, with Intel Xenon 3.6 GHz and NVIDIA GeForce GTX 1060 6GB. In addition to standard self-advection and projection steps, our simulation pipeline consists of density and temperature advection, as well as vorticity confinement, to achieve the desired fire and smoke effect we were interested in. Here we only present results for a 2D implementation.

Performance. For a target Jacobi iteration of 32, which is a typical value for a good convergence in solving the Poisson pressure equation, and a grid resolution of 128×128, we observed $17.4\mu s$ for the two passes of the Poisson filters. The original Jacobi implementation takes about $350\mu s$ for the same task. This means $\approx 20\times$ speed-up is achieved by avoiding most of the CPU workload when solving for pressure on GPU. In total, a complete simulation step takes about $418\mu s$ with the traditional solver, as opposed to $95\mu s$ with our proposed solution. We have included a plot in the video to show how the Poisson filter performance scales when increasing the target iteration. In general, we observed for a fixed resolution when we increment the target Jacobi iteration, the cost of our method increases $0.3\mu s$ per added iteration step, while the traditional solver increases $11\mu s$ per added iteration step. For example, for $itr = 100$ the traditional solver takes $1079\mu s$ whereas our filters only take $37.4\mu s$.

Integration into 3D scene. We use a number of techniques to integrate our fast 2D smoke and fire simulation into a 3D scene, hence calling it a 2.5D simulation. A dynamic camera-based correction technique helps create the illusion of a 3D simulation as it makes a number of transformation adjustments to the simulation texture to make it seem invariant to the camera motion in the scene. The simulation texture can also collide with high-resolution 3D meshes. This is done by applying a half Laplacian kernel to the renderer depth buffer in order to extract the 3D object contours and edges (Figure 2). This is a computationally cheap way to achieve high fidelity 3D collision that only takes $7\mu s$. The same technique can be used to set any 3D object on fire, which provides enormous possibilities for in-game applications. Lastly, we developed a universal system of reinterpreting 3D forces in the scene (wind, motion, gravity, etc.) into a coherent 2D force in the simulation texture *uv* space through a combination of jerk computation and a Perlin noise to compensate for the missing 3rd dimension. For more details, we invite the reader to watch our demo video.

5. Conclusions and Future Work

We present separable Poisson filters to accelerate the projection step in Eulerian fluid simulation. These filters are analytically computed offline and are easy to integrate into any fluid algorithm with a Poisson pressure computation step. As our next step, we are seeking solutions to generalize our method from 2D to 3D, as well as addressing the viscous-diffusion step as it exhibits the same algebraic structure as the Poisson pressure solution obtained from the Jacobi method.

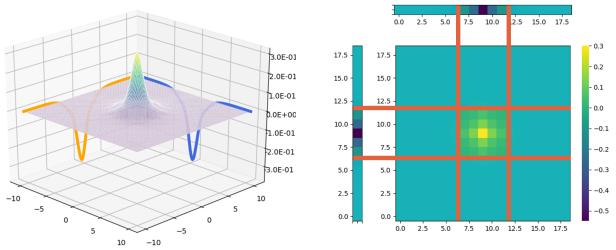

Figure 1: *Poisson filters for $itr = 10$. Left: Full kernel with separable filters. Right: The same filters with 80% truncation (red lines).*

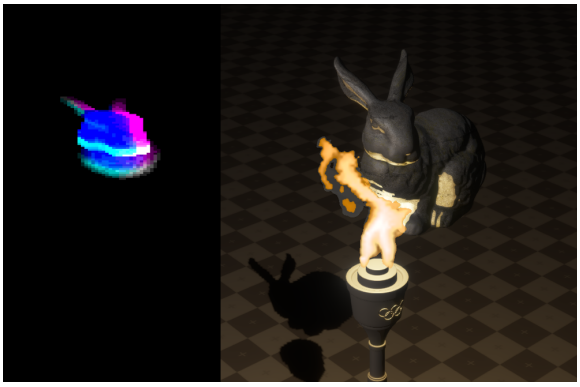

Figure 2: *2D fire colliding with bunny. Left: Processed depth buffer as seen from the simulation point of view. Edges are extracted by a half Laplacian convolutional kernel. Right: The simulation uses the 3D mesh contours for the Neumann boundary conditions.*

References

[CSK18] CUI Q., SEN P., KIM T.: Scalable laplacian eigenfluids. *ACM Trans. Graph.* (2018), 87:1–87:12. 1

[FC06] FIALKA O., CADIK M.: Fft and convolution performance in image filtering on gpu. In *Tenth International Conference on Information Visualisation (IV'06)* (08 2006), pp. 609 – 614. doi:10.1109/IV.2006.53. 1

[Sta99] STAM J.: 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* (1999), Waggenspack W. N., (Ed.), ACM, pp. 121–128. URL: https://doi.org/10.1145/311535.311548, doi:10.1145/311535.311548. 1, 2

[Sta01] STAM J.: A simple fluid solver based on the fft. *Journal of Graphics Tools 6* (10 2001), 43–52. doi:10.1080/10867651.2001.10487540. 1

[TSSP16] TOMPSON J., SCHLACHTER K., SPRECHMANN P., PERLIN K.: Accelerating eulerian fluid simulation with convolutional networks. *CoRR abs/1607.03597* (2016). URL: http://arxiv.org/abs/1607.03597. 1