Supplementary Material

AMIR HOSSEIN RABBANI, Ubisoft Montreal, Canada
JEAN-PHILIPPE GUERTIN, Ubisoft Montreal, Canada
DAMIEN RIOUX-LAVOIE, Ubisoft Montreal, McGill University, Canada
ARNAUD SCHOENTGEN, Ubisoft Montreal, Canada
KAITAI TONG, Ubisoft Montreal, University of British Columbia, Canada
ALEXANDRE SIROIS-VIGNEUX, Ubisoft Montreal, McGill University, Canada
DEREK NOWROUZEZAHRAI, McGill University, Canada

ACM Reference Format:
Amir Hossein Rabbani, Jean-Philippe Guertin, Damien Rioux-Lavoie, Arnaud Schoentgen, Kaitai Tong, Alexandre Sirois-Vigneux, and Derek Nowrouzezahrai. 2022. Supplementary Material. 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, 13 pages. https://doi.org/10.1145/3528233.3530737

1 UNIFIED KERNEL EQUATION

We derive the Jacobi update step for the inverse and forward Poisson's equations using the central difference scheme. As a reminder, the central difference scheme for ∇^2 in $3D$ and for the matrix form of the linear system is

$$\nabla^2 X = \frac{X_l + X_r + X_d + X_u + X_b + X_f - 6X_c}{(\Delta x)^2}, \tag{1}$$

where c is center, l is left, r is right, d is down, u is up, b is back, and f is front, and Δx is the unit size in the discrete space. We use directional indices instead of (i, j, k) to avoid clutter in our derivations. Also note that this equation does not include time, but we will include time in our derivations for diffusivity (forward Poisson) because we are interested in the temporal change of state of the diffuse quantity, whereas we do not include time for pressure (inverse Poisson) because we are only interested in the spatial relationship between the divergence of the field and the corresponding pressure at a specific time.

1.1 Heat Diffusion (Forward Poisson)

We derive the discrete update equation for diffusivity. We only discuss the heat equation, but the same equation is used for density diffusion and viscosity as well. The only difference is that we need to apply the viscosity update step for all 3 dimensions separately as we are dealing with a vector field (as opposed to scalar fields for density and temperature).

The transient heat equation is

$$\frac{\partial T}{\partial t} = \kappa \left[\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right], \tag{2}$$

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

where κ is the thermal diffusivity. Equation (2) is a second order partial differential equation, which means ∇^2 will show up in our equation,

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T \,. \tag{3}$$

The general form of the central difference scheme without considering time is

$$T^{n+1} = T^n + \kappa \nabla^2 T^n \,. \tag{4}$$

We expand ∇^2 and include time to solve for the update step. We arrive at an explicit transient 3D heat conduction formulation by discretizing Equation (2). This gives

$$\frac{T_c^{n+1} - T_c^n}{\Delta t} = \kappa \left[\frac{T_l^n - 2T_c^n + T_r^n}{(\Delta x)^2} + \frac{T_d^n - 2T_c^n + T_u^n}{(\Delta y)^2} + \frac{T_b^n - 2T_c^n + T_f^n}{(\Delta z)^2} \right], \tag{5}$$

assuming $\Delta x = \Delta y = \Delta z$. We solve for T_c^{n+1}, leading to

$$T_c^{n+1} = T_c^n + \frac{\kappa \Delta t}{(\Delta x)^2} \left[T_l^n + T_r^n + T_d^n + T_u^n + T_b^n + T_f^n - 6T_c^n \right]. \tag{6}$$

We are, however, more interested in the **implicit** derivation in order to improve stability, such that

$$T_c^{n+1} = T_c^n + \frac{\kappa \Delta t}{(\Delta x)^2} \left[T_l^{n+1} + T_r^{n+1} + T_d^{n+1} + T_u^{n+1} + T_b^{n+1} + T_f^{n+1} - 6T_c^{n+1} \right]. \tag{7}$$

Note the replacement of n with $n+1$. If $\frac{\kappa \Delta t}{(\Delta x)^2} = \gamma$, rewriting for T_c^{n+1} gives

$$T_c^{n+1} = \left[\frac{1}{1+6\gamma} \right] T_c^n + \left[\frac{\gamma}{1+6\gamma} \right] \left[T_l^{n+1} + T_r^{n+1} + T_d^{n+1} + T_u^{n+1} + T_b^{n+1} + T_f^{n+1} \right]. \tag{8}$$

Refactoring, we get

$$T_c^{n+1} = \left[\frac{\gamma}{1+6\gamma} \right] \left[\frac{T_c^n}{\gamma} + T_l^{n+1} + T_r^{n+1} + T_d^{n+1} + T_u^{n+1} + T_b^{n+1} + T_f^{n+1} \right], \tag{9}$$

and rewriting $\frac{\gamma}{1+6\gamma}$ as $\frac{1}{\frac{1}{\gamma}+6}$ we obtain a simplified equation

$$T_c^{n+1} = \frac{\frac{1}{\gamma} T_c^n + T_l^{n+1} + T_r^{n+1} + T_d^{n+1} + T_u^{n+1} + T_b^{n+1} + T_f^{n+1}}{\frac{1}{\gamma} + 6} \,. \tag{10}$$

By replacing $\frac{1}{\gamma} = \alpha$ and $\frac{1}{\gamma} + 6 = \beta$, we obtain an equation similar in structure to the unified equation mentioned in our paper

$$T_c^{n+1} = \frac{T_l^{n+1} + T_r^{n+1} + T_d^{n+1} + T_u^{n+1} + T_b^{n+1} + T_f^{n+1} + \alpha T_c^n}{\beta}, \tag{11}$$

where $\alpha = \frac{(\Delta x)^2}{\kappa \Delta t}$ and $\beta = \alpha + 6$. Diffusion equation is the same for the transient heat conductivity, density diffusion and viscosity. For the $2D$ version it is the same equation except $\beta = \alpha + 4$ and there is no $T_b^{n+1} + T_f^{n+1}$.

1.2 Poisson-Pressure (Inverse Poisson)

Incompressible fluids must respect the divergence free condition

$$\nabla \cdot u = 0\,, \tag{12}$$

where $\nabla\cdot$ is the divergence operator and u is the flow field. Given the Helmholtz-Hodge decomposition of a divergent fluid field,

$$w = u + \nabla p\,, \tag{13}$$

with w being the divergent field, and applying the divergence operator to both sides and re-arranging, we obtain

$$\nabla \cdot w = \nabla \cdot (u + \nabla p) = \nabla \cdot u + \nabla^2 p\,. \tag{14}$$

Enforcing $\nabla \cdot u = 0$ results in

$$\nabla^2 \cdot p = \nabla \cdot w\,. \tag{15}$$

Since this is a steady equation, we use Equation (1) to get the central difference scheme for the matrix form

$$d = \nabla \cdot W = \frac{P_l + P_r + P_d + P_u + P_b + P_f - 6P_c}{(\Delta x)^2}\,, \tag{16}$$

where d is scalar divergence value. As divergence is given and we are interested in finding the pressure at the current position P_c, we solve for the unknown P_c, giving

$$P_c = \frac{P_l + P_r + P_d + P_u + P_b + P_f - (\Delta x)^2\, d}{6}\,. \tag{17}$$

Much like Equation (11), this equation is also similar in structure to the unified equation mentioned in our paper. Note that we avoided an explicit inversion of ∇^2 in the inverse version. For the $2D$ case it will be the same equation except $\beta = 4$ and there is no $P_b + P_f$.

Finally, we write our unified update equation for both forward and inverse Laplacian

$$X^{n+1} = \frac{X_l^n + X_r^n + X_d^n + X_u^n + X_b^n + X_f^n + \alpha B}{\beta}\,, \tag{18}$$

with α and β values being

- Pressure (inverse): $\alpha = -(\Delta x)^2$, $\beta = 6$
- Diffusion (forward): $\alpha = \frac{(\Delta x)^2}{\kappa \Delta t}$ and $\beta = \alpha + 6$,

and where B is the central thermal or the divergence term depending on the application.

2 COMPATIBILITY CONDITION

Consider a Poisson equation with Neumann boundary condition over a domain Ω with boundary $\partial\Omega$,

$$\nabla^2 \varphi = f \quad \text{in } \Omega\,, \tag{19}$$

$$\nabla \varphi \cdot \mathbf{n} = 0 \quad \text{on } \partial\Omega\,, \tag{20}$$

where $\mathbf{n}$ is the outward pointing boundary normal vector of $\partial\Omega$. Then by integrating each side over Ω, we get

$$\underbrace{\int \cdots \int_{\Omega}}_{d} \nabla^2 \varphi \, \mathrm{d}\mathbf{x} = \underbrace{\int \cdots \int_{\Omega}}_{d} f \, \mathrm{d}\mathbf{x}\,, \tag{21}$$

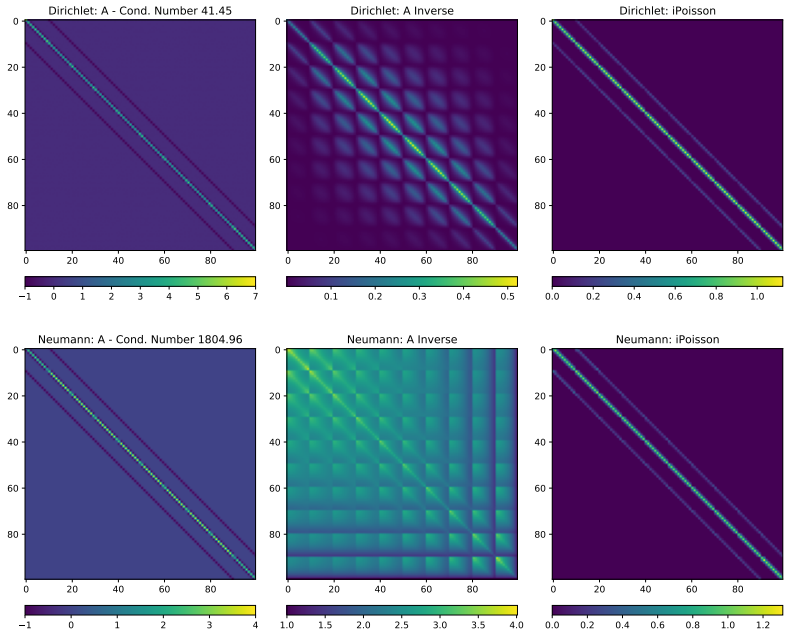

Fig. 1. Incomplete Poisson preconditioner of (Top) Dirichlet and (Bottom) Neumann Laplacian matrix with last element fixed to zero. On the Left we have the Laplacian matrix, center it's inverse and right the incomplete Poisson preconditioner. As we can see, this preconditioner is quite badly adapted to pure Neumann boundary.

where $d \in \{2, 3\}$ is the dimension of the problem. Then, observing that $\nabla^2\varphi = \nabla \cdot (\nabla\varphi)$, using Stokes theorem and the boundary condition, we get that the source term f must satisfy

$$\underbrace{\int \cdots \int_{\Omega}}_{d} f \, \mathrm{d}\mathbf{x} = \underbrace{\int \cdots \int_{\Omega}}_{d} \nabla^2\varphi \, \mathrm{d}\mathbf{x} = \underbrace{\int \cdots \int_{\partial\Omega}}_{d-1} \nabla\varphi \cdot \mathbf{n} \, \mathrm{d}\mathbf{x} = 0 \,. \tag{22}$$

In layman's terms, f should integrate to 0 over the domain for this problem to be well defined.

3 PRECONDITIONED CONJUGATE GRADIENT (PCG)

When solving the Neumann Poisson problem, the resulting linear system is singular by construction. Since PCG works directly with this matrix to compute its preconditioner, we need to ensure that this is not the case by fixing a value on the boundary. This can be achieved by adding a value, say 1, to one of the diagonal entries. This has the effect of setting the value associated to this row to 0. While this makes the system non-singular, it is still poorly conditioned and only becomes worse with higher resolutions. To ensure fairness, we choose to compare against the incomplete Poisson preconditioner [Ament et al. 2010; Sideris et al. 2011], as it is the only one – that we know of – which can easily be used in a GPU compute shader with minimal overhead since we are interested in real-time applications. We found a large decrease in efficiency for PCG when using this preconditioner when using Neumann boundary conditions. After further investigation, we found out that this preconditioner is a poor match for true inverse matrix, which restricts its utility in spite of its GPU-friendliness (see Figure 1). This is unsurprising, as optimal preconditioners for Neumann problems are difficult to obtain [Lee and Min 2021].

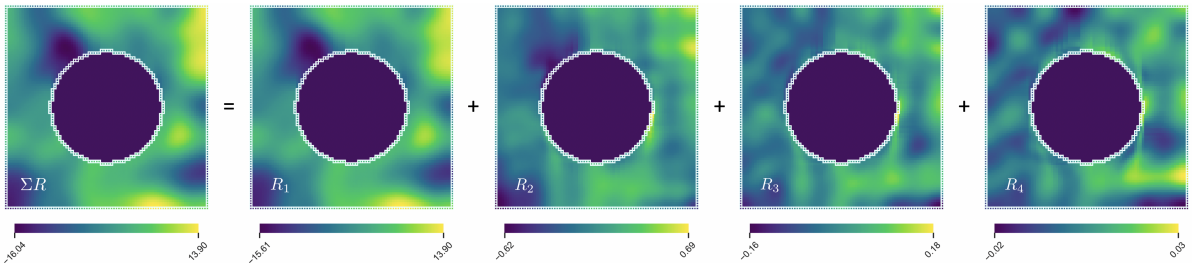

Fig. 2. Poisson-pressure solves with Neumann boundaries (solid circle enforcing $\nabla P = 0$), using the first 4 ranks of 100^{th}-order filters, with random inputs in ± 1 and domain size 81^2. Our solution is a summation of modal solutions, ranging from low ($R1$) to high frequencies ($R4$).

4 2.5D RENDERING

To make our 2D simulation appear more volumetric, we devised a set of heuristics and render filters to give the illusion of depth without introducing an additional dimension to the simulation. This is done by creating an extrusion buffer. These can be created by artists, through advected noise fields or even simply by using a simulation texture such as the advected density field itself. In future work, it would be interesting to utilize machine learning methods to generate these extrusion textures. Once we have this field, we can precompute a single sample single scattering estimation. For multiple scattering, we proceed the same way as in 3D and use our Poisson filters to approximate the diffusion of emission through the density field.

5 3D RENDERING

Rendering uses a simple ray marching algorithm walking through the depth range of the volume or depth buffer, whichever is closest. A precomputation is performed each frame on the full volume to determine single-scattering transmittance from the directional sunlight which is then fetched at each marching step and composited with the ray's accumulated emitted and transmitted radiance. Ray marching taps are snapped to integer steps of the view direction and randomly offset using 3D blue noise to minimize banding artifacts.

Collisions and emission are handled via a combination of analytical functions, precomputed signed distance fields and a real-time approximation using G-Buffer information to extract intersecting regions from the depth buffer and screen-space normals.

6 MODAL SOLUTIONS

In Figure 2 we demonstrate the Poisson-pressure solution as a summation of modal solutions, each of which enforcing the Neumann boundaries. The orthogonality of the modal solutions allows for a parallel implementation.

7 ADDITIONAL COST DATA

In Table 1, we present an additional cost of our solver on fictitious data and without any boundary condition treatment. Filter δ is empirically set to achieve $\approx$ 85% truncation for all filters, where $\delta \in \{10^{-4}, 10^{-1}\}$. Note that our approach is an order of magnitude faster for both the Poisson solver step and the total simulation cost, particularly for higher target iteration counts.

Table 1. Poisson filters (PF) total (GPU+CPU) computational cost compared to Jacobi at different iteration counts (filter orders in case of PF) and resolutions. *Solver* indicates the step that involves the Poisson equation solve, and *Simulation* indicates the total simulation cost. We also highlight the speed-up gain obtained with our filters.

	Resolution		64		256	
Iteration	Dim	Component	Jacobi (ms)	PF (ms)	Jacobi (ms)	PF (ms)
100	2D	Solver	1.12	0.027 (× **41.48**)	1.15	0.038 (× **30.26**)
		Simulation	1.30	0.251 (× **5.17**)	1.42	0.305 (× **4.65**)
	3D	Solver	4.49	0.30 (× **14.98**)	221.69	16.66 (× **13.30**)
		Simulation	5.21	1.01 (× **5.14**)	249.07	44.05 (× **5.65**)
200	2D	Solver	2.38	0.028 (× **85**)	2.589	0.050 (× **51.78**)
		Simulation	2.61	0.252 (× **10.35**)	2.859	0.316 (× **9.05**)
	3D	Solver	8.96	0.40 (× **22.22**)	446.40	23.35 (× **19.12**)
		Simulation	9.67	1.12 (× **8.66**)	473.78	50.73 (× **9.34**)
300	2D	Solver	3.31	0.030 (× **110.3**)	3.66	0.062 (× **59**)
		Simulation	3.56	0.254 (× **14**)	3.94	0.328 (× **12**)
	3D	Solver	13.42	0.48 (× **27.85**)	668.43	28.34 (× **23.59**)
		Simulation	14.13	1.20 (× **11.82**)	695.82	55.72 (× **12.49**)

8 MEMORY FOOTPRINT

Figure 3 compares the memory footprints of our adaptive filters. As we see, the number of important values only increase sublinearly with the order, supporting the sparsity claim on our filters.

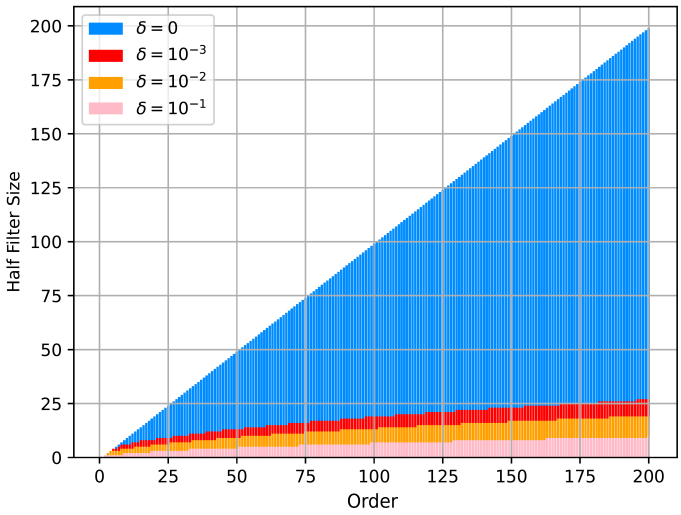

Fig. 3. Low memory footprint using adaptive truncation with thresholds $\delta \in \{0, 10^{-3}, 10^{-2}, 10^{-1}\}$. Half filter sizes are the actual number of filter values required to be stored in memory.

9 TRANSIENT HEAT DIFFUSION

We solve the heat equation on a surface mesh Figure 4. Note that we do not treat mesh curvature in this last example, so we are not solving with the Laplace-Beltrami operator for curved surfaces.

10 PSEUDOCODE

In Algorithm 1, we present our multi-rank convolution algorithm for Poisson filters with mirror marching for Neumann boundary treatment. Considering the Poisson equation $\nabla^2\varphi = f$, the *input* tensor is either f in the inverse Poisson setup, or φ in the forward version. Lines 1–8 apply the same 1D filter in 3 directions of the major tensor axes. Note that while in the first pass (Fiber) the input to the convolution function is the input tensor, the inputs to the following passes are the output of the previous convolution passes. This order is critical in achieving the correct solution.

The presented algorithm is *not* suitable for a parallel (i.e. GPU friendly) implementation, but it is straightforward to do so by making two adjustments: first, replacing the nested loop in lines $10 - 12$ with a multi-threaded texel access; second, replacing the loop in line 3 by encoded ranked convolution operations as 4-wide vectors in the shader code to achieve parallel multi-rank convolution. This encoding makes it possible to apply up to four kernels simultaneously during convolution at no additional cost thanks to SIMD processing. The key insight in the multi-rank convolution in a parallel setup is the orthogonality condition of the modal solutions. In other words, each ranked solution does not depend on the result of other passes.

11 KERNEL, MODES AND FILTERS

Here we visualize kernels, modes and filters for both inverse and forward Poisson for the 2D cases. In Figure 5 we have the full Poisson kernels. In Figures 6, 7, 8, and 9 we have the first and second four ranks of the inverse

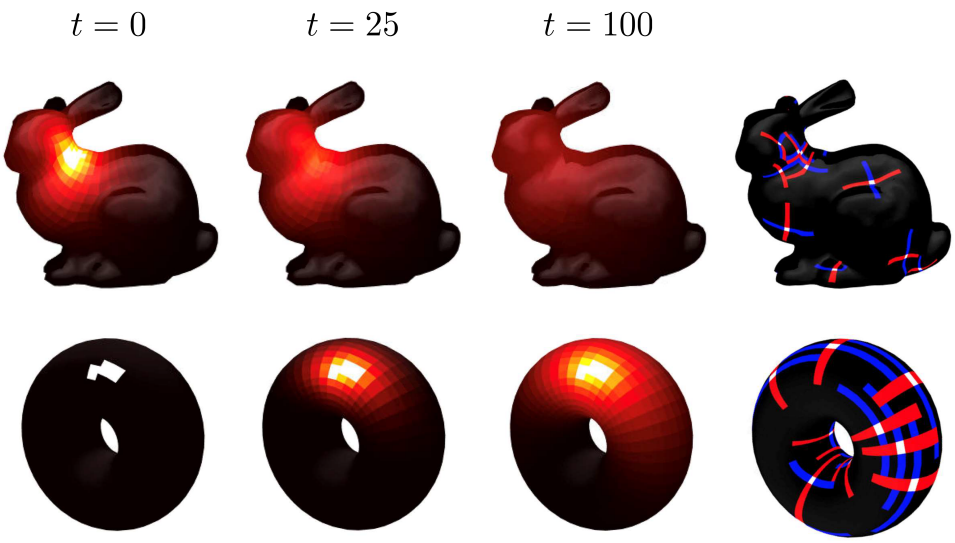

Fig. 4. Transient heat diffusion on a surface mesh without boundary conditions (not accounting for curvature). Left to right: cooling a bunny's (top) and heating up a torus' (bottom) surface for 100 seconds. Rightmost: visualizations of randomly-located horizontal-vertical Poisson filter patches. We apply rank-3 60^{th}-order filters with $\kappa = 0.2$, $\Delta t = 1$, $\Delta x = 1$, and with an adaptive truncation of $\delta = 10^{-2}$.

Algorithm 1 Multi-Rank Convolution with Poisson Filters

procedure SolvePoisson($input$, $filters$, $rank_{max}$) ▷ $input$ is a tensor
  $sum = 0$ ▷ sum is a tensor with the same size as $input$
  for $rank$ **in** $rank_{max}$ **do**
    $output_f$ = ConvolveSingleFilter($input$, $filters[rank]$, z_{axis}) ▷ Fiber Pass
    $output_h$ = ConvolveSingleFilter($output_f$, $filters[rank]$, y_{axis}) ▷ Horizontal Pass
    $output_v$ = ConvolveSingleFilter($output_h$, $filters[rank]$, x_{axis}) ▷ Vertical Pass
    sum += $output_v$ ▷ Modal Solution Contribution
  return sum
procedure ConvolveSingleFilter($input$, $filter$, $orientation$)
  for r **in** $input$ tensor rows **do**
    for c **in** $input$ tensor columns **do**
      for d **in** $input$ tensor depths **do**
        $index = (r, c, d)$
        $output[index]$ = MirrorMarchConvolve($index$, $input$, $filter$, $orientation$)
  return $output$ tensor
procedure MirrorMarchConvolve($index$, $input$, $filter$, $orientation$)
  if $index$ **is in** collider **then**
    return 0
  $center$ = half filter size ▷ Initialize filter $center$ index
  $sum = filter[center] * input[index]$ ▷ Get contribution at $index$
  Initialize $index^+$ and $index^-$ similar to $index$ ▷ Prepare for index marching
  Initialize $step^+$ and $step^-$ depending on $orientation$ ▷ Initialize $step$ directions
  for Δ **in** half filter size **do**
    March($index^+$, $step^+$)
    March($index^-$, $step^-$)
    sum += $filter[center + \Delta] * input[index^+]$ ▷ Accumulation on + filter direction
    sum += $filter[center - \Delta] * input[index^-]$ ▷ Accumulation on − filter direction
  return sum
procedure March($index$, $step$)
  Move $index$ one cell in the direction of $step$
  if $index$ **is in** collider **then**
    Move back $index$ and mirror $step$

and forward filter decompositions, respectively. As we can see, the decomposition form a sequence of low-pass filters with an increasing frequency range.

REFERENCES

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*. IEEE, 583–592.

Byungjoon Lee and Chohong Min. 2021. Optimal preconditioners on solving the Poisson equation with Neumann boundary conditions. *J. Comput. Phys.* 433 (2021), 110189.

Costas Sideris, Mubbasir Kapadia, and Petros Faloutsos. 2011. Parallelized incomplete poisson preconditioner in cloth simulation. In *International Conference on Motion in Games*. Springer, 389–399.

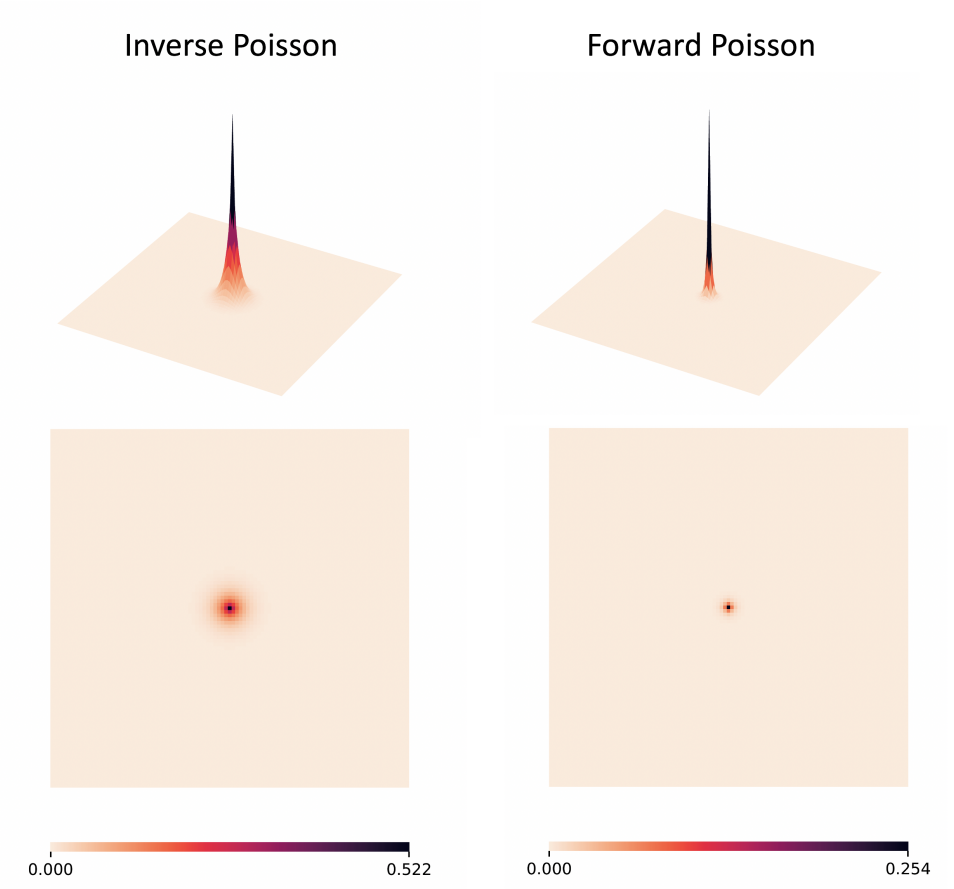

Fig. 5. 2D inverse and forward 50^{th}-order Poisson full kernels, and for forward: $\kappa = 1$, $\Delta t = 1$, $\Delta x = 1$.

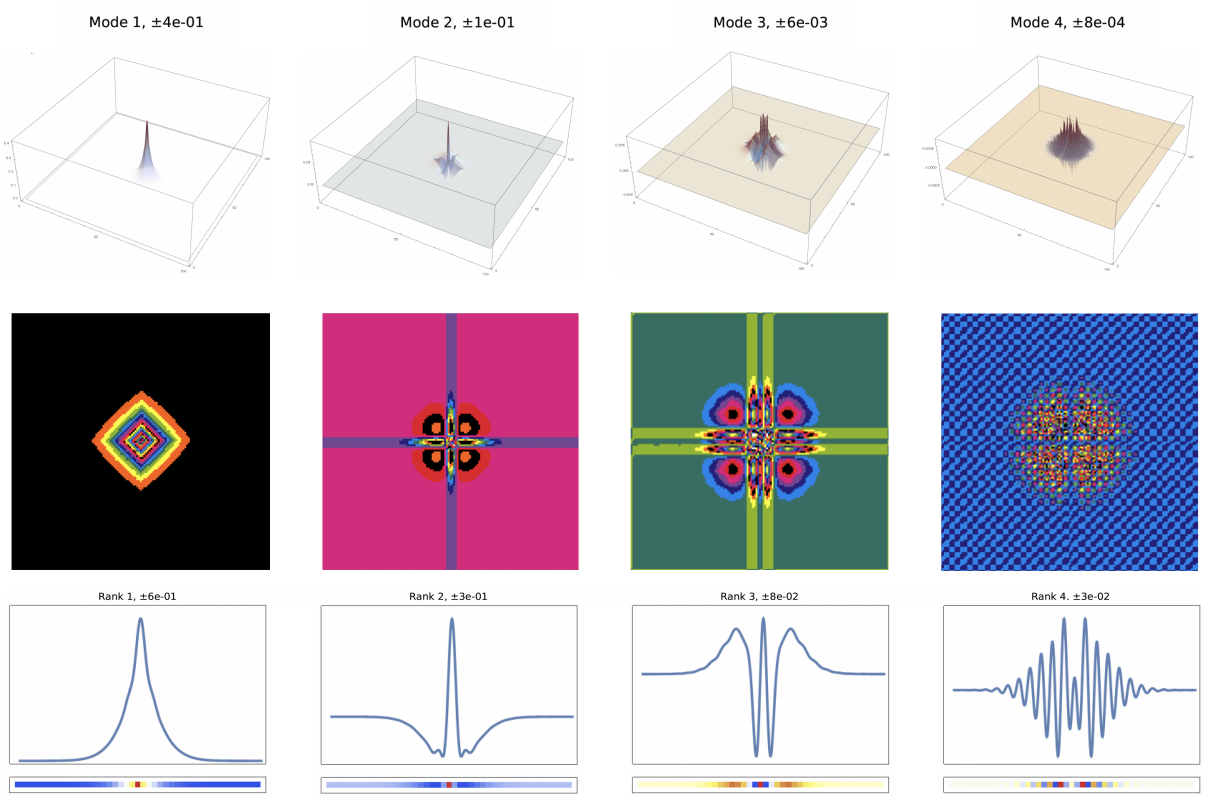

Fig. 6. 2D inverse Poisson: 50^{th}-order kernel modes and filters, ranks 1 to 4.

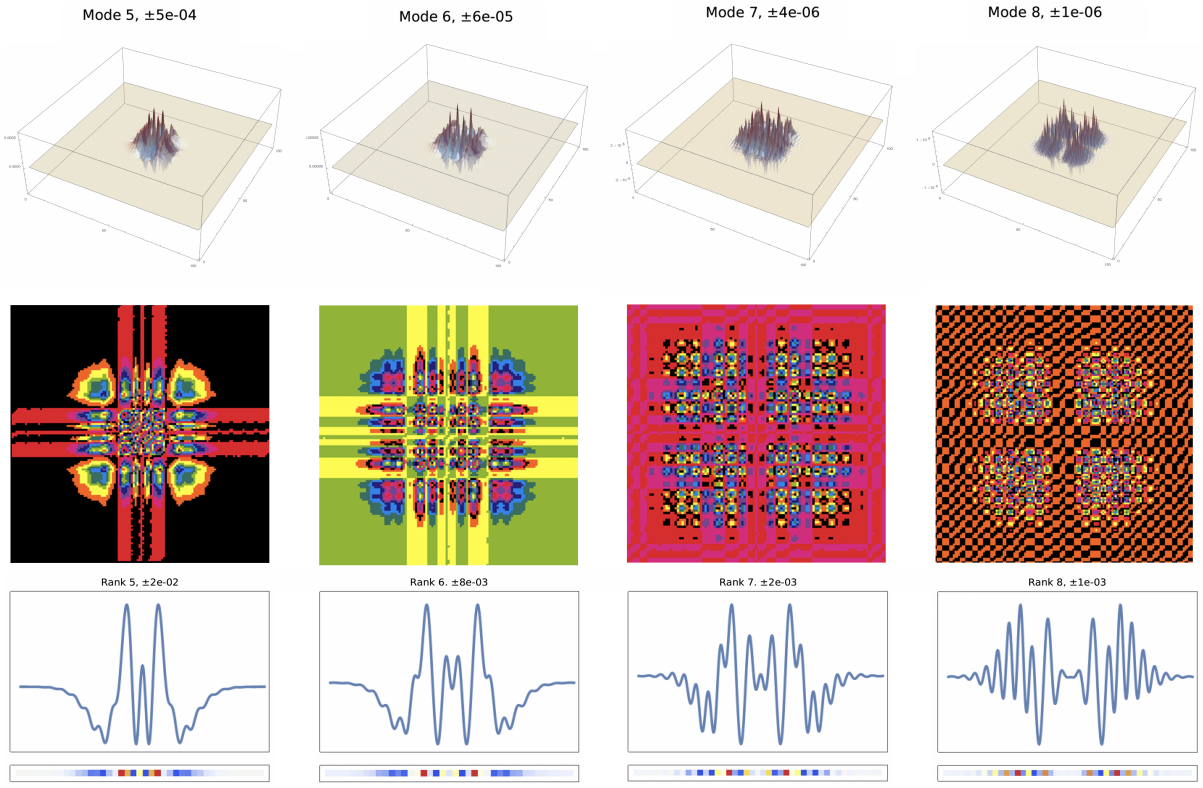

Fig. 7. 2D inverse Poisson: 50^{th}-order kernel modes and filters, ranks 5 to 8.

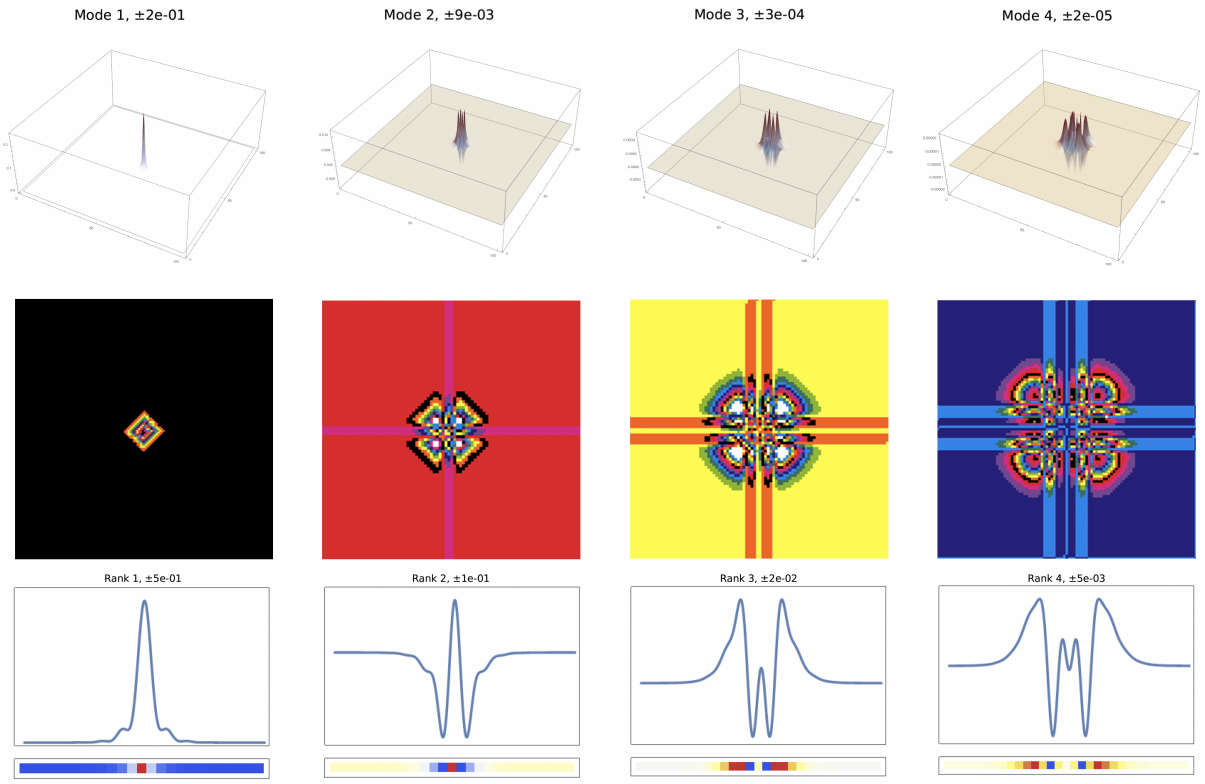

Fig. 8. 2D forward Poisson: 50^{th}-order kernel modes and filters, ranks 1 to 4, $\kappa = 1$, $\Delta t = 1$, $\Delta x = 1$.

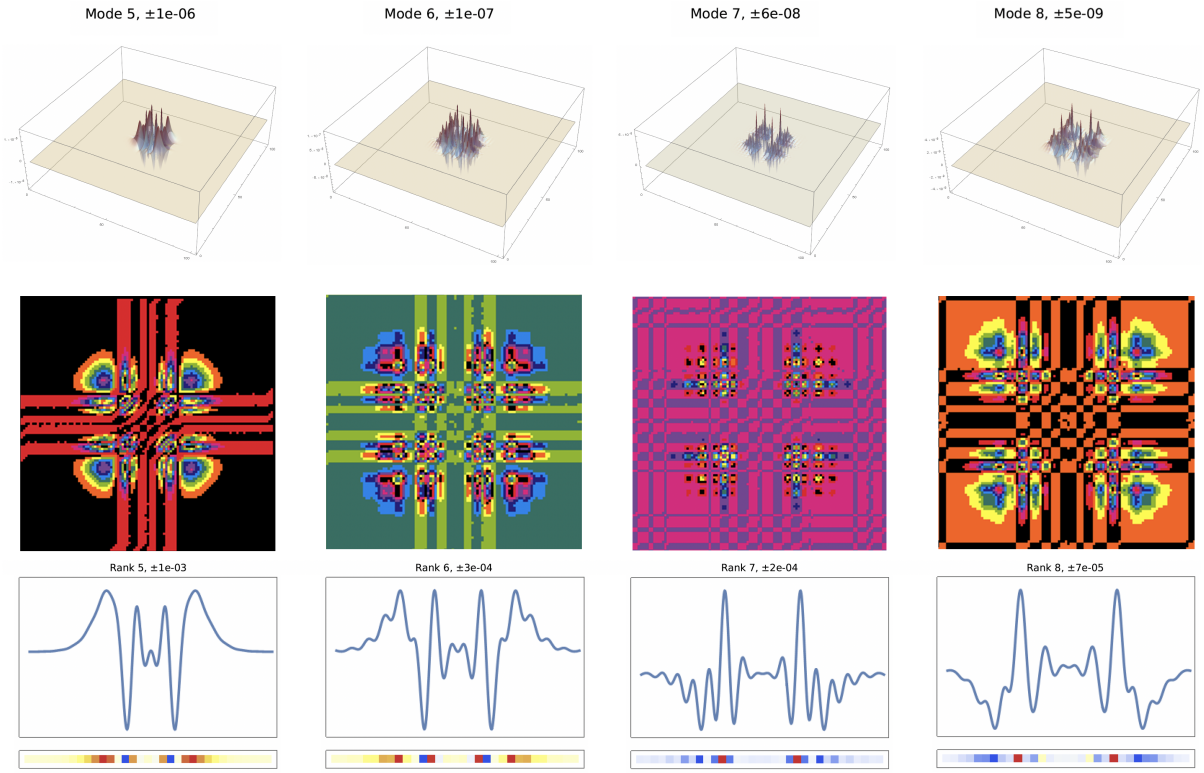

Fig. 9. 2D forward Poisson: 50^{th}-order kernel modes and filters, ranks 5 to 8, $\kappa = 1$, $\Delta t = 1$, $\Delta x = 1$.