High-resolution non-line-of-sight imaging at 60 frames per second via GPU acceleration

Supplementary Material

A. Integration in a SPAD array

In this section, we evaluate the potential results that could be achieved by running our code on current and next-generation live SPAD arrays. We focus on two key metrics: the number of photons our work can process every frame, and the image quality to expect from these photons.

To relate our benchmark to the main paper, our *real-time* NLOS imaging results use the dataset released by Nam et al. [4]. Reading raw photon counts directly from disk allows us to stress-test reconstruction throughput at the maximum frame rate, without being limited by current technology. Notably, this dataset contains approximately 10^6 photons per reconstructed frame.

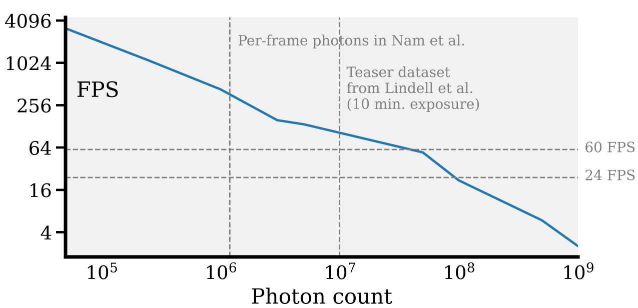


Figure 1. Photon binning time for a dynamic NLOS scene of shape $190 \times 190 \times 208$. The photon count slightly varies per frame; typically, it oscillates around $1.2 \cdot 10^6$.

In this experiment, we take the opposite perspective. Instead of fixing the photon count and reporting the resulting frame rate, we fix the frame rate and determine how many photons our system can process per second. In Figure 1, we show that our implementation can sustain frame rates above 60 FPS while processing at least 1.6 orders of magnitude more photons than in the datasets of Nam et al. [4]. Concretely, the 60-FPS threshold is surpassed with $\sim$ 47M photons, and frame rates remain above 24 FPS for photon counts as large as $\sim$ 96M.

While our work handles several million photons until surpassing the 60-FPS threshold, we also aim to demonstrate that, for a lower number of photons, the hidden scene remains recognizable using both f–k and RSD however, the latter is more resilient to noise, as shown in Figure 2. For example, the reconstructions in the third column are obtained by processing fewer points than in every frame of Nam et al. [4]'s datasets, and the fourth column processes less than half of theirs.

Notably, the dataset of Nam et al. [4] is processed at approximately 400 FPS, whereas the teaser dataset of Lindell et al. [2], which contains 178M photons captured over 180 min, is processed at slightly below 16 FPS. However, shorter exposure times yield visually similar reconstructions, as shown in Figure 2.

Finally, although our experiments read raw data from disk, we also estimate the maximum frame rate supported by the acquisition hardware. Nam et al. [4] report that photon events are streamed from the hardware queue over USB 3.0, whose effective throughput is approximately 500 $\mathrm{MB\,s^{-1}}$ [5]. Since each photon record occupies 4 bytes [4], this bandwidth allows reading up to 125M photon events per second. To put this in context, this throughput would allow acquiring over one hundred frames per second when using the same per-frame photon count as in our dynamic NLOS sequences. This frame rate would be even higher if fewer photons per wall scan are required, as illustrated in Figure 2.

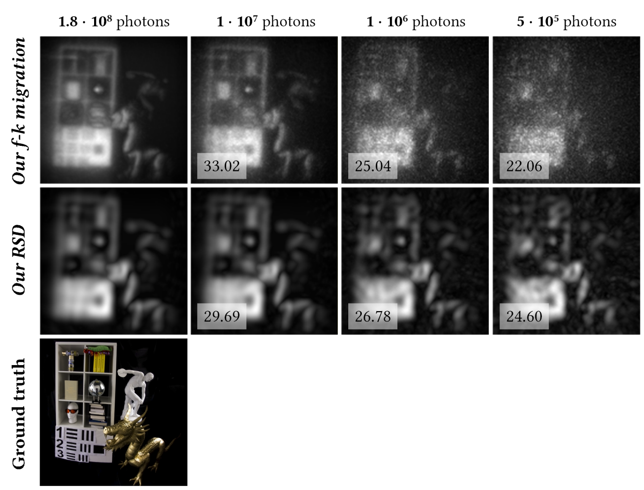


Figure 2. Teaser dataset from f–k [2]. The first and second columns correspond to datasets measured for 180 and 10 minutes, respectively. We downsample the former to 10^6 and $5 \cdot 10^5$ photons to demonstrate that, even with significantly fewer captured photons, hidden scenes remain recognizable, thus enabling high–frame-rate reconstruction and photon binning. The first and second rows show f–k and RSD reconstructions, respectively. The numbers within each reconstruction indicate the Peak Signal-to-Noise Ratio with respect to the most informed reconstruction (i.e., the first column).

B. Further implementation details

B.1. f–k migration implementation

The following algorithms show the pseudocode of the original f–k migration implementation (Algorithm 1) and our version (Algorithm 2). In particular, we aim to highlight the simplicity of our pipeline: whereas Algorithm 1 allocates up to five auxiliary buffers, our method operates exclusively on Ψ and Ψ', two complex-valued buffers of size $(2x, 2y, 2z)$. We additionally merge several steps into single operations; for instance, the Fourier shifts and the initial scaling. Finally, instead of using the squared amplitude, we extract the magnitude of the reconstructed signal, as we found that the hidden scenes become slightly more recognizable.

Algorithm 1. f**–**k **migration [2].** Reconstruction pipeline following prior work; pseudocode shown in a Python-like style. x, y, z represents the spatial (x, y) and temporal dimensions (z). $d_{\max}$ refers to the maximum distance, whereas $\text{apt}_{\text{width}}$ is the physical size in m of the scanned relay-wall.

def ***f–k*_original** $(\Psi, x, y, z, d_{\max}, \text{apt}_{\text{width}})$:
- $(X_w, Y_w, Z_w) \leftarrow \text{mgrid}[-x{:}x,\ -y{:}y,\ -z{:}z]$
- $(X_w, Y_w, Z_w) \leftarrow (X_w/x,\ Y_w/y,\ Z_w/z)$
- $\mathcal{S} \leftarrow \text{tile}(\text{linspace}(0,1,z),\ (x,y,z))$
- $\Psi \leftarrow \Psi \odot \mathcal{S}$ ▷ Initial scaling and zero-padding
- $\Psi' \leftarrow \text{zeros}(2x, 2y, 2z)$
- $\Psi'[{:}x, {:}y, {:}z] \leftarrow \Psi$
- $\widehat{\Psi} \leftarrow \mathcal{F}\{\text{fftshift}(\Psi')\}$ ▷ Forward FFT
- $s \leftarrow x \cdot d_{\max} \,/\, (4z \cdot (\text{apt}_{\text{width}}/2))$
- $Z'_w \leftarrow \sqrt{s^2(X_w^2 + Y_w^2) + Z_w^2}$ ▷ Stolt remapping
- $\Psi' \leftarrow \text{interpn}(X_w, Y_w, Z_w,\ \widehat{\Psi},\ (X_w, Y_w, Z'_w))$
- $\widehat{\Psi} \leftarrow \Psi' \odot \mathbf{1}_{Z_w > 0}$ ▷ Spectral filtering/compensation
- $\widehat{\Psi} \leftarrow \widehat{\Psi} \odot \frac{|Z_w|}{\max(Z'_w)}$
- $\Psi \leftarrow \text{ifftshift}(\mathcal{F}^{-1}\{\widehat{\Psi}\})$ ▷ Inverse FFT
- $f(x_v) \leftarrow \text{max\_z}(|\Psi^2|)$
- **return** $f(x_v)$ ▷ Final reconstruction

Algorithm 2. Our CUDA-based f**–**k **migration** pipeline.

def ***f–k*_ours** $(\Psi, x, y, z, d_{\max}, \text{apt}_{\text{width}})$:
- $\widehat{\Psi}(\cdot,\cdot,\cdot) \leftarrow 0$
- $\widehat{\Psi} \leftarrow \text{scale\_fftshift}(\Psi, \text{distance}, \text{apt}_{\text{width}})$
- $\widehat{\Psi} \leftarrow \mathcal{F}\{\widehat{\Psi}\}$
- $s \leftarrow x \cdot d_{\max} / (4z \cdot \text{apt}_{\text{width}}/2)$
- $\Psi' \leftarrow \text{stolt}(\widehat{\Psi}, s)$
- $\widehat{\Psi} \leftarrow \mathcal{F}^{-1}\{\Psi'\}$
- $f(x_v) \leftarrow \text{ifftshift\_max\_magnitude}(\widehat{\Psi})$
- **return** $f(x_v)$

B.2. RSD implementation

Similarly to the previous section, Algorithms 3 and 4 present both the original RSD implementation and our optimized version. In the original implementation, the initial Fourier transforms are not parallelized: although they could be batched, they are instead executed sequentially. The convolved volume $\mathcal{C}$ is also processed sequentially, weighting each slice $\mathcal{C}(\cdot,\cdot,c)$ by w_c. Moreover, the Fourier-transformed matrices are not padded, which reduces memory consumption at the expense of introducing artifacts.

In contrast, Algorithm 4 highlights the simplicity of our approach: since the RSD kernels are precomputed, the reconstruction reduces to performing batched frequency-domain transforms, convolution, and an inverse Fourier transform.

Algorithm 3. **RSD as implemented by Nam et al. [4].** Reconstruction pipeline following prior work. x, y, z represents the spatial (x, y) and temporal dimensions, $d_{\min}, d_{\max}, \delta d$ are the minimum, maximum and delta distance, and w weights each frequency differently.

def ***rsd*_original** $(\Psi, \mathcal{K}, x, y, z, d_{\min}, d_{\max}, \Delta d, w)$:
- **for** $c \leftarrow 1$ **to** z : ▷ 2D forward FFTs
  - $\widehat{\Psi}_c \leftarrow \mathcal{F}_{2D}\{\Psi(\cdot,\cdot,c)\}$
- $i \leftarrow 0$
- $\Psi(\cdot,\cdot,\cdot) \leftarrow 0$
- **for** $d \leftarrow d_{\min}$ **to** $d_{\max}$, **step** Δd :
  - $\mathcal{C} \leftarrow \widehat{\Psi}_c \odot \mathcal{K}(\cdot,\cdot,\cdot,i)$ ▷ Convolution
  - **for** $c \leftarrow 1$ **to** z :
    - $\Psi(\cdot,\cdot,i) \leftarrow \Psi(\cdot,\cdot,i) + w_c\mathcal{C}(\cdot,\cdot,c)$
  - $\Psi(\cdot,\cdot,i) \leftarrow \mathcal{F}_{2D}^{-1}\{\Psi(\cdot,\cdot,i)\}$ ▷ Inverse FFT
  - $\widehat{\Psi}(\cdot,\cdot,i) \leftarrow |\Psi(\cdot,\cdot,i)|$ ▷ Magnitude at depth i
  - $i \leftarrow i + 1$
- $f(x_v) \leftarrow \text{max\_z}(\widehat{\Psi})$
- **return** $f(x_v)$

Algorithm 4. Our CUDA-based **RSD** pipeline.

def ***rsd*_ours** $(\Psi, \mathcal{K}, x, y, z, w)$:
- $\widehat{\Psi} \leftarrow \mathcal{F}\{\Psi\}$ ▷ Batched 2D forward FFTs
- $\mathcal{C} \leftarrow \text{convolve}(\widehat{\Psi}, \mathcal{K}, x, y, z, w)$
- $\widehat{\Psi} \leftarrow \mathcal{F}^{-1}\{\mathcal{C}\}$ ▷ Inverse FFT
- $\Psi' \leftarrow |\widehat{\Psi}|$
- $f(x_v) \leftarrow \text{max\_z}(\Psi')$
- **return** $f(x_v)$

B.2.1. GPU-driven RSD simplification

This subsection further discusses our enhancements for memory usage and runtime for the RSD-based method. We use as a baseline the implementation by Lindell et al. [2], which is slightly different from that of Nam et al. [4] and follows the algorithm of Liu et al. [3]. We use this baseline as reference for all our offline reconstruction experiments.

The first drawback we addressed was the convolution of the Point Spread Function (PSF), denoted as $\mathcal{S}$ in the following equations, with the signal's phasor representation. The original implementation allocates two complex-valued buffers for the phasor representation, $P_{\cos}$ and $P_{\sin}$, which is suboptimal and allocates twice the required memory. In this formulation, $P_{\cos}$ and $P_{\sin}$ store the cosine and sine components of the phasor and are treated as if they were independent signals: each one is Fourier-transformed, multiplied by the PSF in the frequency domain, and then inversely transformed, and the final complex result is obtained by combining both outputs.

However, these two components are in fact the real and imaginary parts of a single complex phasor. Since the Fourier transform, the pointwise multiplication by $\mathcal{S}$, and the inverse transform are all linear operations, applying them separately to $P_{\cos}$ and $P_{\sin}$ is equivalent to applying them once to their complex combination. Therefore, we found that the following two expressions are equivalent:

$$P' = \mathcal{F}^{-1}\{\tilde{P}_{\cos} \cdot \mathcal{S}\} + i\,\mathcal{F}^{-1}\{\tilde{P}_{\sin} \cdot \mathcal{S}\} \quad (1)$$

$$= \mathcal{F}^{-1}\{(\tilde{P}_{\cos} + i\,\tilde{P}_{\sin}) \cdot \mathcal{S}\} \quad (2)$$

with $\tilde{P}_{\cos} = \mathcal{F}\{P_{\cos}\}$ and $\tilde{P}_{\sin} = \mathcal{F}\{P_{\sin}\}$.

Additionally, another time-consuming step is the construction of the transform operator that maps phasor data from a Cartesian layout to a ring-based layout. In the original implementation, this operator is built as a sparse matrix of size M^3 with $M \leftarrow \max(x, y)$, where each sample index i contributes a value of 1 at position $(i, \lceil\sqrt{i}\rceil)$. The matrix is then iteratively collapsed over $\log_2(\max(x, y))$ iterations; in each iteration, pairs of non-contiguous rows are averaged. After $\log_2 M$ iterations, the matrix is reduced to size $M \times M$, and the accumulated weights have been scaled by a factor of $(\frac{1}{2})^{\log_2 M} = \frac{1}{M}$.

This hierarchical construction is equivalent to directly creating an $M \times M$ matrix and mapping each sample index i to its corresponding ring index $\lceil\sqrt{i+1}\rceil$, with a final weight of $\frac{1}{M\sqrt{i+1}}$. In other words, the hierarchical averaging performed in the original implementation can be collapsed into a single kernel. In fact, constructing the transform operators as in the original formulation required building large sparse matrices (e.g., using the Eigen library) and repeatedly collapsing them on the CPU. After simplifying the algorithm, the overall reconstruction time was reduced to approximately one third.

Finally, note that it is not necessary to construct the inverse operator explicitly; the kernel can access it by simply using transposed indices of the forward transform operator. The forward and backward matrix multiplications were solved with the cuBLAS (Basic Linear Algebra Subprograms) module on top of CUDA.

C. Additional results and benchmarks

C.1. Dynamic reconstruction of a detailed scene

We report additional results for real-time NLOS imaging in another dynamic scene from Nam et al. [4]. Figure 3 shows the hidden scene and our reconstructions, in which the letters 'N', 'L', 'O', and 'S' appear and disappear over time. The scene contains 400 frames total. In the plot, we compare the frames per second for our f–k migration (with and without padding) and our RSD implementations, and the RSD implementation of Nam et al. [4].

In the plot, the best performance corresponds to f–k migration without padding, whereas RSD performs slightly better than f–k migration with padding. In this case, the reconstructions for our RSD look the sharpest, clearly showing the hidden letters.

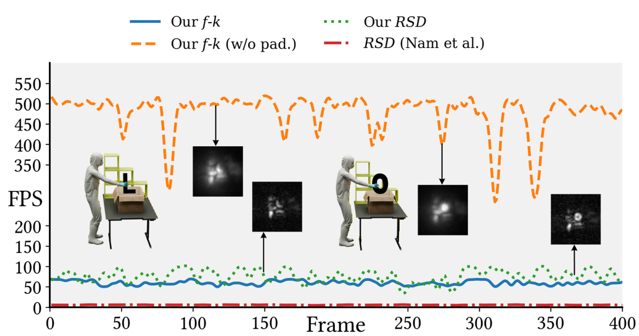


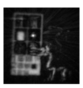
Figure 3. Performance and qualitative comparison between our f–k migration (padded and unpadded), our RSD, and the method of Nam et al. The evaluated dynamic scene spans $190 \times 190 \times 208$ voxels, with RSD applied over 50 depths. For each method, two reconstructed frames are shown next to their ground truth (displayed on the left of each pair) to illustrate differences in fidelity.

C.2. Offline reconstruction

We extend our reported offline reconstruction timings with a comprehensive comparison in Table 1, covering a range of spatial and temporal resolutions. Starting from the original dataset sizes, we downsample the spatial and temporal dimensions. Recall that the f–k migration datasets are 512^3, while the largest Zaragoza datasets are $256 \times 256 \times 4096$. However, our experiments limit the temporal dimension to 2048 bins because 4096 time bins put significant memory pressure.

Besides reconstruction time, we also evaluate throughput in millions of voxels per second. The last two columns report memory usage on both the CPU and GPU. We group them under the label [V]RAM as the baseline f–k and RSD implementations allocate buffers in CPU memory, whereas our optimised variants operate primarily on GPU memory.

Table 1. Performance and memory comparison between our accelerated f–k and RSD implementations and their original counterparts, implemented in Matlab. In addition to reconstruction time, we report throughput in millions of voxels per second, computed as the total number of voxels divided by the reconstruction time. The memory column (Max. [V]RAM) reflects peak CPU or GPU usage: the original methods allocate buffers on the CPU, whereas our optimized versions operate primarily on the GPU. For each dataset and dimensionality, the best-performing implementation is highlighted in bold.

Dataset	Dimensions	Algorithm	Time (s) Ours	Time (s) Original	Throughput (Mvox/s) Ours	Throughput (Mvox/s) Original	Max. [V]RAM usage (GB) Ours	Max. [V]RAM usage (GB) Original
Confocal – f–k migration dataset [2]								
bike 	512×512×512	f–k migration	**4.060 ± 0.10**	146.624 ± 8.467	**33.06**	0.92	**16.500**	52.430
		Phasor fields	8.998 ± 0.37	30.872 ± 0.670	14.92	4.35	21.509	33.579
	256×256×512	f–k migration	**0.157 ± 0.01**	10.695 ± 0.658	**213.42**	3.14	**4.250**	13.107
		Phasor fields	0.249 ± 0.00	6.894 ± 0.195	134.51	4.87	5.501	8.393
	256×256×256	f–k migration	**0.091 ± 0.00**	5.243 ± 0.140	**184.05**	3.20	**2.125**	6.554
		Phasor fields	0.132 ± 0.01	3.428 ± 0.152	127.19	4.89	2.375	4.194
	128×128×512	f–k migration	**0.048 ± 0.00**	2.552 ± 0.109	**174.01**	3.29	**1.125**	3.277
		Phasor fields	0.078 ± 0.01	1.607 ± 0.024	107.40	5.22	1.251	2.101
	128×128×256	f–k migration	**0.032 ± 0.00**	1.253 ± 0.031	**132.21**	3.35	**0.062**	1.638
		Phasor fields	0.048 ± 0.00	0.801 ± 0.027	86.69	5.24	0.625	1.049
	128×128×128	f–k migration	**0.031 ± 0.00**	0.652 ± 0.022	**66.61**	3.22	**0.281**	0.819
		Phasor fields	0.032 ± 0.00	0.408 ± 0.024	65.95	5.14	0.297	0.524
teaser 	512×512×512	f–k migration	**4.062 ± 0.09**	154.730 ± 14.765	**33.04**	0.87	**16.503**	52.433
		Phasor fields	9.064 ± 0.07	31.458 ± 2.647	14.81	4.27	21.504	33.579
	256×256×512	f–k migration	**0.156 ± 0.01**	11.462 ± 0.646	**214.90**	2.93	**4.250**	13.109
		Phasor fields	0.250 ± 0.00	7.155 ± 0.273	134.42	4.69	4.751	8.393
	256×256×256	f–k migration	**0.091 ± 0.00**	5.597 ± 0.197	**184.65**	3.00	**2.125**	6.554
		Phasor fields	0.126 ± 0.00	3.568 ± 0.112	133.66	4.70	2.751	4.194
	128×128×512	f–k migration	**0.047 ± 0.00**	2.713 ± 0.211	**178.12**	3.09	**1.125**	3.277
		Phasor fields	0.079 ± 0.01	1.621 ± 0.056	106.74	5.17	1.251	2.101
	128×128×256	f–k migration	**0.029 ± 0.00**	1.435 ± 0.177	**142.74**	2.92	**0.312**	1.638
		Phasor fields	0.044 ± 0.00	0.822 ± 0.061	94.72	5.10	0.594	1.049
	128×128×128	f–k migration	0.029 ± 0.00	0.672 ± 0.040	72.42	3.12	0.281	0.819
		Phasor fields	**0.028 ± 0.00**	0.409 ± 0.013	**74.29**	5.13	**0.156**	0.524
statue 	512×512×512	f–k migration	**3.834 ± 0.10**	152.567 ± 8.222	**35.01**	0.88	**16.500**	52.430
		Phasor fields	9.067 ± 0.11	32.195 ± 1.857	14.80	4.17	21.501	33.579
	256×256×512	f–k migration	**0.161 ± 0.01**	11.118 ± 0.874	**207.97**	3.02	**4.250**	13.107
		Phasor fields	0.247 ± 0.00	7.063 ± 0.170	135.70	4.75	4.751	8.393
	256×256×256	f–k migration	**0.093 ± 0.00**	5.340 ± 0.209	**181.28**	3.14	**2.125**	6.554
		Phasor fields	0.138 ± 0.01	3.476 ± 0.099	121.74	4.83	2.375	4.195
	128×128×512	f–k migration	**0.045 ± 0.00**	2.747 ± 0.159	**185.58**	3.05	**1.125**	3.277
		Phasor fields	0.074 ± 0.00	1.668 ± 0.040	113.11	5.03	1.251	2.101
	128×128×256	f–k migration	**0.029 ± 0.00**	1.316 ± 0.059	**144.30**	3.19	**0.062**	1.638
		Phasor fields	0.045 ± 0.00	0.830 ± 0.037	94.06	5.05	0.594	1.049
	128×128×128	f–k migration	0.031 ± 0.00	0.686 ± 0.140	67.73	3.06	**0.156**	0.819
		Phasor fields	**0.029 ± 0.00**	0.392 ± 0.002	**72.54**	5.35	**0.156**	0.524

References

[1] Miguel Galindo, Julio Marco, Matthew O'Toole, Gordon Wetzstein, Diego Gutierrez, and Adrian Jarabo. A dataset for benchmarking time-resolved non-line-of-sight imaging, 2019. Publication Title: IEEE International Conference on Computational Photography (ICCP). 5, 6

[2] David B. Lindell, Gordon Wetzstein, and Matthew O'Toole. Wave-based non-line-of-sight imaging using fast f-k migration. *ACM Trans. Graph.*, 38(4):116:1–116:13, 2019. 1, 2, 4

[3] Xiaochun Liu, Ibón Guillén, Marco La Manna, Ji Hyun Nam, Syed Azer Reza, Toan Huu Le, Adrian Jarabo, Diego Gutierrez, and Andreas Velten. Non-line-of-sight imaging using phasor-field virtual wave optics. *Nature*, 572(7771):620–623, 2019. Publisher: Nature Publishing Group. 2

[4] Ji Hyun Nam, Eric Brandt, Sebastian Bauer, Xiaochun Liu, Marco Renna, Alberto Tosi, Eftychios Sifakis, and Andreas Velten. Low-latency time-of-flight non-line-of-sight imaging at 5 frames per second. *Nature Communications*, 12(1):6526, 2021. Publisher: Nature Publishing Group. 1, 2, 3

[5] Richard West, Ahmad Golchin, and Anton Njavro. Real-Time USB Networking and Device I/O. *ACM Trans. Embed. Comput. Syst.*, 22(4):67:1–67:38, 2023. 1

Table 2. Continuation of Table 1.

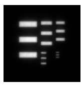
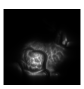
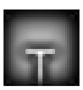
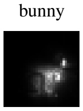
Dataset	Dimensions	Algorithm	Time (s)		Throughput (Mvox/s)		Max. [V]RAM usage (GB)	
			Ours	Original	Ours	Original	Ours	Original
Confocal – Zaragoza dataset [1]								
usaf 	256×256×2048	f–k migration	**1.849 ± 0.07**	139.592 ± 22.222	**72.57**	0.96	**15.823**	52.429
		Phasor fields	6.686 ± 0.07	31.655 ± 1.257	20.07	4.24	17.755	33.621
	256×256×1024	f–k migration	**0.293 ± 0.01**	20.740 ± 1.525	**229.23**	3.24	**7.911**	26.214
		Phasor fields	0.489 ± 0.01	14.870 ± 0.242	137.19	4.51	8.874	16.794
	256×256×512	f–k migration	**0.156 ± 0.01**	10.714 ± 0.292	**214.98**	3.13	**3.955**	13.107
		Phasor fields	0.249 ± 0.00	6.665 ± 0.284	134.70	5.03	4.436	8.393
	128×128×2048	f–k migration	**0.167 ± 0.01**	10.361 ± 0.479	**200.87**	3.24	**4.075**	13.107
		Phasor fields	0.261 ± 0.01	8.209 ± 0.046	128.58	4.09	4.569	8.454
	128×128×1024	f–k migration	**0.086 ± 0.00**	4.976 ± 0.320	**195.57**	3.37	**2.037**	6.554
		Phasor fields	0.133 ± 0.00	3.424 ± 0.165	126.09	4.90	2.641	4.211
	128×128×512	f–k migration	**0.049 ± 0.01**	2.468 ± 0.182	**172.79**	3.40	**1.019**	3.277
		Phasor fields	0.084 ± 0.01	1.611 ± 0.039	99.95	5.21	1.140	2.101
	64×64×2048	f–k migration	**0.051 ± 0.00**	2.590 ± 0.126	**163.78**	3.24	**1.078**	3.277
		Phasor fields	0.091 ± 0.01	3.285 ± 0.091	92.41	2.55	1.213	2.163
	64×64×1024	f–k migration	**0.032 ± 0.00**	1.509 ± 0.234	**130.94**	2.78	**0.299**	1.638
		Phasor fields	0.049 ± 0.00	0.987 ± 0.025	86.48	4.25	0.603	1.065
	64×64×512	f–k migration	**0.029 ± 0.00**	0.605 ± 0.047	**71.81**	3.47	**0.270**	0.819
		Phasor fields	0.032 ± 0.00	0.449 ± 0.039	65.96	4.67	0.285	0.528
bunny 	256×256×2048	f–k migration	**1.793 ± 0.08**	138.419 ± 8.193	**74.87**	0.97	**15.823**	52.429
		Phasor fields	6.641 ± 0.03	33.418 ± 0.673	20.21	4.02	20.632	33.621
	256×256×1024	f–k migration	**0.298 ± 0.01**	22.058 ± 0.971	**224.99**	3.04	**7.911**	26.214
		Phasor fields	0.484 ± 0.01	14.709 ± 0.334	138.74	4.56	8.874	16.793
	256×256×512	f–k migration	**0.157 ± 0.01**	10.384 ± 0.179	**213.76**	3.23	**3.955**	13.107
		Phasor fields	0.249 ± 0.00	7.006 ± 0.258	134.83	4.79	4.436	8.393
	128×128×2048	f–k migration	**0.160 ± 0.01**	10.740 ± 0.568	**210.17**	3.12	**4.075**	13.107
		Phasor fields	0.252 ± 0.01	8.290 ± 0.140	133.00	4.05	4.569	8.454
	128×128×1024	f–k migration	**0.092 ± 0.00**	5.020 ± 0.155	**181.97**	3.34	**2.037**	6.554
		Phasor fields	0.132 ± 0.00	3.410 ± 0.068	126.79	4.92	2.280	4.211
	128×128×512	f–k migration	**0.048 ± 0.00**	2.552 ± 0.034	**175.55**	3.29	**1.019**	3.277
		Phasor fields	0.071 ± 0.00	1.556 ± 0.060	118.41	5.39	1.319	2.101
	64×64×2048	f–k migration	**0.048 ± 0.00**	2.633 ± 0.229	**176.32**	3.19	**1.078**	3.277
		Phasor fields	0.082 ± 0.01	3.293 ± 0.096	102.07	2.55	1.213	2.163
	64×64×1024	f–k migration	**0.032 ± 0.00**	1.232 ± 0.081	**132.92**	3.40	**0.299**	1.638
		Phasor fields	0.050 ± 0.00	1.026 ± 0.042	83.77	4.09	0.603	1.065
	64×64×512	f–k migration	0.032 ± 0.00	0.679 ± 0.080	64.82	3.09	0.270	0.819
		Phasor fields	**0.029 ± 0.00**	0.420 ± 0.013	**72.51**	4.99	**0.149**	0.528
z 	256×256×2048	f–k migration	**2.821 ± 0.08**	142.930 ± 6.273	**47.59**	0.94	**15.823**	52.429
		Phasor fields	11.408 ± 0.10	32.743 ± 1.422	11.77	4.10	17.755	33.636
	256×256×1024	f–k migration	**0.297 ± 0.01**	23.453 ± 0.799	**225.96**	2.86	**7.911**	26.214
		Phasor fields	0.486 ± 0.01	14.576 ± 0.379	138.15	4.60	8.874	16.794
	256×256×512	f–k migration	**0.157 ± 0.01**	10.379 ± 0.201	**213.74**	3.23	**3.955**	13.107
		Phasor fields	0.252 ± 0.00	6.895 ± 0.362	133.15	4.87	4.436	8.400
	128×128×2048	f–k migration	**0.155 ± 0.00**	10.414 ± 0.356	**216.14**	3.22	**4.075**	13.107
		Phasor fields	0.257 ± 0.01	8.356 ± 0.199	130.46	4.02	4.569	8.454
	128×128×1024	f–k migration	**0.089 ± 0.01**	5.169 ± 0.109	**187.59**	3.25	**2.037**	6.554
		Phasor fields	0.143 ± 0.01	3.464 ± 0.044	117.40	4.84	2.280	4.211
	128×128×512	f–k migration	**0.052 ± 0.01**	2.933 ± 0.214	**160.91**	2.86	**1.019**	3.277
		Phasor fields	0.087 ± 0.01	1.567 ± 0.056	96.56	5.35	1.140	2.101
	64×64×2048	f–k migration	**0.048 ± 0.00**	2.525 ± 0.158	**174.56**	3.32	**1.078**	3.277
		Phasor fields	0.087 ± 0.01	3.268 ± 0.043	96.88	2.57	1.213	2.163
	64×64×1024	f–k migration	**0.035 ± 0.01**	1.270 ± 0.082	**119.27**	3.30	**0.539**	1.638
		Phasor fields	0.046 ± 0.00	1.028 ± 0.013	91.90	4.08	0.572	1.065
	64×64×512	f–k migration	0.035 ± 0.01	0.642 ± 0.075	60.53	3.27	**0.270**	0.819
		Phasor fields	**0.032 ± 0.00**	0.416 ± 0.008	**66.29**	5.04	0.285	0.528

Table 3. Continuation of Table 2.

			Time (s)		Throughput (Mvox/s)		Max. [V]RAM usage (GB)	
Dataset	Dimensions	Algorithm	Ours	Original	Ours	Original	Ours	Original
Exhaustive – Zaragoza dataset [1]								
concavities 	16×16×16×16×2048	f–k migration	**0.035 ± 0.01**	0.588 ± 0.036	**55.77**	3.35	**0.231**	0.770
		Phasor fields	0.038 ± 0.01	2.107 ± 0.011	52.03	0.93	0.274	0.558
	16×16×16×16×1024	f–k migration	**0.029 ± 0.00**	0.305 ± 0.019	**33.61**	3.23	**0.115**	0.384
		Phasor fields	0.030 ± 0.00	0.479 ± 0.005	32.66	2.05	0.119	0.263
	16×16×16×16×512	f–k migration	**0.030 ± 0.00**	0.157 ± 0.019	**16.16**	3.13	**0.058**	0.192
		Phasor fields	0.031 ± 0.00	0.153 ± 0.002	15.96	3.22	0.062	0.127
t (in a box) 	16×16×16×16×2048	f–k migration	**0.071 ± 0.08**	0.613 ± 0.056	**27.91**	3.21	**0.231**	0.770
		Phasor fields	0.077 ± 0.09	2.119 ± 0.020	25.41	0.93	0.273	0.559
	16×16×16×16×1024	f–k migration	**0.030 ± 0.00**	0.286 ± 0.016	**33.35**	3.44	**0.115**	0.384
		Phasor fields	0.031 ± 0.00	0.469 ± 0.004	32.24	2.10	0.126	0.262
	16×16×16×16×512	f–k migration	0.032 ± 0.00	0.149 ± 0.006	15.47	3.30	**0.058**	0.192
		Phasor fields	**0.031 ± 0.00**	0.152 ± 0.004	**16.04**	3.24	0.062	0.127
bunny 	16×16×16×16×2048	f–k migration	**0.071 ± 0.08**	0.685 ± 0.104	**27.84**	2.87	**0.231**	0.769
		Phasor fields	0.084 ± 0.07	2.187 ± 0.028	23.38	0.90	0.274	0.560
	16×16×16×16×1024	f–k migration	0.031 ± 0.00	0.348 ± 0.034	32.12	2.83	**0.115**	0.384
		Phasor fields	**0.030 ± 0.00**	0.490 ± 0.015	**33.05**	2.01	0.126	0.262
	16×16×16×16×512	f–k migration	**0.029 ± 0.00**	0.159 ± 0.019	**16.93**	3.09	**0.058**	0.192
		Phasor fields	0.029 ± 0.00	0.158 ± 0.004	16.78	3.11	0.062	0.127