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

Anonymous CVPR submission

Paper ID

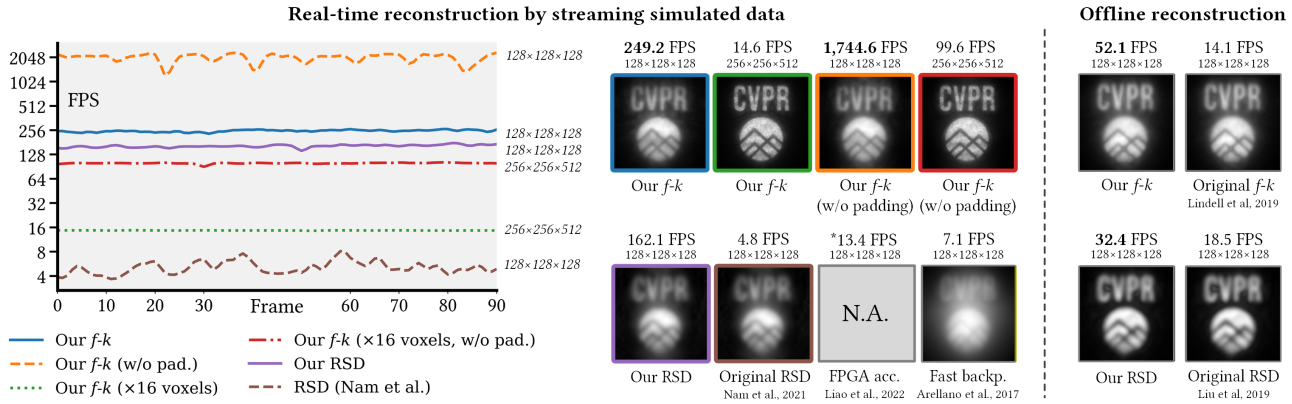

Figure 1. Our implementation of the Rayleigh-Sommerfeld Diffraction (RSD)-based and f–k migration methods outperforms all previous versions by two orders of magnitude. We compare the performance between real-time (left) and offline reconstructions (right). The left plot shows the frame rates (FPS) achieved when reconstructing simulated data, combining results from datasets with size $(128, 128, 128)$ and $(256, 256, 512)$ to emphasize the superior performance of our f–k migration implementation. Thanks to the high throughput of our algorithm, we show that we remain the fastest even processing 16x the data (which naturally gives a clearer reconstruction). Not Available (N.A.) denotes methods whose implementations are not released, and an asterisk (*) denotes frame rates extrapolated from reported results. The right plot compares our GPU-accelerated implementations with the original f–k migration and RSD methods.

Abstract

Non-line-of-sight (NLOS) imaging methods can reconstruct objects hidden around the corner, with potential applications in autonomous driving, medical imaging, remote sensing and others. Yet, real-world deployment remains a major challenge due to the massive amount of data that must be first captured and later processed. The emergence of Single Photon Avalanche Diode (SPAD) arrays, which enable parallel data acquisition, helps address the capture bottleneck. We observe a paradigm shift where the main limitation is moving from data capture to real-time processing at scale. To address this, we develop hardware-accelerated optimizations for two state-of-the-art NLOS imaging algorithms: f–k migration and Rayleigh-Sommerfeld Diffraction (RSD)-based methods. Our approach achieves speedups of three *orders of magnitude under equal conditions, outperforming all previous hardware-accelerated versions including specialized FPGA designs. Moreover, our implementation reduces memory usage by about 60% on average, enabling the reconstruction of even larger scenes. With all these advances, we make use of our real-time NLOS imaging pipeline and introduce a novel algorithm that leverages temporal consistency of moving objects to further improve image quality, paving the way for next-generation NLOS video processing software.*

1. Introduction

Non-line-of-sight (NLOS) imaging methods can reconstruct hidden objects by analysing indirect light scattered on a visible relay wall. Among these, time-of-flight (ToF) NLOS imaging leverages ultra-fast sensors to measure the time of flight of individual photons through the hidden scene by illuminating and capturing individual points on the relay wall. ToF NLOS imaging methods have demonstrated unprecedented imaging capabilities and are promising candidates for practical applications [6, 8] such as motion tracking and geometry reconstruction in areas autonomous navigation to geological and planetary exploration and disaster response.

Still, applications of NLOS imaging are set back due to the huge amounts of data that need to be captured and processed. For reference, current methods can spend minutes to hours capturing and reconstructing a single NLOS image [12]. The main reason is the low Signal-to-Noise Ratio (SNR) inherent to NLOS setups, due to the indirect three-bounce path that each photon follows: from the laser to the relay wall, from the relay wall to the hidden scene and back.

Over the recent years, Single-Photon Avalanche Diode (SPAD) arrays have emerged as a promising solution to parallelize photon acquisition. This greatly alleviates the low SNR problem: for example, work by Nam et al. [17] uses two 16 by 1 SPAD arrays to achieve dynamic NLOS imaging at five frames per second. As the size of SPAD arrays keeps growing [5, 15], we observe a paradigm shift that is currently happening in NLOS imaging where the bottleneck moves from photon acquisition to processing captured information in real-time. Hence several recent works have attempted to optimize the NLOS processing pipeline. Notably, Arellano et al. [2] proposed the use of GPU rasterization for a 1.000x speedup compared to conventional backprojection, bringing the total reconstruction time to a few seconds per frame. Later, Nam et al. [17] achieved frames per second in a GPU using a Rayleigh-Sommerfeld Diffraction (RSD)-based algorithm, while Liao et al. [10] achieved twenty-five reconstructed frames per second by implementing the RSD algorithm in a FPGA.

In this work, we push the limits of NLOS imaging speed and scalability, and accuracy, with two main contributions. First, we present a GPU-accelerated implementation of confocal and non-confocal reconstruction algorithms, achieving rates from hundreds of frames per second on smaller datasets to tens of frames on larger ones. Our experiments demonstrate speedups of up to three orders of magnitude over previous methods, including FPGA-based approaches, all while reducing VRAM usage by 60% on average. Concretely, we optimize an RSD-based method [17] and the f–k migration algorithm [11]. Second, we introduce a novel strategy to combine successive frames of moving objects in an NLOS video in our real-time imaging pipeline. We leverage temporal consistency of NLOS frames by averaging phase-aligned information across time. Since RSD and f–k migration are wave-based methods operating on complex-valued data, this phase-based filtering naturally suppresses random noise while reinforcing static or slowly varying structures.

We showcase our method under two modalities. For *real-time* NLOS imaging, our algorithm simultaneously handles reconstruction for one frame and data processing for the next frame, as in the work by Nam et al. [17]. Conversely, *offline* NLOS imaging focuses on reconstructing already-processed data as fast as possible, as in works by Arellano et al. [2] or Liao et al. [10]. We hope that our method can support future NLOS imaging applications in real-world environments, for that reason we will make our code public upon acceptance.

2. Related Work

Time-of-flight NLOS imaging. We focus on active NLOS imaging, where a laser source illuminates the scene. The first NLOS imaging method relied on a streak camera to capture indirect light [22, 23]. The introduction of SPADs [3], far more affordable than streak cameras, broadened access to NLOS imaging and incentivized the development of multiple innovative reconstruction methods. For our purposes, existing approaches can be broadly grouped according to the capture configuration: **confocal** [11, 19], where the capture device is pointed to the same point in the relay wall as the laser, and **non-confocal capture** [12, 20, 21], which relaxes this restriction. Confocal capture is typically slower because both the laser and detector must steer towards each measurement point. However, reconstruction algorithms based on confocal data can assume certain properties of about light paths which enable useful optimizations that make them more efficient than their non-confocal counterparts [18]. In this work, we implement a complete NLOS processing pipeline that reads raw photon data (from SPAD sensors or simulation), and reconstructs the 3D scene using our optimized GPU implementations of existing wave-based methods. Specifically, we accelerate both the RSD formulation [12] and the f–k migration approach [11]; we support both confocal and non-confocal acquisition.

NLOS imaging optimizations. Many works have proposed strategies to accelerate NLOS imaging; for example, using convolutional versions of plane-to-plane propagation operators [9, 13] or their approximations [1]. Other works focus on reducing memory consumption during reconstruction [14]. A parallel line of research explores deep-learning-based NLOS imaging, where neural networks approximate the mapping from transient data to geometry [4, 16], including dynamic scenes [24]. More relevant to our work are methods that have exploited vectorized hardware for acceleration. Arellano et al. [2] leveraged GPU rasterization to speed up backprojection by three orders of magnitude. Subsequent works, such as NLOS at 5 FPS [17] and FPGA-accelerated NLOS imaging [10] have demonstrated that hardware specialization can substantially reduce runtime, also introducing simple strategies to combine consecutive reconstructed frames. However, most of these systems achieve high performance at the cost of low spatial resolution, which limits their applicability to high-resolution NLOS scenes. In contrast, our work provides optimized GPU implementations of both RSD-based and f–k migration algorithms that are at least two orders of magnitude faster than all prior hardware-accelerated methods, while reconstructing larger NLOS scenes with lower memory consumption, and with a better strategy to combine frames.

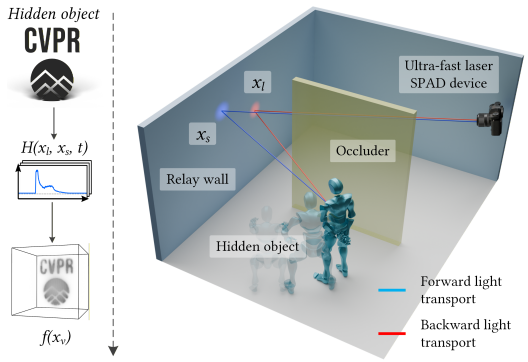

Figure 2. Illustration of the capture setup. An ultrafast laser emits a pulse onto the relay wall, from which part of the light reflects toward the hidden object. The scattered light from the object then bounces back onto the wall and is captured by the SPAD array. On the left, we show from top to bottom the hidden object, its processed histogram captured by the SPAD array, and the reconstructed scene.

3. Background: non-line-of-sight imaging

This section reviews the key principles of active time-of-flight NLOS imaging, along with the theory of f–k migration [11] and the RSD-based method [12], which are the imaging methods we have adapted in our work.

Figure 2 illustrates a NLOS imaging setup. A laser emits ultra-short pulses toward each point $\mathbf{x}_l \in \mathcal{L}$ on a visible *relay* wall. Light scatters at $\mathbf{x}_l$ toward the hidden scene and then returns to the relay wall. An ultra-fast camera captures the indirect illumination response at points $\mathbf{x}_s \in \mathcal{S}$ on the relay wall, yielding a time-resolved impulse response $H(\mathbf{x}_l, \mathbf{x}_s, t)$, where t denotes time of flight from $\mathbf{x}_l$ to $\mathbf{x}_s$.

We distinguish between *confocal capture*, where laser and sensor are directed to the same point (thus $\mathbf{x}_l = \mathbf{x}_s$), and *non-confocal capture*, which relaxes this constraint and enables significantly faster acquisition. On the other hand, confocal capture enables more efficient imaging methods by restricting the possible light paths. The f–k migration method is restricted to confocal data, and the RSD formulation is designed for both modalities. In any case, non-confocal data can be approximately converted to confocal via temporal shifts of $H(\mathbf{x}_l, \mathbf{x}_s, t)$ (see e.g., work by Lindell et al. [11]), hence both methods can work with both capture modalities.

Generally, the reconstruction procedure operates on the impulse response $H(\mathbf{x}_l, \mathbf{x}_s, t)$ to obtain an image $f(\mathbf{x}_v)$ at points $\mathbf{x}_v$ on the hidden scene. Below is a brief overview of how the methods in our work operate:

3.1. NLOS imaging methods

f–k migration. Lindell et al. [11] interpret the hidden scene as a wave field $\Psi(x, y, z, t)$ where each point emits a spherical wave at $t = 0$. The *confocal* impulse response $H(\mathbf{x}_s, \mathbf{x}_s, t)$ measures $\Psi(x_s, y_s, z_s, t)$ at $\mathbf{x}_s = (x_s, y_s, z_s)$ on the relay wall plane at $z_s = 0$ at a later time $t > 0$. The NLOS reconstruction problem is defined as a boundary value problem of this wave field, using Stolt interpolation in the frequency domain to migrate the field from $z = 0$ to $t = 0$, which corresponds to the reconstructed hidden scene $f(\mathbf{x}_v)$:

$$\Psi(x, y, z = 0, t) \overset{\text{Stolt}}{\Longrightarrow} \Psi(x, y, z, t = 0), \quad (1)$$

$$f(\mathbf{x}_v) = \Psi(x_v, y_v, z_v, t = 0). \quad (2)$$

RSD-based method. Liu et al. [13] modulate an arbitrary signal $\mathcal{P}(\mathbf{x}_l, t)$ on top of the impulse response, yielding $\hat{\mathcal{P}}_\omega(\mathbf{x}_l, \mathbf{x}_s) = \mathcal{F}\{P(\mathbf{x}_l, t) *_t H(\mathbf{x}_l, \mathbf{x}_s, t)\}(\omega)$ which represents out-of-focus light waves at the relay wall of angular frequency ω via a convolution $*_t$ and Fourier transform $\mathcal{F}$. Then, NLOS reconstruction is a plane-to-plane propagation that focuses light from the relay wall to the hidden scene:

$$\hat{\mathcal{P}}_\omega(\mathbf{x}_v) = \int_{\mathcal{S}} e^{i\omega t_s} \int_{\mathcal{L}} e^{i\omega t_l} \hat{\mathcal{P}}_\omega(\mathbf{x}_l, \mathbf{x}_s)\, \mathrm{d}\mathbf{x}_l \mathrm{d}\mathbf{x}_s \quad (3)$$

where $t_l = |\mathbf{x}_l - \mathbf{x}_v|/c$ and $t_s = |\mathbf{x}_v - \mathbf{x}_s|/c$ represent times of flight, with c the speed of light. The result $f(\mathbf{x}_v)$ is:

$$f(\mathbf{x}_v) = \int_{-\infty}^{+\infty} \hat{\mathcal{P}}_\omega(\mathbf{x}_v)\, \mathrm{d}\omega. \quad (4)$$

4. Method

In this section we describe the design of our NLOS imaging implementations, which allows us to process and reconstruct high-resolution datasets a frame rate two orders of magnitude higher than all previous work. The key ideas from our work come from careful kernel design that allows to skip storing or even computing intermediate variables. We merge different stages of computation into a single kernel to minimize host-device transfer times, and design a performant kernel scheduling that records computation dependencies for optimized real-time data streaming.

In this section we will differentiate between *real-time* NLOS imaging and *offline* NLOS imaging (Section 1). We describe most of this section with the real-time approach in mind, including our optimizations for the f–k migrationand RSD-based methods, and in the end we describe the differences for offline NLOS imaging.

4.1. Producer-consumer data processing

SPAD devices output a list of timestamps corresponding to photon arrival times, which are processed to form a transient histogram H and later used for reconstruction (see Figure 2). Our work accomplishes this following the producer-consumer (PC) structure from Figure 3. These PC systems are well suited for simultaneously solving the multiple tasks required by a SPAD asynchronously in a multi-threaded environment. Briefly, PC systems are composed of threads that

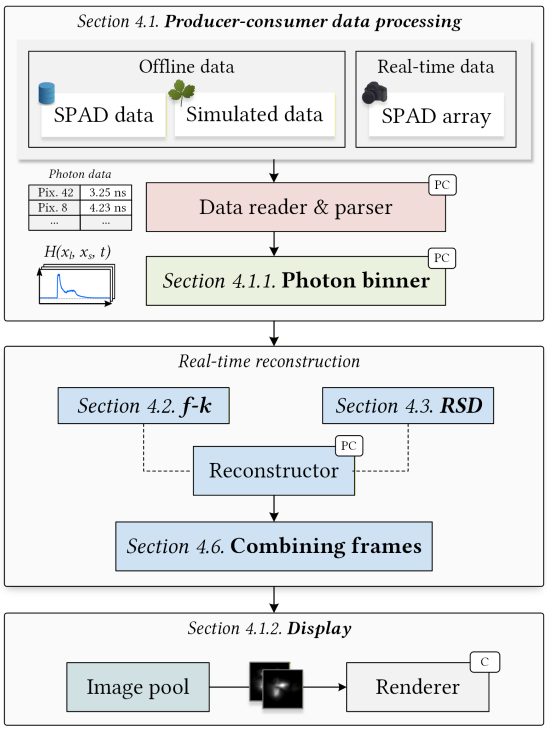

Figure 3. Overview of the producer-consumer scheme. **P** and **C** indicate whether an instance is a consumer, a producer, or both.

(i) **P**roduce data and push them into a queue, (ii) pop data for **C**onsuming, or (iii) perform both operations (**PC**). In a PC system, multiple instances of each worker may exist; however, for simplicity in this explanation, we assume a single instance of each component.

We adapted the work of Nam et al. [17] to implement the PC data processing. Their system supports reading raw photon data from a binary file (*SPAD data*) and photons collected from a SPAD array (*Real-time data*). Additionally, our work supports reading *offline data*, including simulated transient data and existing NLOS datasets [7, 11]. In contrast to the dynamic data acquired by Nam et al. [17], *offline* data is pre-binned into histograms instead of being loaded as a stream of individual photons.

The *Data reader & parser* reads raw photon data and stream the complete record of photons captured by the SPAD array after iterating over every relay wall target. Next, the *Photon binner* accumulates the photons into histograms (one per relay wall target), yielding spatio-temporal data. In the case of RSD the binning is directly performed in the frequency domain by computing the Fourier transform directly. Then, the *Reconstructor* works over a spatio-temporal field (f–k migration) or a phasor field (RSD) and writes the result into a texture from the *Image pool*. For *offline* data, there is no *Data reader & parser*, and the *Photon binner* only copies data into the expected layout (frequency-major for RSDand frequency-minor for f–k migration).

4.1.1. Photon binner

Processing data from the SPAD requires binning the raw photon counts to a histogram before reconstruction. To maximize throughput, we perform this operation on the GPU, whereas Nam et al. [17] solved it on the CPU using multithreading. As discussed in the Appendix, this stage becomes increasingly time-consuming for large photon counts. To accelerate data transfers, photon data are first uploaded to host pinned memory instead of VRAM. This type of memory is locked in physical RAM and cannot be swapped to disk, allowing faster transfers via Direct Memory Access. Although this approach may slightly reduce computational performance during binning, because the data resides outside the GPU's VRAM, it is worthwhile as data transfers are considerably more time-consuming. Then, the uploaded photon data are used to build the histograms utilized during reconstruction.

4.1.2. Display

The purpose of the *Image pool* and *Renderer* modules in Figure 3 is to enable interoperability between OpenGL (displaying reconstructed NLOS images) and CUDA (compute). The *Image pool* maintains a set of buffers that are written by the reconstructor and read by the *Renderer*. The pipeline operates as follows: (i) the reconstructor waits for an available image buffer; once written, (ii) the image is pushed into the presentation queue and removed from the writing queue. Asynchronously, (iii) the renderer waits for an image in the presentation queue; once displayed, (iv) the image is returned to the writing queue for reuse. The *Renderer* writes the contents of an available image into a CUDA surface, which can be both written from CUDA kernels and sampled from OpenGL shaders.

4.2. f–k migration optimizations

The f–k migration method consists of four main stages to reconstruct the hidden scene: (i) distance falloff compensation, (ii) a 3D Fast Fourier Transform (FFT), (iii) Stolt remapping, and (iv) an inverse FFT. The complexity of its memory allocation is $\mathcal{O}(n^3)$ with $n = 8 \cdot \max(x, y, z)$; however, existing implementations allocate several auxiliary arrays for padding, remapping, and FFT operations, which introduces huge memory requirements for high-resolution NLOS scenes. For example, the compensation (i) is often implemented as $\Psi \odot S^{x,y,z}$, which requires pre-computing S, and Stolt remapping proceeds similarly (see Algorithm 1 in the Appendix).

We reduced the overall memory footprint by solving the Stolt remapping on the fly to avoid allocating auxiliary buffers. Following this approach, each thread handles a voxel, evaluates its mapping and accesses the eight voxels required for a trilinear interpolation. Although there is a small overlap

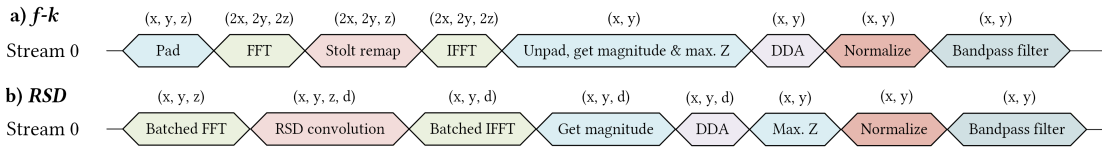

Figure 4. Overview of the CUDA-accelerated f–k migration (top) and RSD (bottom) algorithms. Each node corresponds to a kernel call, and the label above it indicates the number of threads used to perform the computation. The symbols x and y denote the spatial dimensions, z the number of time bins (f–k) or frequencies (RSD), and d the number of propagated depths in RSD.

in memory accesses between neighboring voxels, we did not observe any performance gain from pre-caching surrounding voxels in shared memory, as shown in Figure 6 (left). In the experiment, pre-caching was performed by loading the values within each block, plus a one-voxel neighbourhood, to enable data reuse among all block threads.

Moreover, our f–k algorithm only requires two buffers, Ψ' and Ψ'', of size $(2x, 2y, 2z)$, besides the input $\Psi(x, y, z)$ and the reconstructed image $f(x_v)$. Each processing stage alternately reads from one buffer and writes to the other, following a double-buffering scheme. Note that these buffers double the original dimensions to apply zero-padding and prevent wrap-around artifacts in the FFT, as illustrated in Figure 5. Despite the presence of artifacts, we included the variant without padding in our experiments as a faster alternative, trading reconstruction quality for higher performance. Prior real-time approaches have likewise omitted padding, presumably for similar efficiency reasons [17].

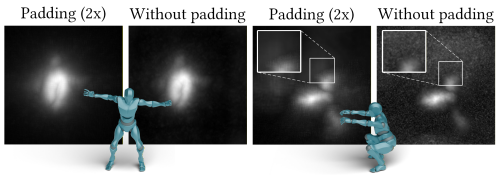

Figure 5. Two scenes reconstructed with and without padding using f–k. The reconstruction on the left side does not show artifacts without padding, whereas the right one does; however, the hidden scene remains recognizable and the versions without padding were computed at much higher speed (theoretically up to 8× faster).

Another key optimization arises from the observation that the original f–k formulation ignores forward transport by setting half of the remapped matrix, Ψ'', to zero (see Line 12 of Algorithm 1 in the Appendix). As illustrated in Figure 4, this reduces the number of active threads in the *Stolt remapping* step to $(2x, 2y, z)$. To enable this optimization, Ψ'' must first be initialized with zeros, and only its half is updated during the *Stolt remapping*. Additional acceleration is obtained by merging operations that were originally performed as separate steps; for example, distance falloff compensation and fftshift. Immediately after the IFFT, we perform unpadding and peak-magnitude search within a single kernel, selecting for each spatial location the spatial slice with maximum backscattered energy.

4.3. RSD-based method optimizations

The inputs to the RSD-based method are the histograms H converted from the time to the frequency domain (the phasors). Note that pipeline further converts such phasors from the 2D spatial to the 2D spatial frequency domain. To align both representations, the data is transformed using 2D FFTs, one per temporal frequency. Nam et al. [17] already leveraged CUDA's FFT implementation, and we further optimized it through CUDA's batched FFT. As discussed earlier, a notable speed-up comes from merging multiple kernels to minimize launch overhead, and batched FFTs enable computing many spatial transforms concurrently.

A major limitation of RSD lies in its memory footprint: it requires precomputing a large complex-valued kernel, $\mathcal{H}(x, y, z, d)$, where d denotes the number of propagated depths. Otherwise, real-time reconstruction would be impractical. Among other drawbacks, we cannot afford padding data for the FFT due to (i) excessive memory allocation and (ii) real-time constraints. Recent works have addressed excessive memory usage by exploiting the kernel's radial symmetry while maintaining comparable or even superior performance [9], though this does not fully resolve the performance drawback. For instance, Jiang et al. [9] reported an execution time of 1.10 s for a scene of size $(190, 190, 93)$ propagated across 50 depths.

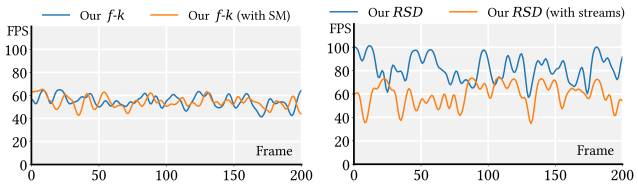

Figure 6. The left plot shows the impact of shared-memory in our f–k implementation, while the right plot compares our RSD pipeline against a multi-stream variant.

Following the FFT, the spatial–frequency data are convolved with the precomputed RSD kernel, producing an intermediate buffer of size (x, y, d). During this operation,

$$\mathcal{I}(x, y, d) = [\Psi'(x, y, z)w(z)] *_{x,y,z} \mathcal{H}(x, y, z, d), \quad (5)$$

$\mathcal{I}$ must be updated with atomic additions, since f complex samples per (x, y) are accumulated into each (x, y, d) element. Weights $w(z)$ follow a Gaussian distribution, hence

having higher weights for middle frequencies. The convolved signal is then transformed via a batched IFFT, performing d two-dimensional inverse FFTs in parallel. Finally, we unpad the resulting data, compute its magnitude, and postprocess it as described in Section 4.5.

Figure 4 provides an overview of our RSD implementation, which resolves all spatial, temporal, and frequency computations simultaneously. In contrast, the method by Nam et al. [17] divides the convolution into two stages: (i) the convolution itself and (ii) depth-wise weighting (scaling by $w(z)$). The weighting stage is executed independently for each frequency and depth, resulting in $f \times d$ separate kernel launches. A straightforward improvement to this approach is to parallelize frequency-wise computations using CUDA streams, i.e., independent command queues that enable asynchronous kernel execution, and to merge multiple convolution kernels into a single one. Figure 6 (right) compares the performance of our RSD implementation with this stream-based variant, showing that executing all frequencies within a single command substantially improves throughput.

4.4. CUDA kernels and graphs

The reconstruction pipelines execute in an infinite loop; thus, optimizing kernel scheduling and minimizing launch overhead is critical. CUDA provides graphs to define dependencies among kernel calls (graph nodes), enabling the GPU to know in advance which kernels, parameters, and grid configurations will be executed. Since a graph must first be recorded before execution, this optimization is performant only for pipelines executed more than once. As illustrated in Figure 4, both f–k migration and RSD methods, including the postprocessing, are recorded in a single CUDA graph.

4.5. Postprocessing of reconstructed frames

The postprocessing stage follows the approach of Nam et al. [17] and proceeds as follows. First, the Depth-Dependent Average (DDA) is computed as a weighted sum of the last three reconstructed frames, with the middle frame having a higher weight. The DDA is then followed by a normalization step to prepare the data for display, and optionally by a bandpass filter that removes values outside the interval $[h_b, h_t]$, with $h_b, h_t \in (0, 1)$. We intentionally omit here operations such as computing the magnitude of complex-valued buffers, since, as mentioned earlier, we opted for fusing as many kernels as possible. Accordingly, magnitude extraction is merged with the unpadding step after the IFFT and the subsequent maximum search, in both the f–k migration and RSD methods.

4.6. Combining reconstructed frames

We introduce a novel strategy that exploits temporal consistency across consecutive NLOS images of the same hidden region to suppress noise and enhance hidden objects. Consider a sequence of n NLOS images. Looking at Equations 2 and 4, our strategy takes the raw output $f_i(\mathbf{x}_v)$ from any algorithm (where the subscript $i \in \{1, \dots, n\}$ represents the frame number in the sequence), and outputs a cleaned frame $\bar{f}_i(\mathbf{x}_v)$ with less noise.

We leverage the fact that both the RSD and f–k migration-methods are wave-based, thus their reconstructions $f_i(\mathbf{x}_v)$ not only represent intensity but also contain the phase of the reconstructed wave. This phase is typically discarded in favour of the intensity, however it contains key additional information. Intuitively, this phase will not change in points $\mathbf{x}_v$ in the hidden scene that correspond to static objects, while image noise will contain random phase that varies between frames. We introduce a coherence metric C_i for each frame, that measures the phase alignment of that frame $f_i(\mathbf{x}_v)$ with respect to the k previous ones:

$$C_i(\mathbf{x}_v) = \frac{\left|\sum_{j=i-k}^{i} f_j(\mathbf{x}_v)\right|}{\varepsilon + \sum_{j=i-k}^{i} |f_j(\mathbf{x}_v)|} \tag{6}$$

where $|f_i(\mathbf{x}_v)|$ represents the amplitude of the wave and ε a small positive constant. With this, we compute our enhanced frame $\bar{f}_i(\mathbf{x}_v)$ as the mean of the k previous frames, weighted by our coherence metric $C_i(\mathbf{x}_v)$:

$$\bar{f}_i(\mathbf{x}_v) = C_i(\mathbf{x}_v)\, \frac{1}{k} \left| \sum_{j=i-k}^{i} f_j(\mathbf{x}_v) \right| \tag{7}$$

4.7. Offline reconstructions

Previously described methods remain similar for offline reconstruction, except that single executions do not benefit from CUDA graph optimization. Moreover, precalculations are not required, making offline implementations better suited for validating design choices, as CUDA graphs are more difficult to test and debug. Also, for a fair comparison, we did not implement the offline RSD following our real-time approach. Instead, we accelerated the RSD implementation released and used by Lindell et al. [11] to validate f–k migration. Both methods are essentially the same, although the RSD kernel construction in Lindell et al.'s implementation of Liu et al's method [12] is considerably more complex. Our optimized version simplifies this construction to make it more GPU-friendly, as discussed in the Appendix.

5. Results and evaluation

Here we showcase the efficiency of our algorithm, measuring execution time and memory usage under different conditions. We split our tests under two categories: first, Section 5.1 deals with *real-time* NLOS imaging, which accounts for simultaneous reconstruction of one frame and processing of the next frame. In Section 5.2 we compare against other approaches optimized for real-time imaging. Later, Section 5.3

Table 1. Average frame time, and peak RAM and VRAM usage for our implementations and [17]. Dataset dimensions are shown above each group of results.

Approach	↓ Frame time	↑ FPS	↓ Peak VRAM	↓ Peak RAM
Dynamic data $190 \times 190 \times 208 \Rightarrow 63$ depths, `newmovement3` dataset				
Our f–k	18.01 ± 7.13 ms	55.52	1,104 MB	1,354 MB
Our f–k (w/o pad.)	2.62 ± 4.13 ms	381.67	302 MB	
Our RSD	17.25 ± 12.09 ms	57.97	3,302 MB	
RSD (Nam et al.)	188.86 ± 21.64 ms	5.29	13,138 MB	
Static data $128 \times 128 \times 128 \Rightarrow 128$ depths				
Our f–k	4.01 ± 0.29 ms	249.37	272 MB	478 MB
Our f–k (w/o pad.)	0.57 ± 0.59 ms	1,754.38	48 MB	
Our RSD	6.17 ± 0.38 ms	162.07	2,120 MB	
RSD (Nam et al.)	207.38 ± 49.08 ms	4.82	8,593 MB	
$256 \times 256 \times 512$				
Our f–k	68.21 ± 0.29ms	14.66	4,352 MB	824 MB
Our f–k (w/o pad.)	10.04 ± 0.40 ms	99.60	768 MB	

deals with *offline* NLOS imaging, which only accounts for reconstruction time. Finally, Section 5.3.1 showcases our phase-aware frame combining approach.

Execution hardware. Unless otherwise mentioned, all the experiments were carried out in a computer with Intel(R) Core(TM) i7-14700KF (3.40 GHz), 64 GB RAM, RTX 4080 SUPER GPU with 16 GB VRAM, and Windows 11 OS. Our implementations utilize C++23 using CUDA 13.0 and OpenMP for CPU multi-threading. Real-time visualization operates on OpenGL 4.6 for rendering and GPGPU (general-purpose computing on GPU).

Adaptation to real SPAD arrays. Our work processes the raw photon records from a SPAD array, instead of streaming from live hardware. Consequently, we read and process photon data as fast as possible from disk, which does not necessarily reflect the transfer rates achievable by current devices. Reading data from disk allows us to stress test our systems at the maximum capacity without being limited by current technology.

5.1. Real-time performance

Our real-time implementation utilizes data captured by Nam et al. [17], whose work we directly compare with, and simulated data. They provide the raw output of their SPAD array, consisting of individual photon timestamps. To ensure a fair comparison, both read and process the same raw output. This allows us to measure how fast each algorithm would theoretically process captured data under the capabilities of next-generation capture hardware. Table 1 shows the performance of our work and theirs while reconstructing the dynamic dataset that is partly displayed within the plot. Additionally, we include here, and in the following experiments, a f–k migration variant that does not pad data for FFT, similarly to the RSD method.

Besides the vastly superior performance of f–k migration without padding, two aspects are worth noting. First, the frame rate fluctuates slightly because our pipeline waits for the CUDA stream that transfer data to the GPU to complete, which can be delayed by other system processes. Second, RSD performs slightly better than f–k migration due to the low depth resolution (63 depths) and the small spatial resolution (190×190). To further stress our system, we carried out the same experiment with simulated data and increased depth resolution. As shown on the left side of Figure 1, f–k migration without padding remains the fastest by an astonishing margin, followed by standard f–k migration and then RSD, which now propagates across 128 depths. The same experiment was repeated with higher spatial, temporal and depth resolution ($256 \times 256 \times 512 \Rightarrow 512$ depths); f–k migration achieved approximately 14 FPS and its unpadded variant reached about 100 FPS, whereas RSD could not allocate its kernel on the GPU. In Section 5.3, we reconstruct datasets of comparable and larger dimensions using an alternative RSD implementation.

The last aspect introduces an additional concern: the peak usage of VRAM and RAM. Table 1 reports the memory statistics collected from the experiments shown in Figure 1, as well as from an additional dynamic NLOS experiment summarized in the Appendix. Real-time approaches make limited use of RAM but are significantly more demanding on the GPU. Nevertheless, f–k migration remains the most efficient method in both VRAM usage and average frame time, requiring about 4 GB for $256 \times 256 \times 512$ voxels.

5.2. Comparison with other methods

We compared our reconstruction throughput against other real-time approaches. Specifically, Table 2 summarizes the performance of our method, Fast Back-Projection [2], and Liao et al. [10]. The latter targets FPGA hardware, and no source code is publicly available; thus, we extrapolated their reported throughput for a $128 \times 128 \times 69$ dataset to other dataset sizes. We evaluate all methods in terms of millions of voxels processed per second and average reconstruction time. For smaller volumes, Fast Back-Projection achieves the lowest throughput, whereas RSD performs well due to the low temporal dimensionality. In contrast, f–k migration becomes superior as the temporal resolution increases. For completeness, Nam et al.'s method could not be tested as it exceeded the available VRAM.

5.3. Offline reconstruction

Figure 7 summarizes the throughput in millions of voxels processed per second for a subset of the offline performance results presented in the Appendix. We evaluated our work on datasets from Lindell et al. [11] and Galindo et al. [7], including confocal and exhaustive captures. The latter were converted to confocal measurements using the Normal Move-

Table 2. Performance of fast reconstruction algorithms, with throughput reported in millions of voxels per second. The *Hardware* column indicates the platform used for each test. Values marked with an asterisk (*) correspond to Liao et al. [10] as their results were extrapolated from the throughput reported in their work. OOM denotes experiments that ran out of memory.

Approach	↑ Mvox/s	↓ Average time (s)	Hardware
Dataset: **Z**, resized to $256 \times 256 \times 512 \Rightarrow 64$ depths			
Our f–k	492.90	0.068 ± 0.00	Nvidia RTX 4080 Ti
Our RSD	62.14	0.54 ± 0.03	
RSD (Nam et al.)	OOM	OOM	
Fast back-projection	1.12	14.962 ± 75.74	
Liao et al.	0.148*	28.3*	Stratix 10 FPGA
Dataset: **Z**, resized to $128 \times 128 \times 256 \Rightarrow 64$ depths			
Our f–k	516.95	0.008 ± 0.00	Nvidia RTX 4080 Ti
Our RSD	619.47	0.007 ± 0.00	
RSD (Nam et al.)	14.95	0.281 ± 0.03	
Fast back-projection	2.50	1.675 ± 0.00	
Liao et al.	28.3*	1.186*	Stratix 10 FPGA

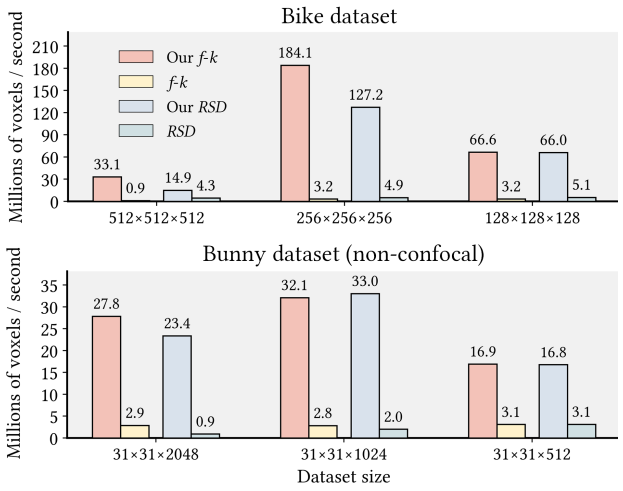

Figure 7. Throughput, in millions of voxels processed per second, for reconstructions from confocal and non-confocal measurements.

out Correction [11]. In addition to reconstruction time, we also recorded peak RAM and VRAM usage.

Across most experiments, our f–k achieves the highest performance, with our RSD surpassing it by only a few ms in 18.5% of the tests. For both cases we are much faster than the original implementations. It is worth to note that both approaches exhibit a bell-shaped trend in Figure 7: throughput improves for medium-sized datasets but decreases for small or large ones. The measured times include both resource allocation and reconstruction, which penalizes smaller datasets due to initialization overhead, while larger datasets are limited by increased spatial and temporal dimensions. We observed that increasing spatial resolution is the main bottleneck, whereas higher temporal resolution adds relatively little overhead. In both approaches, this limitation arises primarily from the 3D FFT and IFFT stages.

5.3.1. Combining reconstructed frames

We showcase in Figure 8 how the frame combining algorithm discussed in Section 4.6 leverages temporal consistency in consecutive NLOS reconstructions, in order to attenuate the noise and enhance the result. We simulate a NLOS capture where the hidden scene contains a humanoid-shaped object that moves from side to side, which corresponds to a speed of 15 captured frames $f_i(\mathbf{x}_v)$ per second, and the whole sequence consists of $n = 60$ frames, $i \in \{1, \ldots, 60\}$.

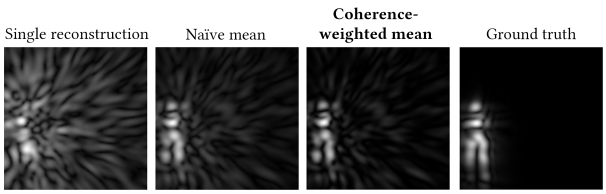

Figure 8. Comparison of reconstruction strategies on a dynamic NLOS sequence. From left to right: single–frame reconstruction, naïve temporal mean, our weighted temporal mean, and ground truth of the final frame.

From left to right, Figure 8 shows the raw reconstructed frame $f_i(\mathbf{x}_v)$, a naïve weighted averaging such as the one used by Nam et al. [17], which does not include our phase alignment term $C_i(\mathbf{x}_v)$ (Equation 7), and our improved version $\bar{f}_i(\mathbf{x}_v)$, which combines the last $k = 8$ frames from Equations 6 and 7. The fourth column contains a simulated ground truth corresponding to the same frame $f_i(\mathbf{x}_v)$ with a much larger number of photons. In comparison, our phase-aware strategy provides a much cleaner result, showing the humanoid and reducing the noise on the outer image regions.

6. Conclusions and limitations

In this paper, we have presented a CUDA-accelerated system for reconstructing high-resolution NLOS scenes from both raw photons and simulated transient data in real-time, as well as for processing existing high-resolution datasets in an *offline* setting. Our real-time pipeline presents frame rates above 50 FPS on dynamic datasets, while the offline pipelines substantially outperform their original implementations, reducing reconstruction time by 98.67% for f–k migration and 78.87% for RSD on the largest dataset.

Although our system outperforms prior work, several challenges remain open. Both f–k migration and RSD rely heavily on *Fourier* operators, which are the most time-consuming stages of our pipelines. Reducing data-type precision to, e.g., 16-bits floats, could cut down time, albeit with potential quality trade-offs. Furthermore, the photon-binning stage continues to be expensive, making partial and asynchronous data uploads a key direction for future improvement. Finally, leveraging additional properties of RSD, including its kernel symmetries, may further reduce memory usage and improve overall performance.

References

[1] Byeongjoo Ahn, Akshat Dave, Ashok Veeraraghavan, Ioannis Gkioulekas, and Aswin Sankaranarayanan. Convolutional Approximations to the General Non-Line-of-Sight Imaging Operator. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7888–7898, 2019. ISSN: 2380-7504. 2

[2] Victor Arellano, Diego Gutierrez, and Adrian Jarabo. Fast back-projection for non-line of sight reconstruction. *Optics Express*, 25(10):11574–11583, 2017. Publisher: Optica Publishing Group. 2, 7

[3] Mauro Buttafava, Jessica Zeman, Alberto Tosi, Kevin Eliceiri, and Andreas Velten. Non-line-of-sight imaging using a time-gated single photon avalanche diode. *Optics Express*, 23(16): 20997–21011, 2015. Publisher: Optical Society of America. 2

[4] Javier Grau Chopite, Patrick Haehn, and Matthias Hullin. Non-Line-of-Sight Estimation of Fast Human Motion with Slow Scanning Imagers. In *Computer Vision – ECCV 2024*, pages 176–194, Cham, 2025. Springer Nature Switzerland. 2

[5] Enrico Conca, Simone Riccardo, Vincenzo Sesta, Davide Portaluppi, Franco Zappa, and Alberto Tosi. Design of a 16 x 16 fast-gated SPAD imager with 16 integrated shared picosecond TDCs for non-line-of-sight imaging. In *Emerging Imaging and Sensing Technologies for Security and Defence IV*, pages 25–32. SPIE, 2019. 2

[6] Daniele Faccio, Andreas Velten, and Gordon Wetzstein. Non-line-of-sight imaging. *Nature Reviews Physics*, 2(6):318–327, 2020. Publisher: Nature Publishing Group UK London. 1

[7] 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). 4, 7

[8] Adrian Jarabo, Belen Masia, Julio Marco, and Diego Gutierrez. Recent advances in transient imaging: A computer graphics and vision perspective. *Visual Informatics*, 1(1):65–79, 2017. 1

[9] Deyang Jiang, Xiaochun Liu, Jianwen Luo, Zhengpeng Liao, Andreas Velten, and Xin Lou. Ring and Radius Sampling Based Phasor Field Diffraction Algorithm for Non-Line-of-Sight Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(11):7841–7853, 2022. 2, 5

[10] Zhengpeng Liao, Deyang Jiang, Xiaochun Liu, Andreas Velten, Yajun Ha, and Xin Lou. FPGA Accelerator for Real-Time Non-Line-of-Sight Imaging. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 69(2):721–734, 2022. 2, 7, 8

[11] 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. 2, 3, 4, 6, 7, 8

[12] 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, 3, 6

[13] Xiaochun Liu, Sebastian Bauer, and Andreas Velten. Phasor field diffraction based reconstruction for fast non-line-of-sight imaging systems. *Nature Communications*, 11(1):1645, 2020. Publisher: Nature Publishing Group. 2, 3

[14] Pablo Luesia-Lahoz, Diego Gutierrez, and Adolfo Muñoz. Zone Plate Virtual Lenses for Memory-Constrained NLOS Imaging. In *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5, 2023. ISSN: 2379-190X. 2

[15] Davide Moschella, Davide Berretta, Alberto Tosi, and Federica Villa. A 64\times 64 SPAD Array For Quantum Ghost Imaging with Integrated TDCs and Event-Driven Readout in a 40 nm CMOS Technology. In *2024 19th Conference on Ph. D Research in Microelectronics and Electronics (PRIME)*, pages 1–4. IEEE, 2024. 2

[16] Fangzhou Mu, Sicheng Mo, Jiayong Peng, Xiaochun Liu, Ji Hyun Nam, Siddeshwar Raghavan, Andreas Velten, and Yin Li. Physics to the Rescue: Deep Non-Line-of-Sight Reconstruction for High-Speed Imaging. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 47(08):6146–6158, 2025. Place: Los Alamitos, CA, USA Publisher: IEEE Computer Society. 2

[17] 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. 2, 4, 5, 6, 7, 8

[18] Matthew O'Toole, David B. Lindell, and Gordon Wetzstein. Real-time non-line-of-sight imaging. In *ACM SIGGRAPH 2018 Emerging Technologies*, pages 1–2, New York, NY, USA, 2018. Association for Computing Machinery. 2

[19] Matthew O'Toole, David B. Lindell, and Gordon Wetzstein. Confocal non-line-of-sight imaging based on the light-cone transform. *Nature*, 555(7696):338–341, 2018. Publisher: Nature Publishing Group. 2

[20] Adithya Pediredla, Akshat Dave, and Ashok Veeraraghavan. SNLOS: Non-line-of-sight Scanning through Temporal Focusing. In *2019 IEEE International Conference on Computational Photography (ICCP)*, pages 1–13, 2019. ISSN: 2472-7636. 2

[21] Oscar Pueyo-Ciutad, Julio Marco, Stephane Schertzer, Frank Christnacher, Martin Laurenzis, Diego Gutierrez, and Albert Redo-Sanchez. Time-Gated Polarization for Active Non-Line-Of-Sight Imaging. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, New York, NY, USA, 2024. Association for Computing Machinery. 2

[22] Andreas Velten, Thomas Willwacher, Otkrist Gupta, Ashok Veeraraghavan, Moungi G Bawendi, and Ramesh Raskar. Recovering three-dimensional shape around a corner using ultrafast time-of-flight imaging. *Nature Communications*, 3(1): 745, 2012. Publisher: Nature Publishing Group UK London. 2

[23] Andreas Velten, Di Wu, Adrian Jarabo, Belen Masia, Christopher Barsi, Chinmaya Joshi, Everett Lawson, Moungi Bawendi, Diego Gutierrez, and Ramesh Raskar. Femto-Photography: Capturing and Visualizing the Propagation of Light. *ACM Transactions on Graphics*, 32(4):1–8, 2013. Publisher: ACM New York, NY, USA. 2

[24] Juntian Ye, Yu Hong, Xiongfei Su, Xin Yuan, and Feihu Xu. Plug-and-Play Algorithms for Dynamic Non-line-of-sight Imaging. *ACM Trans. Graph.*, 43(5):155:1–155:12, 2024. 2