DPF: Differentiable Polyphase Filtering for Large-Kernel Approximation

ZHE CAO*, State Key Lab of CAD&CG, Zhejiang University, China
ZHIZHEN WU*, State Key Lab of CAD&CG, Zhejiang University, China
ZHONGGUI CHEN, School of Informatics, Xiamen University, China
RUI WANG, State Key Lab of CAD&CG, Zhejiang University, China
YUCHI HUO†, State Key Lab of CAD&CG, Zhejiang University, China



Fig. 1. **We propose an approach for rapid large-kernel spatially varying filtering via our Differentiable Polyphase Filtering pipeline.** We compare our method with optimization-based sparse baselines (PST [Schuster et al. 2020], DSKC [Wu et al. 2026]). For anisotropic rotational motion blur and spatially varying ring-shaped bokeh filters, our approach achieves approximately 3000× and 600× speedups over the ground-truth based on dense kernel convolution, respectively, and is about 2× faster than the state of the art.

Large-kernel filtering is a fundamental operation in image post-processing and video effects. While decomposing large kernels into multiple sparse ones is a proven strategy for acceleration on parallel architectures, existing optimization-based approximation methods are constrained to fixed-resolution processing. In this paper, we propose a novel framework based on Polyphase Filtering, which enables the differentiable optimization of sparse kernels across varying resolutions. This approach achieves the fastest approximation to date for large kernels and remains highly effective even for spatially-variant dense kernels. To further enhance efficiency, we introduce Spatial-to-Depth Transformation and Asymmetric Intensity-Decoupled Transformation. Furthermore, to address scenarios with spatially varying kernel sizes, we employ a Layer-Adaptive Filtering strategy that integrates kernels from multiple levels for rapid filtering. Extensive experiments demonstrate that our method significantly outperforms existing state-of-the-art techniques in both computational performance and visual quality.

*Equal contribution
†Corresponding author

Authors' Contact Information: Zhe Cao, caozhe022@qq.com, State Key Lab of CAD&CG, Zhejiang University, China; Zhizhen Wu, zhizhenwu@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Zhonggui Chen, chenzhonggui@xmu.edu.cn, School of Informatics, Xiamen University, China; Rui Wang, ruiwang@zju.edu.cn, State Key Lab of CAD&CG, Zhejiang University, China; Yuchi Huo, huo.yuchi.sc@gmail.com, State Key Lab of CAD&CG, Zhejiang University, China.

This work is licensed under a Creative Commons Attribution 4.0 International License.
SIGGRAPH Conference Papers '26, Los Angeles, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2554-8/2026/07
https://doi.org/10.1145/3799902.3811160

CCS Concepts: • **Computing methodologies** → **Image processing**; **Machine learning algorithms**; **Rendering**.

ACM Reference Format:
Zhe Cao, Zhizhen Wu, Zhonggui Chen, Rui Wang, and Yuchi Huo. 2026. DPF: Differentiable Polyphase Filtering for Large-Kernel Approximation. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Papers (SIGGRAPH Conference Papers '26), July 19–23, 2026, Los Angeles, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3799902.3811160

1 INTRODUCTION

In digital image processing and computer graphics, large-kernel filtering is a fundamental operation that can produce visually striking effects such as bloom, bokeh, and radial or rotational blur. However, the computational complexity associated with large kernel sizes and spatially varying filtering hinders their real-time application on mobile platforms.

One common way to accelerate filtering is kernel decomposition, which approximates a large kernel with a sequence of smaller ones, such as Multi-box [Kovesi 2010] and the sparse multi-pass method Kawase blur [Kawase 2003]. Dual Filtering [Martin et al. 2015] pushes this idea by iterating downsampling and upsampling to cut the number of processed pixels per layer exponentially.

However, Dual Filtering relies on a heuristic approach where the kernel samples at each level only roughly approximate a Gaussian distribution, and its parameters typically require manual tuning. Consequently, a dual-filter configuration cannot reliably approximate a Gaussian blur level with a specific σ. This inherent inaccuracy

leads to visual artifacts and the inability to smoothly control the blur radius, which significantly hinders the adoption of this efficient technique in high-standard industrial applications.

Furthermore, the aforementioned methods are generally limited to Gaussian kernels. To address arbitrary kernels, optimization-based approaches have emerged. Parallel Simulated Tempering (PST) [Schuster et al. 2020] optimizes parameters to approximate kernels, but its high stochasticity severely limits both convergence speed and accuracy. DSKC [Wu et al. 2026] introduced a differentiable gradient descent approach to achieve more efficient optimization. Nevertheless, both PST and DSKC perform filtering at a fixed resolution, which imposes a significant performance bottleneck.

To address these limitations and eliminate visual artifacts, we propose a novel framework for creating high-fidelity, high-performance approximations of large kernels. We introduce *Differentiable Polyphase Filtering*, a method that approximates target kernels using a multi-layer, sparse, and multi-resolution architecture.

Our method integrates the structural benefits of Dual Filtering with the differentiable optimization used in DSKC, combining the speed of sparse multi-resolution sampling with high reconstruction quality. Downsampling, however, changes the system from Linear Shift-Invariant (LSI) [Willems and Polderman 1997] to Linear Periodically Shift-Variant (LPSV) [Crochiere and Rabiner 1981], so the single-impulse supervision in DSKC no longer provides enough constraints for the filtering process. To resolve this limitation, we introduce a polyphase impulse optimization strategy that jointly optimizes multiple phase-specific filters, ensuring that each phase matches the target kernel. The optimized parameters are dataset-independent and can be deployed directly on the mobile platform.

To maximize performance, we introduce two components: a *Spatial-to-Depth Transformation* and an *Asymmetric Intensity Decoupled Sampler*. The *Spatial-to-Depth Transformation* packs neighboring pixels into channels, shrinking the spatial support of the kernel we need to approximate. In parallel, the sampler converts RGB to YUV and treats luminance and chrominance differently: we apply the high-fidelity Spatial-to-Depth Transformation to the information-rich Y channel, while downsampling UV with a standard operator. Together, these steps compress the original three-channel RGB input into a compact four-channel representation, allowing subsequent *Polyphase Filtering* to use standard hardware texture formats efficiently at reduced resolution.

To efficiently support widely used spatially varying filtering, we propose Layer-Adaptive Filtering with Variable Kernel Sizes. In our framework, iterative downsampling and upsampling naturally expand the receptive field of deeper layers. Consequently, small kernels must use fewer layers to avoid over-smoothing, while large kernels utilize deeper layers. We leverage this by adaptively selecting and integrating kernels from different layers, allowing us to generate variable kernel sizes dynamically. This approach simultaneously achieves high performance and superior visual quality.

In summary, our key contributions are:

- We propose Differentiable Polyphase Filtering, a novel framework for approximating large kernels. This approach significantly outperforms existing kernel decomposition techniques in both computational efficiency and reconstruction fidelity.
- We design an acceleration strategy featuring Spatial-to-Depth Transformation and Asymmetric Intensity-Decoupled Sampling. This combination achieves a multiplicative reduction in computational cost by operating on compact representations, while preserving full-resolution visual quality.
- We introduce Layer-Adaptive Filtering, a dynamic mechanism that generalizes our fast algorithm to support spatially varying filtering with variable kernel sizes in real-time.

2 RELATED WORK

2.1 Large Kernel Approximations

Large-kernel filtering is often accelerated by exploiting structure in classical kernels. Recursive Gaussian filters compute each output from a small set of prior samples, making per-pixel cost largely scale-invariant [Van Vliet et al. 1998; Young and Van Vliet 1995]. Constant-time schemes further trade accuracy for speed, including summed-area/box "almost-Gaussian" filters and binomial cascades whose runtime does not grow with radius [Kovesi 2010; Zingl 2020]. On GPUs, multi-pass post-processing reduces large blurs to a few taps per pass, as in Kawase blur [Intel Corporation 2003; Kawase 2003; Martin et al. 2015]; related practice also approximates non-Gaussian bokeh with a small number of separable passes and pass reuse to keep cost stable at large radii [Ferrier and Coffin 2011]. For broader PSFs, prior work uses polygonal apertures, low-rank models [McGraw 2015], and linear-time variable-sized bokeh gather [McIntosh et al. 2012; Moersch and Hamilton 2020], as well as sparse fitting via simulated tempering [Schuster et al. 2020] and differentiable sparse convolution [Wu et al. 2026]. However, many methods are fixed-resolution, non-differentiable, or sample-inefficient at very large radii; our multi-resolution polyphase sparse sampling preserves throughput while enabling differentiable approximation of diverse kernels. Existing works derive sparse kernel representations through statistical heuristics [Leimkühler et al. 2018] or neural network prediction [Gharbi et al. 2019]. More recently, Bálint et al. utilized networks for spatial partitioning and pyramid filtering [Balint et al. 2023]. Unlike these methods that rely on heavy architectures or complex statistical estimations, our approach employs a differentiable training framework to achieve high efficiency.

2.2 Efficient Image Processing

Many methods accelerate per-pixel neighborhood aggregation. This includes constant-time bilateral filtering [Porikli 2008]. They also speed up edge-aware filtering with domain transform formulations [Gastal and Oliveira 2011]. Another option is guided filtering [He et al. 2012]. High-dimensional variants can be reduced via adaptive manifolds [Gastal and Oliveira 2012].

Pyramid-based reconstructions provide fast approximations to edge-preserving filters [Xu et al. 2024]. Learning-based pipelines often gain efficiency by reordering computation. Pixel shuffle is a common choice for upsampling [Shi et al. 2016]. They also reduce arithmetic cost with Winograd minimal filtering [Lavin and Gray 2016]. Some methods cut spatial redundancy via octave convolution [Chen et al. 2019]. Modern CNN backbones have revisited large-kernel designs [Ding et al. 2022; Xie et al. 2023]. However, many methods assume fixed resolution. Many also lack differentiable

cross-scale sparse optimization. They are often tuned for desktop GPUs rather than real-time mobile use. Our framework instead boosts efficiency by YUV decoupling. It also uses channel-to-depth transforms under spatially varying kernels.

2.3 Spatial Varying Filters

Spatially varying filtering is widely used for camera effects with spatially changing point spread functions (PSFs). Early examples include multiview depth-of-field synthesis and non-uniform deblurring [Lee et al. 2009; Whyte et al. 2012]. Later work targets real-time post-processing with structured approximations, from defocus magnification [Kriener et al. 2013] and polygonal bokeh [McIntosh et al. 2012] to visibility-aware multi-layer DoF [Selgrad et al. 2015] and feature-aware motion blur [Guertin et al. 2014]. Recent learning-based methods infer depth/blur parameters and render SV blur with differentiable modules, such as unsupervised depth-and-DoF learning [Kaneko 2021] and lens-conditioned bokeh transfer [Seizinger et al. 2023]. Differentiable sparse-kernel decomposition (DSKC) [Wu et al. 2026] is closest to our work, but focuses on fixed-scale settings rather than ultra-large kernels. Overall, prior SV methods often handle limited PSF families or rely on neural inference, and few target very large, diverse SV kernels; our polyphase framework instead optimizes sparse kernels across resolutions, scaling efficiently to extremely large spatially varying kernels via a hierarchical strategy.

3 PRELIMINARY

Multi-Layer Sparse Kernel. In conventional kernel filtering, the output image I_{out} is generated by convolving the input image I_{in} with a kernel K of size $(2k+1)\times(2k+1)$, as formally defined by:

$$I_{\text{out}}[x,y] = (I_{\text{in}} * K)[x,y] = \sum_{i=-k}^{k}\sum_{j=-k}^{k} I_{\text{in}}[x+i, y+j]\cdot K[i,j]. \quad (1)$$

For large kernels, an efficient alternative is kernel decomposition with a sequence of smaller kernels [Burt 1981] , where the output of one layer serves as the input for the next. Each layer reuses and propagates the signal from the previous layer, which gradually expands the receptive field.

Building on this principle, several methods [Kawase 2003; Schuster et al. 2020; Wu et al. 2026] approximate the target kernel by employing sparse sampling within each layer. Specifically, the sparse kernel K_l in layer l ($l = 1,\dots,L$) is defined by N_l offset-weight samples, $(\mathbf{o}_i, w_i)_{i=1}^{N_l}$. The convolution with such a sparse kernel is a weighted sum of values sampled at these offsets. For a pixel position $\mathbf{p}$, the operation is:

$$(I * K_l)[\mathbf{p}] = \sum_{i=1}^{N_l} w_i \cdot I(\mathbf{p}+\mathbf{o}_i), \quad (2)$$

where $I(\mathbf{p}+\mathbf{o}_i)$ denotes a sample retrieved using bilinear interpolation, with its offset $\mathbf{o}_i$.

The complete operation, consisting of L such layers with kernels $(K_1, K_2, ..., K_L)$, can be expressed as a nested convolution:

$$I_{\text{out}} = (...((I_{\text{in}} * K_1) * K_2) * ... * K_L). \quad (3)$$

This multi-layer filter reduces the number of sampling points per pixel from $(2k+1)^2$ to $\sum_{l=1}^{L} N_l$.

Impulse Response Supervision. To capture the intrinsic characteristics of a filter and prevent overfitting to specific datasets, the optimization of Multi-Layer Sparse Kernels [Schuster et al. 2020; Wu et al. 2026] is typically supervised using a single impulse signal. To obtain the equivalent kernel K, we feed a discrete Dirac delta function δ, which is an image with a single unit-valued pixel at the center, into the filtering pipeline $\mathcal{F}$. The resulting output image is exactly K. By Linear Shift-Invariant (LSI) theory [Willems and Polderman 1997], the impulse response of $\mathcal{F}$ uniquely determines its output for any input I. This is formalized as:

$$\mathcal{F}(\delta) = K \iff \forall I, \mathcal{F}(I) = I * K. \quad (4)$$

4 METHODOLOGY

4.1 Overview

Our method provides a high-fidelity, high-performance approximation for large, dense kernels. We design a two-stage methodology: an offline optimization stage and a real-time filtering stage.

At the offline stage, we introduce a differentiable optimization framework for sparse kernels with multi-resolution. Its core principle is to supervise pixels in polyphase via poly impulse response. We define a dataset-free loss function that measures the disparity with respect to the target kernel with a shift, and we minimize it using gradient descent. This process yields a set of pre-optimized weight and offset parameters.

At runtime, our downsampler and upsampler consist of *Spatial-to-Depth Transformation* and *Asymmetric Intensity-Decoupled Sampler*, respectively, to achieve a large kernel at low resolution. *Asymmetric Intensity-Decoupled Sampler* asymmetrically transforms the input RGB image into a compact half-resolution image. For the intensity channel with more high-frequency details, use the *Spatial-to-Depth Transformation* to compact image pixels for high quality. The compact pixels are filtered using pre-optimized parameters in Polyphase Filtering to achieve multi-resolution filtering. For spatially varying (SV) filtering, our *Layer-Adaptive Filtering* uses layer selection to efficiently compose kernels with different layer counts.

4.2 Differentiable Polyphase Filtering

Our polyphase filtering framework employs a symmetric architecture consisting of L downsampling and L upsampling layers with a factor of 2, indexed by $\{1,\dots,2L\}$. In the l-th layer, it consists of N_l sample points, and we optimise the offset and weight of each sample. We group layers by their output spatial resolution into hierarchy levels. Our pipeline's input image level is 0, and the resolution is R. The output image levels in downsampling are $d = 1, 2, \dots, L$, and the output image levels in upsampling are $d = L-1, L-2, \dots, 0$. For the l-th layer, our structure reduces the complexity from $O(N_l R)$ to $O(N_l R/4^d)$ compared to previous work [Schuster et al. 2020; Wu et al. 2026], minimizing the compute footprint and achieving significant speedups.

Because upsampling breaks linear shift invariance (LSI) [Willems and Polderman 1997], turning the pipeline into a linear periodically shift-variant (LPSV) system [Crochiere and Rabiner 1981], which is equivalent to a collection of distinct phase LSI systems. This prevents a single impulse-response supervision [Schuster et al. 2020; Wu et al. 2026] for impulse a LSI to supervise our filtering system. We observe

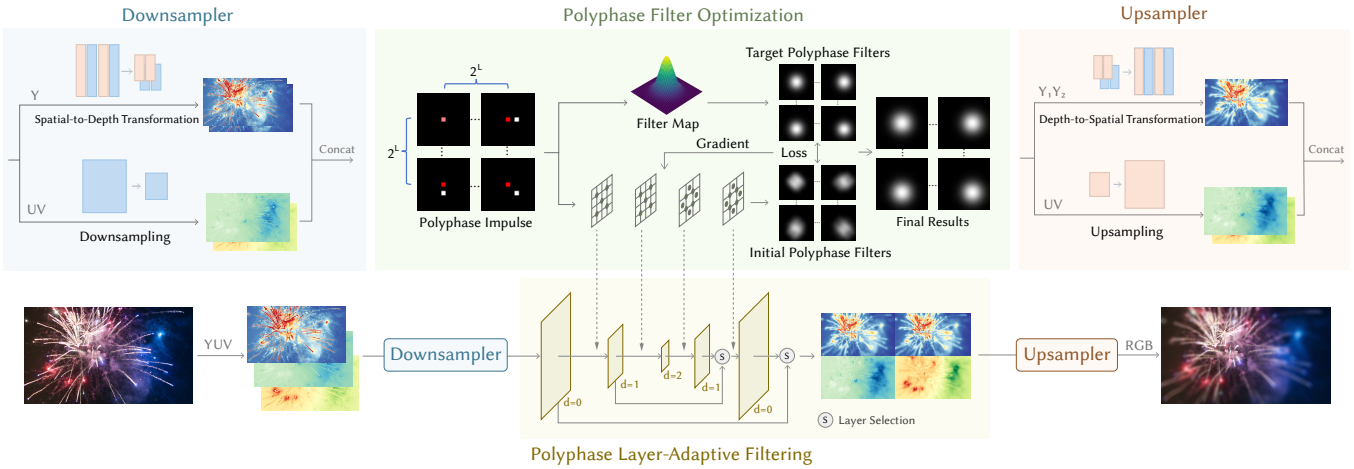


Fig. 2. **An illustration of our Differentiable Polyphase Filtering pipeline.** In the *Polyphase Filter Optimization*, we optimize the parameters of the *Polyphase Filter* with poly impulses and target kernel shifts to ensure pixel results in polyphase. *Downsampler* and *Upsampler* consist of *Spatial-to-Depth Transformation* and *Asymmetric Intensity-Decoupled Sampler* compact pixels into *Polyphase Filtering*. *Polyphase Filtering* uses the optimized parameters of sparse kernels at multiple resolutions. *Layer-Adaptive Filtering Layer* in *Polyphase Filtering* selects layers to preserve details for small kernels.

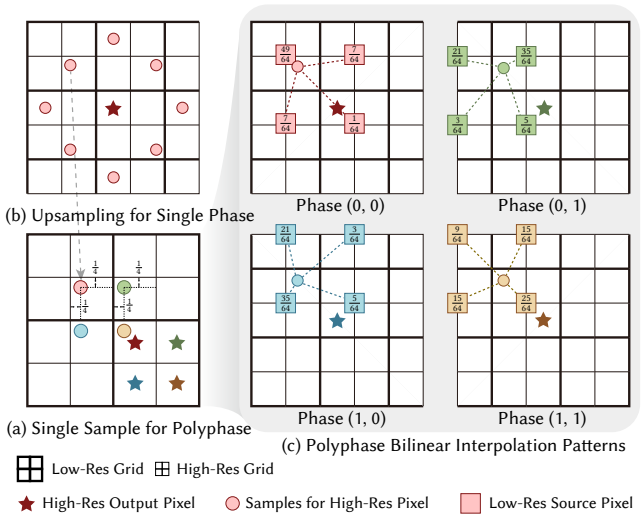


Fig. 3. **Distinct weights and offsets pattern for 2×2 high-res output pixels in upsampling.** Four pixels (stars in (a)) in a 2×2 grid when upsampling have the same set of sample points (circles in (b)). A certain sample (circles in (a)) from sample points for each pixel (stars in (a)) in a 2×2 grid has distinct bilinear interpolation patterns (c), resulting in the weights and offsets contributing to the high-res output pixels in a 2×2 grid being different.

our pipeline with a $P = 2^L$ period on each axis, so we propose to supervise with $P \times P$ impulses in distinct phases.

Specifically, in a single upsampling layer (Fig. 3), every single sample from kernels of a 2×2 high-resolution grid is computed from the same set of four source low-resolution pixels. As a result, the four high-resolution output pixels in that 2×2 block share the same source pixels, but use different weights and sampling offsets. This pattern repeats with a spatial period of 2×2.

For multi-upsampling layers, this recursive composition results in a cumulative period of $P = 2^L$ along each pixel axis. Consequently, for a pixel $\mathbf{p} = (x, y)$, the source low-resolution pixels' weights and offsets depend on its phase index $\phi(\mathbf{p}) = (x \bmod P, y \bmod P)$. The pixels in the same ϕ should be supervised by an impulse response of a specific phase.

We construct a *Polyphase Impulse* that includes all unique phase shifts as inputs. Let $\delta_{\mathbf{s}}$ denote the P^2 discrete Dirac delta functions, which form a family of shifted impulse images. Each $\delta_{\mathbf{s}}$ corresponds to a phase shift $\mathbf{s} = (u, v)$ with $0 \le u, v < P$, and is defined as:

$$\delta_{\mathbf{s}}[x, y] = \begin{cases} 1 & \text{if } (x, y) = (x_c + u, y_c + v) \\ 0 & \text{otherwise} \end{cases}, \tag{5}$$

where (x_c, y_c) is the image center. We feed these polyphase impulses into our filtering pipeline $\mathcal{F}$. And we apply multiple shift operations, $\mathcal{S}$, with shifts s on the reference to obtain a set of shifted reference images. The optimization goal is to minimize the discrepancy between the generated responses and the target kernel K_{target}:

$$\mathcal{L}_{\text{poly}} = \sum_{\mathbf{s} \in [0, P-1]^2} \left\| \mathcal{F}(\delta_{\mathbf{s}}) - \mathcal{S}(K_{\text{target}}, \mathbf{s}) \right\|_2^2. \tag{6}$$

4.3 Spatial-to-Depth Transformation for Neighbor Reuse

Leveraging the local spatial smoothness of large kernels, we propose a Spatial-to-Depth Transformation to accelerate computation by efficiently reusing information from neighboring samples.

Formally, we realize the Spatial-to-Depth Transformation via the Pixel Unshuffle $\mathcal{P}_u(\cdot)$ and Pixel Shuffle $\mathcal{P}_s(\cdot)$ operators [Shi et al. 2016]. For an input I, we employ these operators to enclose the Polyphase Filter $\mathcal{F}(\cdot)$, as formulated in Sec. 4.2. Subsequently, to achieve neighbor reuse, we append a lightweight blending operation

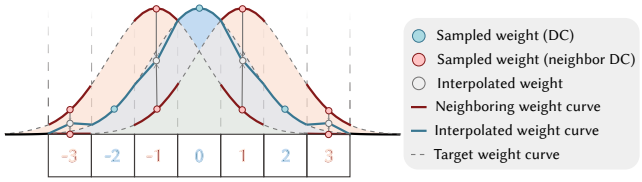


Fig. 4. **Interpolated weights at position 0.** The full convolutional weights centered at position 0 are shown by the blue curve. Replacing it with a dilated convolution samples only the locations $-2, 0, 2$. For the remaining positions $-3, -1, 1, 3$, we approximate their contributions by interpolating from the result of neighboring dilated convolution, reducing the number of sampled points by nearly half.

$\mathcal{B}(\cdot)$. The efficient pipeline $\mathcal{F}_e$ can be expressed as:

$$\mathcal{F}_e(I) = \mathcal{B}\left(\mathcal{P}s\left(\mathcal{F}\left(\mathcal{P}u(I)\right)\right)\right), \tag{7}$$

Through this process, F_e achieves a target kernel size twice that of $\mathcal{F}$, significantly reducing computational costs via the Spatial-to-Depth Transformation. Mathematically, the term $\mathcal{P}s\left(\mathcal{F}\left(\mathcal{P}u(I)\right)\right)$ is equivalent to a dilated convolution, enabling $\mathcal{F}$ to represent any dilated convolution kernel.

Furthermore, a target dense convolution $\mathcal{C}$ can be approximated by decomposing it into an equivalent-sized dilated convolution $\mathcal{D}$ followed by neighbor blending $\mathcal{B}$, expressed as $\hat{\mathcal{C}} = \mathcal{B}\mathcal{D}$. As shown in Fig. 4, the dilated convolution $\mathcal{D}(x)$ samples only half of the positions (blue 1D coordinates at the bottom) and inherits the corresponding weights from the dense target kernel (blue points on the curve at the sampled locations). We reuse the neighboring dilated convolutions, $\mathcal{D}(x-1)$ and $\mathcal{D}(x+1)$, shifted by one pixel, to recover the positions that are missing in the current dilated convolution. Specifically, for each unsampled location (gray), we interpolate its weight using the two corresponding samples from the neighboring dilated convolutions (red).

For an unsampled weight $W(x)$ with $x \in \{-3, -1, 1, 3\}$, the neighboring dilated convolutions provide exactly the two adjacent weights of the dense kernel at x, namely $W(x-1)$ and $W(x+1)$. We reconstruct $W(x)$ by linearly interpolating these adjacent values:

$$\hat{W}(x) = \frac{1}{2}\left[W(x-1) + W(x+1)\right]. \tag{8}$$

To quantify the reconstruction accuracy, we expand the neighbor terms $W(x \pm 1)$ using a Taylor series around the center x. The interpolation error ϵ is derived as:

$$\epsilon = \left|W(x) - \hat{W}(x)\right| \approx \frac{1}{2}\left|W''(x)\right| \cdot T^2, \tag{9}$$

where T denotes the sampling interval. This shows that the reconstruction becomes highly accurate when the kernel has a small second-order derivative, which is common in practice because large-kernel are typically smooth.

Consequently, the full dense convolution $C(x)$ can be decomposed into the dilated convolution $\mathcal{D}(x)$ and a residual term $C_{\text{diff}}(x)$. Based on the neighbor approximation, we estimate the residual as:

$$\begin{aligned} C(x) &= \mathcal{D}(x) + C_{\text{diff}}(x), \\ C_{\text{diff}}(x) &\approx \frac{1}{2}\left[\mathcal{D}(x-1) + \mathcal{D}(x+1)\right]. \end{aligned} \tag{10}$$

and $\mathcal{D} \approx \mathcal{P}s\left(\mathcal{F}\left(\mathcal{P}u(I)\right)\right)$.

4.4 Asymmetric Intensity-Decoupled Sampler

Applying our *Spatial-to-Depth Transformation* Section 4.3 directly to full RGB data would cause a channel explosion (resulting in 12 channels after unshuffling), exceeding the standard 4-channel texture format (RGBA) optimized by GPU hardware.

To balance high-fidelity filtering with tight bandwidth constraints, we propose a decomposition strategy that redistributes information according to perceptual sensitivity and hardware parallelism.

First, we transform the input RGB image into one intensity channel and two chroma channels: $Y = \frac{R+G+B}{3}$, $U = R - Y$, and $V = B - Y$. This separation concentrates most structural detail in Y while leaving U and V comparatively smooth. For the intensity channel, we prioritize fidelity toward the horizontal direction, as the human visual system (HVS) is often more sensitive to horizontal patterns. We therefore apply horizontal pixel unshuffling followed by vertical downsampling, so Y can be carried efficiently via our *Spatial-to-Depth Transformation*.

In parallel, we bilinearly downsample U and V to the same target resolution (0.25× area). Finally, we concatenate these components—two from intensity and two from chroma—into a single compact RGBA texture, which serves as the input for the *Polyphase Filter*.

4.5 Layer-Adaptive Filtering with Variable Kernel Sizes

Spatially varying (SV) filtering is pivotal for achieving expressive visual effects by applying non-uniform kernels across the image domain. Typically, we determine a kernel for each pixel based on a parameter defined in the image's spatial domain. However, spatially varying (SV) filtering is often too expensive in practice. It demands heavy memory bandwidth to fetch per-pixel weights, and its runtime is set by the worst case—pixels that require the largest kernels.

Multi-resolution processing can greatly accelerate SV filtering, but it introduces a key constraint: the chosen layer count sets a lower bound on the kernel sizes the model can represent. Using a uniformly high layer count causes over-blurring for small kernels. Conversely, using a uniformly low layer count to preserve small-kernel detail shrinks the effective receptive field, making it impossible to fit large kernels. To resolve this dilemma, we propose *Layer-Adaptive Filtering*, which adaptively selects an appropriate layer count for each target kernel size and executes kernels with different layer counts within a single SV filtering task.

We initially approximate kernels in parallel using two adjacent levels, L and $L+1$, using the *Polyphase Filtering* described in Sec. 4.2 to determine the optimal layer count for varying target kernel sizes. For small kernels, we choose the smallest level L that satisfies the layer count sufficiency criterion, $\mathcal{L}_L \leq \mathcal{L}_{L+1}$, where $\mathcal{L}_L$ denotes the optimized value of the polyphase loss $\mathcal{L}_{\text{poly}}$ (as defined in Eq. 6) for a pipeline with L layers. We then adopt the optimized result at level L. As the target kernel size increases, the layer-count sufficiency criterion at level L may no longer hold; a crossover point emerges where $\mathcal{L}_L > \mathcal{L}_{L+1}$.

Consequently, we increment the baseline level and compare $L+1$ against $L+2$, selecting $L+1$ as the optimal level for the current

kernel size range. Repeating this procedure constructs a look-up table (LUT) that determines target kernel sizes for the corresponding kernel parameters at the selected decomposition level.

During SV filtering, we use the LUT to map each kernel size in the parameter map to its optimal level, producing a level map $\mathcal{M}(x, y)$. We denote $\max(\mathcal{M})$ as $L_{\max}$. The SV pipeline then performs $L_{\max}$ downsampling and upsampling steps to match the largest kernel.

At upsampling level L_u, we select the input texture for level $L_u + 1$ based on $\mathcal{M}(x, y)$ (Fig. 2). Pixels with $\mathcal{M}(x, y) \leq L_u + 1$ skip deeper processing and take the output from the matching downsampling stage $\mathcal{D}$; otherwise, they take the output from the previous upsampling stage $\mathcal{U}$, enabling a single unified flow across levels. The selected output S is

$$S_{L_u+1}(x, y) = \begin{cases} \mathbf{D}_{L_u+1}(x, y), & \mathcal{M}(x, y) \leq L_u + 1, \\ \mathbf{U}_{L_u}(x, y), & \mathcal{M}(x, y) > L_u + 1\,, \end{cases} \tag{11}$$

Although our method introduces branching, it remains efficient because SV kernel sizes vary smoothly in space (the parameter map is low-frequency). As a result, neighboring pixels within the same GPU warp typically select the same level, so branches stay largely coherent. This coherence limits warp divergence and avoids the instruction-serialization overhead that often accompanies dynamic control flow.

5 EXPERIMENTS

5.1 Setup

Baselines. We compare our method against several baselines. For Gaussian kernels, we include specialized Gaussian approximations such as the Multi-Box method [Kovesi 2010], Kawase blur [Kawase 2003], and Dual Filtering [Martin et al. 2015]. Additionally, we compare against two optimization-based methods that employ multi-pass, fixed-resolution sparse filters: Parallel Simulated Tempering (PST) [Schuster et al. 2020] and Differentiable Sparse Kernel Complex (DSKC) [Wu et al. 2026]. For bokeh kernels and spatially varying kernels, since Multi-Box, Kawase, and Dual Filtering are strictly designed for isotropic Gaussian distributions, we limit our comparison to the general-purpose optimization frameworks, PST and DSKC.

Kernels and Datasets. Our evaluation suite is designed to probe distinct aspects of approximation capabilities. We utilize the classical Gaussian kernel to benchmark the system's ability to efficiently fit fundamental dense operators. Furthermore, we employ a variety of geometric bokeh kernels to assess shape fidelity. These kernels challenge the system to capture distinct morphological features and high-frequency details within a limited computational budget. To guarantee realistic evaluation, all filtering tasks are performed on 1080p real-world photographs.

Implementation and Evaluation Metric. Our offline optimization was performed on an RTX 3090 GPU using PyTorch [Paszke et al. 2019]. Performance measurements were conducted separately on two platforms: a mobile platform equipped with the Snapdragon 8 Gen 3 processor and a PC platform using OpenGL. We report the inference times for the mobile platform in the main text figures, while comprehensive results for both platforms are provided in the supplementary material. We employ the Adam optimizer for all kernel and layer configurations. For Gaussian kernels, the learning rates are set to 0.1, 0.5, 1.0, and 1.5 for configurations ranging from level 1 to 4, respectively; the parameter σ spans from 0.1 to 30 with a step size of 0.1. For geometric kernels, the learning rates for level 1 to 3 are 0.1, 0.5, and 1.5, respectively, with kernel sizes ranging from 1 to 109 pixels at an interval of 2. Our method requires only 4,000 optimization steps per kernel and typically converges within 20 seconds. In comparison, the PST algorithm is executed for 15,000 iterations with 10 parallel candidates (totaling 150,000 steps), and DSKC is run for 15,000 iterations. The initialization scheme follows the approach described in [Wu et al. 2026].

For Gaussian kernel approximation visualization, we utilize the Covariance Frobenius Norm to quantify the discrepancy in kernel diffusion distribution. For all final filtered images in the main text figures, we evaluate visual fidelity using the peak signal-to-noise ratio (PSNR). Furthermore, comprehensive results for all experiments—including PSNR, structural similarity (SSIM), and learned perceptual image patch similarity (LPIPS) [Zhang et al. 2018]—are provided in the supplementary material.

5.2 Gaussian Kernel

To rigorously evaluate the deviations from the ideal distribution of the approximated kernels, we deploy them in a bloom post-processing pipeline. Formally, the bloom effect I_{bloom} is synthesized by compositing the original image I_{in} with a blurred version of its high-intensity components:

$$I_{bloom} = I_{in} + \alpha \cdot (K * \mathcal{T}(I_{in})), \tag{12}$$

where K denotes the approximated Gaussian kernel, α is the intensity scaling factor, and $\mathcal{T}(\cdot)$ represents a thresholding operator that extracts highlights. The visual quality of the bloom relies heavily on the diffusion distribution of light sources.

As shown in Fig. 7, we compare our method against several baselines. Our approach utilizes our proposed *Intensity-Decoupled Polyphase Filtering* with a 3-level hierarchy (comprising 3 downsampling and 3 upsampling stages). Multi-Box approximates a Gaussian with three sequential large-radius box blurs, but runs poorly. Kawase, PST, and DSKC use a 6×4 configuration (6 passes, 4 samples each); even with nearly 2× our inference time, their effective kernels remain non-smooth and anisotropic, producing visible blocky artifacts. Dual Filtering with four multi-resolution levels offers the second-best quality–speed trade-off, but without parameter optimization it cannot faithfully reproduce the peak intensity at the light center or the edge diffusion. We visualize these behaviors by plotting iso-lines of the effective kernels, where our method most closely matches the ground-truth Gaussian.

5.3 Geometric Kernel

To demonstrate the capability of reproducing complex geometric shapes, we employ a bokeh rendering application that combines highlight filtering with background defocus. Unlike the smooth Gaussian kernel, geometric kernels serve as a strict test for high-frequency edge preservation and topological correctness.

As illustrated in Fig. 8, we compare our method (configured with a 2-level hierarchy (2 downsampling and 2 upsampling stages)) against the optimization-based baselines, PST and DSKC (5 passes

with 4 samples each). Geometric kernels pose a significantly greater challenge than isotropic Gaussians. Despite the significantly higher computational overhead compared to our method, baseline methods struggle to recover the kernel structure and details. For Circle, baselines break isotropy; for Star4, they miss concave details and sharp tips; for Ring, only our method preserves the hollow topology. Under the same tight computational budget, they fail to reconstruct smooth target kernels. Matching smoothness with a fixed-resolution pipeline would require much denser sampling, making them inefficient for real-time use compared with our multi-resolution framework. As discussed in Sec. 4.3, discontinuities in the kernel function degrade the kernel approximation accuracy; specifically, our method's PSNR values for Ring and Star4 are approximately 9.5% lower than those of Circle and Side4.

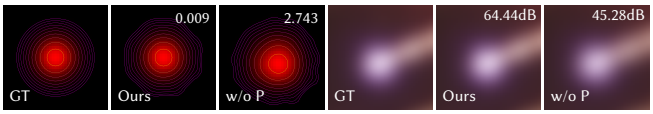


Fig. 5. **Ablation study of the proposed supervision strategy.** We compare the approximated kernels and results of the variant without polyphase-filter-based supervision ($w/o\ P$). Visualizations include iso-contours, PSNR (↑) and the Covariance Frobenius Norm (↓), which quantifies the kernel diffusion distribution.

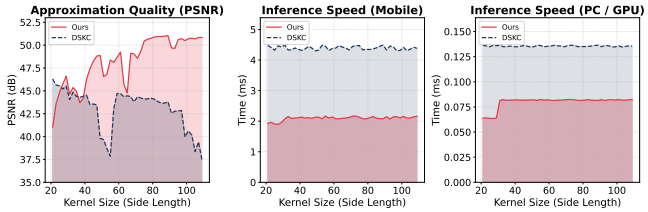


Fig. 6. **Comparison of quality and performance** between our method and the state-of-the-art DSKC [Wu et al. 2026]. We evaluate approximation quality and report the inference performance on both mobile and PC platforms.

5.4 Spatially Varying Kernel

To evaluate the performance of spatially varying (SV) filtering, we utilize a generated depth map to modulate the kernel size per pixel. We apply this varying filter across four distinct geometric kernel types and composite the results for visualization.

As illustrated in Fig. 10, our method leverages our proposed Layer-Adaptive Filtering with Variable Kernel Sizes proposed in 4.5 to dynamically blend kernels from 1-level and 2-level downsampling hierarchies. In contrast, PST and DSKC rely on a static, heavy 5×4 setup for all pixels. The comparison reveals significant disparities: PST exhibits severe instability, resulting in noise-like artifacts across the image. DSKC struggles to optimize for large-radius, leading to blocky artifacts. In contrast, as shown in the macro view (left), our method maintains high fidelity as the kernel size increases. Crucially, it achieves seamless transitions, exhibiting no artifacts or visible popping boundaries at the switching points between different hierarchical levels. Furthermore, as demonstrated in Fig. 11 and Fig. 12, our method remains robust even when handling anisotropic kernels. Finally, regarding liquid and scratched glass effects (Fig. 13), our approach accurately simulates complex physical lighting phenomena.

5.5 Quality and Performance Analysis

As shown in Fig. 6, we compared the performance and quality across different platforms (mobile and PC) as the kernel size increased from small to large. We reported average results of the geometric kernels. Regarding performance, at a kernel size of 30, we observed a degradation when moving from level 1 to 2. In terms of quality, our method generally improves as the kernel size increases, whereas DSKC exhibits the opposite trend, with quality decreasing as the kernel grows larger. The region where the quality of the two methods intersects is between kernel sizes 25 and 40. This phenomenon arises because, below this intersection threshold, a single-level pyramid provides insufficient sampling and an inadequate receptive field, while a two-level pyramid introduces excessive blurring due to the downsampling process.

5.6 Ablation Studies

5.6.1 Supervision strategy. Fig. 5 compares the kernel optimization results by ablating the proposed Polyphase supervision ($w/o\ P$). Without P, the optimized pixels in non-target phases cause the equivalent kernels to exhibit center misalignment and asymmetry.

Table 1. **Ablation study of Spatial-to-Depth Transformation for Neighbor Reuse (T) and Asymmetric Intensity-Decoupled Sampler (Y) across various target kernels.** The approximation quality of Gaussian, Side4, and Ring kernels is evaluated using PSNR (dB), while performance is measured by latency (ms) on both mobile and PC platforms.

Filter	Gaussian		Side4		Ring	
Variant	PSNR ↑	Perf. ↓	PSNR ↑	Perf. ↓	PSNR ↑	Perf. ↓
Ours	**73.45**	**2.21/0.09**	**50.22**	**2.12/0.08**	**49.07**	**2.12/0.08**
$w/o\ T$	52.71	2.45/0.10	47.99	2.29/0.10	39.43	2.29/0.10
$w/o\ Y$	72.61	2.21/0.09	49.92	2.12/0.08	48.95	2.12/0.08

5.6.2 Intensity-Decoupled Dilated Convolution. Table 1 compares the impacts of the Spatial-to-Depth Transformation (T) and Asymmetric Intensity-Decoupled Sampler (Y) components. The results indicate that $w/o\ T$ degrades both quality and efficiency. Furthermore, the intensity-decoupling mechanism provides fidelity gains at negligible computational cost.

5.6.3 Layer-Adaptive Filtering. We compare our proposed method, which dynamically adjusts filter layers against two variants with spatially-uniform layer levels. As illustrated in Fig. 9, the shallow-layer variant fails to approximate large kernels, resulting in noticeable visual artifacts. Conversely, the deep-layer variant leads to over-smoothing in regions with small kernels. Notably, our strategy effectively integrates these multi-depth kernels without incurring any additional computational latency.

6 DISCUSSION AND CONCLUSION

In this paper, we presented a novel Differentiable Polyphase Filtering framework designed for efficient large-kernel approximation. By leveraging our Polyphase Impulse Optimization, we effectively guarantee high fidelity throughout the multi-resolution multi-layer kernel filtering process. To further accelerate inference, we introduced a Spatial-to-Depth Transformation and an Asymmetric Intensity Decoupled Sampler to compact pixel representations and perform asymmetric processing for optimal quality. Additionally, the proposed Layer-Adaptive Filtering extends our method's capabilities to handle spatially varying (SV) filtering tasks.

Extensive experiments demonstrate that our approach achieves significant performance gains, outperforming state-of-the-art methods (e.g., DSKC) with speedups of up to 2× for single-kernel tasks and 2.5× for SV filtering. By effectively resolving the computational bottlenecks of large-kernel filtering, our method proves highly suitable for real-time applications ranging from mobile user interfaces to professional visual effects. It enables the efficient rendering of striking visual phenomena, including bloom, bokeh, and complex motion blurs (Gaussian, radial, or rotational), marking a substantial step forward for real-time rendering systems.

Acknowledgments

This work was partially supported by National Key R&D Program of China (No.2025YFG0101300), NSFC (No. 52532013), and Key R&D Program of Zhejiang (No: 2025C01064). The authors thank the anonymous reviewers for their valuable feedback.

References

Martin Balint, Krzysztof Wolski, Karol Myszkowski, Hans-Peter Seidel, and Rafał Mantiuk. 2023. Neural partitioning pyramids for denoising monte carlo renderings. In *ACM SIGGRAPH 2023 conference proceedings*. 1–11.

Peter J Burt. 1981. Fast filter transform for image processing. *Computer graphics and image processing* 16, 1 (1981), 20–51.

Yunpeng Chen, Haoqi Fan, Bing Xu, Zhicheng Yan, Yannis Kalantidis, Marcus Rohrbach, Shuicheng Yan, and Jiashi Feng. 2019. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution. In *Proceedings of the IEEE/CVF international conference on computer vision*. 3435–3444.

Ronald E Crochiere and Lawrence R Rabiner. 1981. Interpolation and decimation of digital signals—A tutorial review. *Proc. IEEE* 69, 3 (1981), 300–331.

Xiaohan Ding, Xiangyu Zhang, Jungong Han, and Guiguang Ding. 2022. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 11963–11975.

Alex Ferrier and Christina Coffin. 2011. Deferred shading techniques using frostbite in" Battlefield 3" and" Need for Speed the Run". In *ACM SIGGRAPH 2011 Talks*. 1–1.

Eduardo SL Gastal and Manuel M Oliveira. 2011. Domain transform for edge-aware image and video processing. In *ACM SIGGRAPH 2011 papers*. 1–12.

Eduardo SL Gastal and Manuel M Oliveira. 2012. Adaptive manifolds for real-time high-dimensional filtering. *ACM Transactions on Graphics (TOG)* 31, 4 (2012), 1–13.

Michaël Gharbi, Tzu-Mao Li, Miika Aittala, Jaakko Lehtinen, and Frédo Durand. 2019. Sample-based Monte Carlo denoising using a kernel-splatting network. *ACM Transactions on Graphics (ToG)* 38, 4 (2019), 1–12.

Jean-Philippe Guertin, Morgan McGuire, and Derek Nowrouzezahrai. 2014. A Fast and Stable Feature-Aware Motion Blur Filter.. In *High performance graphics*. 51–60.

Kaiming He, Jian Sun, and Xiaoou Tang. 2012. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence* 35, 6 (2012), 1397–1409.

Intel Corporation. 2003. Improve Real-Time GPU-Based Image Blur Algorithms: Kawase Blur and Moving Box Averages. Intel Developer video web page. https://www.intel.com/content/www/us/en/developer/videos/improving-real-time-gpu-based-image-blur-algorithms-kawase-blur-and-moving-box-averages.html Accessed: 2026-01-19.

Takuhiro Kaneko. 2021. Unsupervised learning of depth and depth-of-field effect from natural images with aperture rendering generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 15679–15688.

Masaki Kawase. 2003. Frame buffer postprocessing effects in double-steal (wrechless). In *Game Developers Conference 2003, 3*.

Peter Kovesi. 2010. Fast almost-gaussian filtering. In *2010 International conference on Digital image computing: Techniques and applications*. IEEE, 121–125.

Florian Kriener, Thomas Binder, and Manuel Wille. 2013. Accelerating defocus blur magnification. In *Multimedia Content and Mobile Devices*, Vol. 8667. SPIE, 517–527.

Andrew Lavin and Scott Gray. 2016. Fast algorithms for convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4013–4021.

Sungkil Lee, Elmar Eisemann, and Hans-Peter Seidel. 2009. Depth-of-field rendering with multiview synthesis. *ACM Transactions on Graphics (TOG)* 28, 5 (2009), 1–6.

Thomas Leimkühler, Hans-Peter Seidel, and Tobias Ritschel. 2018. Laplacian kernel splatting for efficient depth-of-field and motion blur synthesis or reconstruction. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–11.

Sam Martin, Andrew Garrard, Andrew Gruber, Marius Bjorge, Renaldas Zioma, Simon Benge, and Niklas Nummelin. 2015. Moving mobile graphics. In *ACM SIGGRAPH 2015 Courses* (Los Angeles, California) *(SIGGRAPH '15)*. Association for Computing Machinery, New York, NY, USA, Article 18. doi:10.1145/2776880.2787664

Tim McGraw. 2015. Fast Bokeh effects using low-rank linear filters. *The Visual Computer* 31, 5 (2015), 601–611.

L McIntosh, Bernhard E Riecke, and Steve DiPaola. 2012. Efficiently Simulating the Bokeh of Polygonal Apertures in a Post-Process Depth of Field Shader. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 1810–1822.

Johannes Moersch and Howard J Hamilton. 2020. Variable-sized, circular bokeh depth of field effects. In *Graphics Interface 2014*. AK Peters/CRC Press, 103–107.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

Fatih Porikli. 2008. Constant time O (1) bilateral filtering. In *2008 IEEE conference on computer vision and pattern recognition*. IEEE, 1–8.

Kersten Schuster, Philip Trettner, and Leif Kobbelt. 2020. High-performance image filters via sparse approximations. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2 (2020), 1–19.

Tim Seizinger, Marcos V Conde, Manuel Kolmet, Tom E Bishop, and Radu Timofte. 2023. Efficient multi-lens bokeh effect rendering and transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 1633–1642.

Kai Selgrad, Christian Reintges, Dominik Penk, Pascal Wagner, and Marc Stamminger. 2015. Real-time depth of field using multi-layer filtering. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games*. 121–127.

Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. 2016. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1874–1883.

Lucas J Van Vliet, Ian T Young, and Piet W Verbeek. 1998. Recursive Gaussian derivative filters. In *Proceedings. Fourteenth International Conference on Pattern Recognition (Cat. No. 98EX170)*, Vol. 1. IEEE, 509–514.

Oliver Whyte, Josef Sivic, Andrew Zisserman, and Jean Ponce. 2012. Non-uniform deblurring for shaken images. *International journal of computer vision* 98, 2 (2012), 168–186.

Jan C Willems and Jan W Polderman. 1997. *Introduction to mathematical systems theory: a behavioral approach*. Vol. 26. Springer Science & Business Media.

Zhizhen Wu, Zhe Cao, and Yuchi Huo. 2026. DISK: Differentiable Sparse Kernel Complex for Efficient Spatially-Variant Convolution. In *The Fourteenth International Conference on Learning Representations*. https://openreview.net/forum?id=bbuxDoRD2D

Chengxing Xie, Xiaoming Zhang, Linze Li, Haiteng Meng, Tianlin Zhang, Tianrui Li, and Xiaole Zhao. 2023. Large kernel distillation network for efficient single image super-resolution. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 1283–1292.

Tianchen Xu, Jiale Yang, Yiming Qin, Bin Sheng, and Enhua Wu. 2024. Fast Approximation to Large-Kernel Edge-Preserving Filters by Recursive Reconstruction from Image Pyramids. (2024).

Ian T Young and Lucas J Van Vliet. 1995. Recursive implementation of the Gaussian filter. *Signal processing* 44, 2 (1995), 139–151.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 586–595.

Alois Zingl. 2020. Efficient Gaussian Blur Algorithm (Extended Binomial Filter for Fast Gaussian Blur). Web page. https://zingl.github.io/blurring.html Last update July 2020.

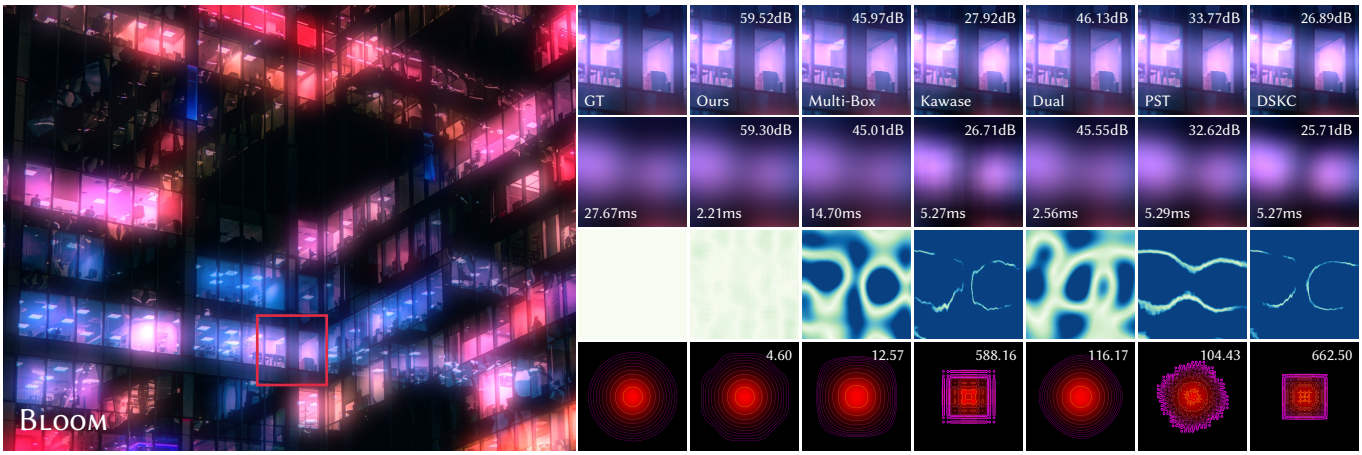


Fig. 7. **Visual comparison of bloom rendering**. We compare our method against traditional Gaussian approximations (Multi-Box [Kovesi 2010], Kawase [Kawase 2003], Dual Filtering [Martin et al. 2015]) and optimization-based sparse baselines (PST [Schuster et al. 2020], DSKC [Wu et al. 2026]). The rows display, from top to bottom: **(1)** Final composited bloom results; **(2)** Filtered highlights annotated with runtime latency (in ms); **(3)** Error maps relative to the ground truth, where deeper blue indicates larger error; **(4)** Kernel approximation visualizations with iso-contours, labeled with the Covariance Frobenius Norm (↓) to quantify the kernel diffusion distribution.

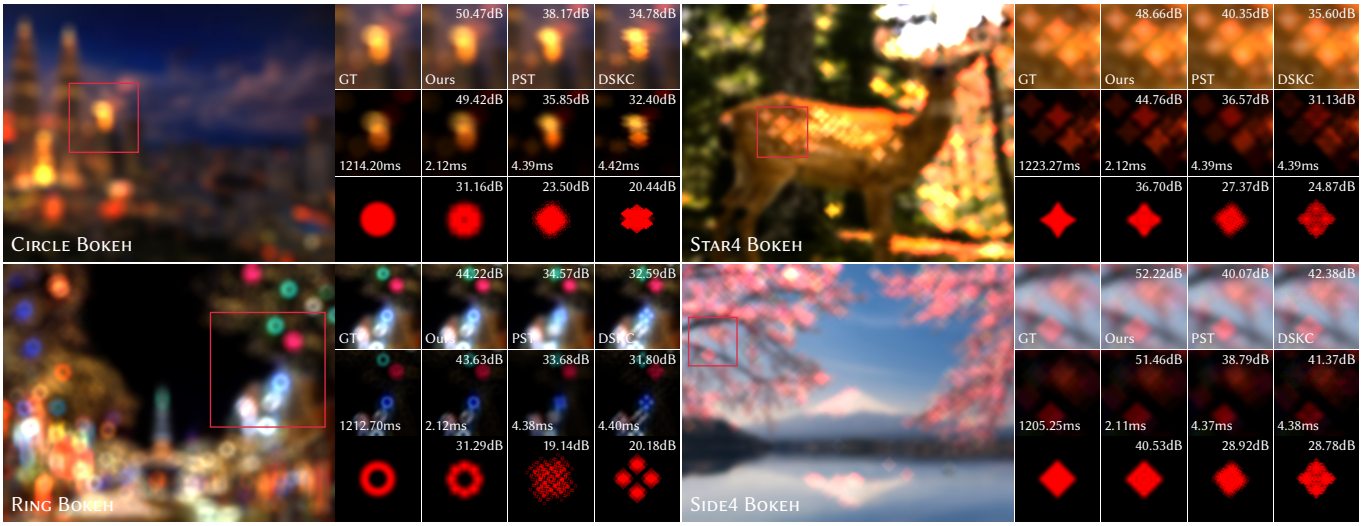


Fig. 8. **Visual comparison of bokeh rendering**. We compare our method against optimization-based sparse baselines (PST [Schuster et al. 2020], DSKC [Wu et al. 2026]). The rows display, from top to bottom: **(1)** Final composited bokeh results; **(2)** Filtered highlights annotated with runtime latency (in ms); **(3)** Kernel approximation visualizations, labeled with PSNR to measure kernel structure and details.



Fig. 9. **Visual comparison between fixed-level variants and our dynamic layer-depth strategy in spatially-varying filtering.** Low-level and high-level configurations produce artifacts (for large kernels) and over-smoothing (for small kernels), respectively, whereas our approach maintains high fidelity across all scales.

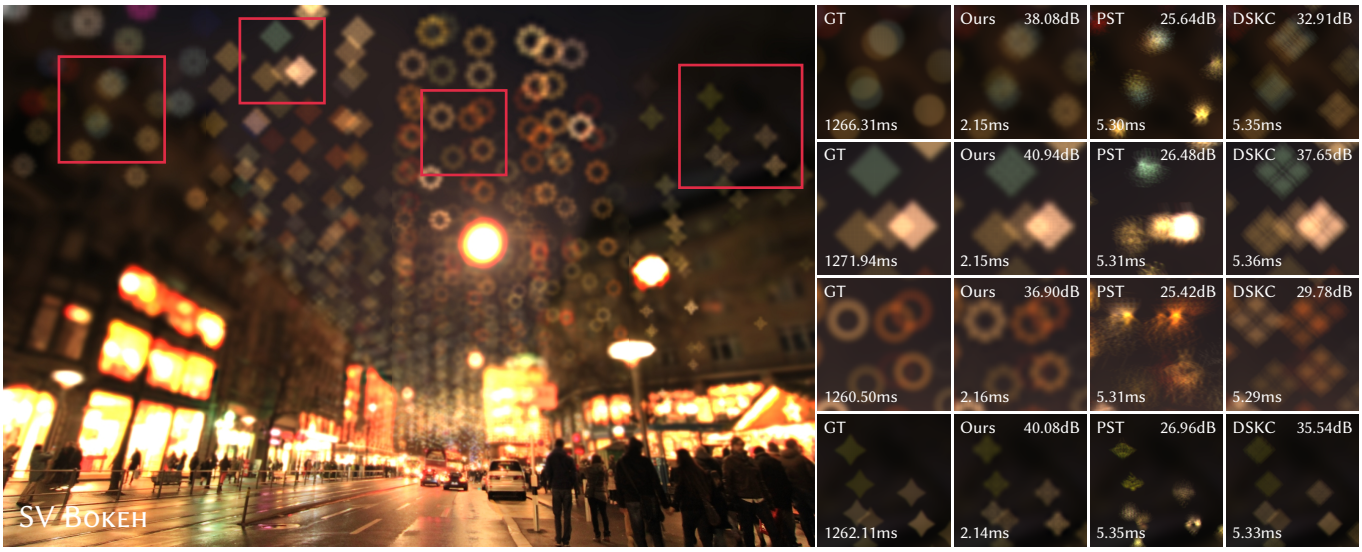


Fig. 10. **Visual comparison of spatially varying bokeh rendering**. We compare our method against optimization-based sparse baselines (PST [Schuster et al. 2020], DSKC [Wu et al. 2026]). Each result is labeled with its corresponding PSNR and runtime latency (in ms).

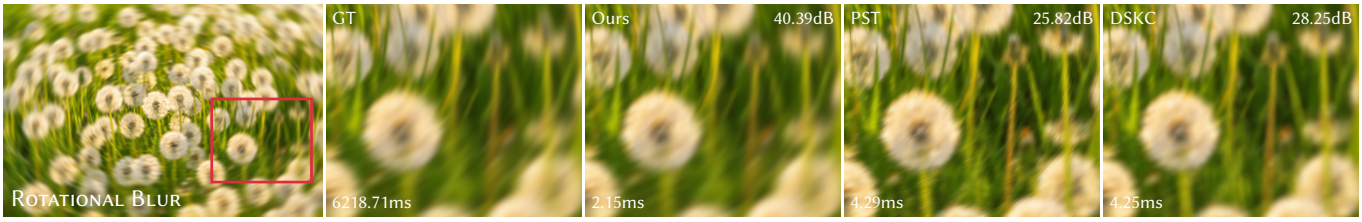


Fig. 11. **Visual comparison of rotational blur**. We compare our method against optimization-based sparse baselines (PST, DSKC).

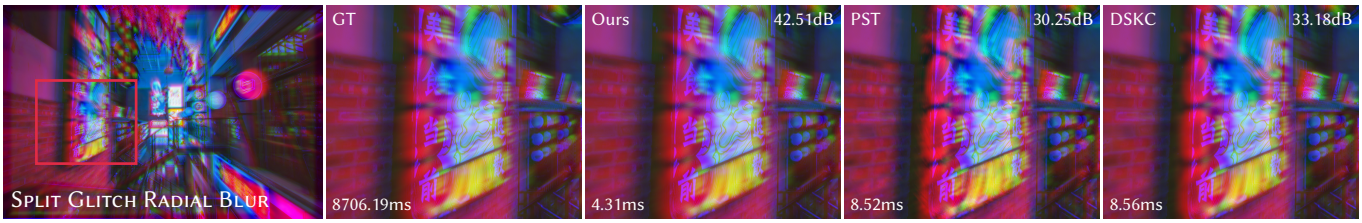


Fig. 12. **Visual comparison of radial blur**. We compare our method against optimization-based sparse baselines (PST, DSKC).

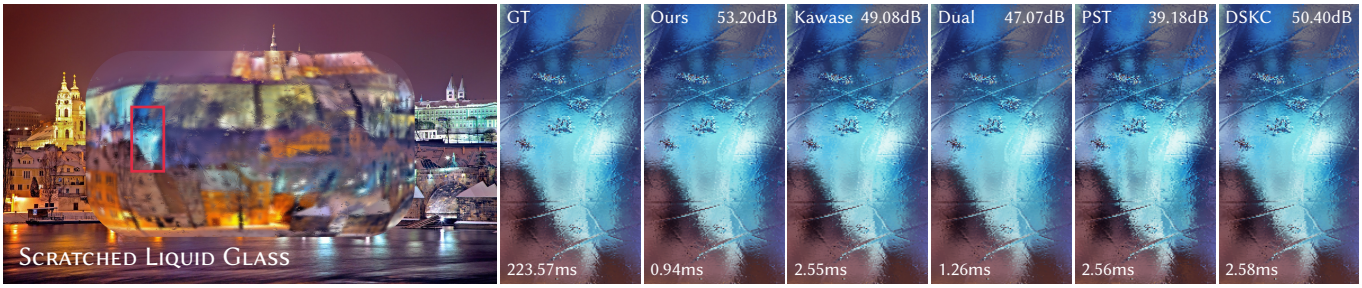


Fig. 13. **Visual comparison of spatially varying Gaussian blur**. We compare our method against optimization-based sparse baselines (PST, DSKC).