Frequency-based Bilateral Filter on Graphics Cards

Simeon Atanasov[1]

Supervisor(s): Elmar Eisemann[1]**, Mathijs Molenaar**[1]

[1]**EEMCS, Delft University of Technology, The Netherlands**

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 23, 2024

Name of the student: Simeon Atanasov
Final project course: CSE3000 Research Project
Thesis committee: Elmar Eisemann, Mathijs Molenaar, Jing Sun

Abstract

The bilateral filter is an edge-aware image filter. While it has a variety of applications, its naive implementation is quadratic in nature, hindering the ability to efficiently process multi-megapixel images. If performance is needed, like in a real-time setting, an approximation is necessary. Current literature on Fourier series-based approximations does not explore the capabilities of graphics processing units (GPUs) as viable platforms for this computational problem. This paper proposes an approach for implementing such filtering on a GPU by conducting a series of separable convolutions, and also investigates the use of different range kernels. Our adaption of the bilateral filter is found to be more two times faster than readily available solutions, with frame times showing that real-time performance is possible for large spatial kernel sizes and image resolutions.

1 Introduction

Bilateral filtering, a term coined by Tomasi and Manduci [1], is an image filter that can preserve edges. Applications vary widely — from simple noise reduction, to low-light photography [2], contrast reduction [3], tone mapping [3], creating cartoon renditions [4], etc. This is achieved through the combination of a spatial and range kernel, ensuring that a given pixel is represented by a weighted combination of pixels in a neighbourhood based on spatial and intensity proximity. One drawback of the filter is the runtime performance – the algorithm has quadratic complexity, stemming from the need to compare pixels in intensity before applying a spatial weighting. In consequence, filter accelerations like performing a separable convolution are not applicable. Specialized solutions are needed.

Several approaches have been presented to increase the speed of the bilateral filter. Durand and Dorsey [3] propose a piecewise-linear approximation with additional subsampling. Chen et al. [5] consider a data structure called a bilateral grid, on which a convolution with a 3D Gaussian kernel is performed. Finally, Gupta et al. [6] propose a Gauss-polynomial approximation of the range kernel combined with the use of OpenCL [7] for executing a GPU-based box filter, followed by post-processing done on the CPU.

The set of approaches, which is the focus of this paper, relies on estimating the range kernel of the bilateral filter using Fourier series [8–12]. They relate closely to the proposal of Annen et al. for calculating shadow maps [13], whose adaptation to the Bilateral Filter was proposed in [14, p. 169]. An advantage of such methods is that to achieve an output, several images need to be filtered independently and summed together, as opposed to having to conduct a per-pixel linear interpolation or 3D convolutions with additional sampling of a data structure.

Currently, literature on frequency-based approaches accounts mainly for the Gaussian range kernel, or ones with quickly attenuating tails [12], but do not account for the possible onset of the Gibbs phenomenon, as observed in [13]. Additionally, the parallelisation aspect of the Fourier series-based approaches has not been explored. Thus, the contributions of the paper are as follows:

1. A GPU-based Frequency-based bilateral filter;
2. An analysis of different range kernels, including some with longer tails, and their impact on execution time;
3. A discussion of different filtering kernel calculations in the spatial domain.

The paper is structured as follows: Section 2 provides background knowledge. Section 3 presents our proposed algorithm. Results are presented in Section 4, while Section 5 is reserved for discussion. Finally, ethical implications are elaborated on in Section 6, followed by conclusion and future work in 7.

2 Background

This section will be presented as follows: Section 2.1 elaborates on linear filtering, and Section 2.2 presents the background of the bilateral filter. Finally, Section 2.3 presents in more detail acceleration approaches.

2.1 Linear Filtering

In linear filtering, a pixel's value is computed as a linear combination of other pixel intensities, as seen in Equation 1:

$$F[\mathbf{p}] = \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\mathbf{p}-\mathbf{q}) I_{\mathbf{q}} \tag{1}$$

where $I_{\mathbf{p}}$ is the intensity of a pixel with coordinates $\mathbf{p}$, $\mathcal{S}$ is the domain of coordinates around pixel $\mathbf{p}$, also called a neighbourhood ($\mathcal{S} \subset \mathbb{Z}^2$), and $G_{\sigma_s}(\mathbf{p}-\mathbf{q})$ is the spatial kernel. This process, also known as *convolution*, is at the foundation of image processing [15]. Because we are working with discrete pixel coordinates, the weights of the spatial kernel can conveniently be organized in a $k \times k$ matrix.

Different functions can be used to define the weights in the matrix depending on the use case. For image blurring, two functions are most prominent – the 2D Gaussian (Equation 2) and the box kernel (Equation 3):

$$G_{\sigma_s}(\mathbf{p}) = \frac{1}{\sigma^2 \cdot 2\pi} e^{\frac{-||\mathbf{p}||^2}{2\sigma_s^2}} \tag{2}$$

$$B_n(\mathbf{p}) = \begin{cases} \frac{1}{n^2} & \text{if } ||\mathbf{p}||_\infty \leq n \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

with n being the kernel size in Equation 3 and $||\mathbf{p}||_\infty$ representing the l_∞ norm – the value of the largest coordinate within the vector $\mathbf{p}$. The amount of blurring is controlled via the parameter σ_s for the Gaussian and n for the box filter, with higher values yielding more blurring.

In the frequency domain, convolution is expressed only as multiplication. Therefore, the effects of different filters can be observed through their Fourier transforms. The Gaussian remains a Gaussian in the frequency domain, meaning that it smoothly attenuates high-frequency components. On the other hand, the box filter becomes a sinc function that exhibits an oscillating behaviour. As a consequence, artefacts within the output will appear, which we typically perceive as lower-quality filtering.

2.2 Bilateral Filter

The bilateral filter (BF) is a method for edge-aware image blurring, defined in [1]. Mathematically, it can be described for an arbitrary pixel with coordinates $\mathbf{p} \in \mathbb{Z}^2$ as follows:

$$BF[\mathbf{p}] = \frac{1}{W_\mathbf{p}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\mathbf{p} - \mathbf{q}) R_{\sigma_r}(I_\mathbf{p} - I_\mathbf{q}) I_\mathbf{q} \quad (4)$$

It differs compared to linear filtering in Equation 1 by the presence of a range kernel $R_{\sigma_r}(I_\mathbf{p} - I_\mathbf{q})$ that weighs each of the pixels in the spatial region $\mathcal{S}$ by a value based on the intensity difference with the centre pixel, and a normalisation factor $W_\mathbf{p}$. It is computed similarly:

$$W_\mathbf{p} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\mathbf{p} - \mathbf{q}) R_{\sigma_r}(I_\mathbf{p} - I_\mathbf{q}) \quad (5)$$

ensuring that the resulting pixel weights sum up to one. The need for this factor arises because of the range kernel – within each neighbourhood, the elements of the spatial kernel will be weighed differently, thus the total norm will vary per filtered pixel.

Both the spatial and the range kernels are usually defined as a Gaussian. Nevertheless, especially for the range kernel, various choices are common, as outlined in Table 1 and shown in Figure 1. Samples of the bilateral filter's output can be observed in Figure 2.

Table 1: A table of possible range kernels. In order to ensure a consistent scale among them, the σ parameter needs to be multiplied by the value in the column labelled 'Scale' [3].

Name	Definition	Scale
Gaussian	$R_\sigma(x) = \mathrm{e}^{\frac{-x^2}{2\sigma^2}}$	1
Tukey	$R_\sigma(x) = \begin{cases} \frac{1}{2}\left(1 - \left(\frac{x}{\sigma}\right)^2\right)^2 & \lvert x\rvert \le \sigma \\ 0 & \text{otherwise} \end{cases}$	$\sqrt{5}$
Huber	$R_\sigma(x) = \begin{cases} \frac{1}{\sigma} & \lvert x\rvert \le \sigma \\ \frac{1}{\lvert x\rvert} & \text{otherwise} \end{cases}$	1
Lorentz	$R_\sigma(x) = \frac{2}{2+\left(\frac{x}{\sigma}\right)^2}$	$\frac{1}{\sqrt{2}}$

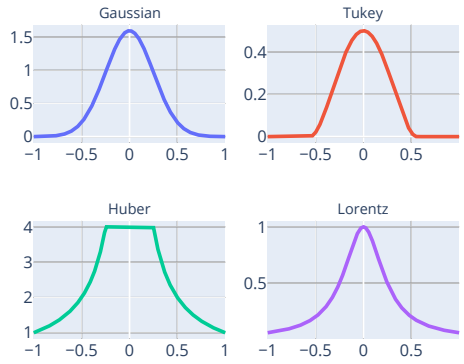
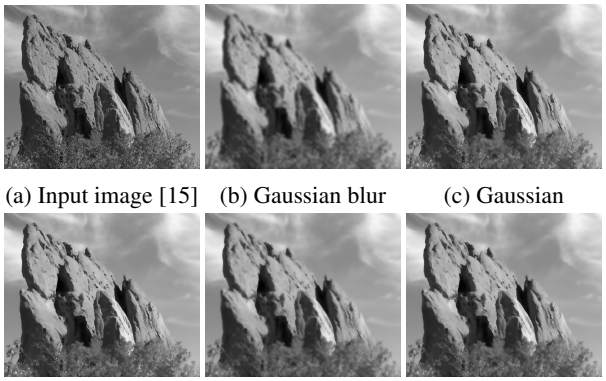


Figure 1: The curves of the different range kernels illustrated in Table 1 for $\sigma_r = 0.25$.



(a) Input image [15] (b) Gaussian blur (c) Gaussian

(d) Tukey (e) Huber (f) Lorentz

Figure 2: Example of an image filtered using the bilateral filter with $\sigma_s = 5, 17 \times 17$ spatial kernel and different range kernels with parameter $\sigma_r = 0.1$. Figure 2b shows as a comparison Gaussian linear blurring

2.3 Approaches for Acceleration

When naively implemented, both linear and bilateral filtering are quadratic algorithms. This subsection explores alternative approaches for executing them that have a lower time complexity.

Linear Filters

For arbitrary spatial kernels, acceleration is achieved by calculating convolutions in the frequency domain, where they are linear operations. By using the Fast Fourier Transform, which is $\mathcal{O}(n \log n)$, the time complexity of linear filtering can be reduced by one order of magnitude.

A property that both the Gaussian and the box filter (Equations 2 and 3) exhibit is *separability*. For example, the Gaussian can be separated like in Equation 6:

$$\begin{aligned} G_{\sigma_s}(\mathbf{p}) &= \frac{1}{\sigma^2 \cdot 2\pi} e^{\frac{-||\mathbf{p}||^2}{2\sigma_s^2}} \\ &= \frac{1}{\sigma^2 \cdot 2\pi} e^{\frac{-p_x^2 - p_y^2}{2\sigma_s^2}} \\ &= \frac{1}{\sigma \cdot \sqrt{2\pi}} e^{\frac{-p_x^2}{2\sigma_s^2}} \cdot \frac{1}{\sigma \cdot \sqrt{2\pi}} e^{\frac{-p_y^2}{2\sigma_s^2}} \end{aligned} \quad (6)$$

This means that a convolution with a separable kernel of size $n \times n$ can be done in two passes – first with a $1 \times n$ kernel and then a $n \times 1$ or vice versa. This so-called *separable convolution* allows performing close to linear-time spatial filtering.

While the box filter is also separable, it enables another approach involving summed-area tables [16]. They can be interpreted as a cumulative distribution over an image, allowing for a constant-time filtering per pixel using two addition and two subtraction operations.

Bilateral Filter

Because of the introduction of $R_{\sigma_r}(I_\mathbf{p} - I_\mathbf{q})$, the bilateral filter is not a linear filter. Therefore, approximations of equation 4 need to be used to reduce the time complexity below quadratic.

One approach to accelerating the bilateral filter, explored by [8–11] is to decompose the range kernel using Fourier series:

$$R_{\sigma_r}(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left(a_k \cos\left(\frac{2\pi k}{T}x\right) + b_k \sin\left(\frac{2\pi k}{T}x\right)\right) \tag{7}$$

where

$$a_k = \frac{2}{T}\int_{-T/2}^{T/2} R_{\sigma_r}(x)\cos\left(\frac{2\pi k}{T}x\right)\mathrm{d}x \tag{8}$$

and

$$b_k = \frac{2}{T}\int_{-T/2}^{T/2} R_{\sigma_r}(x)\sin\left(\frac{2\pi k}{T}x\right)\mathrm{d}x \tag{9}$$

are the Fourier coefficients, and T is the period of approximation.

If the range kernel is an even function (like the examples given in Table 1), the sine component will attenuate to zero. Finally, after limiting the number of coefficients to a value N, the final form of the approximation will be given by:

$$\hat{R}_{\sigma_r}(x) = \frac{a_0}{2} + \sum_{k=1}^{N} a_k \cos\left(\frac{2\pi k}{T}x\right) \tag{10}$$

Now $\hat{R}_{\sigma_r}$ can be substituted within equations 4 and 5. Additionally, from the identity

$$\cos(\alpha-\beta) = \cos(\alpha)\cos(\beta) + \sin(\alpha)\sin(\beta) \tag{11}$$

and from rearranging the obtained terms, the following is yielded:

$$\begin{aligned} BF[\mathbf{p}] \approx & \frac{1}{W_\mathbf{p}}\sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\mathbf{p}-\mathbf{q}) I_\mathbf{q} \frac{a_0}{2} \\ & + \frac{1}{W_\mathbf{p}}\sum_{k=1}^{N} a_k \cos(\xi_k I_\mathbf{p}) \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\mathbf{p}-\mathbf{q}) I_\mathbf{q} \cos(\xi_k I_\mathbf{q}) \\ & + \frac{1}{W_\mathbf{p}}\sum_{k=1}^{N} a_k \sin(\xi_k I_\mathbf{p}) \sum_{\mathbf{q}\in\mathcal{S}} G_{\sigma_s}(\mathbf{p}-\mathbf{q}) I_\mathbf{q} \sin(\xi_k I_\mathbf{q}) \end{aligned} \tag{12}$$

where $\xi_k = \frac{2\pi k}{T}$, and $W_\mathbf{p}$ is computed similarly but without multiplying by $I_\mathbf{q}$. Thus, each coefficient a_k corresponds to four different *component images* – two for the numerator in Equation 12, and two for $W_\mathbf{p}$. According to this formulation, each component image is filtered, and then element-wise multiplied by its unfiltered counterpart and Fourier coefficient.

Importantly, two new variables are introduced: the number of coefficients N and the period of the approximation T. Higher values of N result in a smaller error between the frequency-based bilateral filter and the naive implementation. Yet, a larger T necessitates a higher N, so as to achieve convergence in the Fourier series, leading to higher computation costs.

The advantage of approximating the range kernel using Fourier series is expressed in the presence of four independent linear filtering operations, which are weighted and summed together. The improvement of the time complexity is therefore achieved by using fast algorithms for each convolution.

3 Method

The approach we propose looks at Equation 12 as a sequence of spatial filtering and accumulation operations, each of which is done using a GPU, as outlined in Algorithm 1.

Algorithm 1 GPU implementation of the frequency-based bilateral filter

Require: $R(x)$ – range kernel, σ_s, σ_r, N – number of coefficients, T – period of approximation
1: compute Fourier coefficients $a_0 \dots a_k$
2: precompute lookup table with trigonometric values
3: **for** $k = 0, \dots, N-1$ **do**
4: compute the current component images
5: filter the current 4 component images and accumulate in a common buffer
6: **end for**
7: process the accumulator into the final output
8: **return** final image

Details about each step of Algorithm 1 will be presented separately. Section 3.1 discusses aspects regarding the calculation of coefficients and their number (line 1). Section 3.2 covers the GPU-based filtering (lines 2 to 7).

3.1 Coefficients

When evaluating the coefficients that are used for the filtering, one has to consider their count, the period over which they are calculated, and the method for computing them.

This paper treats images as two-dimensional arrays of floating-point numbers in the range $[0, 1]$. This implies that the values within the argument of the range kernel are in the range $[-1, 1]$. Therefore, for our calculations, a fixed period $T = 2$ is chosen, which excludes the ability to do period length optimisation [8, 10]. The reason for doing so relates to kernels with tails that do not quickly attenuate, which implies that large intensity differences can be overrepresented through the periodicity of the Fourier approximation (see Figure 3).

In order to handle the number of coefficients for arbitrary range kernels through a single calculation, as opposed to iterating over a search space for finding a proper parameter [10, 12] we attempt to use Equation 13:

$$N = \left\lceil \frac{PT}{6\sigma_r} \right\rceil + C \tag{13}$$

This equation increases N as σ_r becomes smaller by ensuring the Fourier series' base functions have P oscillations in the interval $[-3\sigma, 3\sigma]$, which is expected to have the majority of pixel intensities with a high contribution. The reason for doing that is in small sigma values – they introduce high frequency components in the range kernel that need to be added to achieve convergence. If N is insufficient, the Gibbs phenomenon may be observed, as seen in the Huber range kernel in Figure 3.

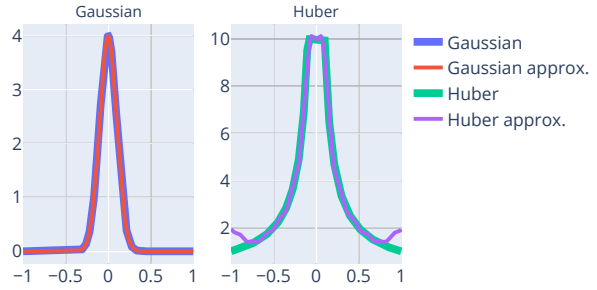


Figure 3: Example of the Gaussian and Huber range kernels with $\sigma_r = 0.1$ estimated with 10 coefficients for a period $T = 1.5$, which does not cover the whole interval $[-1, 1]$. Notice that a pixel intensity difference of ± 1 will be given a disproportionally higher weight in the Huber kernel's approximation (right) compared with the Gaussian (left). Ringing in the Huber kernel can be observed around $x = \pm 0.1$

Finally, the calculation of a_k in equation 8 is handled via numerical integration using the GNU Scientific Library's `gsl_integration_qags` function [17]. This was chosen for ensuring the possibility of working with arbitrary range kernels, which could lead to non-elementary integrals.

3.2 GPU-based Frequency-based Bilateral Filter

Frequency-based bilateral filtering on the GPU starts by copying the input image from the host (CPU) to the device (GPU). Afterwards, for each coefficient, four component images (see Section 2.3) are filtered and added to an accumulator. Finally, the accumulator's values are divided (Equation 12), to yield the final output, which is copied back from the GPU to the host. Each step will be covered separately in more detail.

Calculating Component Images

The input to the algorithm is a single-channel image. From each pixel, four different values are computed. Therefore, it is possible to organize the memory by making a four-channel single-precision component image using the `float4` data type. The channel order used in our implementation is as follows: $\cos(\xi_k I_\mathbf{p})$, $\sin(\xi_k I_\mathbf{p})$, $\cos(\xi_k I_\mathbf{p})\, I_\mathbf{p}$ and $\sin(\xi_k I_\mathbf{p})\, I_\mathbf{p}$, where the first two numbers correspond to the terms of the normalisation factor $W_\mathbf{p}$, and the last two correspond to the numerator of Equation 12.

The calculation of component images would involve many calls to trigonometric functions. Since they are known to be slow on GPUs [18], a lookup table can be used to precompute values of $\cos\left(\frac{2\pi k}{T} I_\mathbf{p}\right)$ and $\sin\left(\frac{2\pi k}{T} I_\mathbf{p}\right)$. The LUT relies on both the limited number of pixel intensities $I_\mathbf{p}$ for an image of type `uint8` and the presence of discrete Fourier coefficients a_k.

Each entry in the lookup table is a value of type `float2` (a vector of two single-precision floating-point numbers), because per a combination of pixel intensity and coefficient index, there is one sine and one cosine value. Regarding memory layout, we opted for making the table with N rows and 256 columns stored in constant memory. Doing so promotes the reusage of values fetched in each cache line within the GPU, which benefits performance [19].

Parallel Spatial Filtering

We chose to implement spatial blurring using separable convolutions, because they allow one to use either the Gaussian or box kernels, both of which are separable (see Section 2.3), and because of the time complexity that allows close to linear time convolution. Combined with the memory layout that uses `float4` values per pixel, one iteration of the separable convolution would be needed to filter four images, contributing to a higher throughput of data. If instead four distinct single-channel images were computed per coefficient, more GPU kernel invocations would be necessary, each of which would contribute to latency.

Separable convolution is implemented using two different GPU kernels – one doing a vertical pass, the other – horizontal. Importantly, Equation 12 requires accumulating the results to a common buffer. We integrated the accumulation into the GPU kernel responsible for the horizontal pass as opposed to the vertical one. The reason for that relates to the row-major memory layout, which gives a lower penalty when reading and writing to and from a GPU's global memory in rows instead of in columns.

For spatial filtering, a choice regarding the conversion from σ_s to a concrete size of a Gaussian convolution kernel is required. To that end, equation 14 is used not only due to its occurrence in the popular image processing library OpenCV[1], but also because values further away from the centre of the kernel quickly attenuate to zero in single-precision floating-point numbers.

$$n = \text{round}(\sigma_s \cdot 1.5) \cdot 2 + 1 \tag{14}$$

Finally, for the GPU-side code, the spatial kernel's coefficients remain unchanged throughout the execution just like the trigonometric lookup table. Therefore, those can be stored in a GPU's constant memory as well, which has a more streamlined pathway to streaming multiprocessors. Since that space is limited, there is an upper bound on both the number of Fourier coefficients and the size of the spatial kernel. In our method, the upper bound for N was set to 28, which allows for at most 1×1024-sized spatial kernel of type `float`.

4 Results

This section presents results regarding the coefficients obtained from the Fourier series in Section 4.1. Finally, runtime performance is shown in Section 4.2.

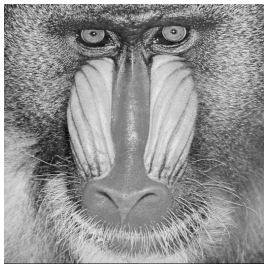
4.1 Number of coefficients

Evaluation of the feasibility of Equation 13 is conducted using a CPU implementation of the naive bilateral filter and a GPU implementation of the frequency-based one. The performance metric will be peak signal-to-noise ratio (PSNR) while excluding the border region of the image due to different possible behaviours depending on the underlying convolution technique.

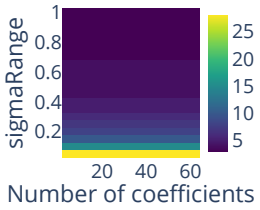
The output of the frequency-based filter is considered viable if its PSNR compared to the naive algorithm is above

[1] https://github.com/opencv/opencv/blob/4.x/modules/imgproc/src/bilateral_filter.dispatch.cpp

50 dB [8]. As seen in Figure 4, when using $P = 4$ with the provided input image, PSNR values range from 50 to 140 dB. While P could be lowered, which would respectively decrease the obtained N, a lower number of coefficients might lead to an overly low PSNR value for some parameter values.



(a) Input image [20]



(b) Number of coefficients using $P = 4$ and $C = 1$

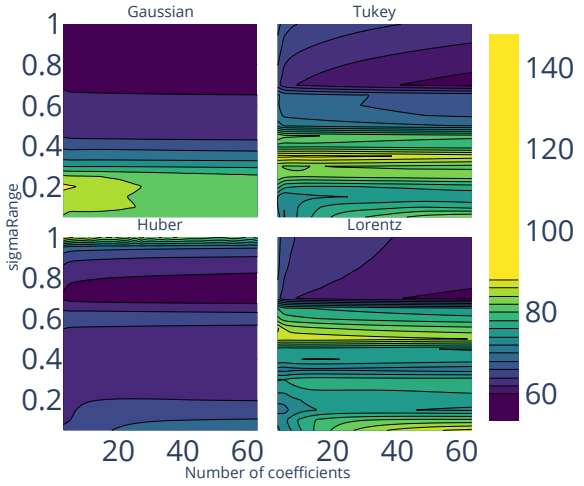


Figure 4: Evaluation of the accuracy of the frequency-based bilateral filter for different range kernel types and different parameters ($\sigma_r \in [0.05, 1]$ when using equation 13 with parameters $P = 4, C = 1$ filter sizing ranging from 3×3 to 63×63) for the image shown in Figure 4a

Moreover, PSNR can vary significantly for other images, especially where regions of high uncertainty (described by a low value of $W_{\mathbf{p}}$ [3]) are present. An example of this occurring can be seen in Figure 5, where the input contains small patches of high intensities surrounded by darker pixels, like shown in Figure 5c. After limiting the frequency-based filter's output to the range $[0, 1]$, salt-and-pepper noise artefacts could be observed in uncertain pixels. For multi-megapixel images, those might not be easily visible, but their presence could be lowering PSNR for an image where the rest of the details are well-preserved – a phenomenon responsible for the trends observed in Figure 6. A possible solution would be to apply an interpolation with the unfiltered image, as noted by Durand and Dorsey [3], as a post-processing step after the frequency-based filter's intensities are limited in the range $[0, 1]$.

Looking closely at Figure 3, slight ringing can be observed for the Huber range kernel despite the high coefficient count. The Gibbs phenomenon observed in this kernel could therefore be responsible for the generally lower PSNR noted for it in Figure 4. This, together with the different behaviours as σ_r is varied in Figure 4, alludes to a deficiency of equation 13 in determining the necessary N, as intrinsic properties of the range kernels are not taken into consideration.

4.2 Runtime performance of the GPU implementation

The variables that need to be isolated for the GPU implementation are the size of the filter and the number of coefficients N. Therefore, unlike for PSNR measurements, the range kernel type and σ_r are not used in the subsequent benchmarks.

The GPU implementation's performance was evaluated on a computer with an NVIDIA RTX 4070 8GB graphics card, AMD Ryzen 7950X processor, and 64 GB of DDR5 memory. When doing so, two steps were taken. Firstly, the GPU and memory clocks were explicitly locked for minimising fluctuations in the observed execution time. Secondly, steps like memory allocation and precomputing values are not taken into the measurements. The image used for testing [21] has a resolution of 4500×3000 px – a rather large size that makes sure the GPU's L2 cache is full, which enables the proper assessment of memory throughput.

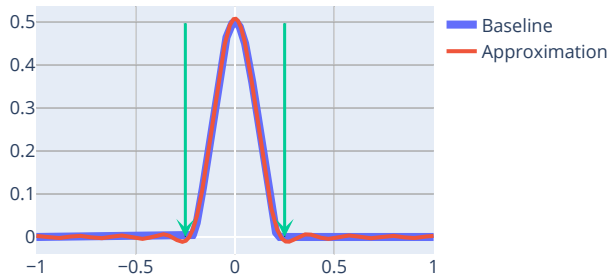
Figure 7 presents the observed runtimes. The importance of minimising the number of coefficients can be seen in the linear growth of the execution time for all tested parameters as more coefficients are added (see Figure 7d). Regarding spatial filtering, separable convolutions would imply the runtime will increase linearly with the kernel size. While this is the case for filter sizes larger than approximately 25×25, smaller ones exhibit a constant behaviour, as seen in Figures 7b and 7c.

In terms of memory performance, Nvidia's Nsight Compute profiler [22] was used to observe the amount of IO operations to and from global memory, with percentages of read and write operations relative to the theoretical bound (number of pixels times the size of the data type) in Figure 8. Notably, it shows a significant discrepancy between the read ratio of the horizontal pass of the separable convolution and the vertical one. The reason for that lies in the observed L2 cache hit ratio, which is observed to be two times lower for the vertical pass as opposed to the horizontal pass.

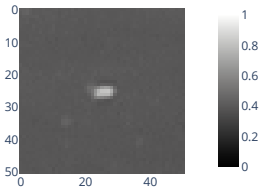
In order to compare performance, we used the GPU-based bilateral filter found in OpenCV[2] as it is a readily available implementation that can handle single-colour images. Figure 9 indicates that the quadratic filter's execution time can in some cases be lower than the frequency-based filter. The advantage of our approach, though, is most apparent for extreme spatial filter sizes, where the difference in execution time can differ by more than a factor of two in the case when $N = 28$, and by more than a factor of 6 times when $N = 10$, shown by Figure 9. This highlights the importance of choosing a number of coefficients, as it can have a significant influence on performance.

[2] https://docs.opencv.org/4.8.0/d0/d05/group__cudaimgproc.html

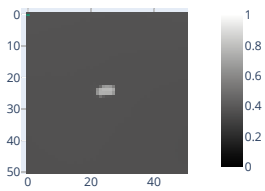
(a) Input image with the location of overshoot marked with a red rectangle; the resolution is 4500×3000px



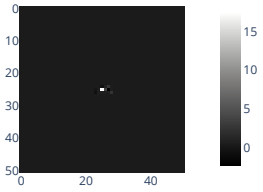
(b) The range kernel overlaid with its approximation. Purple lines mark the intensity difference between the centre pixel and an arbitrary surrounding pixel in 5c



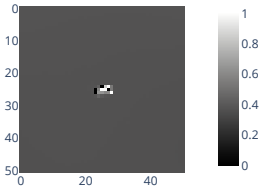
(c) Zoomed-in section of the image



(d) Filtered with the naive implementation



(e) After filtering with the frequency-based filter, before clamping



(f) After filtering with the frequency-based filter and clamping

Figure 5: Example of the occurrence of overshoots after filtering with a kernel size of 51×51, Tukey range kernel ($\sigma_r = 0.1$), approximated with 10 coefficients. Before clamping, the PSNR measured at 36.862 dB, and after clamping – 51.602 dB.

5 Discussion

This work deviates from current literature about the frequency-based bilateral filter in two main ways – the time complexity depends on the filter size and no period length optimisation was applied, which could make convergence of the Fourier series approximation of the range kernel more difficult, as shown in Section 4.1.

Firstly, separable convolutions were chosen as opposed to the FFT for linear filtering after taking into account the time complexities and the amount of input-output operations that could arise in the already highly memory-bound problem of image filtering. The concern with the DFT is the number of transitions to and from the frequency domain (in or approach, per coefficient, 4 FFTs need to be applied, 4 elementwise multiplications carried out, and 4 inverse FFTs afterwards, thus amounting to 8 $\mathcal{O}(n \log n)$ operations).

Secondly, period length optimisation was omitted from this work, though it can provide great results for kernels with quickly attenuating tails (like the Gaussian or in [12]). Here, we explore the effects of other range kernels where this is not guaranteed (like in the Huber or Lorentz kernels, presented in Table 1 and in Figure 3). Unfortunately, the fixed period length leads to difficulties regarding convergence in kernels with points that introduce high-frequency components (like Huber or Tukey) and for small σ_r values. Such cases would require a high number of coefficients, which would negatively impact runtime performance.

An important aspect that was not covered so far is the handling of colour. The colour version of the Equation 4 involves the Euclidean distance of a colour difference vector as the argument of the range kernel [15]. If Fourier series are to be used directly, a rapid growth in the number of coefficients is predicted, thus deeming them infeasible for working with colour images without resorting to other approximations. Some papers like [23, 24] indicated that naively handling channels separately can lead to a sufficient result. Alternatively, the cross-bilateral filter [2] could be adapted to work with the image's intensity as the range kernel, but that approach will lead to blurring across edges with different colours but similar intensity.

Due to time constraints, only PSNR was used as a metric for assessing the precision of the frequency-based bilateral filter's approximations. This excludes metrics tailored towards human perception like SSIM [25] or FLIP [26].

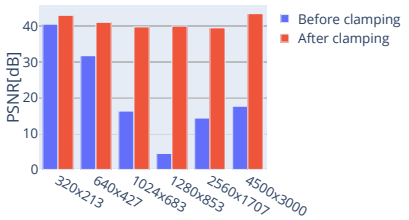
Finally, performance measurements are presented in terms of frame times for a 13.5 MP image. While directly converting them to a measure of frames per second might not seem like great performance, the measure of throughput in MP/s gives a better insight into the capabilities of this approach. The reason for showing frame times is to clearly present asymptotic behaviour.

6 Responsible Research

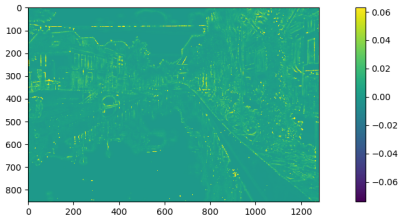
The main challenge to reproducibility comes with the choice of API and device for development. By opting for the proprietary CUDA API, we have implicitly excluded other vendors like AMD or Intel and therefore conformed to the status quo with NVIDIA's dominant position in the GPU market [27]. To cater to other device manufacturers, frameworks like HIP[3]

[3] https://rocm.docs.amd.com/projects/HIP/en/latest/

(a) The input image scaled to 1280x853

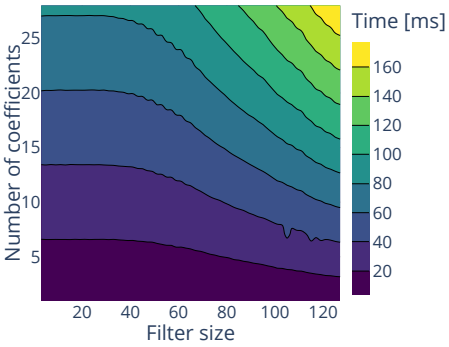


(b) PSNR ratings

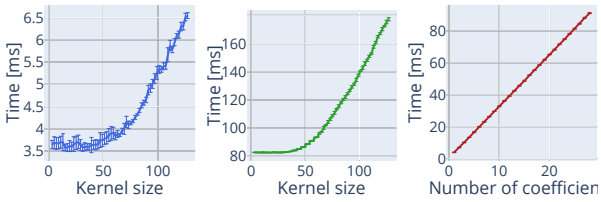


(c) Difference between the naive and clamped frequency-based filter for the image shown in 6a

Figure 6: PSNR ratings for image [21] filtered at different sizes with the parameters $\sigma_s = 8, \sigma_r = 0.1, 6$ coefficients, Gaussian range kernel. The significant drop of the PSNR for larger images in Figure 5 could be attributed to the division by the normalisation factor in highly uncertain areas, which is remedied by clamping (i.e., limiting the output image's pixel vales to be in the range $[0, 1]$)



(a) The entire parameter space



(b) 1 coefficient (c) 28 coefficients (d) 127×127 kernel

Figure 7: Runtime performance of the GPU implementation for kernel sizes from 3×3 to 127×127 and number of coefficients N in the range $[1, 28]$. Subfigures 7b and 7c show the runtime with error bars when N remains constant, and 7d shows the behaviour when the kernel size is fixed at 127×127, and the number of coefficients is modified.

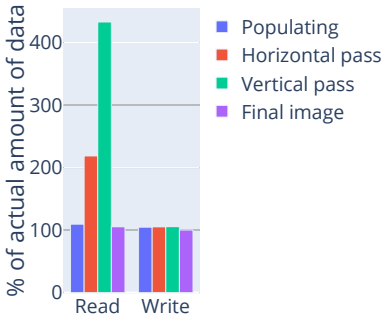


Figure 8: Percentage of read and write operations compared with the image size (in bytes) when filtering a 4500×3000 px image with 10 coefficients, Gaussian range kernel with $\sigma_r = 0.1$, 51×51 spatial kernel; values closer to 100% indicate better efficiency. Horizontal and vertical pass relate to the separable convolution used for spatial filtering of component images. The large difference between the horizontal and vertical pass' read percentages is due to the L2 cache hit ratio

have been created that offer tools[4] for migrating an existing CUDA codebases to it, allowing one to compile for AMD devices. For true platform independence, the provided source code could be translated to the SYCL framework (i.e., via Intel's implementation in their Data Parallel C++ compiler[5]) at the cost of developer intervention.

The Turing microarchitecture of the GPU used for development introduced changes in thread scheduling that could lead to different behaviour on older devices [28]. Since our codebase[6], does not benefit from the updated scheduler, compatibility with older NVIDIA devices is ensured by targeting the older compute capabilities through compiler flags.

Replicating the results on graphics processing units from different vendors may require further tuning of thread blocks sizes, depending on the underlying architecture. The implications for performance relate to global memory accesses and warp/wavefront size, which together affect both the occupancy rate and the memory throughput on the GPU.

Another consideration for reproducibility and comparing PSNR measurements is the omitted borders. This was done intentionally due to the many different strategies for handling the border effect in image filtering. A possible solution is to pad the input images before applying different implementations of the bilateral filter for ensuring consistent handling of border effects.

Finally, during the evaluation, several images have been

[4] https://github.com/ROCm/HIPIFY

[5] https://www.intel.com/content/www/us/en/developer/tools/oneapi/data-parallel-c-plus-plus.html

[6] The project can be found at: https://github.com/simo1427/freq-bf or https://gitlab.ewi.tudelft.nl/cse3000/2023-2024-q4/Eisemann_Molenaar/satanasov-Image-Processing-with-the-Bilateral-Filter

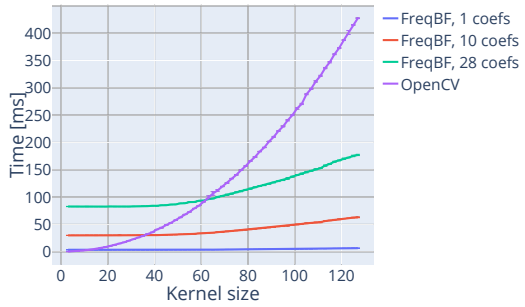


Figure 9: Comparing the performance of our implementation (labelled FreqBF) for different number of coefficients against a reference one provided by OpenCV.

used. Sourcing them was achieved by using imagery from older papers and open databases. The used images are with suitable licences or are allowed to be used for research purposes.

7 Conclusions and Future Work

This paper focused on three areas: mapping the problem of frequency-based bilateral filtering on a GPU, discussing the effects of range kernels with properties different from the usually chosen Gaussian's, and analysing different approaches for doing spatial filtering. By using an accumulator and by doing separable convolutions for component images on a GPU, real-time performance can be a possibility for a large variety of kernel sizes, and the time complexity is also reduced for large filter sizes.

A possible limitation is the use of separable convolution, as it is not providing a time complexity that remains constant as the filter size grows. A solution would be to further experiment with summed area tables, which, combined with a framework for determining the number of coefficients, could very positively affect performance. As a baseline for the performance measurements, other GPU implementations like the bilateral grid [5] could provide an interesting insight into the behaviour of our approach.

A different direction for future work, due to the ubiquity of the Gaussian range kernel, is optimising this approach further by applying mathematical properties, as done by Deng [11]. Finally, the use of half-precision numbers could be attempted, in order to further increase performance at the cost of approximation fidelity.

References

[1] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images", 1998, pp. 839–846. [Online]. Available: https://www.scopus.com/inward/record.uri?eid=2-s2.0-0032319446&partnerID=40&md5=7714e0199daac1e20f0835155cf8ac2e.

[2] E. Eisemann and F. Durand, "Flash photography enhancement via intrinsic relighting", *ACM Trans. Graph.*, vol. 23, no. 3, pp. 673–678, Aug. 2004, ISSN: 0730-0301. DOI: 10.1145/1015706.1015778. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/1015706.1015778.

[3] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images", *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002, ISSN: 0730-0301. DOI: 10.1145/566654.566574. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/566654.566574.

[4] H. Winnemoeller, S. Olsen, and B. Gooch, "Real-time video abstraction", *ACM Trans. Graph.*, vol. 25, pp. 1221–1226, Jul. 2006. DOI: 10.1145/1179352.1142018.

[5] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid", *ACM Trans. Graph.*, vol. 26, no. 3, 103–es, Jul. 2007, ISSN: 0730-0301. DOI: 10.1145/1276377.1276506. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/1276377.1276506.

[6] H. Gupta, D. S. Antony, and G. N. Rathna, "Implementation of gaussian and box kernel based approximation of bilateral filter using opencl", in *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, 2015, pp. 1–5. DOI: 10.1109/DICTA.2015.7371269.

[7] *OpenCL Guide*, Khronos Group. [Online]. Available: https://github.com/KhronosGroup/OpenCL-Guide (visited on 06/20/2024).

[8] K. Sugimoto and S.-I. Kamata, "Compressive bilateral filtering", *IEEE Transactions on Image Processing*, vol. 24, no. 11, pp. 3357–3369, 2015. DOI: 10.1109/TIP.2015.2442916.

[9] S. Ghosh and K. Chaudhury, "On fast bilateral filtering using fourier kernels", *IEEE Signal Processing Letters*, vol. 23, pp. 1–1, Mar. 2016. DOI: 10.1109/LSP.2016.2539982.

[10] S. Ghosh, P. Nair, and K. Chaudhury, "Optimized fourier bilateral filtering", *IEEE Signal Processing Letters*, vol. 25, pp. 1–1, Aug. 2018. DOI: 10.1109/LSP.2018.2866949.

[11] G. Deng, "Fast compressive bilateral filter", *Electronics Letters*, vol. 53, no. 3, pp. 150–152, 2017. DOI: https://doi.org/10.1049/el.2016.3416. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/el.2016.3416. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2016.3416.

[12] Y. Sumiya, N. Fukushima, K. Sugimoto, and S.-i. Kamata, "Extending compressive bilateral filtering for arbitrary range kernel", in *2020 IEEE International Conference on Image Processing (ICIP)*, 2020, pp. 1018–1022. DOI: 10.1109/ICIP40778.2020.9191123.

[13] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz, "Convolution Shadow Maps", in *Rendering Techniques*, J. Kautz and S. Pattanaik, Eds., The Eurographics Association, 2007, ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/051-060.

[14] E. Eisemann, "Optimized representations for the acceleration of display- and collision queries", Ph.D. dissertation, Grenoble Universities (Universit'e Joseph Fourier), 2008. [Online]. Available: http://graphics.tudelft.nl/Publications-new/2008/Eis08.

[15] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, "A gentle introduction to bilateral filtering and its applications", in *ACM SIGGRAPH 2007 Courses*, ser. SIGGRAPH '07, San Diego, California: Association for Computing Machinery, 2007, 1–es, ISBN: 9781450318235. DOI: 10.1145/1281500.1281602. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/1281500.1281602.

[16] J. Hensley, T. Scheuermann, G. Coombe, M. Singh, and A. Lastra, "Fast summed-area table generation and its applications", *Computer Graphics Forum*, vol. 24, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:10623546.

[17] M. Galassi *et al.*, *GNU Scientific Library Reference Manual (3rd Edition)*. Network Theory, 2021, ISBN: 0954612078.

[18] *CUDA C++ programming guide*, Nvidia, May 21, 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (visited on 06/03/2024).

[19] *CUDA best practices guide*, Nvidia, May 21, 2024. [Online]. Available: https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/ (visited on 06/23/2024).

[20] "SIPI Image Database", USC Viterbi School of Engineering. (), [Online]. Available: https://sipi.usc.edu/database/database.php?volume=misc&image=10#top (visited on 06/23/2024).

[21] W. Bulach, *00 0781 Canal in Delft (NL).jpg*, https://commons.wikimedia.org/wiki/File:00_0781_Canal_in_Delft_(NL).jpg, Accessed: 2024-06-11.

[22] *Nsight Compute Documentation*, Nvidia, Apr. 18, 2024. [Online]. Available: https://docs.nvidia.com/nsight-compute/index.html (visited on 06/20/2024).

[23] K. N. Chaudhury, D. Sage, and M. Unser, "Fast $O(1)$ Bilateral Filtering Using Trigonometric Range Kernels", en, *IEEE Transactions on Image Processing*, vol. 20, no. 12, pp. 3376–3382, Dec. 2011, ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2011.2159234. [Online]. Available: http://ieeexplore.ieee.org/document/5872028/ (visited on 04/23/2024).

[24] G. Petschnigg, R. Szeliski, M. Agrawala, M. Cohen, H. Hoppe, and K. Toyama, "Digital photography with flash and no-flash image pairs", *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, Aug. 2004, ISSN: 0730-0301. DOI: 10.1145/1015706.1015777. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/1015706.1015777.

[25] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: From error visibility to structural similarity", *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[26] P. Andersson, J. Nilsson, T. Akenine-Möller, M. Oskarsson, K. Åström, and M. D. Fairchild, "Flip: A difference evaluator for alternating images", *Proc. ACM Comput. Graph. Interact. Tech.*, vol. 3, no. 2, Aug. 2020. DOI: 10.1145/3406183. [Online]. Available: https://doi.org/10.1145/3406183.

[27] "Graphics add-in-board (AIB) supplier shipment share worldwide 2010 to 2023, by quarter [Graph]", Jon Peddie Research. (Mar. 2024), [Online]. Available: https://www.statista.com/statistics/274005/market-share-of-global-graphics-card-shipments-since-3rd-quarter-2010/ (visited on 06/20/2024).

[28] *Turing Tuning Guide*, Nvidia, May 21, 2024. [Online]. Available: https://docs.nvidia.com/cuda/turing-tuning-guide/ (visited on 06/20/2024).