Parallel Recursive Filtering of Infinite Input Extensions

Diego Nehab
IMPA

André Maximo
GE Global Research

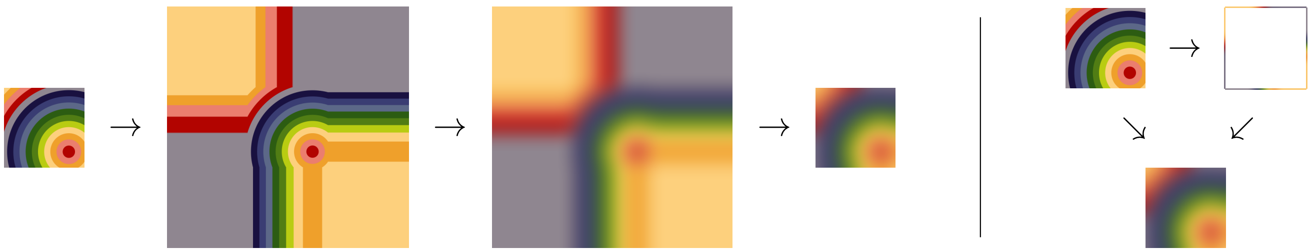


Figure 1: *(Left) Previous recursive filtering algorithms use padding to approximate the effect of boundary conditions. When the filter impulse response decays slowly, the amount of computation over the padding can become prohibitive. (Right) We instead filter infinite extensions exactly. We do so by first obtaining the correct initial feedbacks at the boundaries using explicit formulas. These formulas were designed for easy integration into block-parallel algorithms that run on the GPU. Our new algorithms are not only exact, but also the fastest to date.*

Abstract

Filters with slowly decaying impulse responses have many uses in computer graphics. Recursive filters are often the fastest option for such cases. In this paper, we derive closed-form formulas for computing the exact initial feedbacks needed for recursive filtering infinite input extensions. We provide formulas for the constant-padding (e.g. clamp-to-edge), periodic (repeat) and even-periodic (mirror or reflect) extensions. These formulas were designed for easy integration into modern block-parallel recursive filtering algorithms. Our new modified algorithms are state-of-the-art, filtering images faster even than previous methods that ignore boundary conditions.

Keywords: parallel recursive filtering, infinite extension, GPUs

Concepts: • **Computing methodologies** → **Image processing;**

1 Introduction

Linear time-invariant (LTI) filtering is a fundamental operation in signal and image processing. In the frequency domain, an LTI filter multiplies the transform of the input by the filter's frequency response. In the time domain, it convolves the input with the filter's impulse response. Filters can be designed, for example, to enhance or attenuate high frequencies, or to eliminate or isolate a particular frequency band in the input.

Many applications use LTI filters that can be expressed as linear, constant-coefficient difference equations. Let w_k, z_k, for $k \in \mathbb{Z}$ represent the input and output, respectively. Such filters satisfy

$$\sum_{i=-r}^{r} a_i z_{k-i} = \sum_{i=-s}^{s} b_i w_{k-i}, \text{ with } a_i \text{ and } b_i \text{ design parameters.} \quad (1)$$

SA '16 Technical Papers, December 05–08, 2016, Macao
ISBN: 978-1-4503-4514-9/16/12.
DOI: http://dx.doi.org/10.1145/2980179.2980222

Using standard signal-processing tools, we can decompose (1) into a convolution pass and a causal/anticausal combination of recursive filter passes (see [Oppenheim and Schafer 2010, chapter 6]):

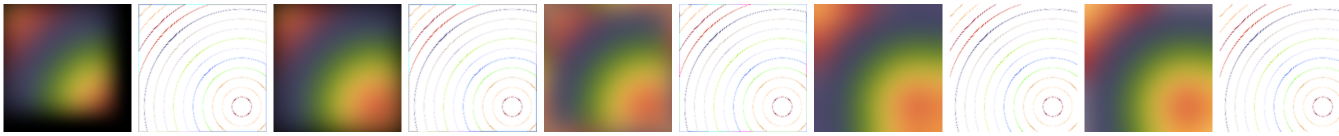
$$x_k = \sum_{i=-s}^{s} c_i w_{k-i}, \quad (2)$$

$$y_k = x_k - \sum_{i=1}^{r} d_i y_{k-i} \quad \text{and} \quad z_k = y_k - \sum_{i=1}^{r} e_i z_{k+i}. \quad (3)$$

The convolution (2) has a finite impulse response (FIR) that is given by the coefficients c_i. In contrast, the recursive filters in (3), characterized by the feedback coefficients d_i and e_i, can have infinite impulse responses (IIR). The process can be extended from 1D to 2D by independently filtering all columns and then all resulting rows.

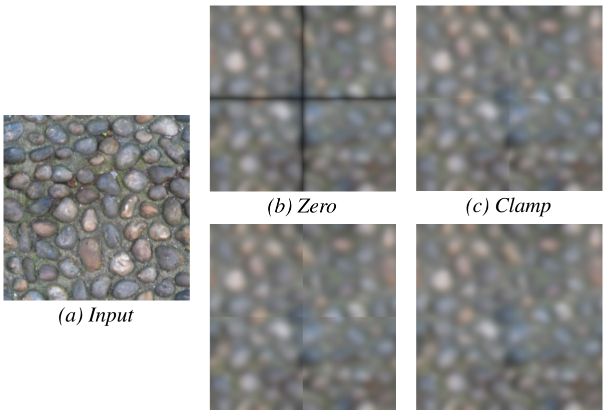
In sequential processing, an input of length n can be filtered in $O(sn + rn)$ time in the time domain or in $O(n \log n)$ time in the frequency domain via the fast Fourier transform (FFT). It follows that we should implement filters with compactly supported impulse responses directly as convolutions. Otherwise, we can try to reproduce the effect of long impulse responses using as few feedback coefficients as possible and proceed by recursive filtering in the time domain. If these alternatives are not viable, we should proceed in the frequency domain using the FFT. The same guidelines apply to parallel processing on modern GPUs using state-of-the-art convolution, FFT, and recursive filtering algorithms [Podlozhnyuk 2007; Govindaraju et al. 2008; Nehab et al. 2011].

Recursive filters can be used to invert the effect of direct convolutions. Almost all generalized sampling algorithms depend on this principle [Nehab and Hoppe 2014]. One of the most important applications is in image quasi-interpolation, where pre-processing with recursive filters enables the design of strategies [Unser et al. 1991; Blu et al. 2001; Condat et al. 2005; Sacht and Nehab 2015] that significantly outperform those typically used in computer graphics applications [Catmull and Rom 1974; Duchon 1979; Mitchell and Netravali 1988]. Recursive filters also play a key role in the optimal approximation of continuous signals by uniformly sampled data, in terms of the L^2 metric [Kajiya and Ullner 1981; Hummel 1983; Unser and Aldroubi 1994]. This idea forms the basis of algorithms for optimal image pyramids [Unser et al. 1993], scaling [Unser et al. 1995a], translation (and rotation) [Unser et al. 1995b], and derivatives [Condat and Möller 2011]. In addition, some of the highest-quality anti-aliasing strategies [McCool 1995; Nehab and Hoppe 2014] are based on post-processing with recursive filters.



(a) Ignore *(b) Zero* *(c) Repeat* *(d) Mirror* *(e) Clamp*

Figure 2: *Infinite extension effect on low-pass filters (left) and high-pass filters (right). Note the different behavior near image boundaries. (a) Feedback assumed to be zero. (b) Extension with zeros. (c) Periodic extension (repeat). (d) Even-periodic extension (mirror/reflect). (e) Extension with boundary colors (clamp-to-edge).*



(a) Input *(b) Zero* *(c) Clamp* *(d) Reflect* *(e) Repeat*

Figure 3: *In this example, a texture (a) that was designed for vertical and horizontal tiling is blurred using a recursive filter. (b) Extending the image with zero, (c) clamping-to-edge, or (d) enforcing even-periodicity can cause serious tiling artifacts (centered for clarity). (e) Respecting the input image periodicity produces correct results.*

(a) Input *(b) Repeat* *(c) Reflect*

Figure 4: *Here, a non-periodic input image (a) is downsampled to 1/8th original resolution and then upsampled back using generalized kernels that require recursive filtering. The periodic extension (b) mixes spatially unrelated data. This causes severe ringing near boundaries. The even-periodic extension (c) avoids the problem.*

Another immediate application is in low-pass filtering. Indeed, recursive approximations to Gaussian filtering [van Vliet et al. 1998] operate in linear time independent of the standard deviation. They are the best alternative in terms of performance, quality, and simplicity. Moreover, "frequency-selective filters of the lowpass, highpass, bandpass, and bandstop types can be obtained from a lowpass discrete-time filter" [Oppenheim and Schafer 2010, chapter 7].

There is, however, a frequently ignored difficulty. To see it, consider finite input and output, i.e., $k \in \{0, \ldots, n-1\}$ in (1)–(3). It is clear that (2) depends on out-of-bounds inputs. Even worse, (3) depends on out-of-bounds *outputs*. Although assuming the input to be zero out-of-bounds often makes sense, setting the initial recursive feedbacks (i.e. outputs) to zero is rarely meaningful. We must be able to extend the input arbitrarily according to a *boundary condition*.

The strategy seen in figure 1, where the boundary colors are extended to infinity, is frequently called "clamp-to-edge". One of its advantages is that it avoids mixing values that are not close together. The downside is that the output does not satisfy the same boundary condition as the input. Another option is to assume the input is periodic (figure 5). The periodic extension simply *repeats* the image, whereas the even-periodic extension *reflects* the image in each repetition, as in a *mirror*. Filtering periodic input results in periodic output. To preserve even-periodicity, the filter must be symmetric.

The most obvious difference between these extension alternatives can be seen near output image boundaries. Figure 2 shows that boundary conditions can cause unrelated data to mix across boundaries when low-pass filtering, or create false edges when high-pass filtering. In other applications, this mixing is not only acceptable but required. For example, in 360° panoramic images, or in textures designed to be tiled, the extension must be periodic. This is shown in figure 3, where blurring an input texture with the wrong extension causes tiling artifacts. Figure 4 shows that the best extension for resampling non-periodic input is the even-periodic extension.

Since each extension has its use, a general solution to this problem must be found. The traditional approach is as follows. Assume the impulse response of the filter is essentially zero outside a compact support from $-\rho$ to ρ. Pad the input on either side with ρ samples selected according to the chosen boundary condition, then filter the extended input from $-\rho$ to $n+\rho-1$ assuming the input and output are both zero outside this extended range.

The issue with padding is that recursive filters are useful precisely because they efficiently implement filters with very long impulse responses. In such cases, respecting boundary conditions becomes particularly important in preventing visible artifacts. At the same time, extending the input on all sides by the effective support width of the impulse response can become computationally prohibitive.

Interestingly, the frequency domain implementation operates on an *infinite* periodic extension of the input. When using the discrete Fourier transform (DFT), the extension is periodic. When using the type-II discrete cosine transform (DCT), the extension is even-periodic [Martucci 1994]. This suggests it should be possible to work with infinite input extensions in the time domain as well!

The difficult part of the problem is determining the initial feedbacks for the recursive filter passes in (3). Our first key contribution is a solution to this problem. We present explicit formulas for computing the initial feedbacks for all infinite extensions seen in figure 2. One caveat is that our formulas require an additional filtering pass over the input. Fortunately, parallel recursive filtering algorithms already perform two passes over the input. By designing our formulas to depend exclusively on quantities available after the first pass, we were able to create novel algorithms that filter infinite extensions at little or no performance penalty. This is our second key contribution.

In summary, we present the first recursive filtering algorithms that compute the exact filtering of infinite input extensions. Moreover, our massively parallel implementations run much faster even than earlier work that ignores the initialization problem at the boundaries.

2 Related work

Since the impulse response of stable LTI filters decay exponentially, the contribution of any input element to a given output becomes negligible as the distance between them increases. Given a specific filter and a target error bound, it is possible to determine the length of input padding needed to respect the error bound over the entire output. This idea is used by Unser et al. [1991] for the initialization of the parallel decomposition of the fast-decaying recursive filters used in B-spline interpolation. Unfortunately, the method becomes impractical in slowly decaying filters (e.g. based on low-pass filters) or at high bit depths (e.g. in floating-point HDR images) due to the additional computation required over the long paddings.

As we mentioned in the introduction, the choice of transform defines the infinite extension implied by the frequency domain implementation. Our focus in this work is on filters with few feedback coefficients, where the time-domain computation can be significantly faster (both asymptotically as well as in practice). Additionally, our method supports constant padding.

Another approach is to interpret (1) as a linear system. The decomposition into the causal and anticausal recursive passes in (3), typically performed by factoring of the Z-transform of (1), is nothing but an LU-style factorization of the corresponding (infinite) Toeplitz matrix. In the finite case, the matrix is not Toeplitz, since the imposition of an input extension disturbs the matrix structure. Nevertheless, we can obtain an LU-factorization and use it to solve the linear system. This idea has been used on narrow-support filters in the case of the even-periodic extension, where coefficients a_i are also symmetric about $i = 0$ [Weickert et al. 1998; Appleton and Talbot 2003]. The only inconvenience is that the feedback coefficients d_{ik} and e_{ik} are now functions of the input index k. Depending on the filter of interest and the target error bound, we may be able to exploit the fact that these coefficients converge as fast as the impulse response decays [Malcolm and Palmer 1974; Nehab and Hoppe 2014].

The final alternative is to compute the exact initialization. Unser et al. [1991] obtain the exact anticausal initialization for the cascaded decomposition of cubic B-spline interpolation filters in the even-periodic extension case (causal initialization is still approximated by padding). Triggs and Sdika [2006] derive exact initialization formulas for cascaded causal and anticausal passes in the clamp-to-edge extension. Our work allows for the exact initialization of cascaded causal and anticausal passes in two dimensions, under different infinite extensions, and for filters of any order. Furthermore, we designed our formulas to integrate seamlessly with state-of-the-art parallel recursive filtering algorithms. Gastal and Oliveira [2015] use the exact initialization of first-order filters in the clamp-to-edge extension to solve the special case of high-order filters whose parallel decompositions consist of sums of first-order filters (i.e., no repeated poles). The initialization of causal-anticausal cascaded decompositions used by parallel algorithms is much more difficult.

Early work on parallel recursive filtering focused on prefix-sums (or *scans*) [Iverson 1962; Stone 1971; Blelloch 1990; Sengupta et al. 2007; Dotsenko et al. 2008; Merrill and Grimshaw 2009]. Nehab et al. [2011] presented a series of algorithms that combine previous parallelization strategies [Stone 1973; Kooge and Stone 1973; Sung and Mitra 1986, 1992] with *overlapping*, a novel IO-saving procedure that leads to greatly improved performance on filter cascades. Chaurasia et al. [2015] then generalized for different cascades types and introduced automatic code generation to explore the best performing combinations. Maximo [2015] generalized to combine the recursive filtering and direct convolution steps. These works, however, assume initial feedbacks to be zero and therefore require the use of padding. Our goal is to provide exact initializations in a form that integrates within these parallelization frameworks, thereby eliminating the final shortcoming.

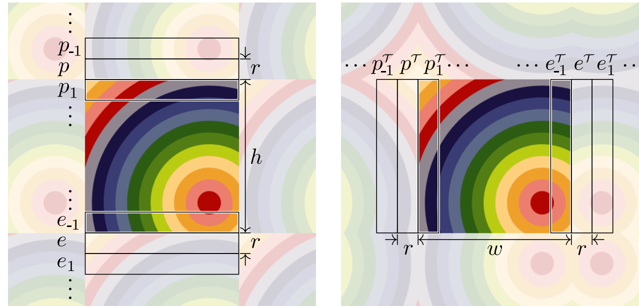


Figure 5: *Periodic extension (left) and even-periodic extension (right) of an $h \times w$ input image. Blocks p, e, p^T and e^T with r rows (and r columns) near input boundaries are also shown.*

3 Background

Given the connection between our work and the parallel recursive filtering framework introduced by Nehab et al. [2011], in the interest of making our text as self-contained as possible, we include below a review of notation, important formulas, and their two key block-parallel algorithms. We begin by defining the initialization problem.

3.1 Problem statement

Our focus is on the cascades of recursive filters that are typical in image-processing applications. These are defined by causal (forward) and anticausal (reverse) recursive filters of order r, to be applied in sequence to the columns and then to the rows of an image.

For column processing, a causal filter $F : \mathbb{R}^r \times \mathbb{R}^h \to \mathbb{R}^h$ takes a *prologue* vector $\boldsymbol{p} \in \mathbb{R}^r$ and an input vector $\boldsymbol{x} \in \mathbb{R}^h$ of size h, and produces an output vector $\boldsymbol{y} = F(\boldsymbol{p}, \boldsymbol{x})$, with the same size as the input. The prologue is $p_{r\text{-}k} = y_{\text{-}k}$, with $k \in \{1, \ldots, r\}$. An anticausal filter $R : \mathbb{R}^h \times \mathbb{R}^r \to \mathbb{R}^h$ is analogous. It takes an input vector $\boldsymbol{x} \in \mathbb{R}^h$ and an *epilogue* vector $\boldsymbol{e} \in \mathbb{R}^r$, and produces an output vector $\boldsymbol{z} = R(\boldsymbol{y}, \boldsymbol{e})$. The epilogue is $e_{k\text{-}1} = z_{h\text{-}1+k}$, with $k \in \{1, \ldots, r\}$. The input/output relationships are given by (3).

The notation for F and R can be "overloaded" to operate independently on all columns of an $h \times w$ input matrix X and an $r \times w$ prologue or epilogue matrix, so that $F\colon \mathbb{R}^{r\times w} \times \mathbb{R}^{h\times w} \to \mathbb{R}^{h\times w}$ and $R\colon \mathbb{R}^{h\times w} \times \mathbb{R}^{r\times w} \to \mathbb{R}^{h\times w}$. For row processing, let F^T and R^T denote the analogous filters that independently operate on all rows of an $h \times w$ input matrix X and an $h \times r$ prologue or epilogue matrix: $F^\mathsf{T}\colon \mathbb{R}^{h\times r} \times \mathbb{R}^{h\times w} \to \mathbb{R}^{h\times w}$ and $R^\mathsf{T}\colon \mathbb{R}^{h\times w} \times \mathbb{R}^{h\times r} \to \mathbb{R}^{h\times w}$.

Given a matrix X, extended to infinity, it is helpful to define notation for selecting blocks of r rows from the extension of X. To that end, let $p_0(X)$ denote the block of r rows preceding the extension of X and let $e_0(X)$ denote the block or r rows following the extension of X. Conversely, let $p_{i\text{-}1}(X)$ and $p_{i+1}(X)$ denote the blocks preceding and following $p_i(X)$, respectively, for $i \in \mathbb{Z}$. Define $e_i(X)$ analogously. Finally, let $p_i^\mathsf{T}(X)$ and $e_i^\mathsf{T}(X)$ denote blocks of r *columns* placed at analogous positions. Assume index $i = 0$ when omitted. These blocks can be seen in figure 5.

With these definitions, the cascaded recursive filter takes an input matrix X and initial feedbacks $p(Y)$, $e(Z)$, $p^\mathsf{T}(U)$, and $e^\mathsf{T}(V)$ to produce an output matrix V, where

$$Y = F\big(p(Y), X\big), \qquad Z = R\big(Y, e(Z)\big), \tag{4}$$

$$U = F^\mathsf{T}\big(p^\mathsf{T}(U), Z\big), \quad \text{and} \quad V = R^\mathsf{T}\big(U, e^\mathsf{T}(V)\big). \tag{5}$$

See figure 6 for an illustration. Given a boundary condition and the input X, our goal is to obtain explicit formulas for the unknown feedbacks $p(Y)$, $e(Z)$, $p^\mathsf{T}(U)$, and $e^\mathsf{T}(V)$ that are needed to generate the finite output V using equations (4)–(5) as if all filter passes were applied to the infinite extension of the input X.

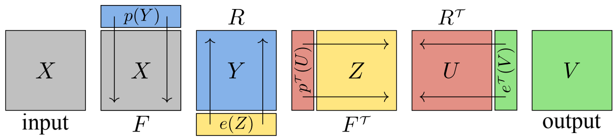


Figure 6: *Cascade of recursive filters applied to an input matrix X. In order to apply the filter passes F, R, $F^{\mathcal{T}}$ and $R^{\mathcal{T}}$, we must determine the initial feedbacks $p(Y)$, $e(Z)$, $p^{\mathcal{T}}(U)$, and $e^{\mathcal{T}}(V)$.*

3.2 Matrix notation and superposition

Grouping input and output elements into r-vectors allows us to use matrices to express the effect of recursive filters in an order-agnostic notation. Consider the causal filter F and let

$$\dot{\boldsymbol{y}}_i = \begin{bmatrix} y_{i\text{-}r} \\ \vdots \\ y_{i\text{-}2} \\ y_{i\text{-}1} \end{bmatrix}, \quad \dot{\boldsymbol{x}}_i = \begin{bmatrix} x_{i\text{-}r} \\ \vdots \\ x_{i\text{-}2} \\ x_{i\text{-}1} \end{bmatrix} \quad \text{and} \quad \ddot{\boldsymbol{x}}_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ x_{i\text{-}1} \end{bmatrix}. \tag{6}$$

It is easy to verify that if

$$\begin{aligned} &\dot{\boldsymbol{y}}_0 = \boldsymbol{p}, \quad \text{then} \\ &\dot{\boldsymbol{y}}_i = \ddot{\boldsymbol{x}}_i + \boldsymbol{A}_F\, \dot{\boldsymbol{y}}_{i\text{-}1}, \quad \text{where} \end{aligned} \tag{7}$$

$$\boldsymbol{A}_F = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ -d_r & -d_{r\text{-}1} & \cdots & -d_2 & -d_1 \end{bmatrix}. \tag{8}$$

A better expression involves $\dot{\boldsymbol{x}}_i$ rather than $\ddot{\boldsymbol{x}}_i$. Iterating (7):

$$\dot{\boldsymbol{y}}_i = \ddot{\boldsymbol{x}}_i + \boldsymbol{A}_F \dot{\boldsymbol{y}}_{i\text{-}1} = \ddot{\boldsymbol{x}}_i + \boldsymbol{A}_F \ddot{\boldsymbol{x}}_{i\text{-}1} + \boldsymbol{A}_F^2 \dot{\boldsymbol{y}}_{i\text{-}2} = \cdots \tag{9}$$

$$= \left(\sum_{k=0}^{r\text{-}1} \boldsymbol{A}_F^k \ddot{\boldsymbol{x}}_{i\text{-}k} \right) + \boldsymbol{A}_F^r \dot{\boldsymbol{y}}_{i\text{-}r} \tag{10}$$

$$= \bar{\boldsymbol{A}}_F \dot{\boldsymbol{x}}_i + \boldsymbol{A}_F^r \dot{\boldsymbol{y}}_{i\text{-}r}. \tag{11}$$

Column k of matrix $\bar{\boldsymbol{A}}_F$, $k \in \{0, 1, \ldots, r-1\}$, is given by the last column of $\boldsymbol{A}_F^{r\text{-}k\text{-}1}$. In other words, the columns of $\bar{\boldsymbol{A}}_F$ are shifts of the first r entries in the impulse response of F. Analogous matrices $\boldsymbol{A}_R$ and $\bar{\boldsymbol{A}}_R$ can be obtained for an anticausal filter R.

In the matrix formulation, each iteration produces r new output elements from the r previous output elements and r new input elements. For example, given a 2nd-order filter:

$$\boldsymbol{A}_F = \begin{bmatrix} 0 & 1 \\ -d_2 & -d_1 \end{bmatrix}, \tag{12}$$

$$\bar{\boldsymbol{A}}_F = \begin{bmatrix} 1 & 0 \\ -d_1 & 1 \end{bmatrix}, \quad \text{and} \quad \boldsymbol{A}_F^2 = \begin{bmatrix} -d_2 & -d_1 \\ d_1 d_2 & d_1^2 - d_2 \end{bmatrix}. \tag{13}$$

We can also use matrices to express a very useful superposition property that allows us to decouple the influences of the feedback and the input on the output. For $b \in \mathbb{Z}_{>0}$:

$$F\big(P_{r \times b}, X_{b \times b}\big) = F(0, X) + F(P, 0) \tag{14}$$

$$= \boldsymbol{A}_{FP}\, P + \boldsymbol{A}_{FB}\, X, \quad \text{and} \tag{15}$$

$$R\big(Y_{b \times b}, E_{r \times b}\big) = R(Y, 0) + R(0, E) \tag{16}$$

$$= \boldsymbol{A}_{RE}\, E + \boldsymbol{A}_{RB}\, Y, \tag{17}$$

with matrices $\boldsymbol{A}_{FP}$, $\boldsymbol{A}_{FB}$, $\boldsymbol{A}_{RE}$, and $\boldsymbol{A}_{RB}$ defined by formulas

$$(\boldsymbol{A}_{FP})_{b \times r} = F(\boldsymbol{I}_{r \times r}, \boldsymbol{0}_{b \times r}), \tag{18}$$

$$(\boldsymbol{A}_{FB})_{b \times b} = F(\boldsymbol{0}_{r \times b}, \boldsymbol{I}_{b \times b}), \tag{19}$$

$$(\boldsymbol{A}_{RE})_{b \times r} = R(\boldsymbol{0}_{b \times r}, \boldsymbol{I}_{r \times r}), \quad \text{and} \tag{20}$$

$$(\boldsymbol{A}_{RB})_{b \times b} = R(\boldsymbol{I}_{b \times b}, \boldsymbol{0}_{r \times b}). \tag{21}$$

3.3 Block-parallel recursive filtering

The block-parallel algorithms assume zero initial feedback in all filtering passes. In section 3.1, we defined matrices Y, Z, U and V in (4)–(5) as the results of filtering infinite extensions. Therefore, to avoid confusion, we must introduce new notation for matrices produced with zero feedback

$$\mathring{Y} = F(0, X), \qquad \mathring{Z} = R(\mathring{Y}, 0), \tag{22}$$

$$\mathring{U} = F^{\mathcal{T}}(0, \mathring{Z}), \quad \text{and} \quad \mathring{V} = R^{\mathcal{T}}(\mathring{U}, 0). \tag{23}$$

Hiding the large latency in accesses to global GPU memory requires a level of parallelism that surpasses the number of independent rows and columns in a typical image. Therefore, input and output must be split into $M \times N$ blocks of size $b \times b$ for parallel processing. Depending on the input size, the last row and column of blocks may, of course, hold smaller blocks. Let $B_{m,n}(X)$ denote the block starting at row mb and column nb of matrix X. Then, define auxiliary matrices $\overset{\circ\circ}{Y}$, $\overset{\circ\circ}{Z}$, $\overset{\circ\circ}{U}$, and $\overset{\circ\circ}{V}$, block by block, so that:

$$\begin{aligned} B_{m,n}(\overset{\circ\circ}{Y}) &= F\big(0, B_{m,n}(X)\big), \\ B_{m,n}(\overset{\circ\circ}{Z}) &= R\big(B_{m,n}(\overset{\circ\circ}{Y}), 0\big), \\ B_{m,n}(\overset{\circ\circ}{U}) &= F^{\mathcal{T}}\big(0, B_{m,n}(\overset{\circ\circ}{Z})\big), \quad \text{and} \\ B_{m,n}(\overset{\circ\circ}{V}) &= R^{\mathcal{T}}\big(B_{m,n}(\overset{\circ\circ}{U}), 0\big). \end{aligned}$$

The distinction between $\overset{\circ\circ}{Y}$ and $\mathring{Y}$ is that, whereas $\overset{\circ\circ}{Y}$ is computed independently per block, $\mathring{Y}$ considers the sequential inter-block dependencies of the image as a whole. Similar distinctions apply to the other pairs $\overset{\circ\circ}{Z}$ and $\mathring{Z}$, $\overset{\circ\circ}{U}$ and $\mathring{U}$, $\overset{\circ\circ}{V}$ and $\mathring{V}$.

Now define an operator T (for *tail*) that extracts the *last* r rows of its argument. In the same manner, let operator H (for *head*) extract the *first* r rows of its argument. Let $P_{m,n}(X) = T\big(B_{m,n}(X)\big)$ and $E_{m,n}(X) = H\big(B_{m,n}(X)\big)$ extract the row perimeters from block $B_{m,n}(X)$. Likewise, let $P^{\mathcal{T}}_{m,n}(X)$ and $E^{\mathcal{T}}_{m,n}(X)$ extract column perimeters from block $B_{m,n}(X)$.

By repeatedly applying superposition properties (14)–(17), Nehab et al. [2011] proved that

$$P_{m,n}(\mathring{Y}) = T(\boldsymbol{A}_{FP})\, P_{m\text{-}1,n}(\mathring{Y}) + P_{m,n}(\overset{\circ\circ}{Y}), \tag{24}$$

$$\begin{aligned} E_{m,n}(\mathring{Z}) = {} & H(\boldsymbol{A}_{RE})\, E_{m+1,n}(\mathring{Z}) \\ & + H(\boldsymbol{A}_{RB}) \boldsymbol{A}_{FP}\, P_{m\text{-}1,n}(\mathring{Y}) \\ & + E_{m,n}(\overset{\circ\circ}{Z}), \end{aligned} \tag{25}$$

$$P^{\mathcal{T}}_{m,n}(\mathring{U}) = P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})\, T(\boldsymbol{A}_{FP})^T + P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U}), \quad \text{where} \tag{26}$$

$$\begin{aligned} P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U}) = {} & \boldsymbol{A}_{RE} \big(E_{m+1,n}(\mathring{Z})\, T(\boldsymbol{A}_{FB})^T\big) \\ & + (\boldsymbol{A}_{RB} \boldsymbol{A}_{FP}) \big(P_{m\text{-}1,n}(\mathring{Y})\, T(\boldsymbol{A}_{FB})^T\big) \\ & + P^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{U}), \quad \text{and} \end{aligned} \tag{27}$$

$$\begin{aligned} E^{\mathcal{T}}_{m,n}(\mathring{V}) = {} & E^{\mathcal{T}}_{m,n+1}(\mathring{V})\, H(\boldsymbol{A}_{RE})^T \\ & + P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U}) \big(H(\boldsymbol{A}_{RB}) \boldsymbol{A}_{FP}\big)^T \\ & + E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V}), \quad \text{where} \end{aligned} \tag{28}$$

$$\begin{aligned} E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V}) = {} & \boldsymbol{A}_{RE} \big(E_{m+1,n}(\mathring{Z}) \big(H(\boldsymbol{A}_{RB}) \boldsymbol{A}_{FB}\big)^T\big) \\ & + (\boldsymbol{A}_{RB} \boldsymbol{A}_{FP}) \big(P_{m\text{-}1,n}(\mathring{Y}) \big(H(\boldsymbol{A}_{RB}) \boldsymbol{A}_{FB}\big)^T\big) \\ & + E^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{V}). \end{aligned} \tag{29}$$

These formulas enable us to filter the input in three basic stages. The first stage collects perimeters for each block, independently and in parallel. The intermediate stage, which is not as parallel, uses the formulas to transform these perimeters into the initial feedbacks to each block. The final stage loads the blocks and initial feedbacks to produce, independently and in parallel, all blocks of the output.

Here is their fully-overlapped *algorithm 5*:

5.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store block perimeters $P_{m,n}(\overset{\circ\circ}{Y})$, $E_{m,n}(\overset{\circ\circ}{Z})$, $P^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{U})$, and $E^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{V})$;

5.2. In parallel for all n, sequentially for each m, compute and store the feedbacks $P_{m\text{-}1,n}(\mathring{Y})$ and $E_{m+1,n}(\mathring{Z})$ according to equations (24) and (25);

5.3. In parallel for all m, sequentially for each n, compute and store feedbacks $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$ and $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$ according to equations (26) and (28);

5.4. In parallel for all m and n, load input block $B_{m,n}(X)$ and all its block feedbacks $P_{m\text{-}1,n}(\mathring{Y})$, $E_{m+1,n}(\mathring{Z})$, $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$, and $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$. Compute and store $B_{m,n}(\mathring{V})$.

The higher level of parallelism and the bandwidth saved by overlapping all filter passes in the first step lead to a 12× improvement in performance relative to previous GPU recursive filtering strategies.

A simpler alternative to the fully-overlapped algorithm was also proposed by Nehab et al. [2011]. It overlaps only causal-anticausal processing and avoids row-column overlapping.

First note that the blocks from matrices $\overset{\circ\bullet}{U}$ and $\overset{\circ\bullet}{V}$, defined by equations (27) and (29), are:

$$B_{m,n}(\overset{\circ\bullet}{U}) = F^{\mathcal{T}}\big(0, B_{m,n}(\mathring{Z})\big), \quad \text{and} \tag{30}$$

$$B_{m,n}(\overset{\circ\bullet}{V}) = R^{\mathcal{T}}\big(B_{m,n}(\overset{\circ\bullet}{U}), 0\big). \tag{31}$$

In other words, the distinction between $\overset{\circ\bullet}{U}$ and $\overset{\circ\circ}{U}$ is that $\overset{\circ\bullet}{U}$ starts from $\mathring{Z}$ rather than $\overset{\circ\circ}{Z}$; it starts after the column processing has finished for the image as a whole, instead of independently by block.

Here is their causal-anticausal overlapped *algorithm 4*:

4.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store block perimeters $P_{m,n}(\overset{\circ\circ}{Y})$, $E_{m,n}(\overset{\circ\circ}{Z})$;

4.2. In parallel for all n, sequentially for each m, compute and store all feedbacks $P_{m\text{-}1,n}(\mathring{Y})$ and then all $E_{m+1,n}(\mathring{Z})$ according to equations (24) and (25);

4.3. In parallel for all m and n, load block $B_{m,n}(X)$ and column block feedbacks $P_{m\text{-}1,n}(\mathring{Y})$ and $E_{m+1,n}(\mathring{Z})$, compute and store $B_{m,n}(\mathring{Z})$, and then compute and store block perimeters $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ using equations (30) and (31);

4.4. In parallel for all m, sequentially for each n, compute and store all feedbacks $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$ and then all $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$ according to equations (26) and (28);

4.5. In parallel for all m and n, load block $B_{m,n}(\mathring{Z})$ and row block feedbacks $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$, and $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$. Compute and store $B_{m,n}(\mathring{V})$.

The step complexity and bandwidth requirements of algorithms 4_r and 5_r increase with filter order r, more so for algorithm 5 than for algorithm 4. For low-order filters, algorithm 5 should have the advantage. Indeed, Nehab et al. [2011] observed that, on a GTX 480 GPU, their 5_1 implementation is much faster than 4_1. Somewhat disappointingly, already for 2nd order the situation was reversed. They attributed this to an "optimization issue that may be resolved with future hardware, compiler, or implementation".

High-order filters can be decomposed in a variety of different ways into cascades of lower-order filters. Each lower-order filter can be implemented using algorithm 4 or 5. For 3rd-order filtering on a GTX 480 GPU, Nehab et al. [2011] had originally proposed a 5_1+4_2 cascade with kernel fusion. One of the motivations for the work of Chaurasia et al. [2015] was the exploration of this universe of different alternatives in search of the best performing combination. In their implementation, 4_3 (which Chaurasia et al. call 3x_3y) was the fastest alternative for 3rd-order filtering on a GTX Titan GPU.

This was the state of the art as we set out to adapt block-parallel recursive filtering algorithms to filter infinite extensions. In section 5.1, we describe an improvement over algorithm 5 that makes full overlapping the fastest alternative for filter orders 1 through 3.

4 Explicit formulas for initial feedbacks

While deriving our formulas for the initial feedbacks, we face a key additional challenge. To enable the design of efficient algorithms, the formulas can depend only on values available between the first and final steps of the block-parallel recursive filtering algorithms.

We follow the same general approach for all extension types. The idea is to iterate the recurrence relations between input and output. Obtaining the output Y_i from input region X_i requires the last part of the output $Y_{i\text{-}1}$ corresponding to region $X_{i\text{-}1}$ preceding X_i in the extension. This iteration process naturally leads to infinite series involving matrix powers. In each case, we prove that these series converge, then find their limits by solving small linear systems.

The convergence and non-singularity proofs, though important, are not needed in understanding the derivations that follow. We have moved them to appendix A in order to avoid breaking the flow of the text. Even though the derivations themselves are a bit involved, the resulting algorithmic steps, summarized at the end of each section, are quite simple to understand and easy to implement.

All derivations focus on column processing by causal-anticausal cascades. Row-column cascades are considered when we describe the complete block-parallel algorithms in section 5.

4.1 Constant padding extension

When clamping to edge, define $C_\uparrow$ to contain r copies of the first row in X, and $C_\downarrow$ to contain r copies of the last row in X. Otherwise, when padding with an arbitrary constant, define them to contain r copies of any desired row-vector in $\mathbb{R}^w$. The boundary condition dictates $p_{\text{-}i}(X) = C_\uparrow$ and $e_i(X) = C_\downarrow$, for $i \in \{0, 1, 2, \ldots\}$.

Start by expressing prologue $p(Y)$, i.e., the initial feedback for the causal pass of recursive filtering, as a function of $p_{\text{-}1}(Y)$, the output directly preceding it:

$$p(Y) = F\big(p_{\text{-}1}(Y), C_\uparrow\big) \tag{32}$$

$$= \boldsymbol{A}^r_F\, p_{\text{-}1}(Y) + \bar{\boldsymbol{A}}_F\, C_\uparrow. \tag{33}$$

Expanding $p_{\text{-}1}(Y)$, then $p_{\text{-}2}(Y)$ and so on k times, we reach:

$$p(Y) = (\boldsymbol{A}^r_F)^k\, p_{\text{-}k}(Y) + \left(\sum_{i=0}^{k\text{-}1} (\boldsymbol{A}^r_F)^i\right) \bar{\boldsymbol{A}}_F\, C_\uparrow. \tag{34}$$

Taking the limit as $k \to \infty$, theorems A.1 and A.2 guarantee that the first term in (34) vanishes and the Neumann series in the second term converges to $\boldsymbol{S}_F = (\boldsymbol{I} - \boldsymbol{A}^r_F)^{\text{-}1}$. Therefore,

$$p(Y) = \boldsymbol{S}_F\, \bar{\boldsymbol{A}}_F\, C_\uparrow. \tag{35}$$

Equation (35) depends only on the constant padding $C_\uparrow$. All required matrices are readily obtained from filter F. The initialization of the causal pass is therefore complete.

To initialize the anticausal pass, we need to find the value of the epilogue $e(Z)$. Developing the equation along the lines of (32)–(34), we obtain

$$e(Z) = R\big(e(Y), e_1(Z)\big) \tag{36}$$

$$= \boldsymbol{A}^r_R\, e_1(Z) + \bar{\boldsymbol{A}}_R\, e(Y). \tag{37}$$

After k expansions, we reach

$$e(Z) = (\boldsymbol{A}^r_R)^k\, e_k(Z) + \left(\sum_{i=0}^{k\text{-}1} (\boldsymbol{A}^r_R)^i\, \bar{\boldsymbol{A}}_R\, e_i(Y)\right). \tag{38}$$

Equation (38) depends on each of the values $e_i(Y)$ that result from the first pass. We can obtain them as functions of $e_{\text{-}1}(Y)$ and the padding $C_\downarrow$ in a form analogous to (34):

$$e_i(Y) = (\boldsymbol{A}_F^r)^{i+1} e_{\text{-}1}(Y) + \left(\sum_{j=0}^{i} (\boldsymbol{A}_F^r)^j\right) \bar{\boldsymbol{A}}_F\, C_\downarrow \tag{39}$$

$$= (\boldsymbol{A}_F^r)^{i+1} e_{\text{-}1}(Y) + \big(\boldsymbol{I} - (\boldsymbol{A}_F^r)^{i+1}\big)\boldsymbol{S}_F \bar{\boldsymbol{A}}_F\, C_\downarrow. \tag{40}$$

Plugging (40) into (38) and rearranging,

$$\begin{aligned} e(Z) &= (\boldsymbol{A}_R^r)^k\, e_k(Z) \\ &+ \left(\sum_{i=0}^{k\text{-}1} (\boldsymbol{A}_R^r)^i\right) \bar{\boldsymbol{A}}_R \boldsymbol{S}_F \bar{\boldsymbol{A}}_F C_\downarrow \\ &+ \left(\sum_{i=0}^{k\text{-}1} (\boldsymbol{A}_R^r)^i \bar{\boldsymbol{A}}_R (\boldsymbol{A}_F^r)^i\right) \boldsymbol{A}_F^r \left(e_{\text{-}1}(Y) - \boldsymbol{S}_F \bar{\boldsymbol{A}}_F C_\downarrow\right). \end{aligned} \tag{41}$$

Taking (41) to the limit as $k \to \infty$, theorems A.1 and A.2 guarantee the first term vanishes and the Neumann series in the second term converges to $\boldsymbol{S}_R = (\boldsymbol{I} - \boldsymbol{A}_R^r)^{\text{-}1}$.

Theorem A.3 then shows that the limit $\boldsymbol{S}_{RF}$ of the series

$$\boldsymbol{S}_{RF} = \lim_{k\to\infty} \sum_{i=0}^{k} (\boldsymbol{A}_R^r)^i \bar{\boldsymbol{A}}_R\, (\boldsymbol{A}_F^r)^i \tag{42}$$

in the last term of (41) satisfies

$$\boldsymbol{S}_{RF} - \boldsymbol{A}_R^r \boldsymbol{S}_{RF} \boldsymbol{A}_F^r = \bar{\boldsymbol{A}}_R. \tag{43}$$

This is a non-singular $(r \times r) \times (r \times r)$ linear system where the only unknowns are the entries of matrix $\boldsymbol{S}_{RF}$. Matrix $\boldsymbol{S}_{RF}$ plays a similar role to matrix $\mathbf{M}$ described by Triggs and Sdika [2006].

We can now write the formula for epilogue $e(Z)$, i.e., the initial feedback for the anticausal pass of recursive filtering:

$$e(Z) = \boldsymbol{S}_{RF} \boldsymbol{A}_F^r\, e_{\text{-}1}(Y) + (\boldsymbol{S}_R \bar{\boldsymbol{A}}_R - \boldsymbol{S}_{RF} \boldsymbol{A}_F^r) \boldsymbol{S}_F \bar{\boldsymbol{A}}_F\, C_\downarrow. \tag{44}$$

It depends only on $e_{\text{-}1}(Y)$, the last r rows of output during the causal pass, and on the constant padding $C_\downarrow$. Since $e_{\text{-}1}(Y)$ can be obtained with a simple modification of the intermediate stages of the block-parallel algorithm, the initialization of the anticausal pass is complete.

Summary of the constant padding extension

Precomputed $r \times r$ matrices
$\boldsymbol{S}_F = (\boldsymbol{I} - \boldsymbol{A}_F^r)^{\text{-}1}$, $\boldsymbol{S}_R = (\boldsymbol{I} - \boldsymbol{A}_R^r)^{\text{-}1}$, $\boldsymbol{S}_{RF}$, where $\boldsymbol{S}_{RF} - \boldsymbol{A}_R^r \boldsymbol{S}_{RF} \boldsymbol{A}_F^r = \bar{\boldsymbol{A}}_R$, $\boldsymbol{S}_F \bar{\boldsymbol{A}}_F$, $\boldsymbol{S}_{RF} \boldsymbol{A}_F^r$, and $(\boldsymbol{S}_R \bar{\boldsymbol{A}}_R - \boldsymbol{S}_{RF} \boldsymbol{A}_F^r)\boldsymbol{S}_F \bar{\boldsymbol{A}}_F$
Inputs
$C_\uparrow$ with r copies of top row of constant padding $C_\downarrow$ with r copies of bottom row of constant padding $e_{\text{-}1}(Y) = H\big(F\big(p(Y), X\big)\big)$ assumed given
Causal and anticausal feedback computation
1. Compute $p(Y)$ from $C_\uparrow$ using (35) 2. Compute $e(Z)$ from $e_{\text{-}1}(Y)$ and $C_\downarrow$ using (44)

4.2 Periodic extension

The periodic extension simply tiles the space with copies of the input image, as shown in figure 5. Given an input matrix $X = X_0$, let $X_{i\text{-}1}$ and X_{i+1} denote the matrices preceding and following X_i in the extension, respectively, for $i \in \mathbb{Z}$.

To find the initial feedback $p(Y)$ to the causal pass, observe that

$$p(Y) = T\Big(F\big(p(Y_{\text{-}1}), X_{\text{-}1}\big)\Big) \tag{45}$$

$$= T\Big(F\big(p(Y_{\text{-}1}), 0\big)\Big) + T\big(F(0, X)\big), \quad \text{since } X = X_{\text{-}1} \tag{46}$$

$$= \boldsymbol{A}_F^h\, p(Y_{\text{-}1}) + e_{\text{-}1}(\mathring{Y}). \tag{47}$$

To see why $T\Big(F\big(p(Y_{\text{-}1}), 0\big)\Big) = \boldsymbol{A}_F^h\, p(Y_{\text{-}1})$, set $\ddot{\boldsymbol{x}}_i = 0$ in equation (7) and iterate h times.

Back to equation (47), we now iterate k times to obtain

$$p(Y) = (\boldsymbol{A}_F^h)^k p(Y_{\text{-}k}) + \sum_{i=0}^{k\text{-}1} (\boldsymbol{A}_F^h)^k\, e_{\text{-}1}(\mathring{Y}). \tag{48}$$

Taking the limit as $k \to \infty$, theorems A.1 and A.2 guarantee that the first term in (48) vanishes and the Neumann series in the second term converges to $(\boldsymbol{I} - \boldsymbol{A}_F^h)^{\text{-}1}$, and so

$$p(Y) = (\boldsymbol{I} - \boldsymbol{A}_F^h)^{\text{-}1}\, e_{\text{-}1}(\mathring{Y}). \tag{49}$$

Equation (49) depends only on $e_{\text{-}1}(\mathring{Y})$, which can be obtained from the intermediate stages of the block-parallel algorithm. This means that the initialization of the causal pass is complete.

The output for the causal pass on periodic input must also be periodic because $p(Y_k) = p(Y_{k\text{-}1})$, for $k \in \mathbb{Z}$. In order to establish an analogy with the causal case, we need additional definitions

$$\dot{Z} = R(Y, 0), \quad \dot{U} = F^{\mathcal{T}}(Z, 0), \quad \text{and} \quad \dot{V} = R^{\mathcal{T}}(U, 0). \tag{50}$$

The distinction between these definitions and those for $\mathring{Z}$, $\mathring{U}$, and $\mathring{V}$ is that the new definitions assume the correct initial feedbacks were used (instead of zero) except for the last filter pass. By analogy

$$e(Z) = (\boldsymbol{I} - \boldsymbol{A}_R^h)^{\text{-}1}\, p_1(\dot{Z}). \tag{51}$$

Since $p_1(\dot{Z})$ can be computed from information available after the intermediate stages of the block-parallel algorithm, the initialization of the anticausal pass is complete.

Summary of the periodic extension

Precomputed $r \times r$ matrices
$(\boldsymbol{I} - \boldsymbol{A}_F^h)^{\text{-}1}$ and $(\boldsymbol{I} - \boldsymbol{A}_R^h)^{\text{-}1}$
Inputs
$e_{\text{-}1}(\mathring{Y}) = T\big(F(0, X)\big)$ assumed given $p_1(\dot{Z}) = H\big(R(Y, 0)\big)$ assumed given
Causal and anticausal feedback computation
1. Compute $p(Y)$ from $e_{\text{-}1}(\mathring{Y})$ using (49) 2. Compute $e(Z)$ from $p_1(\dot{Z})$ using (51)

4.3 Even-periodic extension

Combined causal/anticausal recursive filters are, more often than not, linear phase. A filter has linear phase if and only if its impulse response is symmetric. In the frequency domain, this corresponds to a real frequency response. For recursive filters, this means F and R use the same feedback coefficients $[\, d_1 \ \cdots \ d_r \,]$. This restriction allows us to compute the initial feedbacks for filtering even-periodic infinite extensions as follows.

First, a *conceptual* strategy. Construct the even-periodic extension, depicted in figure 5, by first concatenating the input with its mirror reflection. This process yields an image twice as large, to which we can then apply the simpler periodic extension. We can then use the results we obtained in the previous section.

Let $K(\boldsymbol{X})$ denote the reversion of entries in each column of X and let $X_1 \mathbin{+\!\!\!+} X_2$ denote column-by-column concatenation. Applying equation (49) to the periodic extension of $X \mathbin{+\!\!\!+} K(X)$,

$$p(Y) = (\boldsymbol{I} - \boldsymbol{A}_F^{2h})^{\text{-}1}\, T\big(F(0, X \mathbin{+\!\!\!+} K(X))\big). \tag{52}$$

Moreover, note that

$$\begin{aligned} T\big(F(0, X \mathbin{+\!\!\!+} K(X))\big) &= T\big(F(e_{\text{-}1}(\mathring{Y}), K(X))\big) && (53)\\ &= \boldsymbol{A}_F^h\, e_{\text{-}1}(\mathring{Y}) + T\big(F(0, K(X))\big) && (54)\\ &= \boldsymbol{A}_F^h\, e_{\text{-}1}(\mathring{Y}) + \boldsymbol{K} H\big(R(X,0)\big), && (55)\end{aligned}$$

where $\boldsymbol{K}$ denotes the $r \times r$ *exchange matrix* (i.e., the identity matrix with rows reversed).

One of the values we need, $e_{\text{-}1}(\mathring{Y})$, is readily available after the intermediate passes of the block-parallel algorithm. The other value, $H\big(R(X,0)\big)$, poses a problem because the anticausal filter R is never applied directly over the input X. Instead, it is applied over the result Y of the causal pass. In other words, we have access to $F(0,X)$ and $R\big(F(0,X),0\big)$, but not to $R(X,0)$.

To solve this problem, we use theorem A.4. This key theorem relates the feedbacks needed to equate the two application orders for the cascade, namely $F \circ R$ and $R \circ F$. The theorem states that

$$\mathring{Z} = R\big(F(0,X),0\big) = F\big(P', R(X,E')\big) \tag{56}$$

with

$$P' = R(0, H(\mathring{Z})) \quad \text{and} \quad 0 = F\big(T(\mathring{Z}), E'\big). \tag{57}$$

The new application order gives

$$\begin{aligned} p_1(\mathring{Z}) &= H\Big(R\big(F(0,X),0\big)\Big) && (58)\\ &= H\Big(F\big(P', R(X,E')\big)\Big) && (59)\\ &= \boldsymbol{A}_F^r P' + \bar{\boldsymbol{A}}_F H\big(R(X,E')\big) && (60)\\ &= \boldsymbol{A}_F^r P' + \bar{\boldsymbol{A}}_F \Big(H\big(R(X,0)\big) + \boldsymbol{A}_R^h E'\Big). && (61)\end{aligned}$$

Therefore, given values for P' and E' we can obtain the needed

$$H\big(R(X,0)\big) = (\bar{\boldsymbol{A}}_F)^{\text{-}1}\big(p_1(\mathring{Z}) - \boldsymbol{A}_F^r P'\big) - \boldsymbol{A}_R^h E'. \tag{62}$$

Note that theorem A.5 guarantees that $\bar{\boldsymbol{A}}_F$ is non-singular.

The value of P' comes directly from (57):

$$P' = R\big(0, H(\mathring{Z})\big) = \boldsymbol{A}_R^r\, p_1(\mathring{Z}). \tag{63}$$

To obtain E', first note that

$$T(\mathring{Z}) = T\big(R(\mathring{Y},0)\big) = R\big(T(\mathring{Y}),0\big) = \bar{\boldsymbol{A}}_R\, e_{\text{-}1}(\mathring{Y}). \tag{64}$$

Then, substitute into (57):

$$0 = F\big(T(\mathring{Z}), E'\big) = \bar{\boldsymbol{A}}_F E' + \boldsymbol{A}_F^r \bar{\boldsymbol{A}}_R\, e_{\text{-}1}(\mathring{Y}). \tag{65}$$

The result is that

$$E' = -(\bar{\boldsymbol{A}}_F)^{\text{-}1} \boldsymbol{A}_F^r \bar{\boldsymbol{A}}_R\, e_{\text{-}1}(\mathring{Y}). \tag{66}$$

Substituting (63) and (66) into (62), (62) into (55), and (55) into (52), we obtain the initial feedback for the causal pass:

$$\begin{aligned} p(Y) = (\boldsymbol{I} - \boldsymbol{A}_F^{2h})^{\text{-}1}\Big(&\boldsymbol{K}(\bar{\boldsymbol{A}}_F)^{\text{-}1}(\boldsymbol{I} - \boldsymbol{A}_F^r \boldsymbol{A}_R^r)\, p_1(\mathring{Z}) + {}\\ &\big(\boldsymbol{A}_F^h + \boldsymbol{K}\boldsymbol{A}_R^h (\bar{\boldsymbol{A}}_F)^{\text{-}1} \boldsymbol{A}_F^r \bar{\boldsymbol{A}}_R\big)\, e_{\text{-}1}(\mathring{Y})\Big). \end{aligned} \tag{67}$$

We must now find the initial feedback for the anticausal filter pass. To do so, we first observe that, when the impulse response is symmetric, the result of applying a filter to an even-periodic input is itself even-periodic. From the even-periodicity of the output Z,

$$e_{\text{-}1}(Z) = \boldsymbol{K} e(Z). \tag{68}$$

On the other hand,

$$e_{\text{-}1}(Z) = R\big(e_{\text{-}1}(Y), e(Z)\big) = \bar{\boldsymbol{A}}_R\, e_{\text{-}1}(Y) + \boldsymbol{A}_R^r\, e(Z). \tag{69}$$

Substituting and regrouping, we obtain

$$e(Z) = \boldsymbol{L} e_{\text{-}1}(Y) \quad \text{with} \quad \boldsymbol{L} = (\boldsymbol{K} - \boldsymbol{A}_R^r)^{\text{-}1} \bar{\boldsymbol{A}}_R. \tag{70}$$

Since $\boldsymbol{K} - \boldsymbol{A}_R^r = \boldsymbol{K}(\boldsymbol{I} - \boldsymbol{K}\boldsymbol{A}_R^r)$, theorem A.6 shows that matrix $\boldsymbol{K} - \boldsymbol{A}_R^r$ is non-singular and therefore matrix $\boldsymbol{L}$ always exists.

Finally, we eliminate $e_{\text{-}1}(Y)$ from (70) by recalling that

$$e_{\text{-}1}(Y) = \boldsymbol{A}_F^h\, p(Y) + e_{\text{-}1}(\mathring{Y}) \tag{71}$$

to obtain the final formula

$$e(Z) = \boldsymbol{L}\big(\boldsymbol{A}_F^h\, p(Y) + e_{\text{-}1}(\mathring{Y})\big). \tag{72}$$

Summary of the even-periodic extension

Precomputed $r \times r$ matrices
$(\bar{\boldsymbol{A}}_F)^{\text{-}1}$, $(\boldsymbol{I} - \boldsymbol{A}_F^{2h})^{\text{-}1}$, $\boldsymbol{K}(\bar{\boldsymbol{A}}_F)^{\text{-}1}(\boldsymbol{I} - \boldsymbol{A}_F^r \boldsymbol{A}_R^r)$, $\boldsymbol{A}_F^h$ $\boldsymbol{A}_F^h + \boldsymbol{K}\boldsymbol{A}_R^h(\bar{\boldsymbol{A}}_F)^{\text{-}1}\boldsymbol{A}_F^r \bar{\boldsymbol{A}}_R$, and $\boldsymbol{L} = (\boldsymbol{K} - \boldsymbol{A}_R^r)^{\text{-}1}\bar{\boldsymbol{A}}_R$
Inputs
$e_{\text{-}1}(\mathring{Y}) = T\big(F(0,X)\big)$ assumed given $p_1(\mathring{Z}) = H\big(R(\mathring{Y},0)\big)$ assumed given
Causal and anticausal feedback computation
1. Compute $p(Y)$ from $e_{\text{-}1}(\mathring{Y})$ and $p_1(\mathring{Z})$ using (67) 2. Compute $e(Z)$ from $p(Y)$ and $e_{\text{-}1}(\mathring{Y})$ using (72)

5 Modified block-parallel algorithms

Instead of directly adapting the code provided by Nehab et al. [2011] to use our exact initial feedback formulas, we first made several key improvements. Our modifications increased performance by more than 50% and made the code more general.

5.1 Our improved baseline algorithm

We first made the code agnostic to filter order. This was just a matter of carefully implementing all formulas using C++ templates. We tested our code unmodified up to order 20. It is true that most useful filters are lower order, and that for high orders the best performance may come from cascading low-order filters, as Chaurasia et al. [2015] advocate. However, it is not always possible to filter infinite extensions using arbitrary cascades. The periodic extension can be decomposed arbitrarily because the output of each filter pass preserves the same periodicity. The even-periodic extension offers less freedom because, in order to preserve periodicity, a symmetric high-order causal-anticausal filter pair must be decomposed into a sequence of symmetric causal-anticausal filter pairs. It is unclear how to decompose the constant padding extension, since the output of any filter pass will not, in general, be constant out of bounds.

Then, we performed a few low-level optimizations. For example, we keep all precomputed $b \times r$ matrices in global memory, rather than in constant memory, to prevent serialized access. Other changes were motivated by architectural differences between the GTX 480 GPU that Nehab et al. [2011] targeted and the GTX Titan GPU we (and Chaurasia et al. [2015]) target. The simplest modification was using the new `_ldg` intrinsic to take advantage of the read-only data cache. The GTX Titan has twice as many registers and 6 times more cores per SMX than the GTX 480. To take advantage of these changes, we modified the intra-block steps to perform computations entirely in registers, rather than in shared memory. We also modified the inter-block steps to use multiple computing warps per block.

The most profound change addresses one of the reasons why algorithm 5 performs poorly in higher orders: the lack of parallelism in step 5.3 and the computational cost increase of (27) and (29) with order. We therefore split step 5.3 in two in our new *algorithm 6*:

6.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store block perimeters $P_{m,n}(\mathring{\mathring{Y}})$, $E_{m,n}(\mathring{\mathring{Z}})$, $P^{\mathcal{T}}_{m,n}(\mathring{\mathring{U}})$, and $E^{\mathcal{T}}_{m,n}(\mathring{\mathring{V}})$;

6.2. In parallel for all n, sequentially for each m, compute and store all feedbacks $P_{m-1,n}(\mathring{Y})$ and $E_{m+1,n}(\mathring{Z})$ according to equations (24) and (25);

6.3. In parallel for all m and n, compute and store all $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ according to equations (27) and (29);

6.4. In parallel for all m, sequentially for each n, compute and store feedbacks $P^{\mathcal{T}}_{m,n-1}(\mathring{U})$ and then all $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$ according to equations (26) and (28);

6.5. In parallel for all m and n, load input block $B_{m,n}(X)$ and all its block feedbacks $P_{m-1,n}(\mathring{Y})$, $E_{m+1,n}(\mathring{Z})$, $P^{\mathcal{T}}_{m,n-1}(\mathring{U})$, and $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$. Compute and store $B_{m,n}(\mathring{V})$.

The bulk of the computation of steps 6.2–6.4 is performed in step 6.3. This step runs in parallel for all blocks, rather than in parallel for columns but sequentially for rows. This added parallelism more than makes up for the increased bandwidth requirements.

Using the formulas for the initial feedbacks, we can now adapt algorithm 6 to perform computations on infinite extensions. For each extension type, we must obtain the inputs to the formulas, compute the initial feedbacks that surround the image as a whole, and then use them to correct the internal feedbacks for each block.

As a side note, equations (24)–(29) assume zero initial feedback and operate on $\mathring{Y}$, $\mathring{Z}$, $\mathring{U}$, and $\mathring{V}$. However, the formulas are still valid even if the initial feedbacks were not assumed to be zero. We will therefore replace, on occasion, $\mathring{Z}$ by $\overset{\bullet}{Z}$ or Z etc.

5.2 Constant padding extension

Here, the initial feedback to the causal pass does not depend on the input image as a whole: It can be computed with (35) before the block perimeters are corrected. This, in turn, enables the block-parallel algorithm to compute causal block feedbacks for the constant padding infinite extension at no additional cost. As a natural side effect, we obtain $e_{-1}(Y)$, and with it we can compute the anticausal feedback using (44). This then allows the block-parallel algorithm to compute anticausal block feedbacks at no additional cost.

Before we can proceed to row processing, we must obtain the results of column processing the constant padding of each row. We do this block by block, using formulas

$$\begin{aligned} E^{\mathcal{T}}_{m,n}(Z) &= \boldsymbol{A}_{RE}\, H^{\mathcal{T}}\big(E_{m+1,n}(Z)\big) \\ &\quad + (\boldsymbol{A}_{RB}\, \boldsymbol{A}_{FP})\, H^{\mathcal{T}}\big(P_{m-1,n}(Y)\big) \qquad (73) \\ &\quad + E^{\mathcal{T}}_{m,n}(\mathring{\mathring{Z}}), \quad \text{and} \\ P^{\mathcal{T}}_{m,n}(Z) &= \boldsymbol{A}_{RE}\, T^{\mathcal{T}}\big(E_{m+1,n}(Z)\big) \\ &\quad + (\boldsymbol{A}_{RB}\, \boldsymbol{A}_{FP})\, T^{\mathcal{T}}\big(P_{m-1,n}(Y)\big) \qquad (74) \\ &\quad + P^{\mathcal{T}}_{m,n}(\mathring{\mathring{Z}}). \end{aligned}$$

These formulas come from the first and last r columns in the equation

$$\begin{aligned} R\Big(F\big(P_{r\times b}, X_{b\times b}\big), E_{r\times b}\Big) &= \\ \boldsymbol{A}_{RB}\, \boldsymbol{A}_{FP}\, P &+ R\big(F(0,X),0\big) + \boldsymbol{A}_{RE}\, E. \qquad (75) \end{aligned}$$

Using these processed columns, we can define the constant paddings for rows, $C_{\leftarrow}$ and $C_{\rightarrow}$. Transposed versions of equations (35) and (44) can then be used to obtain the required initial feedbacks for row processing in a similar way to the feedbacks we obtained for column processing.

The result is *algorithm 6^c*:

6^c.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store block perimeters $P_{m,n}(\mathring{\mathring{Y}})$, $E_{m,n}(\mathring{\mathring{Z}})$, $P^{\mathcal{T}}_{m,n}(\mathring{\mathring{U}})$, and $E^{\mathcal{T}}_{m,n}(\mathring{\mathring{V}})$;

6^c.2. In parallel for all m, compute and store block perimeters $E^{\mathcal{T}}_{m,0}(\mathring{\mathring{Z}})$ and $P^{\mathcal{T}}_{m,N-1}(\mathring{\mathring{Z}})$;

6^c.3. In parallel for all n, compute and store $P_{-1,n}(Y)$ from $C_{\uparrow}$ as per (35), then sequentially for all m compute and store $P_{m,n}(Y)$ from $P_{m-1,n}(Y)$ and $P_{m,n}(\mathring{\mathring{Y}})$ as per (24);

6^c.4. In parallel for all n, compute and store $E_{M,n}(Z)$ from $C_{\downarrow}$ and $P_{-1,n}(Y)$ as per (44), then sequentially for all m, compute and store $E_{m,n}(Z)$ from $E_{m+1,n}(Z)$, $P_{m-1,n}(Y)$, and $E_{m,n}(\mathring{\mathring{Z}})$ as per (25);

6^c.5. In parallel for all m and n, compute and store all $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ according to equations (27) and (29);

6^c.6. In parallel for all m, obtain each block of $C_{\leftarrow}$, the constant padding for each row of Z, from $E^{\mathcal{T}}_{m,0}(\mathring{\mathring{Z}})$, $H^{\mathcal{T}}\big(P_{m-1,0}(Y)\big)$, and $H^{\mathcal{T}}\big(P^{\mathcal{T}}_{m+1,0}(Z)\big)$ as per equation (73). Similarly, compute each block of $C_{\rightarrow}$ from $P^{\mathcal{T}}_{m,N-1}(\mathring{\mathring{Z}})$, $T^{\mathcal{T}}\big(P_{m-1,N-1}(Y)\big)$, and $T^{\mathcal{T}}\big(E^{\mathcal{T}}_{m+1,N-1}(Z)\big)$ as per equation (74);

6^c.7. In parallel for all m, compute and store $P^{\mathcal{T}}_{m,-1}(U)$ from $C_{\leftarrow}$ as per (35), then sequentially for all n, compute and store $P^{\mathcal{T}}_{m,n}(U)$ from $P^{\mathcal{T}}_{m,n-1}(U)$, and $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ as per (26);

6^c.8. In parallel for all m, compute and store $E^{\mathcal{T}}_{m,N}(V)$ from $C_{\rightarrow}$ and $P^{\mathcal{T}}_{m,-1}(U)$ as per (44), then sequentially for all n, compute and store $E^{\mathcal{T}}_{m,n}(V)$ from $E^{\mathcal{T}}_{m,n+1}(V)$, $P^{\mathcal{T}}_{m,n-1}(U)$, and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ as per (28);

6^c.9. In parallel for all m and n, load input block $B_{m,n}(X)$ and all its block feedbacks $P_{m-1,n}(Y)$, $E_{m+1,n}(Z)$, $P^{\mathcal{T}}_{m,n-1}(U)$, and $E^{\mathcal{T}}_{m,n+1}(V)$. Compute and store $B_{m,n}(V)$.

Although these modifications seem complex, they translate to very little additional computation and memory bandwidth. As section 6 shows, the result is that the block-parallel algorithm for filtering the infinite extension of an image with constant padding is about as fast as using zero feedback for all passes.

5.3 Periodic extension

In contrast to the constant padding extension, the formula that yields the initial feedback $p(Y)$ for the causal pass of the infinite periodic extension requires the value of $e_{-1}(\mathring{Y})$. This is only available from the block-parallel algorithm after all block perimeters $P_{m,n}(\mathring{\mathring{Y}})$ have been corrected to block feedbacks $P_{m,n}(\mathring{Y})$. Once $p(Y)$ is known, we have to correct them again to obtain $P_{m,n}(Y)$.

Although the second update can be performed in parallel using

$$P_{m,n}(Y) = P_{m,n}(\mathring{Y}) + \boldsymbol{A}_F^{bm}\, p(Y), \qquad (76)$$

our attempts at precomputation of the required matrix powers or their generation on demand always fell short of the performance obtained with sequential updates via equation (24).

For row processing, we simply use transposed equations. The result is *algorithm 6^p*:

6^p.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store $P_{m,n}(\mathring{\mathring{Y}})$, $E_{m,n}(\mathring{\mathring{Z}})$, $P^{\mathcal{T}}_{m,n}(\mathring{\mathring{U}})$, and $E^{\mathcal{T}}_{m,n}(\mathring{\mathring{V}})$;

6^p.2. In parallel for all n, sequentially for each m, compute all $P_{m,n}(\mathring{Y})$ using $P_{m-1,n}(\mathring{Y})$ and $P_{m,n}(\mathring{\mathring{Y}})$ as per (24) until reaching $P_{M-1,n}(\mathring{Y})$. Then, compute and store $P_{-1,n}(Y)$ using (49), and, sequentially again for each m, compute and store all $P_{m,n}(Y)$ using $P_{m-1,n}(Y)$ and $P_{m,n}(\mathring{\mathring{Y}})$ as per (24);

6^p.3. In parallel for all n, sequentially for each m, compute all $E_{m,n}(\overset{\bullet}{Z})$ using $E_{m+1,n}(\overset{\bullet}{Z})$, $P_{m-1,n}(Y)$ and $E_{m,n}(\mathring{\mathring{Z}})$ as per (25) until reaching $E_{0,n}(\overset{\bullet}{Z})$. Then, compute and store $E_{M,n}(Z)$ using (51), and, sequentially again for each m, compute and store all $E_{m,n}(Z)$ using $E_{m+1,n}(Z)$, $P_{m\text{-}1,n}(Y)$ and $E_{m,n}(\overset{\circ\circ}{Z})$ as per (25);

6^{p}.4. In parallel for all m and n, compute and store all $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ according to equations (27) and (29);

6^{p}.5. In parallel for all m, sequentially for each n, compute $P^{\mathcal{T}}_{m,n}(\overset{\bullet}{U})$ using $P^{\mathcal{T}}_{m,n\text{-}1}(\overset{\bullet}{U})$ and $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ according to (26) until reaching $P^{\mathcal{T}}_{m,N\text{-}1}(\overset{\bullet}{U})$. Then, compute and store $P^{\mathcal{T}}_{m,\text{-}1}(U)$ using the transpose of equation (49), and, sequentially again for each n, compute and store all $P^{\mathcal{T}}_{m,n}(U)$ using $P^{\mathcal{T}}_{m,n\text{-}1}(U)$, and $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ as per (26);

6^{p}.6. In parallel for all m, sequentially for each n, compute all $E^{\mathcal{T}}_{m,n}(\overset{\bullet}{V})$ using $E^{\mathcal{T}}_{m,n+1}(\overset{\bullet}{V})$, $P^{\mathcal{T}}_{m,n\text{-}1}(U)$, and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ as per (28) until reaching $E^{\mathcal{T}}_{m,0}(\overset{\bullet}{V})$. Then, compute and store $E^{\mathcal{T}}_{m,N}(V)$ using the transpose of equation (51), and, sequentially again for each n, compute and store all $E^{\mathcal{T}}_{m,n}(V)$ using $E^{\mathcal{T}}_{m,n+1}(V)$, $P^{\mathcal{T}}_{m,n\text{-}1}(U)$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ as per (28);

6^{p}.7. In parallel for all m and n, load input block $B_{m,n}(X)$ and all its block feedbacks $P_{m\text{-}1,n}(Y)$, $E_{m+1,n}(Z)$, $P^{\mathcal{T}}_{m,n\text{-}1}(U)$, and $E^{\mathcal{T}}_{m,n+1}(V)$. Compute and store $B_{m,n}(V)$.

The modifications required by the periodic extension essentially double the cost of the intermediate steps of the algorithm. Fortunately, the total running time is dominated by the first and last steps. See section 6 for the modest resulting performance penalty.

5.4 Even-periodic extension

Both causal and anticausal initial feedbacks $p(Y)$ and $e(Z)$ for the even-periodic infinite extension depend on $e_{\text{-}1}(\mathring{Y})$ and $p_1(\mathring{Z})$. These values are available from the block-parallel algorithm only after all block perimeters $P_{m,n}(\overset{\circ\circ}{Y})$ and $E_{m,n}(\overset{\circ\circ}{Z})$ have been corrected to block feedbacks $P_{m,n}(\mathring{Y})$ and $E_{m,n}(\mathring{Z})$. As in the periodic extension case, after computing $p(Y)$ and $e(Z)$, we sequentially correct the block feedbacks once again.

For row processing, we simply use transposed equations. The result is *algorithm* 6^{e}:

6^{e}.1. In parallel for all m and n, load block $B_{m,n}(X)$ then compute and store $P_{m,n}(\overset{\circ\circ}{Y})$, $E_{m,n}(\overset{\circ\circ}{Z})$, $P^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{U})$, and $E^{\mathcal{T}}_{m,n}(\overset{\circ\circ}{V})$;

6^{e}.2. In parallel for all n, sequentially for each m, compute and store $P_{m,n}(\mathring{Y})$ using $P_{m\text{-}1,n}(\mathring{Y})$ and $P_{m,n}(\overset{\circ\circ}{Y})$ as per (24) until reaching $P_{M\text{-}1,n}(\mathring{Y})$. Then, compute $E_{m,n}(\mathring{Z})$ using $E_{m+1,n}(\mathring{Z})$, $P_{m\text{-}1,n}(\mathring{Y})$ and $E_{m,n}(\overset{\circ\circ}{Z})$ as per (25) until reaching $E_{0,n}(\mathring{Z})$. From $P_{M\text{-}1,n}(\mathring{Y})$ and $E_{0,n}(\mathring{Z})$, compute and store $P_{\text{-}1,n}(Y)$ as per (67) and $E_{M,n}(Z)$ as per (72);

6^{e}.3. In parallel for all n, sequentially for each m, compute and store all $P_{m,n}(Y)$ using $P_{m\text{-}1,n}(Y)$ and $P_{m,n}(\overset{\circ\circ}{Y})$ as per (24), then compute and store all $E_{m,n}(Z)$ using $E_{m+1,n}(Z)$, $P_{m\text{-}1,n}(Y)$ and $E_{m,n}(\overset{\circ\circ}{Z})$ as per (25);

6^{e}.4. In parallel for all m and n, compute and store all $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ according to equations (27) and (29);

6^{e}.5. In parallel for all m, sequentially for each n, compute and store $P^{\mathcal{T}}_{m,n}(\mathring{U})$ using $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$ and $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ as per (26) until reaching $P^{\mathcal{T}}_{m,N\text{-}1}(\mathring{U})$. Then, compute $E^{\mathcal{T}}_{m,n}(\mathring{V})$ using $E^{\mathcal{T}}_{m,n+1}(\mathring{V})$, $P^{\mathcal{T}}_{m,n\text{-}1}(\mathring{U})$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ as per (28) until reaching $E^{\mathcal{T}}_{m,0}(\mathring{V})$. From $P^{\mathcal{T}}_{m,N\text{-}1}(\mathring{U})$ and $E^{\mathcal{T}}_{m,0}(\mathring{V})$, compute and store $P^{\mathcal{T}}_{n,\text{-}1}(U)$ as per (67) and $E^{\mathcal{T}}_{m,N}(V)$ as per (72);

6^{e}.6. In parallel for all m, sequentially each n, compute and store all $P^{\mathcal{T}}_{m,n}(U)$ using $P^{\mathcal{T}}_{m,n\text{-}1}(U)$, and $P^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{U})$ as per (26) then compute and store all $E^{\mathcal{T}}_{m,n}(V)$ using $E^{\mathcal{T}}_{m,n+1}(V)$, $P^{\mathcal{T}}_{m,n\text{-}1}(U)$ and $E^{\mathcal{T}}_{m,n}(\overset{\circ\bullet}{V})$ as per (28);

6^{e}.7. In parallel for all m and n, load input block $B_{m,n}(X)$ and all its block feedbacks $P_{m\text{-}1,n}(Y)$, $E_{m+1,n}(Z)$, $P^{\mathcal{T}}_{m,n\text{-}1}(U)$, and $E^{\mathcal{T}}_{m,n+1}(V)$. Compute and store $B_{m,n}(V)$.

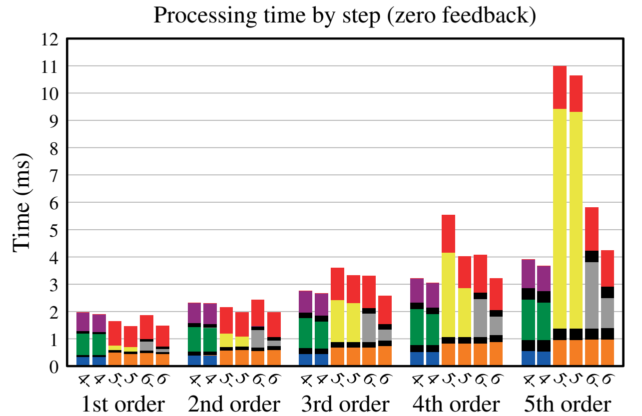


Figure 7: *Time spent on each step of algorithms 4, 5, and 6 for orders 1–5. Each color represents a different kernel. Columns marked with a '-' do not include our low-level optimizations. Note the explosion in cost of step 5.3, in comparison to steps 6.3 and 6.4. These baseline implementations ignore boundary conditions.*

Although the computation of image feedbacks is delayed relative to the periodic extension, the computational cost of the even periodic is similar. This is confirmed by the benchmarks in section 6.

6 Results and discussion

We implemented all algorithms in *C for CUDA* (version 7.0). Benchmarks were run on an NVIDIA GTX Titan with 6GB of DRAM (2688 CUDA cores, 14 SMXs, 192 cores/SMX). Our tests considered single-channel 32-bit floating-point images with random values. Time measurements were repeated 1000 times to reduce variation. Image sizes ranged from 64^2 to 8192^2 pixels, in 64-pixel increments. In figures 8–11, performance numbers are reported in pixel throughput (1 GiP/s = 2^{30} processed pixels per second).

Baseline for comparison We start analyzing the performance of baseline implementations that ignore boundary conditions. Figure 7 shows the time spent on each step of algorithms 4, 5, and 6 when run on a 4096^2 image. Each bar represents a different implementation. Bars marked with a '-' do not use our new low-level optimizations. Each color represents a different kernel, as these are shared across implementations. For example, all implementations use exactly the same second step. The comparison between algorithms 5 and 6 show the benefit of splitting step 5.3 into steps 6.3 and 6.4 for orders 3 and above. The figure suggests that algorithm 4 has the advantage starting at order 4. This is confirmed by figure 8, which shows the throughput of each algorithm for input images of increasing size. As expected, running times increase with order r. The effect is strongest in algorithm 5 and weakest in algorithm 4. Even so, algorithm 4 only becomes significantly faster than algorithm 6 for orders 5 and above. Recall that, in previous implementations, full overlapping was slower than causal-anticausal overlapping already for order 2.

Realistic alternatives for filtering extensions The first competing alternative to our exact formulas is to use an input padding longer than the effective support of the impulse response of the recursive filter of interest. Our implementation of this approach is denoted algorithm 6^{b}. Another option is to formulate a banded linear system, precompute an LU-style factorization, and implement forward- and back-substitution using recursive filters with variable feedback coefficients. This works well for the even-periodic extension, though additional control flow and global memory accesses are needed to manage the variable coefficients. A sequential version of this idea was described in detail by Nehab and Hoppe [2014, section 4.2]. Our 1st-order parallel implementation is denoted algorithm 5^{v}.

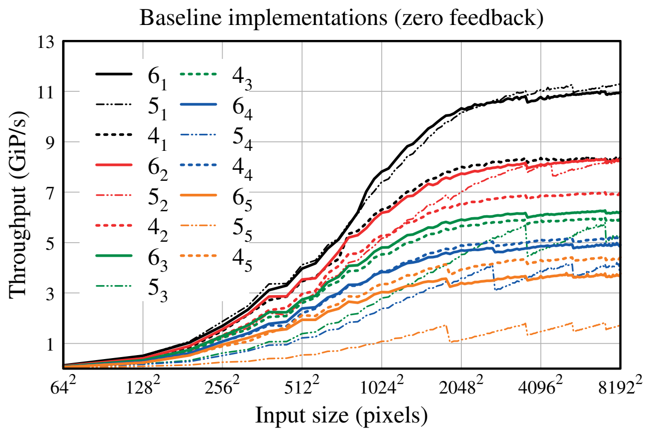


Figure 8: *The arithmetic and step complexity, as well as the bandwidth requirements of algorithms 4–6 vary in a distinct manner with filter order. The plot shows the performance of our baseline implementation in orders 1 to 5 and ignores boundary conditions.*

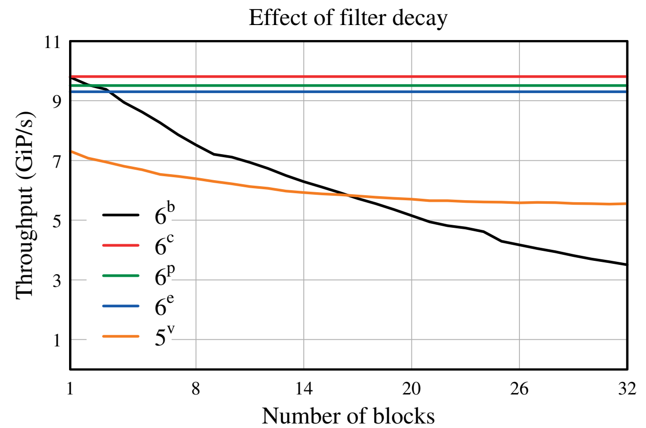


Figure 9: *Effect of recursive filter decay speed (in number of blocks) to the performance of 1st-order algorithm 6 using different extension strategies. Our exact algorithms are oblivious to the decay speed. Alternative methods suffer a performance penalty.*

Effect of impulse response support on performance Our exact algorithms are oblivious to the rate of decay of the recursive filters they implement. (As long as the filters are BIBO stable.) In contrast, when the support of the impulse response grows, competing methods will either require larger padding or the use of varying coefficients throughout larger parts of the input. This means the advantage of using our algorithms increases with the support of the impulse response. To illustrate this, figure 9 shows the performance of filtering a 2048^2 image with filters whose impulse response require an increasing number of blocks to decay to zero within a reasonable tolerance. In contrast to using our explicit formulas, the use of padding or varying coefficients leads to performance loss.

Numerical behavior To test the numerical behavior of our algorithms, we investigated the space of 2nd-order stable recursive filters with real coefficients (i.e., with poles at $\rho e^{\pm i\theta}$ with $0 \leq \rho < 1$). Given a desired error tolerance ϵ and the number of pixels n after which we want the input response to decay to ϵ, it can be shown that the relationship between ρ and θ is given by

$$\rho^{n/2} = \epsilon \sin\theta. \tag{77}$$

With $\epsilon = 10^{-10}$ and $n \in \{32, 64, \ldots, 4096\}$, we selected 300 stratified random values for $\theta \in [0, \pi]$ and computed the corresponding ρ from (77). Using a 512^2 input image, ground truth for the initial feedbacks for all infinite extensions was obtained in double precision using padding long enough for convergence to full floating-point precision. We found that our exact algorithms were consistently within 10^{-9} of ground truth, regardless of extension.

Worst case scenario Figure 10 shows the performance of our algorithms solving the real-world problem of bicubic B-spline interpolation [Unser et al. 1991]. In 8-bit and 16-bit precision, this fast decaying 1st-order recursive filter requires a single block of padding around the input. It is therefore the worst case scenario for our new methods. (Full 32-bit and 64-bit floating-point precision would require 5 and 18 blocks of padding, respectively.) Since the incurred penalty for input padding is negligible, this strategy is about as fast as exact the infinite extension algorithm for constant padding (algorithm 6^c). The periodic (algorithm 6^p) and even-periodic (algorithm 6^e) algorithms come next. The exact algorithm using varying filter coefficients (algorithm 5^v) is the least performant way to deal with boundary conditions. To put these results into context, figure 10 includes lines for the fastest implementations of Nehab et al. [2011] and Chaurasia et al. [2015]. Even though these competing implementations were timed while ignoring boundary conditions, *all* our exact infinite extension implementations are significantly faster.

Target usage scenario Figure 11 shows another real-world application: applying Gaussian blur with a standard deviation that depends on the image resolution. In this test, $\sigma = \frac{n}{6}$ pixels for input resolution n^2. We assume the impulse response decays to zero within approximately 3σ pixels. The recursive filter implementation uses the 3rd-order approximation by van Vliet et al. [1998]. We include in the comparison an exact implementation in the frequency domain using FFT [cuFFT library 2007] as well as direct convolution in the spatial domain [Podlozhnyuk 2007].

Performing the convolution in the spatial domain is clearly slower than in the frequency domain. As the image size and standard deviation increases, our exact algorithms $6_3^{c,e,p}$, become considerably faster than the alternatives. Once again, for added context, note that all our exact implementations are faster than the best implementations of Nehab et al. [2011] and Chaurasia et al. [2015], even when ignoring boundary conditions.[1]

7 Conclusions

Recursive filters enable high-performance filtering with impulse responses that have a wide support. These filters have many applications in computer graphics. They include, for example, a wide variety of low-pass filters (e.g., Butterworth, Chebyshev, and elliptic) that can be easily transformed into other types of frequency-selective filters.

When the impulse response is wide, filtering finite input requires the specification of boundary conditions. The most appropriate boundary conditions depend on the application. Popular alternatives are the constant padding, periodic, and even-periodic infinite extensions.

The traditional approach to implementing boundary conditions for recursive filtering is to augment the input with enough padded data to cover the effective support of the impulse response. This approach becomes computationally prohibitive when the support width is large relative to input size.

We propose instead to compute the initial feedbacks needed for recursive filtering the most popular infinite input extensions, without using any padding. We derive exact formulas for filters of arbitrary order and prove that they are well defined for stable recursive filters. We also show, empirically, that they produce precise results.

[1] Due to some oversight, the code distributed by Nehab et al. [2011] for 3rd-order filtering does not fuse the 5_1 and 4_2 kernels. To their advantage, we fixed this before running our tests. The change made their code perform better than the 3x_3y implementation by Chaurasia et al. [2015].

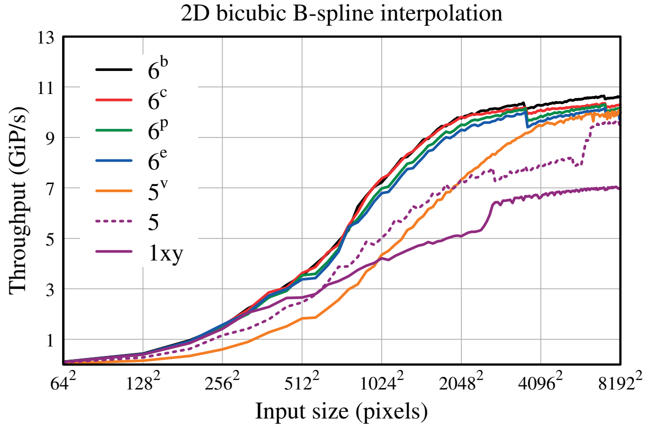


Figure 10: *Throughput of our 1st-order parallel recursive filtering algorithms using different extension strategies. The filters solve the bicubic B-spline interpolation problem.* $6^{c,p,e}$ *use our exact formulas.* 6^{b} *uses padding.* 5^{v} *uses variable coefficients. Competing implementations* 5 *by Nehab et al. [2011] and* 1xy *by Chaurasia et al. [2015] were timed while ignoring boundary conditions.*

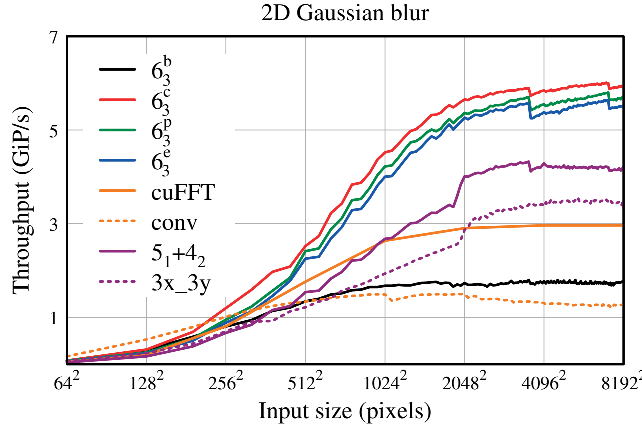


Figure 11: *Throughput of Gaussian blur using our 3rd-order parallel recursive filtering algorithms with different extension strategies.* $6^{c,p,e}$ *use our exact formulas.* 6^{b} *uses padding.* cuFFT *filters in the frequency domain.* conv *is a direct convolution. Competing implementations* $5_1{+}4_2$ *by Nehab et al. [2011] and* 3x_3y *by Chaurasia et al. [2015] were timed while ignoring boundary conditions.*[1]

The formulas depend on information that is only available after going over the entire input data. Fortunately, state-of-the-art block-parallel recursive filtering algorithms already go over the input data twice. By designing our formulas to depend only on information available between these passes, we were able to create new block-parallel algorithms for filtering infinite input extensions with little or no performance penalty.

In summary, our work enables users to obtain the precise results of recursively filtering infinite input extensions in state-of-the-art performance and precision, without ever worrying about the decay rate of the associated impulse response. There is no trade-off involved.

7.1 Future work

A possibility for future work is porting the framework of Chaurasia et al. [2015] to support efficient high-order filtering of infinite input extensions. Theorem A.7 describes how to split a high-order feedback into the feedbacks of lower-order filters and how to merge lower-order feedbacks into the feedback of a higher-order filter. Theorem A.4 describes how to convert the feedbacks for a causal-anticausal chain into the feedbacks for an anticausal-causal chain. Together, these theorems can be used to convert between the feedbacks needed by arbitrary filter decompositions. Still, the details of how to use these theorems to automatically generate efficient implementations for arbitrary decompositions are far from obvious.

We are also interested in investigating the parallelization of recursive filters for which the feedback coefficients can vary on a per pixel basis. Algorithm 5^{v} is a first step in this direction: weights can change along a row for row processing, but all rows must use the same weights. The same is true for column processing. Full support for independent weights would allow us to implement geodesic and edge-aware filters [Gastal and Oliveira 2011; Sun et al. 2014; Gastal and Oliveira 2015; Zhou et al. 2015] that have recently become the basis of a variety of computer graphics applications.

A more immediate direction for future work is porting the framework of [Maximo 2015] to support infinite input extensions.

Acknowledgments

This work has been funded in part by a research scholarship from CNPq and by an INST grant from FAPERJ. NVIDIA Corporation has generously donated the GPUs used in this project.

References

APPLETON, B. and TALBOT, H. 2003. Efficient and consistent recursive filtering of images with reflective extension. In *Scale Space'03*, Springer-Verlag, LNCS 2695, 699–712.

BLELLOCH, G. E. 1990. Prefix sums and their applications. Technical Report CMU-CS-90-190, Carnegie Mellon University.

BLU, T., THÉVENAZ, P., and UNSER, M. 2001. MOMS: Maximal-order interpolation of minimal support. *IEEE Transactions on Image Processing*, 10(7):1069–1080.

CATMULL, E. and ROM, R. 1974. A class of local interpolating splines. In *Computer Aided Geometric Design*, 317–326.

CHAURASIA, G., RAGAN-KELLEY, J., PARIS, S., DRETTAKIS, G., and DURAND, F. 2015. Compiling high performance recursive filters. In *High Performance Graphics*, 85–94.

CONDAT, L., BLU, T., and UNSER, M. 2005. Beyond interpolation: optimal reconstruction by quasi-interpolation. In *IEEE International Conference on Image Processing*, volume 1, 33–36.

CONDAT, L. and MÖLLER, T. 2011. Quantitative error analysis for the reconstruction of derivatives. *IEEE Transactions on Image Processing*, 59(6):2965–2969.

CUFFT library. 2007. URL `http://developer.nvidia.com/cuda-toolkit`. NVIDIA Corporation.

DOTSENKO, Y., GOVINDARAJU, N. K., SLOAN, P.-P., BOYD, C., and MANFERDELLI, J. 2008. Fast scan algorithms on graphics processors. In *Proceedings of the International Conference on Supercomputing*, 205–213.

DUCHON, C. E. 1979. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology*, 18(8):1016–1022.

GASTAL, E. S. L. and OLIVEIRA, M. M. 2011. Domain transform for edge-aware image and video processing. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2011)*, 30(4):69.

GASTAL, E. S. L. and OLIVEIRA, M. M. 2015. High-order recursive filtering of non-uniformly sampled signals for image and video processing. *Computer Graphics Forum*, 34(2):81–93.

GOVINDARAJU, N. K., LLOYD, B., DOTSENKO, Y., SMITH, B., and MANFERDELLI, J. 2008. High performance discrete fourier transforms on graphics processors. In *Proceedings of the IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis*, IEEE.

HUMMEL, R. 1983. Sampling for spline reconstruction. *SIAM Journal on Applied Mathematics*, 43(2):278–288.

IVERSON, K. E. 1962. *A Programming Language*. Wiley.

KAJIYA, J. and ULLNER, M. 1981. Filtering high quality text for display on raster scan devices. *Computer Graphics (Proceedings of ACM SIGGRAPH 1981)*, 15(3):7–15.

KOOGE, P. M. and STONE, H. S. 1973. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Transactions on Computers*, C-22(8):786–793.

MALCOLM, M. A. and PALMER, J. 1974. A fast method for solving a class of tridiagonal linear systems. *Communications of the ACM*, 17(1):14–17.

MARTUCCI, S. A. 1994. Symmetric convolution and the discrete sine and cosine transforms. *IEEE Transactions on Signal Processing*, 42(5):1038–1051.

MAXIMO, A. 2015. Efficient finite impulse response filters in massively-parallel recursive systems. *Journal of Real-Time Image Processing*, 1–9.

MCCOOL, M. D. 1995. Analytic antialiasing with prism splines. In *Proceedings of ACM SIGGRAPH 1995*, 429–436.

MERRILL, D. and GRIMSHAW, A. 2009. Parallel scan for stream architectures. Technical Report CS2009-14, University of Virginia.

MEYER, C. D. 2000. *Matrix Analysis and Applied Linear Algebra*. SIAM, Philadelphia, PA, USA.

MITCHELL, D. P. and NETRAVALI, A. N. 1988. Reconstruction filters in computer graphics. *Computer Graphics (Proceedings of ACM SIGGRAPH 1988)*, 22(4):221–228.

NEHAB, D. and HOPPE, H. 2014. A fresh look at generalized sampling. *Foundations and Trends in Computer Graphics and Vision*, 8(1):1–84.

NEHAB, D., MAXIMO, A., LIMA, R. S., and HOPPE, H. 2011. GPU-efficient recursive filtering and summed-area tables. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH Asia 2011)*, 30(6):176.

OPPENHEIM, A. V. and SCHAFER, R. W. 2010. *Discrete-Time Signal Processing*. Prentice Hall, 3rd edition.

PODLOZHNYUK, V. 2007. Image convolution with CUDA. NVIDIA whitepaper.

SACHT, L. and NEHAB, D. 2015. Optimized quasi-interpolators for image reconstruction. *IEEE Transactions on Image Processing*, 24(12):5249–5259.

SENGUPTA, S., HARRIS, M., ZHANG, Y., and OWENS, J. D. 2007. Scan primitives for GPU computing. In *Proceedings of Graphics Hardware*, 97–106.

STONE, H. S. 1971. Parallel processing with the perfect shuffle. *IEEE Transactions on Computers*, C-20(2):153–161.

STONE, H. S. 1973. An efficient parallel algorithm for the solution of a tridiagonal linear system of equations. *Journal of the ACM*, 20(1):27–38.

SUN, X., MEI, X., JIAO, S., ZHOU, M., LIU, Z., and WANG, H. 2014. Real-time local stereo via edge-aware disparity propagation. *Pattern Recognition Letters*, 49:201–206.

SUNG, W. and MITRA, S. 1986. Efficient multi-processor implementation of recursive digital filters. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, 257–260.

SUNG, W. and MITRA, S. 1992. Multiprocessor implementation of digital filtering algorithms using a parallel block processing method. *IEEE Transactions on Parallel and Distributed Systems*, 3(1):110–120.

TRIGGS, B. and SDIKA, M. 2006. Boundary conditions for Young–van Vliet recursive filtering. *IEEE Transactions on Signal Processing*, 54(5):2365–2367.

UNSER, M. and ALDROUBI, A. 1994. A general sampling theory for nonideal acquisition devices. *IEEE Transactions on Signal Processing*, 42(11):2915–2925.

UNSER, M., ALDROUBI, A., and EDEN, M. 1991. Fast B-spline transforms for continuous image representation and interpolation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(3):277–285.

UNSER, M., ALDROUBI, A., and EDEN, M. 1993. The L_2-polynomial spline pyramid. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):364–379.

UNSER, M., ALDROUBI, A., and EDEN, M. 1995. Enlargement or reduction of digital images with minimum loss of information. *IEEE Transactions on Image Processing*, 4(3):247–258.

UNSER, M., THÉNEVAZ, P., and YAROSLAVSKY, L. 1995. Convolution-based interpolation for fast, high-quality rotation of images. *IEEE Transactions on Image Processing*, 4(10):1371–1381.

VAN VLIET, L. J., YOUNG, I. T., and VERBEEK, P. W. 1998. Recursive Gaussian derivative filters. In *Proceedings of the 14th International Conference on Pattern Recognition*, 509–514 (v. 1).

WEICKERT, J., TER HAAR ROMENY, B. M., and VIERGEVER, M. A. 1998. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*, 7(3):398–410.

ZHOU, M., LIU, Z., HONG, T., WANG, X., WANG, H., and SUN, X. 2015. Efficient O(1) edge-aware filter. In *IEEE International Conference on Image Processing*, 1295–1299.

A Proofs of auxiliary theorems

Theorem A.1. *F is a stable recursive filter in the BIBO sense if and only if the spectral radius $\rho(\boldsymbol{A}_F) < 1$.*

Proof. The key observation is that $\boldsymbol{A}_F$ is the (transposed) *companion matrix* to the denominator of the transfer function of F. In other words, $\lambda \in \sigma(\boldsymbol{A}_F)$ is an eigenvalue of $\boldsymbol{A}_F$ if and only if λ is a *pole* of F. BIBO stability requires $|\lambda| < 1$ for all poles of F (see [Oppenheim and Schafer 2010]). The theorem follows. □

Theorem A.2. *Given a matrix $\boldsymbol{A}$, the following statements are equivalent:*

$$\rho(\boldsymbol{A}) < 1 \;\Leftrightarrow\; \lim_{k\to\infty} \boldsymbol{A}^k = 0 \;\Leftrightarrow\; \sum_{i=0}^{\infty} \boldsymbol{A}^i = (\boldsymbol{I} - \boldsymbol{A})^{-1}. \tag{78}$$

Proof. See a textbook in matrix analysis [Meyer 2000, chapter 7.10]. □

Theorem A.3. *If $\boldsymbol{A}$ and $\boldsymbol{C}$ are two matrices with $\rho(\boldsymbol{A}) < 1$ and $\rho(\boldsymbol{C}) < 1$, then the series*

$$\boldsymbol{S} = \boldsymbol{I} + \boldsymbol{A}\boldsymbol{B}\boldsymbol{C} + \boldsymbol{A}^2\boldsymbol{B}\boldsymbol{C}^2 + \cdots = \sum_{k=0}^{\infty} \boldsymbol{A}^k \boldsymbol{B}\boldsymbol{C}^k \tag{79}$$

converges for all conformable matrices $\boldsymbol{B}$. Furthermore, the limit $\boldsymbol{S}$ satisfies the non-singular linear system

$$\boldsymbol{S} - \boldsymbol{A}\boldsymbol{S}\boldsymbol{C} = \boldsymbol{B}. \tag{80}$$

Proof. From the partial sum

$$\boldsymbol{S}_k = \sum_{i=0}^{k} \boldsymbol{A}^k \boldsymbol{B} \boldsymbol{C}^k, \tag{81}$$

form the telescoping sum

$$\boldsymbol{S}_k - \boldsymbol{A}\boldsymbol{S}_k\boldsymbol{C} = \boldsymbol{B} - \boldsymbol{A}^{k+1}\boldsymbol{B}\boldsymbol{C}^{k+1}. \tag{82}$$

Since

$$\|\boldsymbol{A}^i\boldsymbol{B}\boldsymbol{C}^i\|_F \le \|\boldsymbol{A}^i\|_F\|\boldsymbol{B}\|_F\|\boldsymbol{C}^i\|_F \tag{83}$$

by the submultiplicative property of the Frobenius norm, we can use $\rho(\boldsymbol{A}) < 1$ and $\rho(\boldsymbol{C}) < 1$ and theorem A.2 to obtain the limit

$$\boldsymbol{S} - \boldsymbol{A}\boldsymbol{S}\boldsymbol{C} = \boldsymbol{B}. \tag{84}$$

This is simply a linear system on the entries of $\boldsymbol{S}$. To see that it is non-singular, decompose $\boldsymbol{A}$ and $\boldsymbol{C}$ into their respective Jordan normal forms $\boldsymbol{P_A}\boldsymbol{J_A}\boldsymbol{P_A}^{-1}$ and $\boldsymbol{P_C}\boldsymbol{J_C}\boldsymbol{P_C}^{-1}$. After rearrangement,

$$\boldsymbol{S}' - \boldsymbol{J_A}\boldsymbol{S}'\boldsymbol{J_C} = \boldsymbol{B}', \quad \text{with} \tag{85}$$

$$\boldsymbol{S}' = \boldsymbol{P_A}^{-1}\boldsymbol{S}\boldsymbol{P_C} \quad \text{and} \quad \boldsymbol{B}' = \boldsymbol{P_A}^{-1}\boldsymbol{B}'\boldsymbol{P_C}. \tag{86}$$

Given the solution of this new linear system on the entries s'_{ij} of $\boldsymbol{S}'$, we can obtain the desired $\boldsymbol{S} = \boldsymbol{P_A}\boldsymbol{S}'\boldsymbol{P_C}^{-1}$.

Recall that $\boldsymbol{J_A}$ and $\boldsymbol{J_C}$ are upper bidiagonal, with eigenvalues of $\boldsymbol{A}$ and $\boldsymbol{C}$ in the main diagonal, respectively, and 0 or 1 in the upper diagonal. Arranging the equations on s'_{ij} by decreasing i then increasing j, we see the system is triangular. The pivots are $1 - \lambda_i\gamma_j$, with $\lambda_i \in \sigma(A)$ and $\gamma_j \in \sigma(C)$. These can never be zero, because $\rho(A) < 1$ and $\rho(C) < 1$. The theorem follows. □

Lemma A.1. *Recursive filters F and R commute.*

Proof. Let F^{-1} and R^{-1} denote the discrete convolution inverses of F and R. Let I denote the identity operation. Since discrete convolution commutes,

$$F^{-1} \circ R^{-1} = R^{-1} \circ F^{-1} \quad \Rightarrow \tag{87}$$

$$R \circ F^{-1} \circ R^{-1} = F^{-1} \quad \Rightarrow \tag{88}$$

$$F \circ R \circ F^{-1} \circ R^{-1} = I \quad \Rightarrow \tag{89}$$

$$F \circ R \circ F^{-1} = I \circ R \quad \Rightarrow \tag{90}$$

$$F \circ R = R \circ F. \tag{91}$$

On finite input, appropriate prologues and epilogues must be carefully chosen, as shown in theorem A.4. □

Theorem A.4. *Let F and R be causal and anticausal recursive filters, respectively. Select a boundary condition for the infinite extension of input $\boldsymbol{x}$. Let $\boldsymbol{p}$ and $\boldsymbol{e}$ be the initial feedbacks for filtering the infinite extension of $\boldsymbol{x}$ in the $R \circ F$ order, and $\boldsymbol{e}'$ and $\boldsymbol{p}'$ the initial feedbacks for filtering in the $F \circ R$ order. By lemma A.1,*

$$\boldsymbol{z} = R\big(F(\boldsymbol{p}, \boldsymbol{x}), \boldsymbol{e}\big) = F\big(\boldsymbol{p}', R(\boldsymbol{x}, \boldsymbol{e}')\big). \tag{92}$$

Furthermore, the initial feedbacks are related by equations

$$\boldsymbol{p}' = R\big(\boldsymbol{p}, H(\boldsymbol{z})\big) \quad \textit{and} \quad \boldsymbol{e} = F\big(T(\boldsymbol{z}), \boldsymbol{e}'\big). \tag{93}$$

Proof. Let $\boldsymbol{y} = F(\boldsymbol{p}, \boldsymbol{x})$. Note that $\boldsymbol{p}' = [z_{-1-r} \ \cdots \ z_{-1}]$, that $\boldsymbol{p} = [y_{-1-r} \ \cdots \ y_{-1}]$, and that $H(\boldsymbol{z}) = [z_0 \ \cdots \ z_{r-1}]$. Therefore,

$$\boldsymbol{p}' = [\, z_{-1-r} \cdots z_{-1} \,] \tag{94}$$

$$= R\big([\, y_{-1-r} \cdots y_{-1} \,], [\, z_0 \cdots z_{r-1} \,]\big) \tag{95}$$

$$= R\big(\boldsymbol{p}, H(\boldsymbol{z})\big). \tag{96}$$

The derivation of $\boldsymbol{e} = F\big(T(\boldsymbol{z}), \boldsymbol{e}'\big)$ is analogous. □

Theorem A.5. *The matrices $\bar{\boldsymbol{A}}_F$ and $\bar{\boldsymbol{A}}_R$ associated to causal or anticausal recursive filters F and R are always non-singular.*

Proof. The columns of $\bar{\boldsymbol{A}}_F$ are shifts of the first r entries in the impulse response of F. This makes the matrix lower triangular with 1 in the diagonal. It is therefore non-singular. Let F' be a causal filter using the same feedback coefficients as the anticausal R. It follows that $\bar{\boldsymbol{A}}_R = \boldsymbol{K}\bar{\boldsymbol{A}}_{F'}\boldsymbol{K}$. Since $\boldsymbol{K}$ and $\bar{\boldsymbol{A}}_{F'}$ are non-singular, so is $\bar{\boldsymbol{A}}_R$. □

Theorem A.6. *$\boldsymbol{I} - \boldsymbol{K}\boldsymbol{A}$ is non-singular if $\boldsymbol{A}$ has $\rho(\boldsymbol{A}) < 1$.*

Proof. We show that $\lim_{k\to\infty}(\boldsymbol{K}\boldsymbol{A})^k = 0$, so theorem A.2 on $\boldsymbol{K}\boldsymbol{A}$ proves the required result. Indeed,

$$\lim_{k\to\infty} \boldsymbol{A}^k = 0 \quad \Rightarrow \tag{97}$$

$$\lim_{k\to\infty} \|\boldsymbol{A}^k\|_F = 0 \quad \Rightarrow \tag{98}$$

$$\lim_{k\to\infty} \big\|(\boldsymbol{K}\boldsymbol{A})^k\big\|_F = 0 \quad \Rightarrow \tag{99}$$

$$\lim_{k\to\infty} (\boldsymbol{K}\boldsymbol{A})^k = 0. \tag{100}$$

Theorem A.2 and $\rho(\boldsymbol{A}) < 1$ lead to (97). Orthogonality of $\boldsymbol{K}$ and the submultiplicative property of the Frobenius norm imply $\|(\boldsymbol{K}\boldsymbol{A})^k\|_F \le \|\boldsymbol{A}^k\|_F$, from which (99) follows. □

Theorem A.7. *Let $F_{\boldsymbol{\alpha}}$ and $R_{\boldsymbol{\alpha}}$ be a causal and anticausal recursive filters, with poles $\boldsymbol{\alpha} = (\,\alpha_1, \ldots, \alpha_r\,)$, $r > 1$. If $\boldsymbol{\alpha} \setminus \alpha_i = (\,\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_r\,)$ denotes the same list of poles but with one instance of α_i removed, then*

$$F_{\boldsymbol{\alpha}}(\boldsymbol{p}, \boldsymbol{x}) = F_{\alpha_i}\big(p'_{r-1}, F_{\boldsymbol{\alpha}\setminus\alpha_i}\big([\, p'_0 \ \cdots \ p'_{r-2} \,], \boldsymbol{x}\big)\big) \ \textit{and} \tag{101}$$

$$R_{\boldsymbol{\alpha}}(\boldsymbol{e}, \boldsymbol{x}) = R_{\alpha_i}\big(R_{\boldsymbol{\alpha}\setminus\alpha_i}\big(\boldsymbol{x}, [\, e'_1 \ \cdots \ e'_{r-1} \,]\big), e'_0\big) \tag{102}$$

hold for input $\boldsymbol{x}$, whenever

$$p'_{r-1} = p_{r-1}, \quad [\, p'_0 \ \cdots \ p'_{r-2} \,] = F^{-1}_{\alpha_i}\big([\, p_0 \ \cdots \ p_{r-1} \,]\big) \ \textit{and} \tag{103}$$

$$e'_0 = e_0, \qquad [\, e'_1 \ \cdots \ e'_{r-1} \,] = R^{-1}_{\alpha_i}\big([\, e_0 \ \cdots \ e_{r-1} \,]\big), \tag{104}$$

or, conversely, whenever

$$p_{r-1} = p'_{r-1}, \quad [\, p_0 \ \cdots \ p_{r-2} \,] = F_{\alpha_i}(q, [\, p'_0 \ \cdots \ p'_{r-2} \,]) \ \textit{and} \tag{105}$$

$$e_0 = e'_0, \qquad [\, e_1 \ \cdots \ e_{r-1} \,] = R_{\alpha_i}([\, e'_1 \ \cdots \ e'_{r-1} \,], d), \tag{106}$$

where

$$q = \big(p'_{r-1} - T_1\big(F_{\alpha_i}(0, [\, p_0 \ \cdots \ p_{r-2} \,])\big)\big)/(-\alpha_i)^{r-1} \ \textit{and} \tag{107}$$

$$d = \big(e'_0 - H_1\big(R_{\alpha_i}([\, e_1 \ \cdots \ e_{r-1} \,], 0)\big)\big)/(-\alpha_i)^{r-1}. \tag{108}$$

Proof. Let $\boldsymbol{y} = F(\boldsymbol{p}, \boldsymbol{x})$ and let y_k be vector entries. To prove (103), first observe that $p_{r-1} = y_{-1} = p'_{r-1}$. Then, from (101), note that $[\, p_1 \ \cdots \ p_{r-1} \,] = F_{\alpha_i}\big(p_0, [\, p'_0 \ \cdots \ p_{r-2} \,]\big)$. The desired results comes from inverting the effect of F_{α_i} on both sides, i.e., by applying convolution with $[\, 1 \ \alpha_i \,]$. To prove (105), solve for q in

$$p'_{r-1} = T_1\big(F_{\alpha_i}\big(q, [\, p_0 \ \cdots \ p_{r-2} \,]\big)\big) \tag{109}$$

$$= T_1\big(F_{\alpha_i}(q, 0)\big) + T_1\big(F_{\alpha_i}\big(0, [\, p_0 \ \cdots \ p_{r-2} \,]\big)\big) \tag{110}$$

$$= (-\alpha_i)^{r-1} q + T_1\big(F_{\alpha_i}\big(0, [\, p_0 \ \cdots \ p_{r-2} \,]\big)\big). \tag{111}$$

The proofs for (104) and (106) are analogous. □