The Polynomial Connection between Morphological Dilation and Discrete Convolution

Vivek Sridhar, Keyvan Shahin, Michael Breuß and Marc Reichenbach

***Abstract*—In this paper we consider the fundamental operations dilation and erosion of mathematical morphology. Many powerful image filtering operations are based on their combinations. We establish homomorphism between max-plus semi-ring of integers and subset of polynomials over the field of real numbers. This enables to reformulate the task of computing morphological dilation to that of computing sums and products of polynomials. Therefore, dilation and its dual operation erosion can be computed by convolution of discrete linear signals, which is efficiently accomplished using a Fast Fourier Transform technique. The novel method may deal with non-flat filters and incorporates no restrictions on shape or size of the structuring element, unlike many other fast methods in the field. In contrast to previous fast Fourier techniques it gives exact results and is not an approximation. The new method is in practice particularly suitable for filtering images with small tonal range or when employing large filter sizes. We explore the benefits by investigating an implementation on FPGA hardware. Several experiments demonstrate the exactness and efficiency of the proposed method.**

***Index Terms*—morphological dilation, morphological erosion, max-plus semi-ring, fast Fourier transform, polynomials, FPGA hardware**

I. Introduction

MATHEMATICAL morphology is a highly successful field in image processing. It is concerned with the analysis of shapes and structures in images, see for instance [10]–[12] for an account of theory and applications. The basic building blocks of many of its processes are dilation and erosion. These operations are dual, so that it is convenient to focus on dilation for the construction of algorithms. Modeling images via grey values on a discrete grid, computing dilation means that a pixel value is set to the maximum of the grey values within a filter mask centered upon it. This mask is called structuring element (SE), and it can be either flat or non-flat [18]. A flat SE describes the shape of the mask, while a non-flat SE may additionally contain additive grey value offsets.

An important property of morphological filters is the high efficiency that can be achieved in their implementation. Let us briefly review some efficient algorithms along the lines of their possible classification described in [27]. A first family of schemes aims to reduce the size of the SE or to decompose it, thus reducing the number of comparison operations for evaluation of the maximum respectively minimum over an SE. In a second family of methods a given image is analysed so that redundant operations that may arise in some image parts could be reduced. However, most of these methods are limited in terms of shape, size or flatness of SE, or specific hardware that is needed, cf. [12], [14]–[17], [24].

There are just few fast methods that allow an SE to be of arbitrary shape and size. A very popular example is the classic scheme from [26] that relies on histogram updates. However, as also for [26], the algorithmic complexity of most methods relies inherently on size of the SE, and often also on its shape. Since the SE is moved over an image in implementations relying on sliding window technique, the computational effort also relates to image size.

An alternative construction of fast algorithms relies on the possibility to formulate operations over an SE as convolutions, which may be realized via a fast transform. In a first work [6], binary dilation respectively erosion are represented by convolution of characteristic functions of underlying sets. In [13] this approach was extended in a straightforward way to grey scale images. This was done by decomposing an image into its level sets, and each level set was processed like a binary image. By construction, the method is limited to flat filters of particular shape. A different extension of [6] has been proposed in [7], making use of an analytical approximation of morphological operations. The resulting method is suitable for flat and non-flat SE, without restriction on shape or size. However, as analyzed in [7], [8], this comes at the expense of a shift and smoothing effect in the tonal histogram. In order to address this issue, a novel fast and exact method has recently been proposed [31] which considers the umbra of image and filter as the computational setting for the convolution.

Our Contributions: In this paper we extend the work in [31] to analytically prove the exactness of the proposed method. In particular, we construct a homomorphism between semi-rings of polynomials and max-plus semi-ring of non-negative integers. This allows us to represent computations in the max-plus semi-ring as sums and products of polynomials. We also establish how our constructions and convolution precisely relate to morphological dilation, thus allowing us to utilize the above-mentioned theory. Furthermore we provide extensive additional experiments that demonstrate the exactness of the

Vivek Sridhar and Micheal Breuß are with *Institute for Mathematics*, Brandenburg Technical University, Cottbus, Germany. Email: {sridhviv, breuss}@b-tu.de

Keyvan Shahin and Marc Reichenbach are with *Institute for Computer Science*, Brandenburg Technical University, Cottbus, Germany. Email: {keyvan.shahin, marc.reichenbach}@b-tu.de

This work was partially funded by the European Regional Development Fund (EFRE 85037495) and BTU Graduate Research School (STIBET short-term scholarship for international PhD Students sponsored by the German Academic Exchange Service (DAAD) with funds of the German Federal Foreign Office)

proposed method. In particular, this includes a novel, detailed study of an implementation of our method on FPGA hardware. The results confirm the beneficial theoretical and computational properties of our scheme.

II. Theoretical Background

A. Morphological Operations

An N-dimensional grey-value image is a function $f : F \rightarrow \mathbb{L}$. Thereby $F \subseteq \mathbb{Z}^N$ is the set of the (in general, N-dimensional) indices of pixels in the image, also known as domain of the image. Furthermore, $\mathbb{L} = \{0, 1, \cdots l\}$, $l > 0$, is the tonal range of the image. In case of the common 8-bit grey-value image, $\mathbb{L}$ is the set of integers in the range $[0, 255]$. Similarly, a morphological grey-value filter, flat or non-flat, can be defined as $b : B \rightarrow \mathbb{L}$, $B \subseteq \mathbb{Z}^N$. Let us note that $b(\cdot) \geq 0$ (which affects some formula). The mask domain B denotes the domain of the structuring element.

The dilation of image f by structuring elements b is denoted by $f \oplus b$ and is computed for each x in its domain $F \oplus B$ as

$$(f \oplus b)(x) = \max\{f(x-y)+b(y) \,|\, (x-y) \in F \wedge y \in B\} \quad (1)$$

Here, $F \oplus B = \{x_F + x_B | x_F \in F \wedge x_B \in B\}$. $F \oplus B$ is the Minkowski addition of set of indices. This also corresponds to dilation of F by B considering them as binary images (see, [18]). In practice, we are only interested in the indices contained the original image, therefore, for each $x \in F$, we use, with $f \oplus b := (f \oplus b)(x)$:

$$f \oplus b = \begin{cases} 0 & \text{if } x \notin F \oplus B \\ \max\{f(x-y) + b(y) \,|\, (x-y) \in F \wedge y \in B\} \\ \quad \text{otherwise} \end{cases} \quad (2)$$

Example 1. *(Adopted from [31].) Consider a 1-dimensional image* $f = \begin{bmatrix} \underline{3} & 0 & 7 & 6 & 2 & 7 \end{bmatrix}$ *and filter* $b = \begin{bmatrix} 1 & \underline{2} & X & 0 \end{bmatrix}$. *Here,* $\underline{a}$ *denotes that* a *is exactly the entry that is located at the index* 0*, i.e. at the spatial origin. The symbol* X *is used for indices not in the domain.*

From (2)*, we have* $(f \oplus b)(0) = \max\{f(0) + b(0), f(1) + b(-1)\} = \max\{3 + 2, 1 + 0\} = 5$. *Similarly,* $(f \oplus b)(2) = \max\{f(0) + b(2), f(2) + b(0), f(3) + b(-1)\} = \max\{3, 9, 7\} = 9$ *and so on.*

Thus, we get, $(f \oplus b) = \begin{bmatrix} \underline{5} & 8 & 9 & 8 & 8 & 9 \end{bmatrix}$.

The other fundamental morphological operation is erosion. Erosion of image f by structuring element b is denoted as $f \ominus b$ and is computed for each x in its domain $F \ominus B$ as

$$(f \ominus b)(x) = \min\{f(x+y) - b(y) \,|\, x+y \in F \wedge y \in B\} \quad (3)$$

Grey-value dilation and erosion are *dual* operations, i.e. $f \oplus b = -((-f) \ominus \breve{b})$. Here, $\breve{b}$ is the reflection of structuring element about origin (of Z^N) and $(-f)$ denotes the negative of image, i.e. $(-f)(x) = l - f(x)$, $\forall x \in F$ (see, [20]). Due to the duality, the task of computing grey-value erosion can be reduced to that of computing grey-value dilation in linear time. Thus, in this paper, we focus on computing dilation.

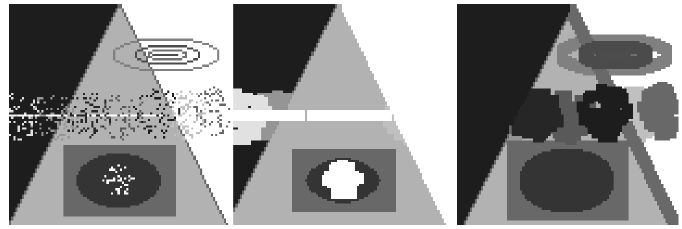

Figure 1. **: Left** *Sample image* of size 99×99. **Centre:** Dilation. **Right:** Erosion, with a 5×5 flat filter.

B. Polynomials and Max-plus semi-ring

The connection between grey-value morphological dilation and the *tropical semi-ring* $(\mathbb{R}_{\max}, \max, +)$, here $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$, has been previously explored in [21]. In particular, the computation at each pixel of dilation for grey-value images using a non-flat structuring element, is of form $\max\{a_1 + b_1, a_2 + b_2 \cdots a_m + b_m\}$ (see Example 1). This computation takes place in the semi-ring $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$, as $\mathbb{L} \subset \mathbb{Z}^{+\text{ve}}_{\max} = \mathbb{N} \cup \{0\} \cup \{-\infty\}$. Note, that the pixel not in the domain of image or SE can be assumed to have the value $-\infty$.

The theory that we develop in this section allows us to reduce the problem of computation in semi-ring $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$ to problem of computing sums and products of polynomials over the real field $(\mathbb{R}, +, \cdot)$. Our construction in the next section (Section III), allows us to compute sums and products of polynomials as convolution of real numbers (or integers). Thus, we are able to utilize Fast Fourier Transforms (or Fast Number Theoretic Transforms) to speed up the computations.

We begin by recalling the definition of the algebraic structure *semi-ring*, which is ring without an additive inverse (see, Chapter 1 of [22]).

Definition 1. ***Semi-ring*** *A semi-ring* $(R, +, \cdot)$ *is a non-empty set* R*, equipped with two binary operators* $+$ *and* $\cdot$ *such that they satisfy the following properties:*

i. ***Closure of addition:*** $a + b \in R$, $\forall a,\, b \in R$.
ii. ***Associativity of addition:*** $(a + b) + c = a + (b + c)$, $\forall a, b, c \in R$
iii. ***Existence of additive identity:*** $\exists 0 \in R$ *such that* $a + 0 = 0 + a = a$, $\forall a \in R$.
iv. ***Commutativity of addition:*** $a + b = b + a$, $\forall a, b \in R$.
v. ***Closure of multiplication:*** $a \cdot b \in R$, $\forall a, b \in R$.
vi. ***Associativity of multiplication:*** $a(bc) = (ab)c$, $\forall a, b, c \in R$.
vii. ***Existence of multiplicative identity:*** $\exists 1 \in R$ *such that* $a1 = 1a = a$, $\forall a \in R$
viii. ***Distributive laws:*** *Multiplication left and right distributes over addition, i.e. for all* $a, b, c, \in R$*, we have,*
  a) ***Left distribution*** $a(b + c) = ab + ac$.
  b) ***Right distribution*** $(a + b)c = ac + bc$.
ix. $a \cdot 0 = 0 \cdot a = 0$, $\forall a \in R$.

In addition to the above properties, if $ab = ba$, $\forall a, b \in R$*, then the semi-ring is said to be commutative.*

We now establish that $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$ is a semi-ring.

Proposition II.1. $(\mathbb{Z}^{+ve}_{\max}, \max, +)$ *is a commutative semi-ring.*

Proof. We know that $(\mathbb{R}_{\max}, \max, +)$ forms a semi-ring known as *tropical semi-ring* [21].

Clearly, $\mathbb{Z}^{+\text{ve}}_{\max} = \mathbb{N} \cup \{0\} \cup \{-\infty\} \subset \mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$. To establish that $\mathbb{Z}^{+\text{ve}}_{\max}$ is a semi-ring (i.e. sub-semiring of $(\mathbb{R}_{\max}, \max, +)$), we show the closure of operators and existence of identities in $\mathbb{Z}^{+\text{ve}}_{\max}$ (see, Chapter 1 of [22]).

i. By definition of $\mathbb{Z}^{+\text{ve}}_{\max}$, the identity of operator '+', is $0 \in \mathbb{Z}^{+\text{ve}}_{\max}$ and the identity of operator 'max', is $-\infty \in \mathbb{Z}^{+\text{ve}}_{\max}$.
ii. We prove the closure of operators.
As for the whole numbers, $\mathbb{N} \cup \{0\}$ is closed under operator '+'. Therefore, for any $a, b \in \mathbb{N} \cup \{0\}$, $a + b \in \mathbb{N} \cup \{0\} \subset \mathbb{Z}^{+\text{ve}}_{\max}$.
If $a = -\infty$ or $b = -\infty$, then $a + b = -\infty \in \mathbb{Z}^{+\text{ve}}_{\max}$. Thus, $\mathbb{Z}^{+\text{ve}}_{\max}$ is closed under '+'.
Similarly, for any $a, b \in \mathbb{N} \cup \{0\}$, $\max\{a, b\} \in \mathbb{N} \cup \{0\}$. If $a = -\infty$, then $\max\{a, b\} = b$ and if $b = -\infty$, then $\max\{a, b\} = a$. Thus, $\mathbb{Z}^{+\text{ve}}_{\max}$ is closed under 'max'.

Since $(\mathbb{R}_{\max}, \max, +)$ is commutative semi-ring, $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$ is also commutative. □

Consider the ring of polynomials, $(\mathbb{R}[x], +, \cdot)$, over the field of real numbers, $(\mathbb{R}, +, \cdot)$. $\mathbb{R}[x]$ consists of all polynomials, in a single variable $x \in \mathbb{R}$, with real coefficients and non-negative integers as powers. Let $\mathbb{P} \subset \mathbb{R}[x]$, be the set of polynomials with non-negative real coefficients, e.g. 0, $3x^2 + 2$, x, 1.5, $7.3x + 9$ etc.

Proposition II.2. *$(\mathbb{P}, +, \cdot)$ is a commutative semi-ring.*

Proof. $(\mathbb{R}[x], +, \cdot)$ is a commutative ring [23]. To prove the proposition, we show that $(\mathbb{P}, +, \cdot)$ is a sub-semiring of $(\mathbb{R}[x], +, \cdot)$. We need the closure of operators and existence of identities in $\mathbb{P}$ (see, Chapter 1 of [22]).

i. The additive and multiplicative identity of $(\mathbb{R}[x], +, \cdot)$ are 0 and 1, respectively. By definition of $\mathbb{P}$, $0 \in \mathbb{P}$ and $1 \in \mathbb{P}$.
ii. The set of non-negative real numbers, $\{x | x \in \mathbb{R}, x \geq 0\}$ is closed under addition and multiplication. The set of non-negative integers, $\mathbb{N} \cup \{0\}$ is closed under addition. Thus, for any $p_1, p_2 \in P$, $p_1 + p_2 \in \mathbb{P}$, as coefficients of $p_1 + p_2$ are non-negative real numbers. Similarly, $p_1 \cdot p_2 \in \mathbb{P}$, as coefficients of $p_1 \cdot p_2$ are non-negative and powers are non-negative integers.

Since, $(\mathbb{R}[x], +, \cdot)$ is commutative, $(\mathbb{P}, +, \cdot)$ is a commutative semi-ring. □

Let $\delta : \mathbb{R}[x] \to \mathbb{Z}^{+\text{ve}}_{\max}$ be the degree function, i.e.

$$\delta(p) = \begin{cases} -\infty & \text{if } p = 0 \\ \text{highest power with} & \\ \text{non-zero coefficient} & \text{otherwise.} \end{cases} \tag{4}$$

We know that (see, Chapter 3 of [23]):

i. $\delta(p_1 \cdot p_2) = \delta(p_1) + \delta(p_2)$, $\forall p_1, p_2 \in \mathbb{R}[x]$.
ii. $\delta(p_1 + p_2) \leq \max\{\delta(p_1), \delta(p_2)\}$, $\forall p_1, p_2 \in \mathbb{R}[x]$.

We now show that $\delta(.)$ is a homomorphism from $\mathbb{P}$ onto $\mathbb{Z}^{+\text{ve}}_{\max}$.

Proposition II.3. *$\delta(.) : \mathbb{P} \to \mathbb{Z}^{+ve}_{\max}$ is a homomorphism from semi-ring $(\mathbb{P}, +, \cdot)$ onto semi-ring $(\mathbb{Z}^{+ve}_{\max}, \max, +)$.*

Proof. We first show that $\delta(.) : \mathbb{P} \to \mathbb{Z}^{+\text{ve}}_{\max}$ is a surjection.

For $-\infty \in \mathbb{Z}^{+\text{ve}}_{\max}$, we have $0 \in \mathbb{P}$ such that $\delta(0) = -\infty$. Similarly, for $0 \in \mathbb{Z}^{+\text{ve}}_{\max}$, we have $1 \in \mathbb{P}$, such that $\delta(1) = 0$. For any $a \in \mathbb{N}$, we have $x^a \in \mathbb{P}$, such that $\delta(x^a) = a$.

For $\delta(.)$ to be a homomorphism between the semi-rings, it needs to map the identities and preserve the operations (see, Chapter 9 of [22]).

i. We have $\delta(0) = -\infty$ and $\delta(1) = 0$. Therefore, $\delta(.)$ maps the additive and multiplicative identities of $(\mathbb{P}, +, \cdot)$ to the corresponding identities of $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$.
ii. We already have established $\delta(p_1 \cdot p_2) = \delta(p_1) + \delta(p_2)$, $\forall p_1, p_2 \in \mathbb{P}$.
We also know $\delta(p_1 + p_2) \leq \max\{\delta(p_1) + \delta(p_2)\}$, $\forall p_1, p_2 \in \mathbb{P}$. We prove equality. If $p_1 = 0$, then $\delta(p_1 + p_2) = \delta(p_2) = \max\{\delta(p_1), \delta(p_2)\}$, as $\delta(p_1) = -\infty$. The case $p_2 = 0$ works analogously.
Let $p_1 \neq 0$ and $p_2 \neq 0$. Then, $\delta(p_1) = a$ and $\delta(p_2) = b$, for some $a, b \in \mathbb{N} \cup \{0\}$. Since we are dealing with commutative semi-rings, without loss of generality, we can assume $a \geq b$. Since coefficients of p_1 and p_2 are non-negative, coefficient of x^a is non-zero in $p_1 + p_2$. Therefore, $\delta(p_1 + p_2) \geq a = \max\{\delta(p_1), \delta(p_2)\}$. This means $\delta(p_1 + p_2) = \max\{\delta(p_1), \delta(p_2)\}$
$\therefore \delta(p_1 + p_2) = \max\{\delta(p_1), \delta(p_2)\}$, $\forall p_1, p_2 \in \mathbb{P}$. □

We define an injection, $\delta'(.) : \mathbb{Z}^{+\text{ve}}_{\max} \to \mathbb{P}$, which computes an inverse of $\delta(.) : \mathbb{P} \to \mathbb{Z}^{+\text{ve}}_{\max}$

$$\delta'(a) = \begin{cases} 0 & \text{if } a = -\infty \\ 1 & \text{if } a = 0 \\ x^a & , \forall a \in \mathbb{N} \end{cases} \tag{5}$$

Clearly, one can observe that:

$$\delta(\delta'(a)) = a, \, \forall a \in \mathbb{Z}^{+\text{ve}}_{\max}. \tag{6}$$

Proposition II.4. *$\delta'(.) : \mathbb{Z}^{+ve}_{\max} \to \mathbb{P}$ is an injection.*

Proof. Let $\delta'(a_1) = \delta'(a_2)$, for any $a_1, a_2 \in \mathbb{Z}^{+\text{ve}}_{\max}$.
If $\delta'(a_1) = \delta'(a_2) = 0$, then $a_1 = a_2 = -\infty$, by definition.
If $\delta'(a_1) = \delta'(a_2) = 1$, then $a_1 = a_2 = 0$, by definition.
Otherwise, $\delta'(a_1) = \delta'(a_2) \Rightarrow x^{a_1} = x^{a_2} \Rightarrow a_1 = a_2$. □

The next theorem is the main result of this section, which allows us to describe computations in $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$ semi-rings in terms of sums and products of polynomials.

Theorem II.5. *The following equality is true:*

$$\max\{a_1 + b_1, a_2 + b_2 \cdots a_k + b_k\} \; = \; \delta(\sum_{i=i}^{m} p_{a_i} \cdot p_{b_i}) \tag{7}$$

where,

i. $m \in \mathbb{N}$ and $m \geq 2$
ii. $a_i, b_i \in \mathbb{Z}^{+ve}_{\max}$, and
iii. $p_a = \delta'(a)$, for $a \in \mathbb{Z}^{+ve}_{\max}$

Proof. We prove the theorem using the principle of mathematical induction. We make use of the semi-ring structures of $(\mathbb{Z}^{+\text{ve}}_{\max}, \max, +)$ and $(\mathbb{P}, +, \cdot)$ (see, Proposition II.1 and Proposition II.2), especially the associativity of operators (see, Definition 1) and the homomorphism, $\delta(.)$, between the semi-rings (Proposition II.3).

For $m = 2$, we have,

$$\begin{aligned}&\max\{a_1 + b_1, a_2 + b_2\}\\&= \max\{\delta(\delta'(a_1)) + \delta(\delta'(b_1)), \delta(\delta'(a_2)) + \delta(\delta'(b_2))\}\\&= \max\{\delta(p_{a_1}) + \delta(p_{b_1}), \delta(p_{a_2}) + \delta(p_{b_2})\}\\&= \max\{\delta(p_{a_1} \cdot p_{b_1}), \delta(p_{a_2} \cdot p_{b_2})\}\\&= \delta((p_{a_1} \cdot p_{b_1}) + (p_{a_2} \cdot p_{b_2}))\end{aligned}$$

Thus, Equation (7) holds for $m = 2$.

Let the statement in Equation (7) hold for $m = k$, $k \in \mathbb{N}, k \geq 2$.

We have,

$$\max\{a_1 + b_1, a_2 + b_2, \cdots, a_k + b_k\} = \delta(\sum_{i=1}^{k} p_{a_i} \cdot p_{b_i})$$

Let $a_0 = \max\{a_1 + b_1, a_2 + b_2, \cdots, a_k + b_k\}$. Similarly, let $p_0 = \sum_{i=1}^{k} p_{a_i} \cdot p_{b_i}$. Clearly, due to closure of operators in semi-rings, $a_0 \in \mathbb{Z}^{+\text{ve}}_{\max}$ and $p_0 \in \mathbb{P}$.

We now show that Equation (7) holds for $m = k + 1$. For any $a_{k+1}, b_{k+1} \in \mathbb{Z}^{+\text{ve}}_{\max}$, we have

$$\begin{aligned}&\max\{a_1 + b_1, a_2 + b_2, \cdots, a_k + b_k, a_{k+1} + b_{k+1}\}\\&= \max\{\max\{a_1 + b_1, a_2 + b_2, \cdots, a_k + b_k\}, a_{k+1} + b_{k+1}\}\\&= \max\{a_0, a_{k+1} + b_{k+1}\}\\&= \max\{\delta(\delta'(a_0)), \delta(\delta'(a_{k+1})) + \delta(\delta'(b_{k+1}))\}, \text{ by (6)}\\&= \delta(p_{a_0} + (p_{a_{k+1}} \cdot p_{b_{k+1}}))\end{aligned}$$

We have $\delta(p_0) = a_0 = \delta(\delta'(a_0)) = \delta(p_{a_0})$. Thus, we get:

$$\begin{aligned}&\delta(p_{a_0} + (p_{a_{k+1}} \cdot p_{b_{k+1}}))\\&= \max\{\delta(p_{a_0}), \delta(p_{a_{k+1}} \cdot p_{b_{k+1}})\}\\&= \max\{\delta(p_0), \delta(p_{a_{k+1}} \cdot p_{b_{k+1}})\}\\&= \max\{\delta(\sum_{i=1}^{k} p_{a_i} \cdot p_{b_i}), \delta(p_{a_{k+1}} \cdot p_{b_{k+1}})\}\\&= \delta((\sum_{i=1}^{k} p_{a_i} \cdot p_{b_i}) + (p_{a_{k+1}} \cdot p_{b_{k+1}}))\\&= \delta(\sum_{i=1}^{k+1} p_{a_i} \cdot p_{b_i})\end{aligned}$$

i.e.

$$\max\{a_1 + b_1, a_2 + b_2, \cdots, a_k + b_k, a_{k+1} + b_{k+1}\} = \delta(\sum_{i=1}^{k+1} p_{a_i} \cdot p_{b_i})$$

Here, we have shown that, Equation (7) holds for $m = k+1$, if it holds for $m = k$. Thus, Equation (7) holds for $m \in \mathbb{N}$, $m \geq 2$. □

C. Discrete Linear Convolutions

In this section we briefly discuss some basic notions and properties of linear discrete convolution which are utilized in our proposed method. The content of this section is extracted from part of [31], focusing on the techniques that are relevant for the extensions in the current paper.

One-Dimensional Discrete Linear Convolution: Consider two 1-dimensional discrete signals $f : F \to \mathbb{R}$ and $g : G \to \mathbb{R}$, where $F, G \subseteq \mathbb{Z}$. The convolution of f and g, denoted by $f \circledast g$, results in a 1-dimensional discrete signal $h : \mathbb{Z} \to \mathbb{R}$, computed by:

$$\begin{aligned}h[k] &= (f \circledast g)[k] = (g \circledast f)[k]\\&= \sum_{i=-\infty}^{\infty} f[i]g[k - i] \ , \quad \forall k \in \mathbb{Z}\end{aligned} \tag{8}$$

In (8), f and g are sufficiently *padded* with 0s, i.e, $f[i] = 0$, if $i \notin F$, and $g[i] = 0$, if $i \notin G$.

If F and G are finite sub-intervals of $\mathbb{Z}$, we might be interested in the output $h = (f \circledast g)$, only over a finite subset $H \subseteq \mathbb{Z}$. This subset is determined by the *mode* of convolution [9]. In this work, we utilize the *full mode* of convolution, in which the output $h_{full} = (f \circledast_{full} g)$ omits all the elements of the linear discrete convolution whose computation only involves *padded* parts of the inputs:

$$\begin{aligned}h_{full}[k] &= (f \circledast_{full} g)[k]\\&= \sum_{i\,:\, i \in F \wedge (k-i) \in G} f[i]g[k - i]\end{aligned} \tag{9}$$

Example 2. *Let us clarify the meaning of the notions from above by giving explicit formulae for corresponding convolutions of two signals of finite lengths n_1+1 respectively n_2+1. Consider two signals $f : [0, n_1] \to \mathbb{R}$ and $g : [0, n_2] \to \mathbb{R}$. The linear discrete convolution of the two signals is given by*

$$h[k] = (f \circledast g)[k] = \begin{cases} \sum_{i=\max\{0,k-n_2\}}^{\min\{n_1,k\}} f[i]g[k - i] \\ \quad \text{if } 0 \leq k \leq n_1 + n_2 \\ 0 \quad \text{otherwise} \end{cases} \tag{10}$$

The full mode of convolution is given by

$$(f \circledast_{full} g)[k] = \sum_{i=\max\{0,k-n_2\}}^{\min\{n_1,k\}} f[i]g[k - i], \quad \forall k \in [0, n_1 + n_2] \tag{11}$$

Example 3. *Let us consider another example of full convolution, involving discrete 1-dimensional signals with negative indices. Let $f_1 : [-1, 7] \to \mathbb{R}$ and $g_1 : [-3, 5] \to \mathbb{R}$.*

$$(f_1 \circledast_{full} g_1)[k] = \sum_{i=\max\{-1,k-5\}}^{\min\{7,k+3\}} f_1[i]g_1[k - i], \quad \forall k \in [-4, 12] \tag{12}$$

Multi-Dimensional Linear Discrete Convolution: Multi-dimensional linear discrete convolution is a straightforward extension of 1-dimensional convolution. Let $\tilde{f} : \tilde{F} \to R$ and $\tilde{g} : \tilde{G} \to R$ be N-dimensional discrete signals, i.e. $\tilde{F}, \tilde{G} \subseteq \mathbb{Z}^N$. Setting $\theta_j := k_j - i_j$, the convolution $(\tilde{f} \circledast \tilde{g}) = \tilde{h} : \mathbb{Z}^N \to \mathbb{R}$ is defined as:

$$\begin{aligned}\tilde{h}[k_1, k_2, \ldots, k_N] &= (\tilde{f} \circledast \tilde{g})[k_1, k_2, \ldots, k_N]\\&= \sum_{i_1=-\infty}^{\infty} \cdots \sum_{i_N=-\infty}^{\infty} \tilde{f}[i_1, \ldots, i_N]\tilde{g}[\theta_1, \ldots, \theta_N]\end{aligned} \tag{13}$$

valid for all $k_1, k_2, \dots k_N \in \mathbb{Z}$ within all the θ_j, and where $\tilde{f}$ and $\tilde{g}$ are sufficiently *padded*.

If $\tilde{F}$ and $\tilde{G}$ are finite and N-dimensional rectangles, i.e. $\tilde{F} = F_1 \times F_2 \times \dots F_N$, $\tilde{G} = G_1 \times G_2 \times \dots G_N$, where all the $F_1, \dots, F_N, G_1, \dots, G_N$ are sub-intervals of $\mathbb{Z}$, the finite domain of interest of output $\tilde{H} \subseteq \mathbb{Z}^N$ is specified by the *mode* of convolution along each dimension, similar to 1-dimensional case.

III. Proposed Method

Let us first sketch the general proceeding of our novel method. We consider in a general setting an N-dimensional grey-value image $f : F \to \mathbb{L}$ and filter (flat or non-flat) $b : B \to \mathbb{L}$, $F, B \subseteq \mathbb{Z}^N$. First we construct $(N+1)$-dimensional arrays, f_{Um} and b_{Um}, for f and b respectively, such that, we have a 1-dimensional vector which represents the coefficients of the polynomial ($\delta'(f(x))$ or $\delta'(b(x))$) corresponding to each pixel (x). The constructions f_{Um} and b_{Um} are similar to the classic notion of the umbra of an image, cf. [18], to which we make a reference with the subscript.

By Theorem II.5 (and Equation (7)), the constructed arrays allow us to transfer the problem of morphological dilation (computing maxima of sum) in N dimensions to that of convolution in $(N+1)$ dimensions. There we can use an efficient method such as the FFT to compute the convolution, $(f \oplus b)_{Um}$, of f_{Um} and b_{Um}. The dilated image, $(f \oplus b)$, can be obtained by appropriately projecting the $(N+1)$-dimensional array $(f \oplus b)_{Um}$ on N-dimensions, i.e. by computing $\delta(p_x)$ at each $x \in F$, where p_x is the polynomial whose coefficients constitute $(f \oplus b)_{Um}(x)$.

The detailed description of our proposed method is given as follows.

Step 1: Let $l_R = \max_{x \in F}\{f(x)\} + \max_{x \in B}\{b(x)\}$. Construct two $(N+1)$-dimensional arrays f_{Um} and b_{Um}. The first N dimensions, referred to as the *domain-dimensions*, of f_{Um} and b_{Um} consists of all (N-dimensional) indices of F and B, respectively. The last dimension, referred to as the *range-dimension* consists of indices $\{0, 1, \dots l_R\}$.

The *range-dimension*, $f_{Um}(x)$ (or $b_{Um}(x)$), represents the coefficients of polynomial $p_x = \delta'(f(x))$ (or $p_x = \delta'(b(x))$) corresponding to the pixel value of image (or filter) at position x. The values in f_{Um} and b_{Um} are determined by the following two equations:

$$f_{Um}(x, y) = \begin{cases} 1 & \text{if } x \in F \text{ and } f(x) = y \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

$$b_{Um}(x, y) = \begin{cases} 1 & \text{if } x \in B \text{ and } b(x) = y \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

Note that the indices of *range-dimension* start from 0. The above construction makes it possible to have image and filter of any shape and including *gaps* in the domain.

We pad or fill the domain appropriately so that F and B are (hyper-) rectangles. For example, if $x_0 \notin B$, then, we can define $b(x_0) = -\infty$, i.e. $\delta'(b(x_0)) = \delta'(0) = 0$, thus, $b_{Um}(x_0, y) = 0$, $\forall y \in \{0, 1, \dots l_R\}$. Thus, the constructed arrays f_{Um} and b_{Um} will always be $(N+1)$-dimensional (hyper-) rectangles in shape, regardless of the shape of image domain F and filter domain B.

Example 4. *(Adopted from [31].) Consider the image f and filter b in Example 1. Then f_{Um} and b_{Um} are 2-dimensional arrays (meaning that we have matrices here), with the column (range dimension) having indices $0, 1, \dots 9$, since f has the range of values in $[0, 7]$ and b in $[0, 2]$. Since the image f in Example 1 has six pixels, f_{Um} has six corresponding columns.*

$$f_{Um} = \begin{bmatrix} 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 0&0&1&0&0&1 \\ 0&0&0&1&0&0 \\ 0&0&0&0&0&0 \\ 0&0&0&0&0&0 \\ 1&0&0&0&0&0 \\ 0&0&0&0&1&0 \\ 0&0&0&0&0&0 \\ \underline{0}&1&0&0&0&0 \end{bmatrix} \quad b_{Um} = \begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&1&0&0 \\ 1&0&0&0 \\ 0&\underline{0}&0&1 \end{bmatrix}$$

Let us note that we write the column entries having in mind the umbra notion, which means that numbering is from left to right (as in f) and from bottom to top (so that row number k from the bottom corresponds to the grey value k, with the added range for holding the possible filtering results when using a non-flat structuring element as by b in the example).

Step 2: We calculate $(f \oplus b)_{Um}$ by taking the linear convolution of f_{Um} and b_{Um}, by using *full mode* on the *domain dimensions* and the *range dimension*. The working of convolution of umbras is explained in the next subsection (see, Subsection III-A)

Within our implementation, this step is sped up using Fast Fourier Transform (FFT).

Example 5. *Continuing Example 4, we demonstrate the computation of $(f \oplus b)_{Um}(2)$. We have, from Equation (19):*

$$\begin{aligned} & (f \oplus b)_{Um}(2) \\ = \; & f_{Um}(0) \circledast_{full} b_{Um}(2) + f_{Um}(1) \circledast_{full} b_{Um}(1) + \\ & f_{Um}(2) \circledast_{full} b_{Um}(0) + f_{Um}(3) \circledast_{full} b_{Um}(-1) \end{aligned}$$

so that

$$(f \oplus b)_{Um}(2) = \begin{bmatrix} 0\\0\\0\\0\\0\\0\\1\\0\\0\\0 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix} + \begin{bmatrix} 1\\0\\0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix} + \begin{bmatrix} 0\\0\\1\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix} = \begin{bmatrix} 1\\0\\1\\0\\0\\0\\1\\0\\0\\0 \end{bmatrix}$$

Step 3: $(f \oplus b)$ is determined from $(f \oplus b)_{Um}$ for each $x \in F$, using the following equation

$$(f \oplus b)(x) = \begin{cases} \max\{y \,|\, (f \oplus b)_{Um}(x, y) \geq 1\} & \text{if } x \in F \oplus B \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where, clearly,

$$x \notin F \oplus B = \{x \,|\, x - x_b \in F \text{ for some } x_b \in B\} \\ \Leftrightarrow \nexists y : (f \oplus b)_{Um}(x, y) \geq 1$$

Equation (16), in essence, computes $\delta(p_x)$, for each $x \in F$ where p_x is the polynomial whose coefficients constitute the *range dimension* $(f \oplus b)_{Um}(x)$.

Thus $(f \oplus b)$ calculated by (16) is the same as defined in (2).

Example 6. *Continuing Example 5, after **Step 2**, we have,*

$$(f \oplus b)_{Um} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ \underline{0} & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

(where we consequently find $(f \oplus b)_{Um}(2)$ in the third column). Using (16)*, we easily obtain $(f \oplus b)$ from the matrix above by taking the highest row index in which the value 1 appears in each column, starting from bottom row equivalent to grey value zero. Thus we get $(f \oplus b) = \begin{bmatrix} \underline{5} & 8 & 9 & 8 & 8 & 9 \end{bmatrix}$, compare Example 1.*

This demonstrates how the proposed method can be used to compute the exact *dilation of an image by a non-flat filter.*

A. Convolution of Umbras

In this subsection, we justify why the convolution performed in **Step 2** of our method produces the desired result.

i. Products of polynomials corresponds to *full mode* of convolution of it's coefficients [19].
Let $p, q \in \mathbb{P}$, $p(x) = \sum_{j=0}^{n} f_j x^j$, $q(x) = \sum_{j=0}^{m} g_j x^j$.
Let f and g be 1-dimensional signals on $[0, n]$ and $[0, m]$ respectively, with $f[j] = f_j$, $\forall j \in [0, n]$ and $g[j] = g_j$, $\forall j \in [o, m]$. Then, we get

$$c(x) = p(x)q(x) = \sum_{j=0}^{m+n} h_j x^j \\ \text{where, } h_j = \sum_{k=\max\{0,j-m\}}^{\min\{n,j\}} f[k]g[j-k] \tag{17}$$

That is, coefficients of c are given by the 1-dimensional signal h, defined on $[0, m+n]$, where $h = f \circledast_{full} g$ (compare, Equations (17) and (11)).

ii. We now focus on the convolution of *umbras*, f_{Um} and b_{Um}. Note that, by construction (in **Step 1**), F and B are finite N-dimensional (hyper-)rectangles and L is the finite set $\{0, 1, \cdots l_R\}$. Therefore, the set of indices, $F \times L$ and $B \times L$ are *finite* $(N+1)$-dimensional (hyper-)rectangles. Thus, we can write

$$\begin{aligned}
&(f_{Um} \circledast b_{Um})[k_1, k_2 \cdots k_{N+1}] \\
&= \sum_{i_1=-\infty}^{\infty} \cdots \sum_{i_{N+1}=-\infty}^{\infty} f_{Um}[i_1, i_2 \cdots i_{N+1}] \\
&\qquad b_{Um}[\theta_1, \theta_2 \cdots \theta_{N+1}] \\
&= \sum_{i_1=-\infty}^{\infty} \cdots \sum_{i_N=-\infty}^{\infty} \Big\{ \sum_{i_{N+1}=-\infty}^{\infty} f_{Um}[i_1, \cdots i_{N+1}] \\
&\qquad b_{Um}[\theta_1, \cdots \theta_{N+1}] \Big\} \\
&= \sum_{i_1=-\infty}^{\infty} \cdots \sum_{i_N=-\infty}^{\infty} \big\{ (f_{Um}[i_1, \cdots i_N, :] \circledast \\
&\qquad b_{Um}[\theta_1, \cdots \theta_N, :])(k_{N+1}) \big\}
\end{aligned} \tag{18}$$

Here, $\theta_j = k_j - i_j$ and $f_{Um}[k_1, \ldots, k_N, :]$ and $b_{Um}[\theta_1, \ldots, \theta_N, :] = b_{Um}[k_1 - i_1, \ldots, k_N - i_N, :]$ are 1-dimensional signals, with index of every dimension, except the last, fixed.
Computing (18) for all indices of the last dimension (*range* dimension) we obtain,

$$\begin{aligned}
&(f_{Um} \circledast b_{Um})[k_1, \cdots, k_N, :] \\
&= \sum_{i_1=-\infty}^{\infty} \cdots \sum_{i_N=-\infty}^{\infty} \big\{ (f_{Um}[i_1, \cdots i_N, :] \circledast b_{Um}[\theta_1, \cdots \theta_N, :]) \big\}
\end{aligned} \tag{19}$$

iii. Taking *full mode* of convolution along each dimension in Equation (19), we obtain

$$\begin{aligned}
&(f_{Um} \circledast_{full} b_{Um})[k_1, \cdots, k_N, :] \\
&= \sum_{i_1 \in F_1 \wedge \theta_1 \in B_1} \cdots \sum_{i_N \in F_N \wedge \theta_N \in B_N} \big\{ (f_{Um}[i_1, \cdots i_N, :] \circledast_{full} \\
&\qquad b_{Um}[\theta_1, \cdots \theta_N, :]) \big\}
\end{aligned} \tag{20}$$

We have

$$\begin{aligned}
&\{i_j | i_j \in F_j \wedge \theta_j \in B_j\} \\
&= \{i_j | i_j \in F_j \wedge k_j - i_j \in B_j\} \\
&= \{k_j - i_j | k_j - i_j \in F_j \wedge i_j \in B_j\} \\
&= \{\theta_j | \theta_j \in F_j \wedge i_j \in B_j\}
\end{aligned}$$

Thus, we can rewrite the Equation (20) as

$$\begin{aligned}
&(f_{Um} \circledast_{full} b_{Um})[k_1, \cdots, k_N, :] \\
&= \sum_{\theta_1 \in F_1 \wedge i_1 \in B_1} \cdots \sum_{\theta_N \in F_N \wedge i_N \in B_N} \big\{ (f_{Um}[\theta_1, \cdots \theta_N, :] \\
&\qquad \circledast_{full} b_{Um}[i_1, \cdots i_N, :]) \big\}
\end{aligned} \tag{21}$$

F_j and B_j are closed sub-intervals of $\mathbb{Z}$, let $F_j = [a_1, a_2]$ and $B_j = [b_1, b_2]$. We show that the set of indices i_j, and thus set of indices $\theta_j = k_j - i_j$, satisfying $\theta_j \in F_j \wedge i_j \in B_j$ form a closed sub-interval of $\mathbb{Z}$.

$$\begin{aligned} i_j \in B_j &\Rightarrow b_1 \leq i_j \leq b_2 \\ k_j - i_j \in F_j &\Rightarrow a_1 \leq k_j - i_j \leq a_2 \\ &\Rightarrow k_j - a_2 \leq i_j \leq k_j - a_1 \\ \therefore \max\{k_j - a_2, b_1\} \leq & i_j \leq \min\{k_j - a_1, b_2\} \end{aligned}$$

Consider a fixed $x = (k_1, k_2 \cdots k_N) \in \mathbb{Z}^N$. Let $I_j = \{i_j | \theta_j \in F_j \wedge i_j \in B_j\}$ and $\Theta_j = \{\theta_j | \theta_j \in F_j \wedge i_j \in B_j\}$, for $j = 1, 2, \cdots N$. I_js and Θ_js are closed sub-intervals of $\mathbb{Z}$ and thus, $\prod_{j=1}^N I_j$ and $\prod_{j=1}^N \Theta_j$ are N-dimensional (hyper-)rectangles.
Moreover, if $y = (i_1, i_2, \cdots, i_N) \in \prod_{j=1}^N I_j$, then, clearly,
$x - y = (k_1 - i_1, k_2 - i_2, \cdots, k_N - i_N) \in \prod_{j=1}^N \Theta_j$
Since, $F = \prod_{j=i}^N F_j$ and $B = \prod_{j=i}^N B_j$, we get

$$\begin{aligned} &\{(x-y, y) | (x-y) \in F \wedge y \in B\} \\ &= \{(x-y, y) | (x-y) \in \prod_{j=1}^N \Theta_j \wedge y \in \prod_{j=1}^N I_j\} \end{aligned} \tag{22}$$

From Equations (21) and (22), we get

$$\begin{aligned} &(f_{Um} \circledast_{full} b_{Um})(x) \\ &= (f_{Um} \circledast_{full} b_{Um})[k_1, \cdots, k_N, :] \\ &= \sum_{\theta_1 \in F_1 \wedge i_1 \in B_1} \cdots \sum_{\theta_N \in F_N \wedge i_N \in B_N} \big\{ (f_{Um}[\theta_1, \cdots \theta_N, :] \circledast_{full} \\ &\qquad\qquad b_{Um}[i_1, \cdots i_N, :]) \big\} \\ &= \sum_{(x-y) \in \prod_{j=1}^N \Theta_j \wedge y \in \prod_{j=1}^N I_j} f_{Um}(x-y) \circledast_{full} b_{Um}(y) \\ &= \sum_{(x-y) \in F \wedge y \in B} f_{Um}(x-y) \circledast_{full} b_{Um}(y) \end{aligned} \tag{23}$$

Clearly, the above equation holds for every $x \in \mathbb{Z}^N$ such that $x - y \in F$ for some $y \in B$, i.e. for every $x \in F \oplus B$. Compare Equation (23) and the formula for dilation, Equation (1). In (23), to compute $(f_{Um} \circledast_{full} b_{Um})$ at x, we take sum of product of polynomials $(f_{Um}(x-y) \circledast_{full} b_{Um}(y))$ at exactly those pair of indices $(\{(x-y, y) | (x-y) \in F \wedge y \in B\})$ on which the maximum of sum of pixel values $f(x-y) + b(y)$ is computed for finding value of dilated image $(f \oplus b)$ at x.

Thus, our constructions in **Step 1** and convolution in **Step 2** allows us to effectively apply Theorem II.5 to compute dilation of images.

B. Time Complexity

Let n_i be the size of the image and n_f be the size of the filter and the range of the grey-values be $[0, n_r - 1]$. **Step 1** takes $\mathcal{O}(2n_i n_r + 2n_f n_r)$. For convolution using FFT in **Step 2**, it takes $\mathcal{O}(2n_i n_r \log(2n_i n_r)) + \mathcal{O}2n_i n_f \log(2n_i n_f))$. **Step 3** takes $\mathcal{O}(2n_i n_r)$. In total, the time complexity is $\mathcal{O}(2n_i n_r \log(2n_i n_r) + 2n_i n_f \log(2n_i n_f))$.

In practice, we have $n_f \leq n_i$ and n_r is a small constant, say c ($c = 256$ and $c = 16$, in case of 8-bit and 4-bit grey-value images, respectively). Therefore, the overall time complexity is $\mathcal{O}(4cn_i \log(cn_i)) = \mathcal{O}(n_i \log(n_i))$.

For the classical dilation, when we use non-flat filter with no restrictions, the time complexity is $\mathcal{O}(n_f n_i)$. As $n_f \to n_i$, we get $\mathcal{O}(n_f n_i) \to \mathcal{O}(n_i^2)$. Therefore, for large images with relatively large filter size (w.r.t. image size) or small range of pixel values (e.g., $[0, 15]$ in 4-bit image or $[0, 255]$ in 8-bit image), our proposed method is more suitable. The experiments in the following sections confirm this.

IV. Experimental Evaluation

In this section we first discuss the computational performance of the proposed method in comparison to several alternative methods for dilation/erosion. After that we demonstrate the qualitative properties of our new exact method in comparison with the computationally related approach based on analytic fast approximation introduced in [7].

A. Quantitative Performance Evaluation

For our discussion, let the image be of size n_i and filter of size n_f, given again in terms of the number of corresponding pixels, with $n_f \leq n_i$. Let us briefly describe all the algorithms that we compare.

We consider the Proposed method (capitalized here for better identification) for 4-bit and 8-bit tonal range, where the latter corresponds to standard grey value range. Though the asymptotic time complexity, $\mathcal{O}(n_i \log n_i)$, remains the same for higher rates, the tonal range of image and filter has a significant effect on the run time of our method. For computations within our scheme, we have used fft.fftn() and fft.ifftn() from NumPy package [2], for FFT and inverse FFT respectively. All computations are performed on a modern workstation (Intel® Xeon ® W-2125 Processor, Fedora Linux 36 (64-bit), 64GB RAM).

The first method for comparison is a naive implementation of classical dilation (denoted here as Classical), i.e. computing by sliding the filter over each pixel. Furthermore, we employ SciPy routine ndimage.grey_dilation() from SciPy package [25] for comparison. This is a highly efficient implementation of the histogram sliding window described in [26], using the approach described in [5] to compute the min and max. The run-time of naive implementation and SciPy method is independent of the tonal range of the image. Therefore, we only test the run-time for 8-bit tonal range.

Note that the worst case time complexity of naive implementation and SciPy method with non-flat filter, even if optimised in implementation, is theoretically still $O(n_f \times n_i)$.

In the first experiment, see Figure 2 *top*, we evaluate the average time taken by varying the filter size on a fixed size of image. The filters and images are filters generated using numpy.random.randint(), with range of values from 0 to 255 for 8-bit Proposed, Classical and SciPy method and with range of values from 0 to 15 for 4-bit Proposed method. The size of images are 512×512. It is clearly visible that the SciPy method and the Classical dilation behave linearly with respect to size of filter. For filter size 32×32 they take about 0.24 and 0.42 seconds absolute computation time on our computer,

Table I
FPGA RESOURCE UTILIZATION

Tonal Range	FFT Size	BRAM	DSP	FF	LUT	URAM
2-bit tonal range	32	112 (5%)	10 (0%)	29636 (1%)	29366 (3%)	2 (0%)
	64	68 (3%)	10 (0%)	33900 (1%)	33246 (3%)	50 (10%)
	128	116 (5%)	10 (0%)	38736 (2%)	44075 (4%)	50 (10%)
	256	392 (20%)	10 (0%)	39772 (2%)	42415 (4%)	50 (10%)
	512	650 (33%)	10 (0%)	43572 (2%)	50655 (5%)	50 (10%)
	1024	1194 (61%)	10 (0%)	47382 (2%)	61894 (6%)	50 (10%)
3-bit tonal range	32	208 (10%)	10 (0%)	31817 (1%)	35821 (3%)	2 (0%)
	64	116 (5%)	10 (0%)	36137 (2%)	38645 (4%)	98 (21%)
	128	212 (10%)	10 (0%)	41077 (2%)	49928 (5%)	98 (21%)
	256	488 (25%)	10 (0%)	42217 (2%)	48324 (5%)	98 (21%)
	512	746 (38%)	10 (0%)	46121 (2%)	57126 (6%)	98 (21%)
	1024	1290 (66%)	10 (0%)	50050 (2%)	68711 (7%)	98 (21%)
4-bit tonal range	32	400 (20%)	10 (0%)	39752 (2%)	52866 (5%)	2 (0%)
	64	212 (10%)	10 (0%)	44195 (2%)	53578 (5%)	194 (41%)
	128	404 (20%)	10 (0%)	49332 (2%)	65769 (7%)	194 (41%)
	256	680 (35%)	10 (0%)	50693 (2%)	64845 (7%)	194 (41%)
	512	938 (48%)	10 (0%)	54806 (3%)	74219 (8%)	194 (41%)
	1024	1482 (76%)	10 (0%)	58929 (3%)	86470 (9%)	194 (41%)
5-bit tonal range	32	788 (40%)	10 (0%)	51967 (2%)	82938 (9%)	2 (0%)
	64	408 (21%)	10 (0%)	56623 (3%)	79420 (8%)	386(83%)
	128	792 (40%)	10 (0%)	62187 (3%)	93431 (10%)	386(83%)
	256	1068 (55%)	10 (0%)	63964 (3%)	93857 (10%)	386(83%)
	512	1326 (68%)	10 (0%)	68493 (3%)	104387 (11%)	386(83%)
	1024	1870 (96%)	10 (0%)	73032 (4%)	117996 (13%)	386(83%)

respectively, up to around 25 seconds for filter size 256×256. The time taken for Proposed method remains constant with the size of the filter, taking on average about 0.76 seconds in 4-bit settings and about 12.5 in 8-bit setting.

In the second experiment, Figure 2 *bottom*, we evaluate the time taken for dilation on varying sizes of images. The image sizes$n_i = n \times n$ increases from 128×128 to 4096×4096. The corresponding filter sizes are $n_f = \lfloor \frac{n}{10} \rfloor \times \lfloor \frac{n}{10} \rfloor$. The images and the filters are generated using numpy.random.randint(), with range of values from 0 to 255, except for 4-bit Proposed method, where the range of values is 0 to 15. As expected, the Proposed method in 4-bit and 8-bit settings perform in $\mathcal{O}(n_i \log n_i)$ time. In 4-bit setting, the Proposed method takes 0.42 seconds for dilation of 128×128 image by 12×12 filter, and 15 seconds for dilation of 2048×2048 image by a 200×200 filter. In 8-bit settings, the Proposed method takes 0.7 and 261.2 seconds, respectively. We have, in the second experiment, $n_f = \lfloor \frac{n}{10} \rfloor \times \lfloor \frac{n}{10} \rfloor \approx \frac{n_i}{100}$. Thus the time complexity of SciPy method and naive method is $\mathcal{O}(n_i n_f) = \mathcal{O}(n_i^2)$. This is also reflected by their run-times in the second experiment.

We observe from the above experiments that our Proposed method is significantly faster than the other methods in the narrow tonal range, as e.g. in the 4-bit setting. The Proposed method is faster than SciPy method and Classical method, also in the usual 8-bit setting, if the ratio of filter size to image size is in the larger range, as seen in first experiment or when working on larger images, even keeping the ratio of filter size to image size constant, as in second experiment.

Let us note that we have not employed GPUs in the above experiments. It is surely worth pointing out that attempts to use GPUs to compute grey value morphology usually incorporates restrictions on symmetry and/or flatness of the filter, see discussions in [15], [24]. However, there are several efficient implementations of FFT and inverse FFT on the GPU, see e.g. [28]. Therefore, our method could be sped up utilising the GPUs, without any restrictions on the filter, making it even more competitive in the large tonal range.

B. Quantitative Results of the Hardware Implementation

In this work, we accelerated the proposed method by implementing it on hardware and observing the results. We used a modern Xilinx FPGA board, the Versal VCK190 Evaluation Board, which houses an XCVC1902-2M FPGA device, to implement a hardware representation of our method. The IP core designed for this method was written in C++ and synthesized using Xilinx Vitis HLS 2022.2. The results from the hardware implementation were compared to the Python code to ensure the functionality of the design. For our hardware tests, we chose images and filters as squares with sizes of each edge in a manner, to form padded images and filters with edge sizes that are powers of 2 to fit the FFT cores optimally. We used Xilinx FFT IP core for 1-dimensional FFTs (forward and inverse) to eventually implement 3-dimensional FFTs.

We tested our method for tonal ranges of 2-bit, 3-bit, 4-bit, and 5-bit, with the filter size set to 5×5, and changed the image size from 28×28 up to 1020×1020. The FPGA resource utilization summary is shown in Table I for these scenarios. It can be seen that the highest resource utilization is in memory blocks, which are used to store the contents of 2-dimensional and 3-dimensional arrays at different points of the procedure. Choosing a higher tonal range limits the largest image size that can be processed, considering the BRAM and URAM resources available on the FPGA.

Figure 3 shows the execution time for the four tonal range scenarios and different FFT sizes. The FFT size addresses the size of the 1-D FFTs processing the x-axis and y-axis of the padded image and filter. The execution time data shows that, in

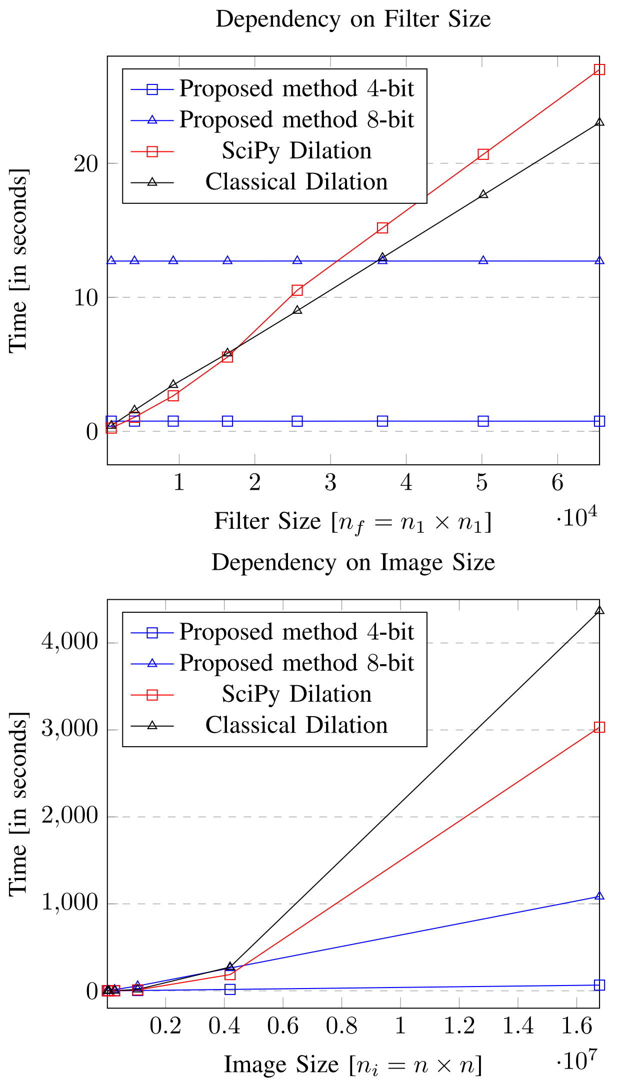

Figure 2. Evaluation of algorithmic time complexity of our method. **Top:** Varying filter sizes on an image of size 512×512. Let us note that the lower axis is given in factors of 10^4. **Bottom:** Varying image sizes $n_i = n \times n$, with filter size $n_f = \lfloor \frac{n}{10} \rfloor \times \lfloor \frac{n}{10} \rfloor$. Let us note that the lower axis is given in factors of 10^7.

each scenario, when the FFT size doubles, the execution time almost doubles as well, which is consistent with the results from the Python implementation of this method.

C. Qualitative Comparison to Previous Fourier Approach

The significant advantage of our method is the fact that we are able to compute the *exact* dilation of an image of size n_i by any non-flat filter of size $n_f \leq n_i$ in $\mathcal{O}(n_i \log n_i)$ time. The Fast Analytical Approximation proposed in [7], is also asymptotically $\mathcal{O}(n_i \log n_i)$ as the new Proposed method and, in fact, has faster run-times in practice. The downside of Fast Approximation method is the non-trivial grey-value shift, that has been studied in detail in [8], which comes along with a smoothing effect in the tonal histogram.

To demonstrate the exactness of the Proposed method and its difference from Fast Approximation, we use a non-flat filter.

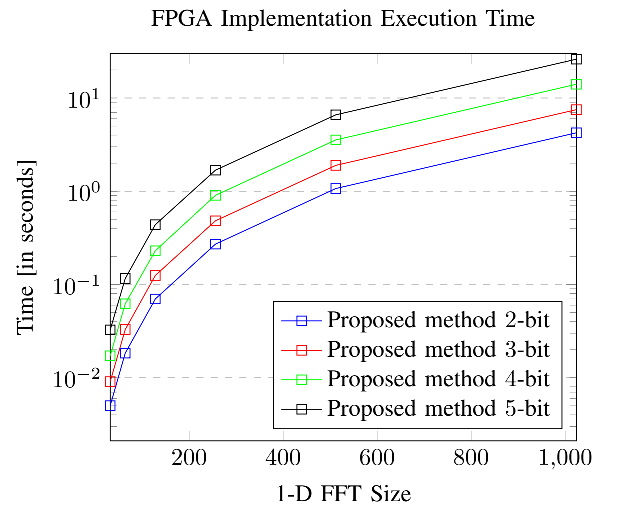

Figure 3. Execution time of the proposed method with different tonal ranges and for different image sizes

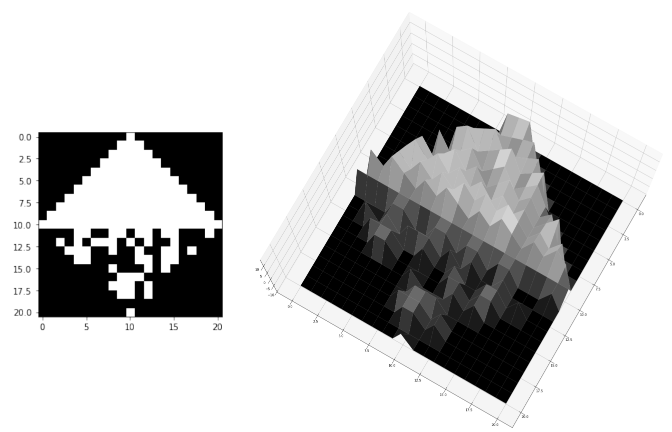

Figure 4. : **Left** The base shape of filter. **Right:** Umbra of the filter of image.

The shape of the filter and its umbra (again after [18]) is shown in Figure 4.

We perform the demonstration of dilation quality on a 512×512 image of *Peppers*, see Figure 5 (left). The Figure 5 (right) shows the dilation of the *Pepper* image with non-flat filter in the classical way, i.e. as described in (2).

Figure 5. : **Left:** *Pepper* image of size 512×512 **Right:** Classical Dilation with the non-flat filter (as described in Figure 4.

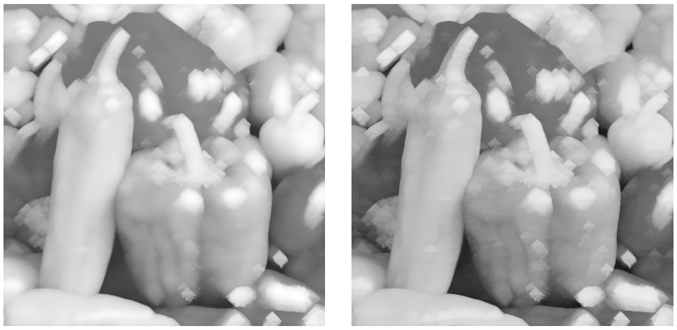

Figure 6. Dilation of *Pepper* image of size 512×512 with the non-flat filter **Left:** Fast Approximation. **Right:** Proposed Method.

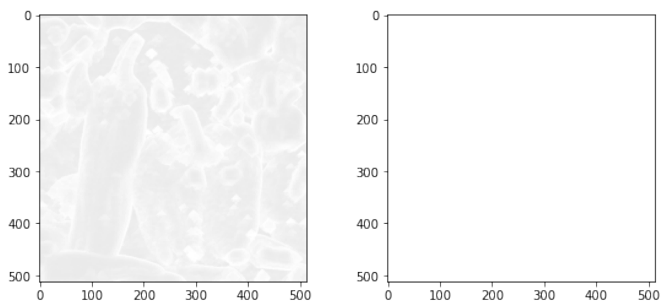

Figure 7. Negative of absolute difference (pixel-wise) with Classical Dilation of *Pepper* image with the non-flat filter **Left:** Fast Approximation. **Right:** Proposed Method.

We compare the Proposed method (in 8-bit settings) and Fast Approximation, see Figure 6, with the classical exact dilation.

The *exactness* of the Proposed method is confirmed visually by looking at the images Figure 5 (right) and Figure 6 (right). This is quantitatively confirmed, in Figure 7, by taking the absolute difference, pixel-by-pixel, of the result of classical dilation with that of fast approximation and proposed method. It is also evident by overlaying the histograms of the images, see Figure 8, that there are no artefacts or dilation accuracy degradation by the boundary treatment inside the FFT.

The positive grey-value shift in Fast Approximation is as expected apparent in Figure 8. Moreover, our proposed method is also successful in preserving the minute details which are lost in Fast Approximation, compare images in Figure 6.

To demonstrate the exactness the proposed method, we perform two experiments, see Figure 9, in 8-bit settings. We measure the average absolute difference in pixel value with classical dilation with fast approximation and the proposed method. In the first experiment, we generate, 100 pair of random 99×99 images and filter, for each filter sizes 3×3, $5 \times 5 \cdots 17 \times 17$, using numpy.random.randint(), with range of values from 0 to 255.Similarly, in the second experiment, we generate, 100 pair of random 5×5 filters and images, for each image sizes 100×100, $200 \times 200 \cdots 600 \times 600$. As expected the Proposed method has 0 average absolute error, i.e. it exactly equates with classical dilation. We can also observe that the absolute error in Fast Approximation increases logarithmically with respect to filter size.

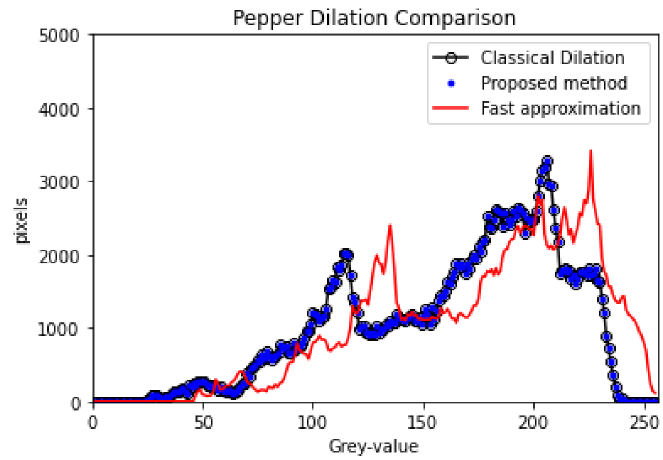

Figure 8. Histogram of dilated images.

V. Conclusion

We have proposed a novel method to *exactly* compute morphological dilation of an image, of size n_i, with any arbitrary non-flat filter, of size $n_f \leq n_i$, in $\mathcal{O}(n_i \log n_i)$ time. As erosion is dual to dilation, it is straightforward to compute erosion analogously.

Because our novel scheme is exact, any useful morphological filter combinations like opening, closing, Beucher gradient, and so on, can be easily combined in a novel and fast way. We conjecture that this may make our algorithm a novel and useful basis for many practical applications.

We have established homomorphism between the max-plus semi-ring of non-negative integers and a semi-ring of polynomials set in the real field. This theory could serve as a foundation to future research relating max-plus semi-rings and plus-prod semi-rings (see e.g. [35], [21]).

Our proposed method allows us to explore some of well-engineered techniques developed for computing convolution in future work. For example, heading for real time applications in very large images, say $n_i \geq 10^9$, some partitioned convolution algorithm [4] can be implemented. We may improve the runtime of our implementation by employing GPUs to calculate FFT and inverse FFT, see e.g. [28], and thus speeding up **Step 2** of our method.

The $(N+1)$-dimensional arrays f_{Um} and b_{Um} constructed in Step 1,see (15) and (16), satisfy the sparsity conditions mentioned in [3]. Therefore, the performance, especially for broader tonal ranges, may also be improved by using Sparse-FFT to compute the convolution in Step 2, see [3].

As we have shown, the new method appears to be particularly useful for narrow tonal ranges, but at the same time it is evident that the method may be highly efficient for standard tonal ranges when exploring advanced computational techniques and hardware as mentioned. By the presented FPGA implementation we highlight the potential usefulness of the new method for many possible applications.

References

[1] Agarwal, R.C. and Burrus, C.S., 1975. Number theoretic transforms to implement fast digital convolution. Proceedings of the IEEE, 63(4),

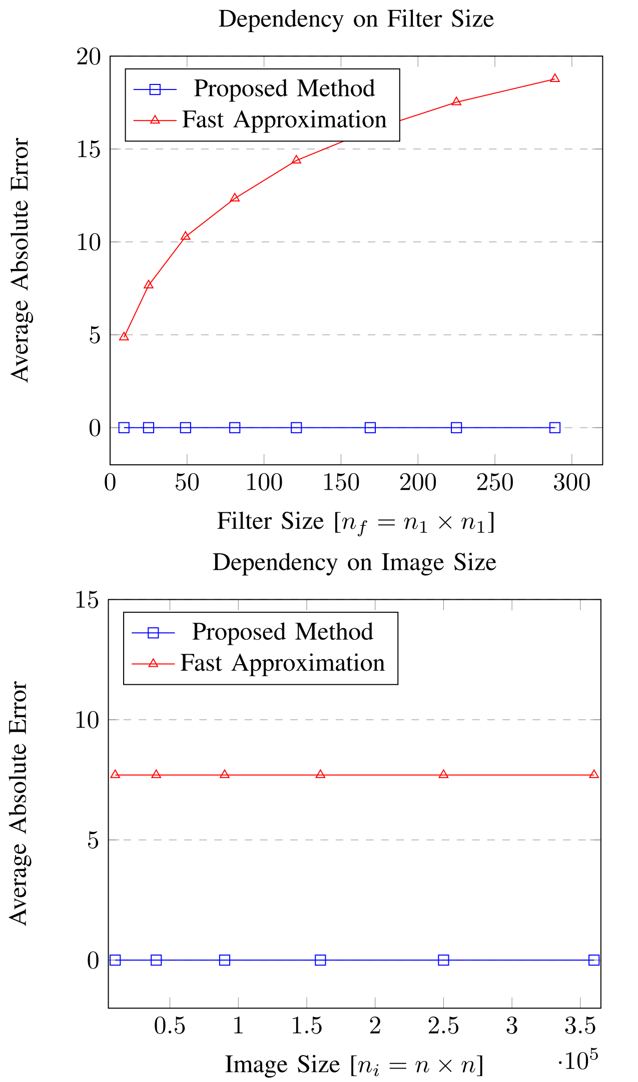

Figure 9. Average *absolute* difference in pixel value with Classical **Top:** Varying filter sizes $n_f = n_1 \times n_1$, on an image of size 99×99. **Bottom:** Varying image sizes $n_i = n \times n$, with filter size 5×5.

pp.550-560.
[2] Harris, C.R., Millman, K.J., Van Der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J. and Kern, R., 2020. Array programming with NumPy. Nature, 585(7825), pp.357-362.
[3] Bringmann, Karl, Nick Fischer, and Vasileios Nakos. "Sparse nonnegative convolution is equivalent to dense nonnegative convolution." In Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing, pp. 1711-1724. 2021.
[4] Wefers, Frank. Partitioned convolution algorithms for real-time auralization. Vol. 20. Logos Verlag Berlin GmbH, 2015.
[5] Douglas, S.C., Running max/min calculation using a pruned ordered list. IEEE Transactions on Signal Processing, 44(11), pp. 2872-2877. 1996.
[6] Tuzikov, A.V., Margolin, G.L. and Grenov, A.I., 1997. Convex set symmetry measurement via Minkowski addition. Journal of Mathematical Imaging and Vision, 7(1), pp.53-68.
[7] Kahra, M., Sridhar, V. and Breuß, M., 2021, May. Fast morphological dilation and erosion for grey scale images using the Fourier transform. In International Conference on Scale Space and Variational Methods in Computer Vision (pp. 65-77). Springer, Cham.
[8] Sridhar, V., Breuß, M. and Kahra, M., 2021, October. Fast Approximation of Color Morphology. In International Symposium on Visual Computing (pp. 488-499). Springer, Cham.
[9] SciPy Documentation, https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.convolve.html. Last accessed 2 Feb 2021.
[10] Serra, J. and Soille, P. eds., 2012. Mathematical morphology and its applications to image processing (Vol. 2). Springer Science and Business Media.
[11] Najman, L. and Talbot, H. eds., 2013. Mathematical morphology: from theory to applications. John Wiley and Sons.
[12] Roerdink, J.B., 2011, July. Mathematical morphology in computer graphics, scientific visualization and visual exploration. In International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (pp. 367-380). Springer, Berlin, Heidelberg.
[13] Kukal, J., Majerová, D. and Procházka, A., 2007. Dilation and erosion of gray images with spherical masks. In the Proceedings of the 15th Annual Conference Technical Computing
[14] Déforges, O., Normand, N. and Babel, M., 2013. Fast recursive grayscale morphology operators: from the algorithm to the pipeline architecture. Journal of Real-Time Image Processing, 8(2), pp.143-152.
[15] Moreaud, M. and Itthirad, F., 2014, April. Fast algorithm for dilation and erosion using arbitrary flat structuring element: Improvement of urbach and wilkinson's algorithm to GPU computing. In 2014 International Conference on Multimedia Computing and Systems (ICMCS) (pp. 289-294). IEEE.
[16] Lin, X. and Xu, Z., 2009, May. A fast algorithm for erosion and dilation in mathematical morphology. In 2009 WRI World Congress on Software Engineering (Vol. 2, pp. 185-188). IEEE.
[17] Van Herk, M., 1992. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. Pattern Recognition Letters, 13(7), pp.517-521.
[18] Haralick, R.M., Sternberg, S.R. and Zhuang, X., 1987. Image analysis using mathematical morphology. IEEE transactions on pattern analysis and machine intelligence, (4), pp.532-550.
[19] Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. MIT press, 2022.
[20] Sridhar, Vivek, and Michael Breuß. "Sampling of Non-flat Morphology for Grey Value Images." In International Conference on Computer Analysis of Images and Patterns, pp. 88-97. Springer, Cham, 2021.
[21] Maragos, Petros. "Tropical geometry, mathematical morphology and weighted lattices." In International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing, pp. 3-15. Springer, Cham, 2019.
[22] Golan, Jonathan S. Semirings and their Applications. Springer Science and Business Media, 2013.
[23] Herstein, I. N. (1975). Topics in Algebra. Wiley. ISBN 0471010901.
[24] Thurley, M.J. and Danell, V., 2012. Fast morphological image processing open-source extensions for GPU processing with CUDA. IEEE journal of selected topics in signal processing, 6(7), pp.849-855.
[25] P. Virtanen et al.: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods **17**(3), 261–272, (2020)
[26] Van Droogenbroeck, M. and Talbot, H., 1996. Fast computation of morphological operations with arbitrary structuring elements. Pattern recognition letters, 17(14), pp.1451-1460.
[27] Van Droogenbroeck, M. and Buckley, M.J., 2005. Morphological erosions and openings: fast algorithms based on anchors. Journal of Mathematical Imaging and Vision, 22(2), pp.121-142.
[28] Moreland, K. and Angel, E., The FFT on a GPU. In Proceedings of the ACM SIGGRAPH/Eurographics conference on Graphics hardware, pp. 112-119. 2003.
[29] Jones, R., 1999. Connected filtering and segmentation using component trees. Computer Vision and Image Understanding, 75(3), pp.215-228.
[30] Agarwal, R.C. and Burrus, C., 1974. Fast convolution using Fermat number transforms with applications to digital filtering. IEEE Transactions on Acoustics, Speech, and Signal Processing, 22(2), pp.87-97.
[31] Sridhar, Vivek, and Michael Breuß. "An Exact Fast Fourier Method for Morphological Dilation and Erosion Using the Umbra Technique." In 2022 19th Conference on Robots and Vision (CRV), pp. 190-196. IEEE, 2022.
[32] Pollard, J.M., 1971. The fast Fourier transform in a finite field. Mathematics of computation, 25(114), pp.365-374.
[33] Rader, C.M., 1972, January. The number theoretic DFT and exact discrete convolution. In IEEE Arden House Workshop on digital signal processing. Harriman NY.
[34] Krizek, M., Luca, F. and Somer, L., 2002. 17 Lectures on Fermat numbers: from number theory to geometry. Springer Science and Business Media.
[35] Cohen, Guy, Stéphane Gaubert, and Jean-Pierre Quadrat. "Max-plus algebra and system theory: where we are and where to go now." Annual reviews in control 23 (1999): 207-219.