Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Leveraging the geodesic distance transform on interval-valued maps to compute the tree of shapes

Edwin Carlinet

edwin1.carlinet@epita.fr

EPITA

Baptiste Esteban

EPITA

Nicolas Boutry

EPITA

Gonzalo Romero-Garcia

EPITA

Research Article

Keywords: Tree of Shapes, Distance transforms, Interval-valued maps, Geodesics

Posted Date: February 26th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-8888553/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Leveraging the geodesic distance transform on interval-valued maps to compute the tree of shapes

Edwin Carlinet, Baptiste Esteban, Nicolas Boutry, Gonzalo Romero-Garcia

EPITA Research Laboratory, 14-16 rue Voltaire, FR-94270 Le Kremlin-Bicêtre, France.

*Corresponding author(s). E-mail(s): edwin1.carlinet@epita.fr;
Contributing authors: baptiste.esteban@epita.fr;
nicolas.boutry@epita.fr; gonzalo.romero-garcia@epita.fr;

The tree of shapes (ToS) is a self-dual and contrast-invariant hierarchical representation of images, making it highly suitable for various image processing tasks such as filtering, segmentation, and object detection. Computing the ToS has been shown to be linear-time under certain conditions, but encounters challenges when applied to high-dynamic range (HDR) images or suffers from scalability issues on massively parallel architectures. This efficient algorithm relies on a specific topological framework that allows *continuous* topological properties on discrete images. Interval-valued maps ensure the correctness of the ToS construction by providing a continuous representation of the image. In this paper, we introduce geodesics on interval-valued maps and leverage the resulting distance maps to compute the ToS. By utilizing the geodesic distance transform, we can efficiently compute the ToS on high-dynamic range images and overcome the scalability issues associated with existing algorithms. The proposed approach allows for a more efficient computation of the ToS, making it suitable for processing large-scale images and enabling its application in various image processing tasks.

Tree of Shapes, Distance transforms, Interval-valued maps, Geodesics

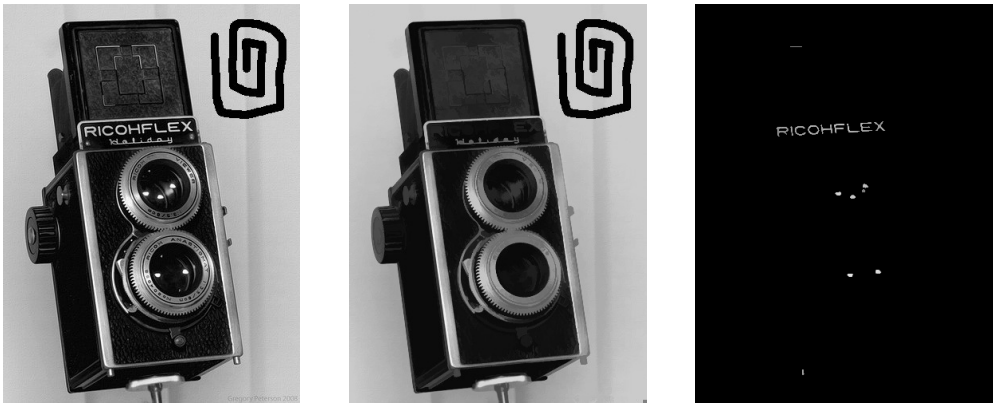


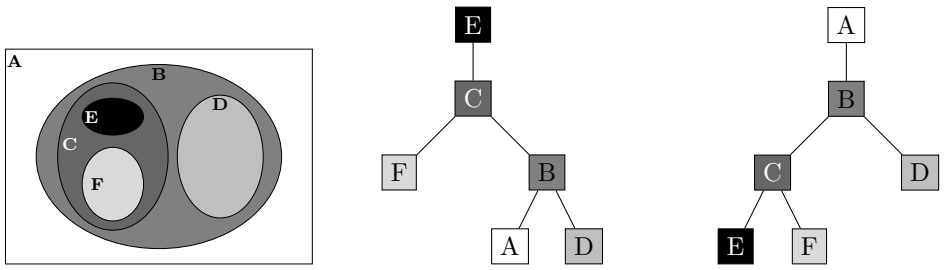
Fig. 1: Illustration of a grain filters with the ToS. Left: original image. Middle: image simplified with a grain filter. Right: residual image after filtering.

1 Introduction

Hierarchical representations of image [1] are extensively employed in image processing. These representations enable the description of objects of interest at multiple levels of detail, based on the inclusion relationships of image regions. Among them, the tree of shapes (ToS) encodes the inclusion relationships of an image's connected components and, consequently, its level lines. It is utilized in various tasks, such as image simplification [2], visual saliency [3], feature extraction [4], and image classification [5]. Figure 1 demonstrates a simple ToS-based application, where a grain filter is used as a pre-filtering step to accelerate text detection. Components are selected according to their size, resulting in a residual image that contains text characters.

The Fast Level Line Transform (FLLT) [6] introduced an algorithm to construct the Tree of Shapes (ToS) by merging the min-tree and max-tree [7]. Subsequently, the Fast Level Set Transform (FLST) [8] employed a region-growing approach to flood an image from its extrema, thereby organizing its level sets. In [9], the author presented a top-down method that initiates from the image border and follows the level lines. Géraud et al. [10] first sorted the level lines of an image based on their inclusion levels and then applied a Union-Find procedure to derive the ToS. Similarly, [11] and [12] adapted the algorithm from [10] by initially constructing an order map and then extracting the ToS from this map using a max-tree, resulting in the first linear and parallel ToS algorithms.

While these algorithms perform well on most images when running on "regular" CPUs, they encounter challenges when applied on some types of images or with specific architectures. For instance, the parallelization strategy adopted in [12] uses a dynamic parallelism (threads are spawned during the algorithm) with a divide-and-conquer approach which is not suitable for massively parallel architectures. The same goes for the algorithm presented in [11] which achieves linear time complexity under specific conditions : the high-performance data structures used in these algorithms rely on a low quantization of pixel values.



(a) An image f (b) Max-tree $\mathcal{T}_{\geq}(f)$ (c) ToS $\mathcal{T}_{\circ}(f)$

Fig. 2: An image and its max-tree and tree of shapes

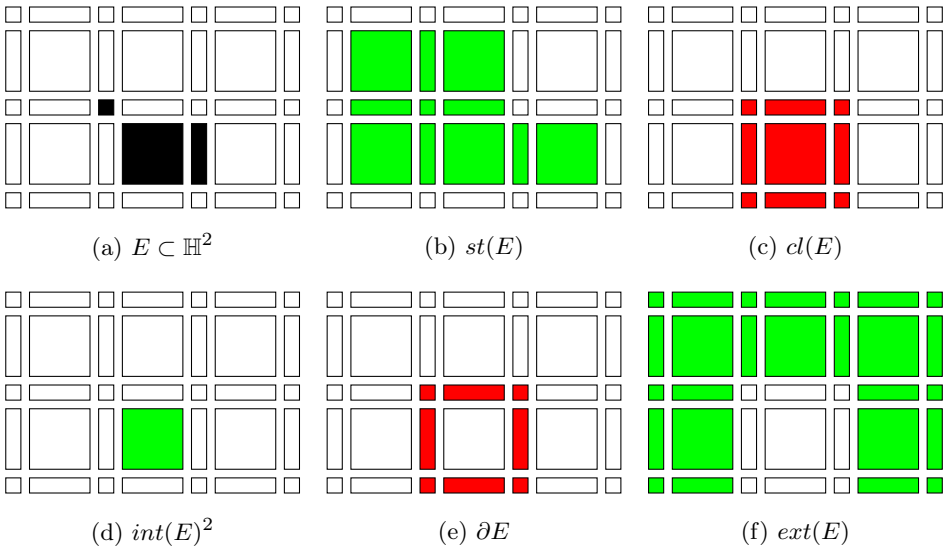
Despite those limitations, the correctness of those algorithms relies on a specific topological framework that allows *continuous* topological properties on discrete images. The interval-valued maps on cubical complexes [13] is a key element that we leverage in this paper. We define geodesics in this context and show how the resulting distance maps can be used to compute the ToS in an efficient way on massively parallel architectures, and on HDR images. This work is an extension of our previous works [14, 15]. Compared to these works, we provide an in-depth topological foundation of the proposed approach and prove its equivalence with the Tree of Shapes definition. We also provide a simple direct application of the produced distance maps for interactive image segmentation.

This paper is structured as follows: Section 2 provides the definition and the topological background necessary to understand the topological framework used in [10, 11] to compute the Tree of Shapes. Section 3 introduces geodesics on interval-valued maps, adapts the Fast Marching algorithm to compute the corresponding distance map and proves the correctness of the algorithm. Section 4 proves the equivalence between the ToS of an image and the max-tree of its distance map produced by the Fast Marching algorithm. Finally, Section 5 presents how this new approach enables both the computation of the ToS on massively parallel architectures and the handling of HDR images. It also provides illustrations of applications of the produced distance maps.

2 Mathematical background

2.1 A rough definition of the tree of shapes

Let $f : \Omega \to \mathcal{V}$ be an image defined on a domain $\Omega \subset \mathbb{Z}^n$ and whose value set $\mathcal{V}$ is endowed with an order relationship $<$. Let $[f \geq \lambda] = \{p \in \Omega \mid f(p) \geq \lambda\}$ be the upper thresholds set and $[f < \lambda] = \{p \in \Omega \mid f(p) < \lambda\}$ be the lower thresholds set, with $\lambda \in \mathcal{V}$. The set of upper connected components (so called *peak components*) is denoted by $\mathfrak{C}^{\geq} = \cup_{\lambda \in \mathcal{V}} \{X \in CC([f \geq \lambda])\}$ and the set of lower connected components by $\mathfrak{C}^{<} = \cup_{\lambda \in \mathcal{V}} \{X \in CC([f < \lambda])\}$, with the operator $CC(X)$ returning the set of connected components in a set X. The max-tree $\mathcal{T}_{\geq}$ (resp. the min-tree $\mathcal{T}_{<}$) encodes the inclusion relationship of the components in $\mathfrak{C}^{\geq}$ (resp. $\mathfrak{C}^{<}$).



(a) $E \subset \mathbb{H}^2$ (b) $st(E)$ (c) $cl(E)$

(d) $int(E)^2$ (e) ∂E (f) $ext(E)$

Fig. 3: Some topological operations on a set E of a 2D cubical complex grid. Open sets are in green, closed sets in red.

A shape $\mathcal{S}$ is a connected component of f with its holes filled and belongs to the set $\mathfrak{C}^\circ = \{\text{sat}(X) \mid X \in \mathfrak{C}^< \cup \mathfrak{C}^\geq\}$ with $\text{sat}(X)$ the hole-filling (saturation) operator. The tree of shapes $\mathcal{T}_\circ$ encodes the inclusion relationship of the shapes of $\mathfrak{C}^\circ$, and thus the level lines of f. Fig. 2 illustrates a max-tree $\mathcal{T}_\geq$ and a tree of shapes $\mathcal{T}_\circ$ computed on an image.

An attentive reader will have certainly noticed that we have casually ignored the issues of dual connectivities in the previous definitions. This problem has been widely addressed before, and we will simply assume that f is well-composed [16]. This translates in 2D to an equivalence between 4 and 8 connectivities for level cuts. If f is not well-composed, the inclusion relationship of shapes does not create a tree. Nevertheless, any image can be made well-composed upstream. The interested reader can refer to [13, 17].

2.2 Interval-valued plain maps and cellular grids

Najman and Géraud [13] introduced the theoretical foundations for computing the tree of shapes free of any topological issues. To this aim, the image is not considered in a regular $\mathbb{Z}^n$ grid but in an n-dimensional space of cubical complexes. In 2D, this space is composed of 2-dimensional cells or 2-faces (the original pixels) but also of cells of lower dimensions: 1-dimensional cells or 1-faces (edges between pixels) and 0-dimensional cells or 0-faces (corners between edges). This space is endowed with a topology (Aleksandrov topology) that respects the inclusion relationships between the different cells. Let σ and τ be two cells, $\gamma \leq \tau$ if σ is a face of τ. The closure of a cell σ, denoted by $cl(\sigma)$, is defined as the set of all cells σ such that $\tau \leq \sigma$. For

instance, the closure of a 2-dimensional cell (a pixel) contains the pixel itself, its four edges, and its four corners. The closure of a 1-dimensional cell (an edge) contains the edge itself and its two corners. In contrast, the Star of a cell σ, denoted by $st(\sigma)$, is the set of all simplices τ such that $\sigma \leq \tau$. The neighborhood of a cell σ is defined as $\mathcal{N}(\sigma) = st(\sigma) \cup cl(\sigma)$. Examples of these topological operations are illustrated in fig. 3.

These definitions extend naturally to sets of cells. A set X is closed if for any cell τ in X, all cells σ in $cl(\tau)$ are also in X (X is stable downwards). Dually, a set O is open if for any cell σ in O, all cells τ in $st(\sigma)$ are also in O (O is stable upwards).

$$st(X) = \bigcup_{\sigma \in X} st(\sigma) \qquad cl(X) = \bigcup_{\tau \in X} cl(\tau) \qquad \mathcal{N}(X) = \bigcup_{\sigma \in X} \mathcal{N}(\sigma)$$

Khalimsky grids are a particular instance of those grids where the cells are indexed by integers. From the sets $\mathbb{H}^1_0 = a; a \in Z$ and $\mathbb{H}^1_1 = a, a+1; a \in Z$, we can define $\mathbb{H}^1 = \mathbb{H}^1_0 \cup \mathbb{H}^1_1$. It generalizes to higher dimensions by defining $\mathbb{H}^n = \mathbb{H}^1 \times \mathbb{H}^1 \times ... \times \mathbb{H}^1$ (n times). We say that h is a d-face of $\mathbb{H}^n$ and that d is the dimension of h if h contains d elements of $\mathbb{H}^1_1$ and $n-d$ elements of $\mathbb{H}^1_0$. In the following, by abuse of notation, coordinates in $\mathbb{Z}^n$ will also refer to the corresponding cells in $\mathbb{H}^n$.

Definition 1. *Let Ω be some topological space. We say that Ω is* unicoherent *when for any A, B closed connected subsets of Ω satisfying $A \cup B = \Omega$, then the intersection $A \cap B$ is connected.*

$\mathbb{H}^n$ is known to be unicoherent, unicoherency can be expressed using open connected sets. In the following, we will note $\mathbb{H}_\Omega$ the Khalimsky grid restricted to the domain Ω of the original image, $\mathbb{H}_\Omega$ being also unicoherent.

Proposition 1. *When A, B are two open connected subsets of $\mathbb{H}_\Omega$ so that $A \cup B = \mathbb{H}_\Omega$, then their intersection is connected.*

Definition 2. *For any subset A of $\mathbb{H}_\Omega$, we define the boundary of A:*

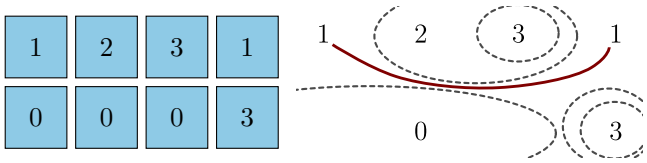
$$\partial A := cl\,(A) \cap cl\,(\mathbb{H}_\Omega \setminus A)$$

which is always a (possibly empty) ***closed*** *set.*

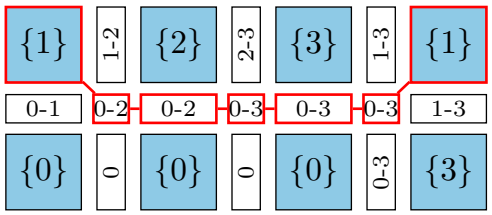
The second key concept introduced in [13] is the set-valued map [18]. Instead of assigning a single value to each cell of the grid, a set of values (actually an interval of values) is assigned to each cell.

Definition 3. *The* span immersion *of f is the plain set-valued map $F : \mathbb{H}_\Omega \to \mathcal{P}(\mathcal{V})$ that associates to each cell $(i, j) \in \mathbb{H}_\Omega$:*

$$F(i,j) = \begin{cases} \{f(\frac{i}{2}, \frac{j}{2})\} & \text{if } i \text{ and } j \text{ are even (2-cell)} \\ [\min f(\tau), \max f(\tau)], \tau \in st(i,j) & \text{if } i \text{ or } j \text{ is odd} \end{cases}$$



(a) Example image and its level lines



(b) Interval-valued map after immersion

Fig. 4: Illustration of the immersion and level set distances. The top-left and top-right "one" pixels are not connected in (a) but they belong to the same level set. In the interval-valued map (b), the level line "One" can be materialized and connects "physically" the two pixels "One": the distance between these two pixels is thus zero.

Immersion is depicted in Figure 4 where original pixels (2-faces) are depicted as blue squares. Faces of lower dimensions and intervals jointly enable to represent *physically* all possible level lines passing between adjacent pair of original pixels. For example, the saddle point $\begin{smallmatrix} 3 & 1 \\ 0 & 3 \end{smallmatrix}$ has level lines from 0 to 3 that pass through the center (depending on the interpolation), while only level lines from 1 to 3 can pass between the two top pixels.

The notion of upper and lower thresholds sets extends naturally to set-valued maps. A strict lower threshold set noted $[F \lhd \lambda]$ and non-strict lower threshold set noted $[F \trianglelefteq \lambda]$ are defined as follows:

$$[F \lhd \lambda] = \{p \in \mathbb{H}_\Omega \mid \forall \alpha \in F(p), \alpha < \lambda\}$$
$$[F \trianglelefteq \lambda] = \{p \in \mathbb{H}_\Omega \mid \exists \alpha \in F(p), \alpha \leq \lambda\}$$

Definition 4. *We say that an interval-valued map is* Alexandrov well-composed, *shortly AWC, when each connected component of each of the boundaries of any of its threshold sets are discrete* $(n-1)$*-surfaces.*

Definition 5. *Let A be some open subset of $\mathbb{H}_\Omega$. We define the* internal boundary *as:*

$$\partial_{int} A := st(\partial A) \cap A$$

which is also an open set as intersection of two open sets.

Definition 6. *Let A, B be two open subsets of $\mathbb{H}_\Omega$, we define:*

$$A \bigcap B := \partial_{int} A \cap \partial_{int} B.$$

Definition 7. *Let p_∞ be some fixed element of $\partial\mathbb{H}_\Omega$. We define as* saturation *of a given set $A \subseteq \mathbb{H}_\Omega$ the set:*

$$\mathrm{sat}(A) := \mathbb{H}_\Omega \setminus CC(\mathbb{H}_\Omega \setminus A, p_\infty)$$

Using these definitions and given the properties of the AWC set-valued map F, Najman and Géraud [13] demonstrated that shapes defined as saturated connected components of the family of sets $\{[F \triangleleft \lambda]\}_{\lambda \in \mathcal{V}}$ and $\{[F \triangleright \lambda]\}_{\lambda \in \mathcal{V}}$ are open and connected or disjoint.

Definition 8. *We denote by $\mathcal{T}(F)$ the (existing and unique) tree of shapes of the given image F:*

$$\mathcal{T}(F) := \{\mathrm{sat}(\Gamma)\ ;\ \Gamma \in CC([F \rhd \lambda]) \cup CC([F \lhd \lambda]),\ \lambda \in \mathbb{R}\}$$

supplied with the natural inclusion-based parenthood relation parent*, that is, $\mathcal{S}_{par} \in \mathcal{T}(F)$ is said to be the (direct)* parent *of $\mathcal{S} \in \mathcal{T}(F)$ in $\mathcal{T}(F)$ when $\mathcal{S} \subsetneq \mathcal{S}_{par}$ and there exists no $\mathcal{S}' \in \mathcal{T}(F)$ satisfying $\mathcal{S} \subsetneq \mathcal{S}' \subsetneq \mathcal{S}_{par}$. In this case, we write:*

$$\mathcal{S}_{par} = parent_{\mathcal{T}(F)}(\mathcal{S}).$$

We naturally extend this notation to parents of order $m \geq 1$ by using the notation $parent^m(\mathcal{S})$ for the m^{th} order parent of $\mathcal{S}$; for sake of simplicity, we will consider $parent^0(\mathcal{S})$ to be the shape $\mathcal{S}$ itself.

2.3 Algorithms for constructing the tree of shapes

This section describes the different steps to compute the tree of shapes based on the algorithms presented in [10] and [11]. To ensure the root of the tree encloses the whole image, the image is padded with a border whose value is usually set to the median value of the image bordering pixels so that the self-duality of the resulting tree is maintained.

Immersion. The *immersion* step involves embedding the image into a set-valued map on a Khalimsky grid as seen in section 2.2.

Propagation. The propagation depicted in fig. 5 aims at sorting the pixels of the image. It processes the previously computed set-valued map F, starting from a root point p_∞ with an associated value v_∞. In [10], this step outputs a sorted array of

Algorithm 1: WaveFront Propagation from [10]

Function `Propagation`(F, p_∞, v_∞):

$Q \leftarrow \varnothing$; $S \leftarrow [\,]$
$\mathcal{F}(p) \leftarrow$ `undef` **for all** p
$\lambda \leftarrow v_\infty$
`Push`(Q, v_∞, p_∞)
$\mathcal{F}(p_\infty) \leftarrow v_\infty$
while Q *not empty* **:**
  $(\lambda, p) \leftarrow$ `Pop`(Q, λ)
  `Append`(S, p)
  foreach $n \in \mathcal{N}(p)$ ***such that*** $\mathcal{F}(n)$ *is undef* **do**
    $\mathcal{F}(n) \leftarrow$ `proj` (λ, `F` (n))
    `Push`(Q, $\mathcal{F}(n)$, n)
return $(\mathcal{F}, S)$

Algorithm 2: Wavefront Propagation from [11]

Function `Propagation`(F, p_∞, v_∞):

$Q \leftarrow \varnothing$
$\mathrm{Ord}(p) \leftarrow -1$ **for all** p
$\lambda \leftarrow v_\infty$, $d \leftarrow 0$
`Push`(Q, v_∞, p_∞)
$\mathrm{Ord}(p_\infty) \leftarrow 0$
while Q *not empty* **:**
  $(\lambda_{\text{new}}, p) \leftarrow$ `Pop`(Q, λ)
  if $\lambda_{new} \neq \lambda$ **:**
    $d \leftarrow d + 1$
    $\lambda \leftarrow \lambda_{\text{new}}$
  foreach $n \in \mathcal{N}(p)$ *such that* $Ord(n) = -1$ **do**
    $\lambda' \leftarrow$ `proj` (λ, `F`(n))
    `Push`(Q, λ', n)
    $\mathrm{Ord}(n) \leftarrow d$
return *Ord*

Note: `proj(x, [a,b])` is defined in definition 9 as $\texttt{proj}(x, [a, b]) = \max(a, \min(x, b))$.

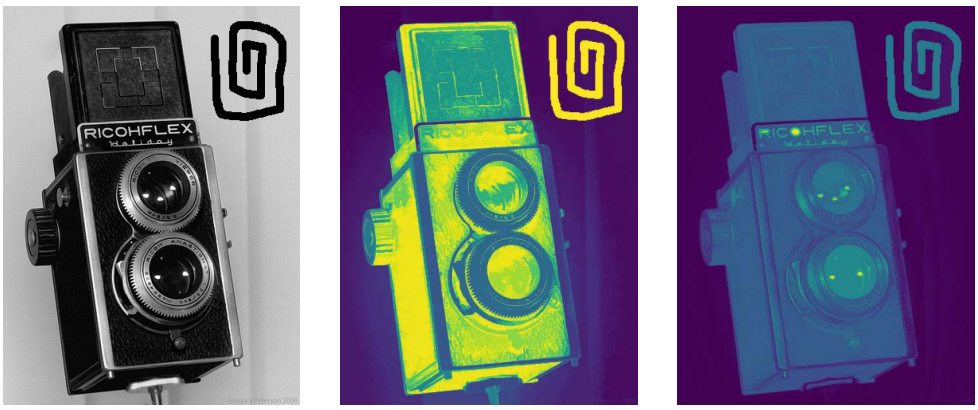
Fig. 5: Propagation algorithm from [10] (left) and order map algorithm from [11] (right). [10] outputs a pair $(\mathcal{F}, S)$ suitable for the Union-Find procedure while [11] outputs the order map Ord suitable for any max-tree algorithm.

pixels S while in [11], it outputs a mapping Ord that associates for each pixel p the wavefront step number (somewhat related to the depth of the pixel in the final ToS). The two algorithms are very similar and differ in the way the "sorting" is stored. In these algorithms, λ denotes the current wavefront level, and d is the wavefront counter. Each time the wavefront level is updated, d is incremented, and the current wavefront level is stored in the order map Ord. Both algorithms rely on a priority queue Q whose efficiency is crucial for the overall performance. The queue is implemented as a hierarchical queue, which is a set of queues $Q_0, Q_1, \ldots, Q_n$ where n is the number of levels in the image.

- Push(Q, v, p) appends a point p to the queue Q_v at level v
- Pop(Q, v) retrieves the point p at the nearest level l of v. It first finds the closest (from v) non-empty queue Q_l and then pops and removes a point p from this queue. The state of the wavefront λ is then updated to l.

The resulting order map Ord, computed during the propagation on the *camera* image, is illustrated in fig. 6b, with values ranging from 0 to 5700.

Tree construction. In [10], the sorted array S is used to build the ToS by immersion using the Union-Find algorithm [19]. In [11], a max-tree is constructed from the



(a) (b) (c)

Fig. 6: The order map (b) computed on the camera image using algorithm 2 and the distance transform image (c) using algorithm 3. The order map values range from 0 to 5700, while the distance transform values range from 0 to 600.

order map using either a linear or quasi-linear algorithm, depending on the maximum value of the order map. After constructing the tree, post-processing steps are typically performed, such as "un-immersion" to return to the original domain and associating the nodes with their original grayscale values.

To adapt this process to H/XDR images, the challenging part is to adapt the propagation algorithm. Indeed, efficient max-tree algorithms have been proposed for High Dynamic Range images [20, 21], but the propagation step consists of continuously traversing a hierarchical queue back and forth to order the pixels. One could obviously tackle this issue by using an ordered set instead of a hierarchical queue, but this would not be efficient (all operations would be $O(\log n)$). A similar issue goes when it comes to parallelization. The max-tree is known to be massively parallelizable [22]. The *immersion* process is also trivially parallelizable (the output depends locally on the 2x2 neighborhood of the original image). However, the propagation step is inherently sequential; the ordering being given by the pixel queue.

As a result, we need to rethink the way the propagation step is performed. This is the topic of the next section where we propose to use the geodesic distance transform on interval-valued maps as an alternative to the order map computed during the propagation step.

3 Geodesic distance on interval-valued maps

We are interested in this section in finding an alternative to the order map computed during the propagation step. The intuition here is that any leveling of the order map (that preserves the ordering relationship between any pair of neighboring pixels) can be used as an order map. Here we propose to use the geodesic distance transform

from the border on an interval-valued image. Instead of counting the number of level-lines (as done in fig. 5), we instead sum the differences between the level-lines on the path. This computes the gray-weighted distance transform [23] or the total variation distance [24] on the interval-valued image F. More formally, let $\Pi(x)$ be the set of paths from the border to the pixel x in the image, the distance map D is defined as:

$$D(x) = \min_{\mathcal{F} \leftarrow F, \pi \in \Pi(x)} TV(\pi) \tag{1}$$

where $TV(\pi)$ is the total variation of the path $\pi = [\pi_1, \pi_2, ..., \pi_n]$, defined as

$$TV(\pi) = \sum_{i=1}^{n-1} |\mathcal{F}(\pi_i) - \mathcal{F}(\pi_{i+1})|$$

$\mathcal{F} \leftarrow F$ stands for any map $\mathbb{H}_\Omega \to \mathcal{V}$ such that $\forall p, \mathcal{F}(p) \in F(p)$. Note that with this distance, marching on the pixels of the same level set does not increase the distance as illustrated in fig. 4 where the distance between the two pixels "One" is zero. The values of 0- and 1-faces are unknowns and must be chosen in the intervals defined by F.

Proposition 2. *For all $d \in \mathbb{R}^+$, the lower level set $[D \leq d]$ are connected and closed. Conversely, the upper level sets $[D > d]$ are open.*

Proof Let us prove that $[D \leq d]$ is path-connected by showing that every point in $[D \leq d]$ can be connected to the border by a path entirely contained in $[D \leq d]$.

Let $x \in [D \leq d]$ and π_x^* be an optimal path from the border to x. Since the distance is computed from the border, for all $x \in \mathbb{H}_\Omega$, D is non-decreasing along the optimal path from $\partial\mathbb{H}_\Omega$ to x. Thus, all the cells in the path π_x^* have their distance value less than or equal to d and then all the cells are included in $[D \leq d]$.

Now, let us prove that $[D \leq d]$ is closed by proving that $\forall x \in [D \leq d]$, $cl(x) \subseteq [D \leq d]$. Suppose $x \in [D \leq d]$ and $y \in cl(x)$. Then, we have $F(y) \subseteq F(x)$ (by definition 3) and thus, we can reach y from the border without increasing the distance.

By definition, the set $[D > d]$ is the complement of the set $[D \leq d]$ which is closed as we just proved. Then, $[D > d]$ is open. □

Theorem 1. *Let X be a hole of the set $[D \leq d]$, $d \in \mathbb{R}^+$, then X is a shape, i.e., there exists $Y \subseteq \mathbb{H}_\Omega$ and $\lambda \in \mathcal{V}$ s.t. $X = \mathrm{sat}(Y)$ with $Y \in CC([F \triangleleft \lambda])$ or $Y \in CC([F \triangleright \lambda])$.*

This is the theorem that establishes the equivalence between the ToS and the max-tree of the distance map. It is proven in section 4.

Definition 9. *$proj(x, Y) : E \times \mathcal{P}(E) \to E$ is the* projection *operator of a value x on the interval $Y = [a, b]$ that solves $\min_{\alpha \in Y} |x - \alpha|$. It is defined as:*

$$proj(x, [a, b]) = \begin{cases} a & \text{if } x < a \\ b & \text{if } x > b \\ x & \text{otherwise} \end{cases} \tag{2}$$

Proposition 3. *Given an interval-valued map F constant on its border ($F(x) = \{c\} \forall x \in \partial\Omega$), let π_x^* be an optimal path from the border to the cell x whose length is $D(x)$ for a some map $\mathcal{F}_x$. There exists an optimal map $\mathcal{F}_x^*$ with values $\mathcal{F}(\pi_{i+1}) = proj(\mathcal{F}(\pi_i), F(\pi_{i+1}))$, $i \in [1, n-1]$ that also achieves the distance $D(x)$ on the same path π_x^*.*

Proof The projection operator is a greedy choice to build $\mathcal{F}$ that minimizes the variations along the path. Given $\mathcal{F}_x$, we can replace sequentially each value along the path by the projection of the previous value without increasing the total variation. Thus, we can build a map $\mathcal{F}_x^*$ that achieves the same distance $D(x)$. □

Proposition 4. *Any optimal path π_x^* from the border to the cell x implies $\mathcal{F}(x) = proj(\mathcal{F}(\pi_{n-1}), F(x))$.*

Proof If $\mathcal{F}(x)$ is not the projection of $\mathcal{F}(\pi_{n-1})$ on $F(x)$, we can replace it by this value and decrease the total variation induced by the last segment of the path, which contradicts the optimality of the path. □

Proposition 5. *There exists an optimal map $\mathcal{F}^*$ that achieves the distance $D(x)$ for all $x \in \mathbb{H}_\Omega$ and this map is unique.*

Proof The problem has an optimal substructure as the distances are summed along paths and the distances are non-negative. It follows that for any cell y in the optimal path π_x^* from the border to x, the sub-path from the border to y is also optimal for y. Using proposition 4, it follows by induction that $\mathcal{F}^*(x)$, $\mathcal{F}^*(\pi_{n-1})$, $\mathcal{F}^*(\pi_{n-2})$, ... all the way back to the border are uniquely defined by the projection operator. Thus, there exists a unique optimal map $\mathcal{F}$ that achieves the distance $D(x)$ for all $x \in \mathbb{H}_\Omega$. □

Because the distance to x only depends on the distance to its neighbors closer to the border, and based on propositions 3 and 5, D can be seen as the solution of the following discrete Eikonal equation:

$$U(x) = \begin{cases} 0 & \text{if } x \in \partial\mathbb{H}_\Omega \\ \min_{y \in \mathcal{N}(x)} U(y) + |\mathcal{F}(x) - proj(\mathcal{F}(y), F(x))| & \text{otherwise} \end{cases} \tag{3}$$

Eikonal equations were designed to model a wavefront propagation in a medium. In our context, the medium is simply a landscape where the gray levels are the elevations.

There are generally two classes of algorithms to solve Eikonal equations: Fast Sweeping Methods [25, 26] and Fast Marching Methods [27]. The Fast Marching Method is essentially a generalization of the Dijkstra algorithm. It works on a grid and maintains a priority queue of front points sorted by their current best-known distance.

Algorithm 3: FMM for interval-valued images

Function `dt`(F, p_∞, v_∞)**:**
- Initialize Q
- $D(p) \leftarrow +\infty$ **for all** p
- $\mathcal{F}(p_\infty) \leftarrow v_\infty$
- `Push`(Q, p_∞, 0)
- **while** Q *not empty* **:**
  - $(p, d) \leftarrow$ `Pop`(Q)
  - **foreach** $q \in \mathcal{N}(p)$ ***such that*** $D(q) = +\infty$ **do**
    - $\mathcal{F}(q) \leftarrow$ `proj` $(\mathcal{F}(p), F(q))$
    - $d_{pq} \leftarrow |\mathcal{F}(p) - \mathcal{F}(q)|$
    - $D(q) \leftarrow d + d_{pq}$
    - `Push`(Q, q, $D(q)$)
- **return** F, D

Algorithm 3 shows our adaptation of the algorithm to compute the distance. It outputs the distance map D and the level line map $\mathcal{F}$. $D(p)$ holds the distance to the border, and $\mathcal{F}(p)$ holds the value chosen in the interval $F(p)$ to reach this distance.

Pixels are classified into three categories: Accepted (pixels whose distance is finalized), Front (pixels that are neighbors of accepted pixels and have a tentative distance), and Far (pixels that have not been reached yet). The status of each pixel follows Far $\rightarrow$ Front $\rightarrow$ Accepted.

It begins by initializing D where each point is assigned a tentative distance value: zero for source points (p_∞ or the image border) and infinity for all others (all points are initially Far). Source points are then added to the priority queue Q that represents the Front. The algorithm iteratively moves the closest Front pixel (from Q) to Accepted and computes the distances of its Far neighbors, which become Front (they are inserted into Q).

There is nothing fancy in this algorithm, it is a classical distance transform very similar to fig. 5. The proof of correctness follows the one of Dijkstra's algorithm or the FMM. However, there is a subtle difference: once a cell goes from *Far* to *Front*, its distance is computed and will never be updated anymore. This property classically holds for elements in *Accepted*, but in this framework, it also holds for elements in *Front*. This is illustrated in fig. 7. On the regular grid, the distance between the gray pixels is 5, thus depending on which one is taken first from Q, the target might be enqueued with a distance $5 + 5 = 10$. Only latter, when the other gray pixel is dequeued, the distance to the target will be updated to 6. On the Khalimsky grid, the target is at distance 2 because the level line "One" can be materialized and connects "physically" the source and the target. The continuity implied by the intervals allows to take a "goat path" through the hillsides.

Proposition 6. *Once a cell x is moved from Far to Front, $D(x)$ is the correct geodesic distance from the border to x, and $\mathcal{F}(x)$ holds the value in $F(x)$ allowing*

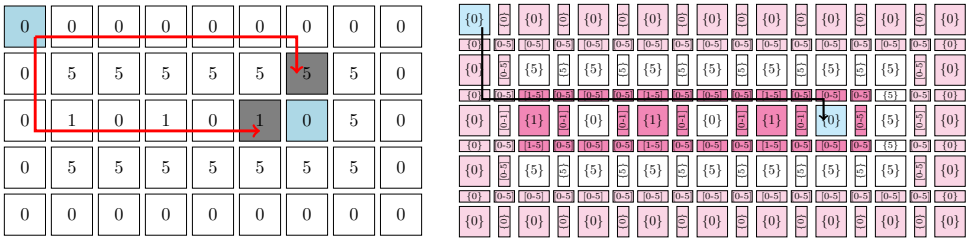


Fig. 7: Fast Marching on a regular grid (left) and on a Khalimsky grid with intervals (right). Source and target are in blue. On the left, the gray pixels are at distance 5 from the source and the target is at distance 6. On the right, the light red pixels are at distance 0, and dark red pixels are at distance 1 from the source. The target is at distance 2 because the level line "One" can be materialized and connects "physically" the source and the target.

to reach this distance. Or equivalently, $\forall x \in$ FRONT *with current candidate distance* $\tilde{D}(x)$*, then* $\tilde{D}(x) = D(x)$*.*

Proof An *update* in Dijkstra's algorithm occurs when a shorter path to a node already in *Front* is found. Let d be the current wavefront level. We have $\forall x \in \text{FRONT}, \tilde{D}(x) \geq d$ and $\forall x \in \text{ACCEPTED}, \tilde{D}(x) = D(x) \leq d$. Let $x \in \text{FRONT}$.

•If $\tilde{D}(x) = d$, then $D(x) = \tilde{D}(x) = d$ because the distance being increasing, $\tilde{D}(x)$ cannot be improved by any path passing through a FRONT pixel.

•Otherwise, $\tilde{D}(x) > d$. We know by theorem 1 that holes are shapes, and thus, that pixels of FRONT that belong to the same hole of x are is in the same monotonic section (either $CC([F \triangleright \lambda])$ or $CC([F \triangleleft \lambda])$). On a monotonic section, the distance between two cells a and b is exactly $|\mathcal{F}(a) - \mathcal{F}(b)|$ and **the continuity** of F ensures that any monotonic curve (path) from a to b has the same length. Thus, any path that reaches x through a FRONT pixel of the same hole as x cannot offer a shorter path than the one through the ACCEPTED pixels, and thus, cannot update the distance to x. □

4 Equivalence between the max-tree of the distance map and the tree of shapes

We will now prove formally that the tree of shapes $\mathcal{T}_S$ of the original image F is the same as $\mathcal{T}_D = \mathcal{T}_{\geq}(\mathbf{D})$ the max-tree of the distance map $\mathbf{D}$ of F. For this matter, we will proceed by double inclusion, first proving that any shape is a peak component of the distance map, and then proving that any peak component is a shape.

4.1 Mathematical preliminaries

Let us first establish some preliminary definitions and properties that will be useful for the proof of the equivalence between the tree of shapes and the max-tree of the distance map. The proofs of these properties are given in the appendix.

Definition 10. *We define as* proper value *of a shape $\mathcal{S} \in \mathcal{T}_{\mathcal{S}}$ the scalar value in $\mathbb{Z}$ such that it is the value used for the reconstruction of U starting from its tree of shapes. In other words, when $\mathcal{S}$ is $\mathbb{H}_\Omega$,*

$$\lambda^*(\mathcal{S}) := \lambda_\infty,$$

when $\mathcal{S}$ is an upper shape, then

$$\lambda^*(\mathcal{S}) := \min\left\{\lfloor U(p) \rfloor \ ; \ p \in \partial_{int}(\mathcal{S})\right\},$$

and when $\mathcal{S}$ is a lower shape, then

$$\lambda^*(\mathcal{S}) := \max\left\{\lceil U(p) \rceil \ ; \ p \in \partial_{int}(\mathcal{S})\right\}.$$

Proposition 7. *Let $\mathcal{E}$ be an element of $\mathcal{T}_{\mathbf{D}}$, then $\mathcal{E}$ is saturated, that is,*

$$Sat(\mathcal{E}) = \mathcal{E}.$$

Proposition 8. *Let $\mathcal{E}$ be some element of $\mathcal{T}_{\mathbf{D}}$, then its interior boundary $\partial_{int}(\mathcal{E})$ is connected.*

Proposition 9. *Let X be some saturated subset of $\mathbb{H}_n$ which does not contain p_∞. Then, we have the following property:*

$$Sat(\partial_{int}(X)) = X.$$

Proposition 10. *Let $\mathcal{E}$ be some element of $\mathcal{T}_{\mathbf{D}}$ which does not contain p_∞. Then, we have the following property:*

$$Sat(\partial_{int}(\mathcal{E})) = \mathcal{E}.$$

Proof This is a direct consequence of proposition 2 and proposition 7 applied to proposition 9. □

4.2 A shape is a peak component of the distance map

Theorem 2. *Any shape $\mathcal{S}$ element of $\mathcal{T}_{\mathcal{S}}$ is also an element of the max-tree $\mathcal{T}_{\mathbf{D}}$.*

Proof We are going to prove by induction of the depth $k \geq 0$ in $\mathcal{T}_{\mathcal{S}}$ that any shape $\mathcal{S}$ is an element of $\mathcal{T}_{\mathbf{D}}$.

Initialization ($k = 0$): when $\mathcal{S}$ is of depth 0, then $\mathcal{S} = \mathbb{H}_\Omega$ (the root of $\mathcal{T}_\mathcal{S}$), which is equal to $[\mathbf{D} \geq 0]$ since $\mathbf{D}$ is positive or equal to 0 everywhere on $\mathbb{H}_\Omega$. Since $\mathbb{H}_\Omega$ is connected (thanks to its unicoherency), $[\mathbf{D} \geq 0]$ is connected, and then $\mathcal{S}$ belongs to $CC([D \geq 0])$, that is, $\mathcal{S} \in \mathcal{T}_\mathbf{D}$.

Initialization ($k = 1$): when a shape $\mathcal{S}$ has a depth equal to 1 in $\mathcal{T}_\mathcal{S}$, then its parent shape $\mathcal{S}_{\text{par}}$ is $\mathbb{H}_\Omega$ and we have the property:

$$\partial_{int}(\mathcal{S}) \cap \Delta\mathbb{H}_\Omega = \emptyset.$$

When $\mathcal{S}$ is of nature $\triangleright$, then $\lambda^*(\mathcal{S}) > \lambda_\infty$, and when $\mathcal{S}$ is of nature $\triangleleft$, then $\lambda^*(\mathcal{S}) < \lambda_\infty$. In both cases, we have the two following properties:

- For any $p \in \partial_{int}(\mathcal{S})$, we have $\mathbf{D}(p) \geq 1$: either $\mathcal{S}$ is of nature $\triangleright$ and $\lambda^*(\mathcal{S}) = \min_{p \in \partial_{int}(\mathcal{S})}\{\lfloor U(p) \rfloor\} \geq \lambda_\infty + 1$, or $\mathcal{S}$ is of nature $\triangleleft$ and $\lambda^*(\mathcal{S}) = \max_{p \in \partial_{int}(\mathcal{S})}\{\lceil U(p) \rceil\} \leq \lambda_\infty - 1$, both cases inducing $d_{pq}(p)$ greater than or equal to one. Since $\mathbf{D}(\mathbb{H}_\Omega^{*,\mathcal{T}_\mathcal{S}}) = 0$ and , for all $p \in \partial_{int}(\mathcal{S})$, $d_{pq}(p) \geq |\lambda^*(\mathcal{S}) - \lambda_\infty|$, we obtain:

$$\forall p \in \partial_{int}(\mathcal{S}),\ \mathbf{D}(p) := \mathbf{D}(\mathbb{H}_\Omega^{*,\mathcal{T}_\mathcal{S}}) + d_{pq}(p) \geq |\lambda^*(\mathcal{S}) - \lambda_\infty| \geq 1.$$

- For any $p \in \partial\mathcal{S}$, $\mathbf{D}(p) = 0$ since p belongs to

$$\mathbb{H}_\Omega^{*,\mathcal{T}_\mathcal{S}} = CC([U \equiv \lambda_\infty], p_\infty),$$

  whose distance transform $\mathbf{D}$ is valued at 0 by the algorithm.

The direct consequence of these two properties is that $\mathcal{S} \in CC([\mathbf{D} \geq 1])$, thus $\mathcal{S} \in \mathcal{T}_\mathbf{D}$. $\mathcal{C}$

Heredity ($k \geq 2$): We assume that the property is true for any depth $\ell \in [0, k-1]$ and we want to show that it implies that the same property is true for a depth equal to k.

Let $\mathcal{S}$ be some shape of depth k at least equal to 2 in $\mathcal{T}_\mathcal{S}$. By induction hypothesis, $\mathcal{S}_{\text{par}}$ is a distance cut since it is a shape of depth lower than k, thus we have:

$$\mathcal{S}_{\text{par}} \in CC([\mathbf{D} \geq \mathbf{D}_{min}(\mathcal{S}_{\text{par}})]).$$

- First, we assume now that we are in the case:

$$\mathcal{S}_{\text{par}} \pitchfork \mathcal{S} = \emptyset,$$

  then $\partial\mathcal{S} \subsetneq \mathcal{S}_{\text{par}}^{*,\mathcal{T}_\mathcal{S}}$. However,

$$\mathcal{S}_{\text{par}}^{*,\mathcal{T}_\mathcal{S}} = \bigcup_{p \in \partial_{int}(\mathcal{S}_{\text{par}})} CC([U \equiv \lambda^*(\mathcal{S}_{\text{par}})], p) \cap \mathcal{S}_{\text{par}},$$

  whose distance transform $\mathbf{D}$ is set at $\mathbf{D}_{min}(\mathcal{S}_{\text{par}})$ by the algorithm. Thus we obtain that:

$$\forall p \in \partial\mathcal{S},\ \mathbf{D}(p) = \mathbf{D}_{min}(\mathcal{S}_{\text{par}}).$$

  Moreover, during the propagation on $\mathcal{S}_{\text{par}}$, we obtain the valuation of $\mathbf{D}$ on every face of $\partial_{int}\mathcal{S}$:

$$\forall p \in \partial_{int}\mathcal{S},\ \mathbf{D}(p) := \mathbf{D}_{min}(\mathcal{S}_{\text{par}}) + |\lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}_{\text{par}})| \geq \mathbf{D}_{min}(\mathcal{S}_{\text{par}}) + 1,$$

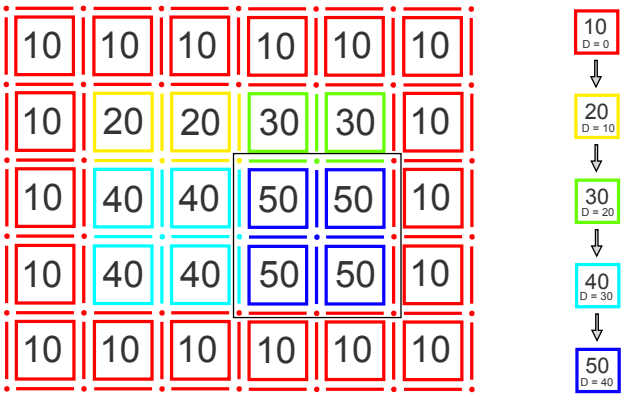


Fig. 8: Order of selection of the points in the boundary of a shape using the distance transform algorithm (we use the same notation as in the previous figures). Starting from the depicted plain map F, we can observe that the distance map on the shape $\mathcal{S}_{50}$ is valuated in this order: first, the red component corresponding to $\mathbb{H}_\Omega = \mathcal{S}_{10}$ minus $\mathcal{S}_{20}$ is processed, so that the red part of the boundary of $\mathcal{S}_{50}$ is processed (and a first part of the internal boundary of $\mathcal{S}_{50}$ is valuated). We continue this way with $\mathcal{S}_{20}$, $\mathcal{S}_{30}$ until $\mathcal{S}_{40}$. In other words, the boundary of $\mathcal{S}_{50}$ has been processed in the order of inclusion of the shapes of F.

which allows us to conclude that $\mathcal{S} \in CC([\mathbf{D} \geq \mathbf{D}_{min}(\mathcal{S}_{\text{par}})+1])$, leading to $\mathcal{S} \in \mathcal{T}_{\mathbf{D}}$.

- Secund, we assume that we are in the case:

$$\mathcal{S}_{\text{par}} \bigcap \mathcal{S} \neq \emptyset.$$

In this context, we need to define the parent $\mathcal{S}^M_{\text{par}}$ of $\mathcal{S}$ in $\mathcal{T}_{\mathcal{S}}$ of minimal order $M \geq 1$ satisfying the relation:

$$\partial\mathcal{S} \subseteq \mathcal{S}^M_{\text{par}}.$$

This leads to the definition of the sequence

$$\mathcal{S}^m_{\text{par}} := \text{parent}^m_{\mathcal{T}_{\mathcal{S}}}(\mathcal{S})$$

for $m \in [1, M]$.

Note that such a $\mathcal{S}^M_{\text{par}}$ necessarily exists since $\mathbb{H}_\Omega$ is parent of all the elements of $\mathcal{T}_{\mathcal{S}} \setminus \{\mathbb{H}_\Omega\}$ and satisfies for every shape $\mathcal{S} \in \mathcal{T}_{\mathcal{S}}$ the relation $\partial\mathcal{S} \subseteq \mathbb{H}_\Omega$.

We will remark that thanks to the induction hypothesis, all the parents of $\mathcal{S}$ are distance cuts since they belong to $\mathcal{T}_{\mathcal{S}}$ and they have a depth lower than k.

Intuitively, these parents correspond to all the shapes "surrounding" $\mathcal{S}$ and from which we will valuate $\partial_{int}\mathcal{S}$ during the propagation (see Figure 8). Indeed,

propagating on $\mathcal{S}_{\text{par}}^M$ will set $\mathbf{D}$ at each position

$$p \in \partial_{int}(\mathcal{S}) \cap \mathcal{N}^*(\mathcal{S}_{\text{par}}^M)$$

at

$$\mathbf{D}(p) := \mathbf{D}_{min}(\mathcal{S}_{\text{par}}^M) + |\lambda^*(\mathcal{S}_{\text{par}}^M) - \lambda^*(\mathcal{S})|,$$

and will set $(\mathcal{S}_{\text{par}}^M)^{*,\mathcal{T}_\mathcal{S}}$ at $\mathbf{D}_{min}(\mathcal{S}_{\text{par}}^M)$. Then, the propagation will continue on $\mathcal{S}_{\text{par}}^{M-1}$ to set $\mathbf{D}$ at each position

$$p \in \partial_{int}(\mathcal{S}) \cap \mathcal{N}^*(\mathcal{S}_{\text{par}}^{M-1}) \setminus \mathcal{N}^*(\mathcal{S}_{\text{par}}^M)$$

at

$$\mathbf{D}(p) := \mathbf{D}_{min}(\mathcal{S}_{\text{par}}^{M-1}) + |\lambda^*(\mathcal{S}_{\text{par}}^{M-1}) - \lambda^*(\mathcal{S})|,$$

and will set $(\mathcal{S}_{\text{par}}^{M-1})^{*,\mathcal{T}_\mathcal{S}}$ at $\mathbf{D}_{min}(\mathcal{S}_{\text{par}}^{M-1})$. This will continue this way until it propagates all over $\mathcal{S}_{\text{par}}^1$ to set each position p belonging to

$$\partial_{int}(\mathcal{S}) \cap \mathcal{N}^*(\mathcal{S}_{\text{par}}^1) \setminus \left(\bigcup_{m \in [2,M]} \mathcal{N}^*(\mathcal{S}_{\text{par}}^m) \right)$$

and to set $\mathbf{D}$ at every point in $(\mathcal{S}_{\text{par}}^1)^{*,\mathcal{T}_\mathcal{S}}$ at $\mathbf{D}_{min}(\mathcal{S}_{\text{par}}^1)$. It will finish by valuating $\mathbf{D}$ all over $\partial_{int}(\mathcal{S})$ in the same way (depending on the possible shapes it contains).

Note now that for any $m \in [0, M-2]$, we have the property:

$$\mathcal{S}_{\text{par}}^m \bigcap \mathcal{S}_{\text{par}}^{m+1} \neq \emptyset$$

(otherwise we would have a lower value M). We can then deduce that the nature of the shapes $\mathcal{S}_{\text{par}}^m$ for $m \in [1, M-1]$ are the same as the one of $\mathcal{S}$.

Let us now assume without constraint that $\mathcal{S}$ is of nature $\triangleright$ (the other case can be treated in a symmetrical manner). We obtain then that:

$$(\lambda^*(\mathcal{S}_{\text{par}}^m))_{m \in [0,M]}$$

is a decreasing sequence.

The propagation occurs in this way. While the propagation occurs all over the different parent shapes $(\mathcal{S}_{\text{par}}^m)^{*,\mathcal{T}_\mathcal{S}}$ for $m \in [1, M]$ by increasing order of depth, the points q belonging to

$$\partial_{int}(\mathcal{S}) \cap \mathcal{N}^*((\mathcal{S}_{\text{par}}^m)^{*,\mathcal{T}_\mathcal{S}})$$

are set from some $p \in \mathcal{N}^*((\mathcal{S}_{\text{par}}^m)^{*,\mathcal{T}_\mathcal{S}})$ in this way:

$$\mathbf{D}(q) := \mathbf{D}(p) + d_{pq}(p,q) = \mathbf{D}_{min}(\mathcal{S}_{\text{par}}^m) + d_{pq}(p,q).$$

Moreover,

$$d_{pq}(p,q) \geq |\lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}_{\text{par}}^m)| \geq \lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}_{\text{par}}^m),$$

which leads to:

$$\mathbf{D}(q) \geq \mathbf{D}_{min}(\mathcal{S}^m_{\rm par}) + \lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}^m_{\rm par}).$$

Case $m = 1$: We can remark that the elements q of $\partial_{int}(\mathcal{S}) \cap \mathcal{N}^*(\mathcal{S}_{\rm par})$ valued by $p \in \mathcal{S}^{*,\mathcal{T}_\mathcal{S}}_{\rm par}$ satisfy:

$$\mathbf{D}(q) \geq \mathbf{D}_{min}(\mathcal{S}_{\rm par}) + 1$$

since $\lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}_{\rm par}) \geq 1$.

Case $m = 2$: Let us remark that the points $p \in (\mathcal{S}^2_{\rm par})^{*,\mathcal{T}_\mathcal{S}}$ from which we value the points $q \in \partial_{int}(\mathcal{S}_{\rm par}) \cap \mathcal{S}^{*,\mathcal{T}_\mathcal{S}}_{\rm par}$ have proceeded this way:

$$\mathbf{D}(q) := \mathbf{D}_{min}(\mathcal{S}^2_{\rm par}) + \lambda^*(\mathcal{S}_{\rm par}) - \lambda^*(\mathcal{S}^2_{\rm par}).$$

Since we have the relation:

$$\min_{q \in \partial_{int}(\mathcal{S}_{\rm par}) \cap \mathcal{S}^{*,\mathcal{T}_\mathcal{S}}_{\rm par}} \mathbf{D}(q) = \min_{q \in \partial_{int}(\mathcal{S}_{\rm par})} \mathbf{D}(q) = \mathbf{D}_{min}(\mathcal{S}_{\rm par}),$$

we have also:

$$\mathbf{D}_{min}(\mathcal{S}_{\rm par}) := \mathbf{D}_{min}(\mathcal{S}^2_{\rm par}) + \lambda^*(\mathcal{S}_{\rm par}) - \lambda^*(\mathcal{S}^2_{\rm par}),$$

or in other words:

$$\mathbf{D}_{min}(\mathcal{S}_{\rm par}) - \lambda^*(\mathcal{S}_{\rm par}) = \mathbf{D}_{min}(\mathcal{S}^2_{\rm par}) - \lambda^*(\mathcal{S}^2_{\rm par}),$$

Consequently, the elements q of $\partial_{int}(\mathcal{S}) \cap \mathcal{N}^*(\mathcal{S}^2_{\rm par})$ valued by $p \in (\mathcal{S}^2_{\rm par})^{*,\mathcal{T}_\mathcal{S}}$ satisfy:

$$\mathbf{D}(q) \geq \mathbf{D}_{min}(\mathcal{S}^2_{\rm par}) + \lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}^2_{\rm par}) = \mathbf{D}_{min}(\mathcal{S}_{\rm par}) + \lambda^*(\mathcal{S}) - \lambda^*(\mathcal{S}_{\rm par}) \geq \mathbf{D}_{min}(\mathcal{S}_{\rm par}) + 1.$$

Case $m \geq 3$: we can continue this way to obtain the same relation for any q in $\partial_{int}(\mathcal{S})$:

$$\mathbf{D}(q) \geq \mathbf{D}_{min}(\mathcal{S}_{\rm par}) + 1.$$

Whatever the (finite) value $M \geq 1$, we will then have that $\mathbf{D}$ is greater than or equal to $\mathbf{D}_{min}(\mathcal{S}_{\rm par}) + 1$ and that $\mathbf{D}$ is lower than $\mathbf{D}(\mathcal{S}_{\rm par}) + 1$ on $\partial\mathcal{S}$, leading to $\mathcal{S} \in CC([\mathbf{D} \geq \mathbf{D}(\mathcal{S}_{\rm par}) + 1])$, that is, $\mathcal{S} \in \mathcal{T}_\mathbf{D}$.

□

4.3 A peak component of the distance map is a shape

Lemma 1. *Let E be some element of $\mathcal{T}_\mathbf{D} \setminus \{\mathbb{H}_\Omega\}$ and whose parent in $\mathcal{T}_\mathbf{D}$ is also in $\mathcal{T}_\mathcal{S}$ and denoted by $\mathcal{S}_{\rm par}$. We have either that, for any $p \in \partial_{int}(E)$, $F(p) \rhd \lambda^*(\mathcal{S}_{\rm par})$ and we say that E is* of F-nature $\rhd$*, or, for any $p \in \partial_{int}(E)$, $F(p) \lhd \lambda^*(\mathcal{S}_{\rm par})$ and we say that E is* of F-nature $\lhd$*.*

Proof Let us assume that $p \in \partial_{int}(E)$ satisfies $F(p) \equiv \lambda^*(\mathcal{S}_{\text{par}})$, then $p \in [\mathbf{D} = \mathbf{D}_{min}(\mathcal{S}_{\text{par}})]$ (contradiction). Consequently, $F(p) \not\equiv \lambda^*(\mathcal{S}_{\text{par}})$, that is, either $F(p) \rhd \lambda^*(\mathcal{S}_{\text{par}})$ or $F(p) \lhd \lambda^*(\mathcal{S}_{\text{par}})$. □

Now, let us assume that there exist $p, q \in \partial_{int}(E)$ such that $F(p) \rhd \lambda^*(\mathcal{S}_{\text{par}})$ and $F(q) \lhd \lambda^*(\mathcal{S}_{\text{par}})$ (we call this Property (**P**)). Since, by Proposition 8, $\partial_{int}(E)$ is connected, we can assume without constraint that p and q are neighbors. However, due to the fact that F is a span-based immersion, then p neighbor of q implies that either $F(p) \subseteq F(q)$ or $F(q) \subseteq F(p)$. Both cases are impossible by Property (**P**), so we have a contradiction. This concludes the proof. □

Theorem 3. *Any peak component E element of $\mathcal{T}_{\mathbf{D}}$ is also an element of $\mathcal{T}_{\mathcal{S}}$.*

Proof We proceed by induction on the threshold $k \in [0, \max_{p \in \mathbb{H}_\Omega} \mathbf{D}(p)]$ used to compute $E \in CC([\mathbf{D} \geq k])$.

Initialization ($k = 0$): The (only) set E of depth $k = 0$ is $\mathbb{H}_\Omega$ and belongs then to $\mathcal{T}_{\mathcal{S}}$.

Initialization ($k = 1$): Let E be some connected component of $[\mathbf{D} \geq 1]$. Since E admits as parent in $\mathcal{T}_{\mathbf{D}}$ the set $\mathbb{H}_\Omega$ which is a shape, we can use Lemma 1 to say that E is of F-nature $\rhd$ or $\lhd$. Let us assume we are in the case $\rhd$ (the other can be treated in a similar way). We have then:

$$\partial_{int}(E) \subseteq [U \rhd \lambda_\infty],$$

which leads to the fact that there exists some

$$\Gamma \in CC([U \rhd \lambda_\infty])$$

with $\partial_{int}(E) \subseteq \Gamma$ (thanks to the connectivity of $\partial_{int}(E)$ obtained by Proposition 8). Also, we have that for any $p \in \partial E$, then $\mathbf{D}(p) = 0$, so $\Gamma \subseteq E$.

We observe then that, thanks to the increasingness of the sat operator, to proposition 10, and to Proposition 7:

$$E = \text{sat}(\partial_{int}(E)) \subseteq \text{sat}(\Gamma) \subseteq \text{sat}(E) = E,$$

thus $E = \text{sat}(\Gamma)$ and then belongs to $\mathcal{T}_{\mathcal{S}}$.

Heredity ($k \geq 2$): Let E be some connected component of $[\mathbf{D} \geq k]$. Let E_{par} be its parent in $\mathcal{T}_{\mathbf{D}}$, then E_{par} is a connected component of $CC([D \geq k'])$ with $k' < k$, so it is a shape of F by the induction hypothesis.

- Firstly, let us treat the case where

$$E \bigcap E_{\text{par}} = \emptyset,$$

  then it is equivalent to the case $k = 1$ seen in the initialization part, we just have to replace λ_∞ and $\mathbb{H}_\Omega$ with $\lambda^*(E_{\text{par}})$ and E_{par} respectively.
- Secondly, let us treat the case where

$$E \bigcap E_{\text{par}} \neq \emptyset.$$

We assume that we are in the case where E has its F-nature set as $\rhd$. Also, we look for the parent $\text{parent}^M_{\mathcal{T}_\mathbf{D}}(E)$ of E in $\mathcal{T}_\mathbf{D}$ of minimal depth $M \geq 0$ such that it contains ∂E. From it, we define the sequence $(E^m_{\text{par}})_{m\in[1,M]} = (\text{parent}^m_{\mathcal{T}_\mathbf{D}}(E))_{m\in[1,M]}$.

Intuitively, we can say that the union of these sets surrounds E:

$$\mathcal{N}^*(E) \subseteq \bigcup_{m\in[1,M]} (E^m_{\text{par}})^{*,\mathcal{T}_\mathbf{D}},$$

which means that every element $q \in \partial_{int}(E)$ has been valuated relatively to some $p \in (E^m_{\text{par}})^{*,\mathcal{T}_\mathbf{D}}$ for some $m \in [1, M]$. Indeed, propagating all over E^M_{par}, then E^{M-1}_{par}, and so on until E_{par} valuates $\partial_{int}(E) \cap \mathcal{N}^*(E^M_{\text{par}})$, $\partial_{int}(E) \cap \mathcal{N}^*(E^{M-1}_{\text{par}}) \setminus \mathcal{N}^*(E^M_{\text{par}})$, and so on until

$$\partial_{int}(E) \cap \mathcal{N}^*(E_{\text{par}}) \setminus \bigcup_{m\in[2,M]} \mathcal{N}^*(E^m_{\text{par}}).$$

The distance transform being set all over $\partial_{int}(E)$, the value $\mathbf{D}_{min}(E)$ exists in $\mathbb{N}$. Necessarily, for each $q \in \partial_{int}(E) \cap \mathcal{N}^*(E^M_{\text{par}})$, d_{pq} has been used this way:

$$d_{pq}(q) := |\lambda^*(E^M_{\text{par}}) - \lambda^{\mathcal{E}}_{\text{min}}|$$

where $\lambda^{\mathcal{E}}_{\text{min}} := \min_{r\in\partial_{int}(E)}\{\lfloor F(r)\rfloor\}$ (due to the F-nature $\rhd$ of E). This value $d_{pq}(q)$ implies that we obtained:

$$\mathbf{D}(q) := \mathbf{D}(p) + d_{pq}(q) = \mathbf{D}_{min}(E^M_{\text{par}}) + d_{pq}(q) = \mathbf{D}_{min}(E^M_{\text{par}}) + \lambda^{\mathcal{E}}_{\text{min}} - \lambda^*(E^M_{\text{par}}).$$

In other words:

$$\forall q \in \partial_{int}(E),\ \mathbf{D}(q) - \mathbf{D}_{min}(E^M_{\text{par}}) = \lambda^{\mathcal{E}}_{\text{min}} - \lambda^*(E^M_{\text{par}})$$

$\mathcal{N}$

We can re-iterate the reasoning for $p \in (E^{M-1}_{\text{par}})^{*,\mathcal{T}_\mathbf{D}}$ and $q \in \partial_{int}(E) \cap \mathcal{N}^*(E^{M-1}_{\text{par}}) \setminus \mathcal{N}^*(E^M_{\text{par}})$ and so on until we obtain for any $m \in [1, M]$:

$$\forall q \in \partial_{int}(E),\ \mathbf{D}(q) - \mathbf{D}_{min}(E^m_{\text{par}}) = \lambda^{\mathcal{E}}_{\text{min}} - \lambda^*(E^m_{\text{par}}). \quad (F)$$

Since, for any $m \in [1, M]$, the value $\lambda^{\mathcal{E}}_{\text{min}} - \lambda^*(E^m_{\text{par}})$ is greater than 0, then we can deduce that:

$$\forall q \in \partial_{int}(E),\ F(q) \rhd \max_{m\in[1,M]} \lambda^*(E^m_{\text{par}}).$$

Additionally, $\partial E \subseteq \mathcal{N}^*(E)$ where any q has been set by some p in some E^m_{par} relatively to the value $\lambda^*(E^m_{\text{par}})$ for some $m \in [1, M]$ (see formula (F)). In other words,

$$\forall p \in \partial E,\ F(p) \trianglelefteq \max_{m\in[1,M]} \lambda^*(E^m_{\text{par}}).$$

Let us define $\lambda^{sol} := \max_{m\in[1,M]} \lambda^*(E^m_{\text{par}})$. We can now define Γ the connected component of $[U \rhd \lambda^{sol}]$ containing $\partial_{int}(E)$. Then, $\partial_{int}(E) \subseteq \Gamma$, implying that

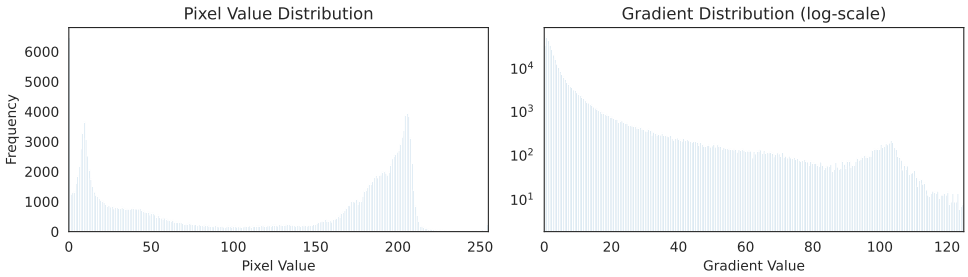


Fig. 9: Distribution of the grayscale values and distribution of gradients of the camera image. The histogram is not uniform and has a long tail: most edges have low gradient values.

$E = \text{sat}(\partial_{int}(E)) \subseteq \text{sat}(\Gamma)$. Furthermore, every neighbor r of E and not in E cannot belong to Γ since $F(r) \not\triangleright \lambda^{sol}$. The consequence is that we have $\Gamma \subseteq E$, so $\text{sat}(\Gamma) \subseteq \text{sat}(E) = E$, leading finally to $E = \text{sat}(\Gamma)$, that is, $E \in \mathcal{T}_{\mathcal{S}}$.

□

5 Beyond the distance transform

5.1 Leveraging the distance transform for ToS computation on XDR images

Algorithm 3 inserts elements into the priority queue Q using absolute distance values. It can be slightly modified to push distances relative to the current wavefront instead of absolute distances. This is possible because the distance is monotonically increasing from the border to any pixel in the image. The distance is bounded as the difference between two level lines is bounded (255 on 8-bit grayscale images). A priority queue can thus be implemented with a circular array of queues to store the pixels as in [28] and enables O(1)-amortized time queue operations. Q stores an active state d that is the current wavefront distance. Then, PUSH and POP used in algorithm 3 just need to be modified to adopt the following rules (an implementation of the circular bucket queue is given in algorithm 4) :

- $\text{PUSH}(Q, p, d_{pq})$ inserts p into the queue $(d+d_{pq}) \bmod 256$, where d_{pq} is the relative distance to the current wavefront.
- $\text{POP}(Q) \to (p, d)$ iterates (circularly) over the queues until it finds a non-empty queue. The internal wavefront distance d is updated accordingly. It then pops the first pixel p from the active queue.

One may wonder why it is preferable to use the distance transform instead of the order map. The answer is simple: *almost none* for low-dynamic range images (except for the fact the distance transform image has generally a lower dynamic and max-tree algorithms run faster on LDR images). However, for extreme dynamic range (XDR) images, using gradients instead of grayscale values is a game-changer. Indeed, while

Algorithm 4: `CircularBucketQueue`

```
class CircularBucketQueue:
    queues ← array of 256 lists
    distance ← 0
    count ← 0
    Function push(p, dist):
        queues[dist mod 256].append(p)
        count ← count + 1
    Function top_distance():
        k ← distance mod 256
        d ← 0
        while queues[k] is empty :
            k ← (k + 1) mod 256
            d ← d + 1
        return distance +d
    Function pop():
        // Update the wavefront distance
        distance ← top_distance ()
        p ← queues[distance mod 256].pop()
        count ← count −1
        return (p, distance)
```

Algorithm 5: `LazyQueue`

```
class LazyQueue:
    data ← [ ]
    validItems ← 0
    minDistance ← +∞
    Function push(p, dist):
        if dist < minDistance :
            minDistance ← dist
        data.append((p, dist))
    Function top_distance():
        return minDistance
    Function pop():
        heapify(data[validItems . . .])
        (p, dist) ← heappop (data)
        validItems ← len (data)
        _, minDistance ← data [0]
        return (p, dist)
```

Fig. 10: Circular bucket queue (left) and lazy queue (right). The circular bucket queue is used for distances within 255 units, while the lazy queue is used for more distant ones.

grayscale levels could be uniformly distributed, the distribution of the gradients (i.e. distances between two level lines) is very unbalanced as illustrated in fig. 9. This is a rather good property as many techniques exist to efficiently implement priority queues when priorities are high-quantized but not uniformly distributed. For instance, *bucket queues* are very efficient when the distribution of the priorities is known and can be pre-computed (this is a generalization of the hierarchical queues with non-uniform buckets). *Hybrid queues* enable maintaining a fast-access structure for high-priority items, and a slower structure for low-priority items, and *lazy queues* defer the update of the queue until necessary. Implementation usually combines many of these techniques to achieve a good performance. For instance, the Fast Marching Method has been optimized with approximated bucket queues [29, 30]. More recently, in [31], the authors combine bucket queue, lazy queues, and a fast-access cache to compute efficiently the α-tree on extreme dynamic range pixel dissimilarities.

The implementation from [31] could probably be adapted to our problem, but contrary to [31], we need to handle *dynamic* updates of the queue. Indeed, in the α-tree algorithm, once the edges are pushed their priorities are fixed. In our case, the *relative* distances change as soon as the wavefront moves. To come up with this issue, we use a simple hybrid strategy that:

Quantization	8 bits	16 bits	24 bits
Immersion	2%	<2%	<2%
Propagation	56%	56%	64%
Max-tree	42%	43%	34%
Proc. Time (MPix/s)	7.47 ± 0.55	1.84 ± 0.13	1.44 ± 0.50

Table 1: Time distribution and total processing time of the ToS algorithm from [11] depending on the quantization.

- stores the pixels in a fast `CircularBucketQueue` for nearby distances within a relative distance $d \leq Q_{th}$
- otherwise, fall backs to the `LazyQueue` that is a deferred-priority queue designed to optimize the trade-off between insertion and extraction costs. Rather than maintaining strict heap order with every insertion, elements are simply appended to a list, and the queue defers sorting until a `dequeue` operation is called. Bulk insertions in a heap with Floyd's algorithm have a better amortized complexity than individual insertions.

The *dequeue* operation of our hybrid queue selects the element with the smallest distance from two underlying queues. It compares the top distances from both queues, updates the global distance to the minimum of the two, and pops the corresponding element. If the selected element comes from the lazy queue, it synchronizes the circular queue's internal distance to ensure consistency for future operations. This hybrid design balances performance and flexibility across a wide range of priority distances. Pseudo-code for the two queues is shown in fig. 10.

We conducted a performance comparison of the ToS computation based on the proposed distance transform (algorithm 3) and the one based on the order transform (algorithm 2). To this aim, we used three different kinds of data: a set of 50 grayscale aerial pictures encoded on 8 bits whose size range from 2430x3500 to 3289x3500 from the database of the Netherlands Institute of Military History[1], a set of 5 images encoded on 16 bits from the Sentinel-2 database[2] from which the first channel has been extracted, and a set of 50 color images encoded on 8 bits whose values have been used as 24 bits values. The CPU of the machine used for the experiments is an Intel Xeon Silver 4000 @ 2.20GHz. Note that the preliminary steps of the algorithm (adding a constant border and immersion) are not included in the performance measurement (they are negligible compared to the ToS computation time).

Table 1 shows the impact of the quantization on the performance of algorithm 2 from [11]. It can be seen that the *propagation* step is the most time-consuming one, accounting for 56% of the total computation time on 8-bit images, and this part is even more important on high-quantized images when the implementation requires red-black trees to keep the pixels in the front ordered. The (quasi-)linear ToS algorithm is no more linear for high-quantized images as the red-black tree requires logarithmic complexity for insertion and deletion operations. This is a major drawback of the ToS

[1] https://beeldbank.nimh.nl/
[2] https://data.inpe.br/stac/browser/collections/S2_L2A-1

Quantization	8 bits	16 bits	24 bits
Carlinet et al. [11]	**7.47** ± 0.55	1.84 ± 0.13	1.44 ± 0.50
CircularBucketQueue	7.07 ± 0.42	4.44 ± 0.25	N/A
HybridQueue	7.07 ± 0.42	**4.44** ± 0.25	**2.26** ± 0.72
hit rate	*100%*	*96.9%*	*57.54%*

Table 2: Timing comparison (MPix/s) of the order map computation algorithm 2 vs our distance transform computation algorithm 3 with a hybrid queue. *hit rate* is the percentage of pixels that were processed using the circular bucket queue (fast) when using the hybrid queue.

algorithm based on the order map that makes it not suitable for high-dynamic range images. This also urges the need for a more efficient propagation step.

Table 2 exposes timings of the proposed approach based on a distance transform implementation that relies on a hybrid queue to handle efficiently low edges and postpones the update of the queue for high edges. They show that our implementation takes the best of the two worlds: it is almost as fast as algorithm 2 from [11] on 8 bits, and it is significantly faster on 16 bits and 24 bits images (two to three times faster). Performance should probably be improved using a more sophisticated hybrid queue as in [31]. To be short, any strategy that would allow to maximize the hit-rate of the circular bucket queue would be beneficial as it turns most logarithmic operations into constant ones. The hit-rate of the circular bucket queue is shown in the last row of table 2 and shows that the hybrid queue is fine on 16-bits images but rather bad on 24-bits images. An analysis of the distribution of the edges could be used to adapt the threshold Q_{th} to the image and would probably improve the hit-rate.

5.2 ToS and distance transform on massively parallel architectures

In section 2.3, we have described the ToS computation as a two-step process: first computing an order map, then computing a max-tree from this order map. To make this process massively parallel, both steps must be parallelizable. The max-tree computation has been shown to be massively parallelizable [22], and the min/max interpolation are trivially parallelizable (as this is a 2x2 stencil). It remains the *Order* map computation that is not trivially parallelizable as it relies on a wavefront propagation.

In section 3, we have proposed to replace the order map by a distance transform computed on an interval-valued map. This distance transform can be computed using the Fast Marching Method that is inherently sequential. Alternatives exist to compute the distance transform without any front propagation. The Fast Sweeping Method [32] for solving the Eikonal equation or [25] for computing the grey-weighted distance transform are iterative methods that perform multiple passes over the image until convergence, each pass performing a Gauss-Seidel update of the distance map.

Reformulating the ToS computation using the distance transform instead of the order map enables the use of these iterative methods to compute the distance map in a

Algorithm 6: Gauss-Seidel iteration for distance transform computation

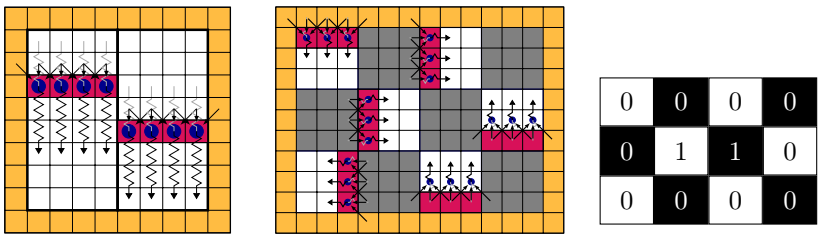
Fn $GSIter_{dir,\mathcal{X},\mathcal{M}}$(***in*** F, ***in out*** $\mathcal{F}$, ***in out*** D)
- $change \leftarrow$ **false**
- **foreach** pixel p in $\mathcal{X}$ in direction dir **do**
  - **foreach** neighbor n in mask $\mathcal{M}$ **do**
    - $\lambda \leftarrow \text{proj}(\mathcal{F}(n), F(p))$
    - $d_{\text{new}} \leftarrow |\mathcal{F}(n) - \lambda| + D(n)$
    - **if** $d_{new} < D(p)$ **:**
      - $D(p) \leftarrow d_{\text{new}}$
      - $\mathcal{F}(p) \leftarrow \lambda$
      - $change \leftarrow$ **true**
- **return** $change$

Algorithm 7: Iterative Distance Transform with FSM for interval-valued images

Function $FSM(F,\ p_\infty,\ v_\infty)$
- $change \leftarrow$ **false**
- Initialize $\mathcal{F}$ and D
- $\mathcal{F}(p_\infty) \leftarrow v_\infty$
- $D(p_\infty) \leftarrow 0$
- **While** $change$
  - **foreach** Col $\mathcal{C}$ in **parallel do**
    - $change \leftarrow \text{GSIter}_{\downarrow,\mathcal{C},\blacksquare}(F, \mathcal{F}, D)$
  - **foreach** Col $\mathcal{C}$ in **parallel do**
    - $change \leftarrow \text{GSIter}_{\uparrow,\mathcal{C},\blacksquare}(F, \mathcal{F}, D)$
  - **foreach** Row $\mathcal{R}$ in **parallel do**
    - $change \leftarrow \text{GSIter}_{\rightarrow,\mathcal{R},\blacksquare}(F, \mathcal{F}, D)$
  - **foreach** Row $\mathcal{R}$ in **parallel do**
    - $change \leftarrow \text{GSIter}_{\leftarrow,\mathcal{R},\blacksquare}(F, \mathcal{F}, D)$
- **return** $\mathcal{F}, D$

Fig. 11: Distance transform computation using the Fast Sweeping Method. The image is swept in four directions (top-down, bottom-up, left-right, right-left) until convergence.

massively parallel way. Parallelizing the distance transform(s) is not a novel idea... In 1968, [33] proposed a parallel version of distance transforms, the idea is to iteratively update the distance of each pixel based on its neighbors until convergence, in **parallel**. To avoid data races, the algorithm runs with a double buffer strategy and needs to synchronize after each round. Let n be the number of pixels in the image, and P the number of processors. This approach has a maximal parallelism, since with $P = n$ number of processors, all pixels can be updated in parallel at each round. In practice, the number of processors is much less than n, and very few pixels need to be updated at each round. In other words, very few processors are making progress at each round. More recently, [34, 35] have proposed a more efficient parallelization strategy that augments the work by processor (or work per thread). Instead of updating a single pixel at each round, each processor is responsible for processing a whole column or a whole row of the image. During a pass, each processor updates the distance based on the distances of its neighbors from the previous row/column as illustrated in algorithm 7 and fig. 12 (a). To solve Eikonal equations, similar ideas have been proposed, as in [36] where the authors propose to process the four directions in parallel (parallel sweeping) and a domain decomposition (tiling) to further increase the parallelism. While these implementations leverage parallel architectures, they require more rounds to converge (compared to the Fast Marching Method) and they do not fully exploit the memory hierarchy of modern architectures. We propose two parallelization strategies that overcome these issues.



(a) Column/Row Blocks (b) 2D Blocks (c) Priority of blocks

Fig. 12: Parallel strategies for the iterative distance transform. (a) Two blocks of 4 threads during the top-down pass. Each thread processes a column. Threads of the same block are synchronized after each line update. (b) Image split in 3x3 blocks colored with a chessboard pattern. This round only updates the white blocks. Blocks are independent, a block of threads updates a block of the image by repeating the four passes until convergence. (c) The priority assigned to each block of (b) based on the distance to the border. Priorities are used to schedule blocks in the task-based implementation.

Algorithm 8: Block-based parallel distance transform.

Function *BlockDistanceTransform*:
    color ← white;
    Initialize block *status* to *inactive* except *white* border blocks;
    while *Not converged* :
        for each Active color *block*, ***in parallel*** :
            Iterate algorithm 7 until convergence of the current block;
            if N/E/W/S *border has been updated* :
                Set *N/E/W/S* neighboring block to *active*;
            Set current block to *inactive*;
        Swap current *color*;
return *D*

Block-based parallelization *(Block)*. It extends the tiling strategy of [36] to better exploits the GPUs programming model. Instead of processing the image row by row or column by column, the distance transform is computed block by block taking advantage of the cache hierarchy (fast CUDA shared-memory). We associate a *status* to each block: *active* or *inactive*. An active block is a block that is adjacent to a block that has been updated during the previous round. To avoid data-race between values that are read/written by adjacent blocks, we use an odd-even communication strategy. Blocks have colors using a chessboard pattern. At each round, we update all *white active* blocks, then all *black active* blocks, and so on... as illustrated in fig. 12b The algorithm is then as follows:

Method	MPixels/s	#Rounds	Speed-up
CPU	12.3 ± 1	1	1
GPU - Naïve	1.1 ± 0.5	1130 ± 480	0.1
GPU - Block	25.7 ± 8.3	1835 ± 802	1.9
GPU - Queue	85.9 ± 19.8	2196 ± 843	6.7

Table 3: Performance comparison of the distance transform with different parallelization strategies.

Task-based parallelization *(Task Queue).* Using the classical CUDA programming model, the block-based approach has two drawbacks. First, at each round, each block is scheduled even if inactive (it runs but exits immediately). Second, blocks are scheduled in any order and if the image is complex, there might be a lot of active blocks at each round. As we compute a distance transform from the border, we want to prioritize the blocks close to the border as they are the most likely to get their final state. On the contrary, the most "inner" blocks are likely to be invalidated more often. We thus propose to switch to a CUDA "task-based" parallelization with priority queue to schedule the blocks. The priority is *based* on the distance of the block to the border as shown on fig. 12c. Each "CUDA block" (*worker*) retrieves a *physical* block from the queue, processes it, and pushes its neighbors to the queue as previously. The number of workers is set according to the number of Streaming Multiprocessors (SM) of the GPU, and each worker will process a maximum predefined number of blocks (#border blocks / #workers). Thus, it *may not* process all *active* blocks at each round, only the most important ones that are likely to converge and not be invalidated anymore. Note that the priority is not "just" the distance of the block from the border. The value is mapped to [0-63] in order to get one queue per priority that leverages fast concurrent lock-free queue updates. Compared to the block-based approach, the queue-based approach requires more advanced lock-free programming techniques.

We have conducted a performance comparison of the proposed distance transform algorithms and also compared against the CPU propagation algorithm. The bench dataset contains 639 grayscale aerial pictures (with sizes ranging from 2430×3500 to 3289×3500) from the database of the Netherlands Institute of Military History [3]. The CPU is an Intel Xeon Silver 4200 @ 2.20GHz and the GPU is an NVidia Quadro RTX 8000, the timings do not include memory transfers between host and device memories.

The results are reported in table 3. It shows that the naïve GPU distance transform is much slower than the CPU propagation. This was expected because it requires many rounds to converge (with 4 passes over the entire image at each). The block-based parallelization is much faster than the naïve one, due to data locality and cache reuse (processing the blocks in shared memory). The queue-based is the fastest, even if it requires more rounds to converge, each round is much faster because it only processes the most important blocks. The speed-up reaches 84x compared to the naïve GPU implementation and 6.7 compared to the CPU propagation.

[3] https://beeldbank.nimh.nl/

Method	ToS on CPU	ToS on GPU
Immersion	82 (2%)	0.22 (<1%)
Propagation/DT	2921 (56%)	440 (97%)
Max-tree	2186 (42%)	9.32 (2%)
Total	5189	450

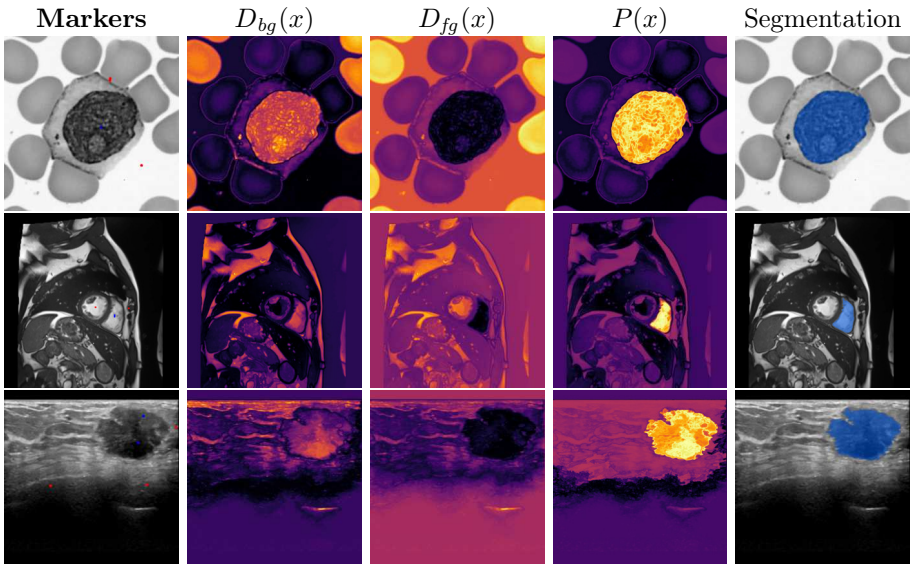
Table 4: Distribution of the computation time (in ms) of the sequential ToS algorithm on CPU and the massively parallel ToS on GPU.

However, this benchmark is **unfair** to the CPU algorithms for two reasons. First, we did not use the parallel version from [12]. In their work, the authors did not benchmark the contribution of the parallelization of each step (they consider the speed-up of the whole pipeline). In our tests, the parallelization of fig. 5 with many threads did not scale well because the work in the queues is unbalanced. The speed-up, at the level of the propagation, was not significant. The performance could probably be improved by a better workload balancing, but we did not investigate this. Second, the GPU algorithms and parallelization strategies proposed in this paper can be implemented on the CPU using a thread-level parallelism **and** a data-level parallelism, i.e. using suitable SIMD instructions. However, the implementation is much less straightforward, and the compiler auto-vectorization has failed in issuing the SIMD code required for such implementation. Nevertheless, even if the CPU version could be improved, the speed-up would be at best the number of cores of the CPU, so our GPU implementation would still be challenging to beat.

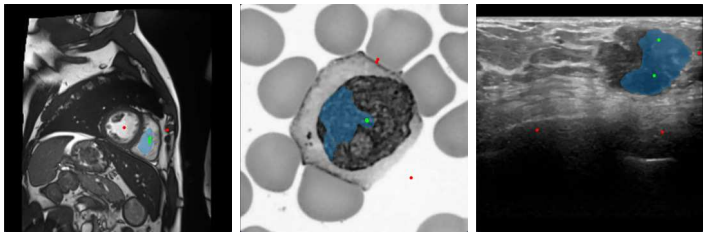
Table 4 shows the distribution of the computation time of the ToS algorithm on CPU (sequential) and GPU with the Queue-based distance transform. We do not want to claim that the GPU version outperforms the CPU version, even if it does in this experiment (it should be benchmarked against an optimized parallel version). Our GPU distance transform is a direct replacement of the propagation algorithm, but does not gain as much as the immersion/max-tree steps. It is now by far the bottleneck of the ToS algorithm. However, even if we do not get a 10x speed-up on this step, the overall speed-up is still significant and the ToS can now be fully computed on the GPU without any memory movement to the Host. The last point was our main motivation in this work as the memory transfer that goes forth and back between the host and the device in image processing pipelines is often the bottleneck.

5.3 Leave the ToS aside ! A simple application of distance maps on interval-valued images

Object segmentation is a classical application of the distance transform. In [37, 38], the authors use the geodesic distance from two sets of seeds to segment the image into two parts using two background and foreground probability maps. In [39], the tree of shapes is used as a structure to compute and efficiently update such distance



(a) Sample results with the geodesic distance transform computed with algorithm 3



(b) Sample results with ScribblePrompt[4]

Fig. 13: Seeded segmentation results with the proposed geodesic distance transform on three samples from the ScribblePrompt demos dataset.

maps interactively, and in [40, 41], the authors use this same tree to compute the minimum-barrier distance and increase the robustness of the segmentation.

By extending algorithm 3 to consider a set of seeds instead of the image border as the source of the distance transform (i.e. $p_\infty = \{\text{seeds}\}$), we can now compute a geodesic distance-map-based segmentation on HDR images and leverage massively parallel implementations to compute it efficiently. The method can be summarized as follows:

- Compute $D_{fg}(x)$ and $D_{bg}(x)$, the geodesic distances from the foreground and background seeds respectively, using the proposed distance transform algorithm.
- Compute the probability map $P(x) = \frac{D_{bg}(x)}{D_{fg}(x)+D_{bg}(x)}$.
- Threshold the probability map to get the segmentation mask. The threshold (by default 0.5) can be set by the user.

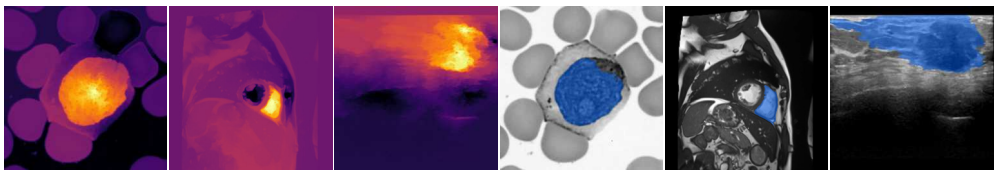


Fig. 14: Segmentation results with geodesic distances computed on the original image instead of the interval-valued map using the same seeds as in fig. 13. The segmentation is slightly degraded due to digitalization effects.

While this approach is somewhat outdated and outperformed by deep learning methods [42, 43], it is still remarkable that such simple techniques can sometimes achieve better results while being much less computationally expensive and much more interpretable. The samples taken from the ScribblePrompt demos dataset [43] are illustrated in fig. 13 and compared with their deep learning counterparts (between 15M for U-Net-based and 150M parameters for SAM-based models). It is worth noting that geodesics not computed on the interval-valued map have *incorrect* distances because of digitization effects. This might slightly degrade the segmentation results, as shown in fig. 14.

6 Conclusion

In this paper, we have provided the theoretical foundations of geodesics on interval-valued maps and shown their usefulness in overcoming the limitations of the state-of-the-art ToS computation algorithm. A practical implementation of the distance transform was given by slightly extending the Fast Marching Method and the Fast Sweeping Method to fit the specificities of our problem. We have shown that the proposed distance transform can be efficiently implemented on massively parallel architectures, and is able to process high-dynamic range images: the two main limitations of the state-of-the-art ToS computation algorithm. This results in significant speed-ups (2 to 3 times faster) compared to the state-of-the-art ToS computation algorithm on high-dynamic range images, and enables the computation of the ToS on GPU without any memory transfer to the host (with also a significant speed-up compared to the CPU version). More importantly, we have proven formally that the proposed approach is mathematically sound, that the algorithms are correct, and that the max-tree of the distance map is equivalent to the ToS of the original input.

The current work opens several perspectives. First, the algorithmic aspects of this work that enable fast computation of the distance transform on accelerators or with high-quantized data could be used in many applications outside image processing (physics, simulation, path planning, etc.). Second, we think that interval-valued maps on Khalimsky grids are beautiful mathematical objects that link continuity and discreteness, and deserve to be further studied. In particular, how the work on measurability, continuity, integrability (and more generally on function analysis) of set-valued maps by Aubin and Frankowska [18] extends to a topological space like the Khalimsky grid, which also extends continuity to digital spaces. The computation of geodesics was

motivated by the need to overcome the limitations of the Tree-of-Shapes algorithm, but the properties of these maps could probably solve other discrete-related problems in digital geometry. Finally, as we have opened the door to alternative algorithms for the *propagation* step of the ToS computation, it is to be hoped that we will eventually be able to compute the ToS in a single pass by extending one of these algorithms.

References

[1] Bosilj, P., Kijak, E., Lefèvre, S.: Partition and inclusion hierarchies of images: A comprehensive survey. Journal of Imaging **4**(2), 33 (2018)

[2] Xu, Y., Géraud, T., Najman, L.: Hierarchical image simplification and segmentation based on mumford-shah-salient level line selection. Pattern Recognition Letters **83**, 278–286 (2016)

[3] Ôn Vũ Ngọc, M., Boutry, N., Fabrizio, J., Géraud, T.: A minimum barrier distance for multivariate images with applications. Computer Vision and Image Understanding **197-198**, 102993 (2020)

[4] Bosilj, P., Kijak, E., Lefèvre, S.: Beyond MSER: Maximally stable regions using tree of shapes. In: Proceedings of the British Machine Vision Conference (BMVC), pp. 1–13 (2015)

[5] Cavallaro, G., Mura, M.D., Benediktsson, J.A., Plaza, A.: Remote sensing image classification using attribute filters defined over the tree of shapes. IEEE Transactions on Geoscience and Remote Sensing **54**(7), 3899–3911 (2016)

[6] Monasse, P., Guichard, F.: Fast computation of a contrast-invariant image representation. IEEE Transactions on Image Processing **9**(5), 860–872 (2000)

[7] Salembier, P., Oliveras, A., Garrido, L.: Antiextensive connected operators for image and sequence processing. IEEE Transactions on Image Processing **7**(4), 555–570 (1998)

[8] Caselles, V., Monasse, P.: Geometric Description of Images as Topographic Maps. Lecture Notes in Mathematics. Springer, ??? (2010)

[9] Song, Y.: A topdown algorithm for computation of level line trees. IEEE Transactions on Image Processing **16**(8), 2107–2116 (2007)

[10] Géraud, T., Carlinet, E., Crozet, S., Najman, L.: A quasi-linear algorithm to compute the tree of shapes of nD images. In: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM 2013), pp. 98–110. Springer, ??? (2013)

[11] Carlinet, E., Crozet, S., Géraud, T.: The tree of shapes turned into a max-tree: A simple and efficient linear algorithm. In: IEEE International Conference on

Image Processing (ICIP), pp. 1488–1492. IEEE, ??? (2018)

[12] Crozet, S., Géraud, T.: A first parallel algorithm to compute the morphological tree of shapes of nd images. In: 2014 IEEE International Conference on Image Processing (ICIP), pp. 2933–2937 (2014)

[13] Najman, L., Géraud, T.: Discrete set-valued continuity and interpolation. In: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM), pp. 37–48 (2013). Springer

[14] Carlinet, E., Esteban, B.: A tree of shapes computation algorithm for massively parallel architectures. In: IEEE International Conference on Image Processing (ICIP), pp. 2169–2174 (2025). IEEE

[15] Carlinet, E., Esteban, B.: The tree of shapes as a distance transform: building the tos on high-dynamic range images. In: International Conference on Discrete Geometry and Mathematical Morphology (DGMM), pp. 303–314 (2025). Springer

[16] Latecki, L., Eckhardt, U., Rosenfeld, A.: Well-Composed sets. Computer Vision and Image Understanding **61**(1), 70–83 (1995)

[17] Boutry, N., Géraud, T., Najman, L.: A tutorial on Well-Composedness. Journal of Mathematical Imaging and Vision **60**(3), 443–478 (2018)

[18] Aubin, J.-P., Frankowska, H.: Set-Valued Analysis. Springer, ??? (2009)

[19] Berger, C., Géraud, T., Levillain, R., Widynski, N., Baillard, A., Bertin, E.: Effective component tree computation with application to pattern recognition in astronomical imaging. In: IEEE International Conference on Image Processing (ICIP), vol. 4, p. 41 (2007). IEEE

[20] Wilkinson, M.H.F.: A fast component-tree algorithm for high dynamic-range images and second generation connectivity. In: IEEE International Conference on Image Processing (ICIP), pp. 1021–1024 (2011)

[21] Moschini, U., Meijster, A., Wilkinson, M.H.: A hybrid shared-memory parallel max-tree algorithm for extreme dynamic-range images. IEEE transactions on pattern analysis and machine intelligence **40**(3), 513–526 (2017)

[22] Blin, N., Carlinet, E., Lemaitre, F., Lacassagne, L., Geraud, T.: Max-Tree computation on GPUs. IEEE Transactions on Parallel and Distributed Systems **33**(12), 3520–3531 (2022)

[23] Kimmel, R., Kiryati, N., Bruckstein, A.M.: Sub-pixel distance maps and weighted distance transforms. Journal of Mathematical Imaging and Vision **6**, 223–233 (1996)

[24] Aubert, G., Kornprobst, P.: Mathematical Problems in Image Processing: Partial Differential Equations and the Calculus of Variations. Springer, ??? (2010)

[25] Toivanen, P.J.: New geodosic distance transforms for gray-scale images. Pattern Recognition Letters **17**(5), 437–450 (1996)

[26] Zhao, H.: A fast sweeping method for Eikonal equations. Mathematics of computation **74**(250), 603–627 (2005)

[27] Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proceedings of the National Academy of Sciences **93**(4), 1591–1595 (1996)

[28] Dial, R.B.: Algorithm 360: shortest-path forest with topological ordering [H]. Communications of the ACM **12**(11), 632–633 (1969)

[29] Yatziv, L., Bartesaghi, A., Sapiro, G.: O(N) implementation of the fast marching algorithm. Journal of computational physics **212**(2), 393–399 (2006)

[30] Rasch, C., Satzger, T.: Remarks on the $\mathcal{O}(n)$ implementation of the fast marching method. IMA Journal of Numerical Analysis **29**(3), 806–813 (2009)

[31] Ryu, J., Trager, S.C., Wilkinson, M.H.: A fast alpha-tree algorithm for extreme dynamic range pixel dissimilarities. IEEE Transactions on Pattern Analysis and Machine Intelligence **46**(5), 3199–3212 (2023)

[32] Tsai, Y.-H.R., Cheng, L.-T., Osher, S., Zhao, H.-K.: Fast sweeping algorithms for a class of hamilton–jacobi equations. SIAM journal on numerical analysis **41**(2), 673–694 (2003)

[33] Rosenfeld, A., Pfaltz, J.L.: Distance functions on digital pictures. Pattern Recognition **1**(1), 33–61 (1968)

[34] Criminisi, A., Sharp, T., Siddiqui, K.: Interactive geodesic segmentation of n-dimensional medical images on the graphics processor. Radiological Society of North America (RSNA) (2009)

[35] Asad, M., Dorent, R., Vercauteren, T.: FastGeodis: Fast generalised geodesic distance transform. The Journal of Open Source Software **7**(79), 4532 (2022)

[36] Zhao, H.: Parallel implementations of the fast sweeping method. Journal of Computational Mathematics, 421–429 (2007)

[37] Bai, X., Sapiro, G.: A geodesic framework for fast interactive image and video segmentation and matting. In: IEEE 11th International Conference on Computer Vision (ICCV), pp. 1–8 (2007)

[38] Criminisi, A., Sharp, T., Blake, A.: Geos: Geodesic image segmentation. In: European Conference on Computer Vision (ECCV), pp. 99–112 (2008). Springer

[39] Dubrovina, A., Hershkovitz, R., Kimmel, R.: Image editing using level set trees. In: IEEE International Conference on Image Processing (ICIP), pp. 4442–4446 (2014). IEEE

[40] Géraud, T., Xu, Y., Carlinet, E., Boutry, N.: Introducing the Dahu pseudo-distance. In: International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing (ISMM 2017), pp. 55–67 (2017). Springer

[41] Ôn Vũ Ngọc, M., Carlinet, E., Fabrizio, J., Géraud, T.: The Dahu graph-cut for interactive segmentation on 2D/3D images. Pattern Recognition **136**, 109207 (2023)

[42] Liu, Q., Xu, Z., Bertasius, G., Niethammer, M.: Simpleclick: Interactive image segmentation with simple vision transformers. In: IEEE/CVF International Conference on Computer Vision, pp. 22290–22300 (2023)

[43] Wong, H.E., Rakic, M., Guttag, J., Dalca, A.V.: Scribbleprompt: fast and flexible interactive segmentation for any biomedical image. In: European Conference on Computer Vision, pp. 207–229 (2024). Springer

A Proofs for Section 4 (Equivalence between the max-tree of the distance map and the tree of shapes)

Proposition 11. *Any subset X of $\mathbb{H}_\Omega$ is saturated iff it does not admit any internal hole.*

Proof. The set X is saturated when $Sat(X) = X$, which is equivalent to:

$$X = \mathbb{H}_\Omega \setminus \mathcal{CC}(\mathbb{H}_\Omega \setminus X, p_\infty),$$

which is equivalent to:

$$\mathbb{H}_\Omega \setminus X = \mathcal{CC}(\mathbb{H}_\Omega \setminus X, p_\infty),$$

equivalent to the fact that X admits no internal hole. □

Proposition 12 (Proposition 7 in [GBC+22])**.** *Let X be some open subset of $\mathbb{H}_n$. Then we can reformulate its interior boundary as:*

$$\partial_{int}(X) = st(X) \cap st(\mathbb{H}_n \setminus X).$$

Proposition 13. *Let $X \subset \mathbb{H}_n$ be some (non-empty) open saturated connected set which does not contain p_∞. Then, its interior boundary $\partial_{int}(X)$ is connected.*

Proof. By Proposition 12, the interior boundary of an open set X can be written $\partial_{int}(X) = X \cap st(\mathbb{H}_n \setminus X)$ since $X = st(X)$. Because X is saturated, $\mathbb{H}_n \setminus X = \mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty)$, which leads to:

$$\partial_{int}(X) = X \cap st(\mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty)).$$

Let us denote $A = X$ and $B = st(\mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty))$. We can observe that A and B are open connected sets. Furthermore, A is saturated, then $A = \mathbb{H}_n \setminus \mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty)$, and B contains $\mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty)$, which leads to:

$$A \cup B \supseteq (\mathbb{H}_n \setminus \mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty)) \cup \mathcal{CC}(\mathbb{H}_n \setminus X, p_\infty) = \mathbb{H}_n,$$

thus $A \cup B$ covers $\mathbb{H}_n$. By Proposition 1, the interior boundary of X is connected. The proof is done. □

□

Proof. We will proceed *reductio ad absurdum*. Let us assume that $\mathcal{E}$ element of $\mathcal{CC}([\mathbf{D} \geq k])$ for some $k \in \mathbb{N}$ is not saturated, which means by proposition 11 that $\mathcal{E}$ admits some internal hole $H \neq \emptyset$.

□

Proposition 8. *Let $\mathcal{E}$ be some element of $\mathcal{T}_{\mathbf{D}}$, then its interior boundary $\partial_{int}(\mathcal{E})$ is connected.*

Proof. Since $\mathcal{E}$ is open by proposition 2, saturated by proposition 7 and connected by definition, then, by Proposition 13, we obtain that $\partial_{int}(\mathcal{E})$ is connected.

□

Proposition 9. *Let X be some saturated subset of $\mathbb{H}_n$ which does not contain p_∞. Then, we have the following property:*

$$Sat(\partial_{int}(X)) = X.$$

Proof. When we are in the case where $\partial_{int}(X) = X$, then we immediately obtain the equality we look forward since X is saturated. At the contrary, when $\partial_{int}(X) \subsetneq X$, we can decompose the space $\mathbb{H}_\Omega$ in this way: we denote by $\mathcal{EXT}$ the connected component of $\mathbb{H}_\Omega \setminus \partial_{int}(X)$ which contains p_∞ (this "exterior component" is not empty since $p \notin \partial_{int}(X)$) and by $\{\mathcal{INT}_j\}_{j\in J}$ the (possibly several) connected component(s) of $\mathbb{H}_\Omega \setminus \partial_{int}(X)$ which do not contain p_∞ (the "interior(s) components(s)"). Let us prove with a double inclusion that, with this partitioning of $\mathbb{H}_\Omega$:

$$\mathbb{H}_\Omega = \partial_{int}(X) \sqcup \mathcal{EXT} \sqcup \bigsqcup_{j\in J} \mathcal{INT}_j,$$

we can obtain the relation $Sat(\partial_{int}(X)) = X$. □

References for the Appendix

[GBC+22] Thierry Géraud, Nicolas Boutry, Sébastien Crozet, Edwin Carlinet, and Laurent Najman. A proof of the tree of shapes in nD. *arXiv preprint arXiv:2206.05109*, 2022.