Computational Geometry 42 (2009) 45–59

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Optimal simplification of polygonal chains for subpixel-accurate rendering

Lilian Buzer

Université Paris-Est, Unité Mixte CNRS-ESIEE, UMR 8049, Department of Computer Science, ESIEE, 2 bd Blaise Pascal, Cité Descartes, BP 99, 93162 Noisy-Le-Grand Cedex, France

A R T I C L E   I N F O

Article history:
Received 28 November 2006
Received in revised form 13 March 2008
Accepted 17 March 2008
Available online 28 March 2008
Communicated by P. Agarwal

Keywords:
Polygonal chain
Approximation
Accurate rendering
Digital line-segment
Near-linear time
Breadth-first traversal
Query method

A B S T R A C T

For a given polygonal chain, we study the min-# problem, which consists of finding an approximate and ordered subchain with a minimum number of vertices under a given approximation criterion. We propose the first near-linear time algorithm for the min-# problem that ensures optimality in the number of vertices and that retains the shape of the input polygonal chain at the same time. Previous algorithms with near-linear time performance produced geometrical inconsistencies and former methods used to preserve the features of the original chain required a quadratic time complexity. When we set the approximation error equal to the pixel pitch, our simplification criterion guarantees that the rendering of the simplification lies at most half a pixel away from the original chain, contrary to other methods that do not offer a direct control over the rendering. Thus, we avoid producing visible topological inconsistencies. Moreover, our method is based on basic data structures, which leads to a simple and efficient implementation.

1. Introduction

A *polygonal chain* in the Euclidean plane is defined as an ordered list of vertices such that any two consecutive vertices are connected by a line-segment. A polygonal chain whose line-segments are non-self-intersecting is called *simple*. The problem of approximating a simple polygonal chain arises in many applications including geographic information systems (GIS), cartography, computer graphics, data compression or medicine [28]. Recently, the need for accurate vectorization system emerges [19]. Enhancing the quality of conversion has been recognized as an important factor in advancing the research in this field [30].

Let $P = (p_1, p_2, \ldots, p_n)$ denote a simple polygonal chain. An *approximation* P' of P is an *ordered subchain* $(p_{i_1}, p_{i_2}, \ldots, p_{i_m})$ of P such that $i_1 < \cdots < i_m$, $p_{i_1} = p_1$ and $p_{i_m} = p_n$. Thus, the vertices of any approximation are an ordered subset of the vertices of P. It is easily seen that the approximations are not required to be simple. We use the notation P_i^j, where $i < j$, to denote the subchain $(p_i, \ldots, p_j)$ of P. A *simplification criterion* C is associated with a non-negative real-valued function δ_C. This function assigns to any polygonal chain s a value called the *error* of s. By an abuse of language, the *error of an edge* $p_i p_j$ corresponds to the error of the subchain P_i^j of P. The *error of an approximation* P' of P is defined as follows:

$$\Delta_C(P') = max_{1 \leqslant j < m} \delta_C\left(P_{i_j}^{i_{j+1}}\right)$$

It represents the maximum error of each of the edges of P'. We say that P' is an ϵ-approximation of P if $\Delta_C(P')$ is less than ϵ. Thus, a subchain P_i^j can be replaced by a line-segment $p_i p_j$ in an ϵ-approximation of P if $\delta_C(P_i^j)$ is less than ϵ (see Fig. 1 for an example). Two main problems have been considered in the literature:

E-mail address: Buzerl@esiee.fr.

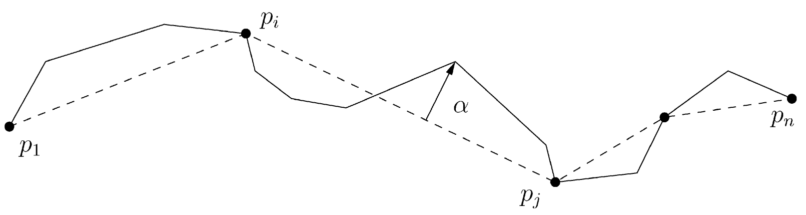

Fig. 1. A polygonal chain P (in bold) and one of its approximations (dashed); the error of the line-segment p_ip_j, under the tolerance zone criterion and under the L_2 metric, is equal to α.

- **Min-ϵ problem**: Given a polygonal chain P and an integer k, find among all the approximations of P with at most k vertices an approximation with a minimum error.
- **Min-# problem**: Given a polygonal chain P and a real number $\epsilon \geqslant 0$, find among all the ϵ-approximations of P an ϵ-approximation with a minimum number of vertices.

In this paper, we focus on the second problem. Our main contribution consists of setting up the first near-linear time algorithm for the min-# problem that ensures optimality in the number of vertices and that retains the shape of the input polygonal chain at the same time. When ϵ corresponds to the pixel pitch, our approximation criterion guarantees that the original chain lies at most half a pixel away from the rendering of the simplified chain. Thus, we can optimally choose ϵ to achieve a subpixel-accurate rendering while obtaining a minimum number of segments. Only our approximation criterion makes this functionality available.

In Section 2, we recall the existing approximation criteria and we classify them relative to their ability to preserve the features of the input polygonal chain. Then in Section 3, we choose to present the optimal methods that compute a simplification with a minimum number of segments as well as the heuristics. In Section 4, we describe our new simplification criterion and its connection to the definition of the digital line-segments. Then, we prove that the rendering of the simplified polygonal chain lies at most half a pixel away from the original chain, when ϵ equals the screen pixel pitch. In Section 5, we show how to preprocess some key information on the vertical and horizontal thickness of the subchains we consider. In the same way, in Section 6, we study how to efficiently precompute some information in order to optimize the queries on the location of the line-segment endpoints. In Section 7, we design our algorithm and show that it finds an optimal simplification by performing at most n queries that have a logarithmic time complexity. In the last section, we present some experiments and we compare our method with two well-known algorithms.

2. Approximation criteria

2.1. Metrics

Let us examine the most commonly used metrics involved in the definitions of the approximation criteria. For each of these metrics, we recall the expression of the distance between two points $u = (u_x, u_y)$ and $v = (v_x, v_y)$:

- *The Manhattan distance* ($L1$) where $L_1(u, v) = |u_x - v_x| + |u_y - v_y|$;
- *The Euclidean distance* ($L2$) where $L_2(u, v) = \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2}$;
- *The Max distance* (L_∞) where $L_\infty(u, v) = \max\{|u_x - v_y|, |u_x - v_y|\}$;
- *The uniform metric* (*also known as Chebyshev metric*) where $d(u, v) = |u_y - v_y|$ if $u_x = v_x$ or $d(u, v) = \infty$ otherwise.

2.2. Tolerance regions

An ϵ-approximation can be defined in an equivalent way. For a given error bound ϵ, we can consider that an approximation criterion associates two vertices u and v with a set of *tolerance regions* $\Upsilon_\epsilon(u, v)$ in the Euclidean plane. Thus, an approximation P' is defined as an ϵ-approximation of P if for each edge p_ip_j of P' there exists a tolerance region in $\Upsilon_\epsilon(p_i, p_j)$ that contains P_i^j. We present a representative but non-exhaustive list of the approximation criteria used in the literature. This list is designed to reflect, at best, the work done by all the researchers interested in the min-# problem during the last two decades.

The error of a subchain P_i^j can be formulated as the maximum distance of the vertices of P_i^j to a geometric object $T_{p_ip_j}$. In this case, a single tolerance region is associated with two vertices p_i and p_j. This region can be defined by $\{p \in \mathbb{R}^2 \mid distance(T_{p_ip_j}, p) \leqslant \epsilon\}$. We recall that the distance between a geometrical object T and a point p is equal to $\min_{q \in T} d(q, p)$. Under the **segment distance criterion**, $T_{p_ip_j}$ corresponds to the line-segment connecting p_i to p_j. Under the **infinite beam criterion** (also called the parallel strip criterion), $T_{p_ip_j}$ is defined as the line passing trough p_i and p_j. The three metrics L_1, L_2 and L_∞ have been used to evaluate the distance function. As the segment distance criterion under the L_2 metric has been quite popular, it has been named the **tolerance zone criterion**. For the special case where the input

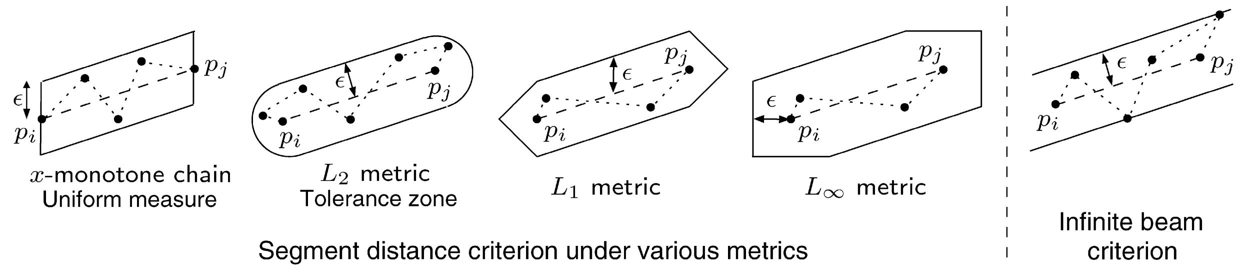

Fig. 2. Approximation criteria linked to a single tolerance region.

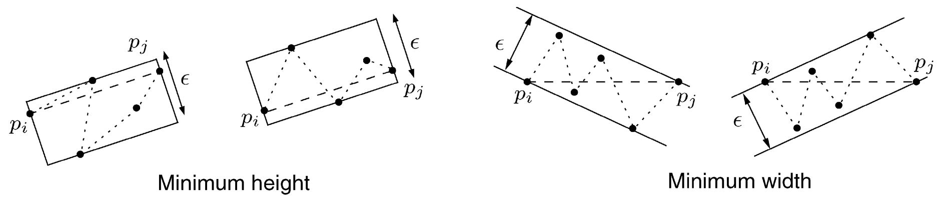

Fig. 3. Approximation criteria linked to multiple tolerance regions.

chain is an x-monotone chain, we can consider that the input chain and the simplified chain correspond to two functions that map x into y. In this way, the uniform metric can be used to compute the error. Thus, the criterion is called the **uniform measure criterion**. In Fig. 2, we present for each of these criteria the unique tolerance region associated with two given vertices.

The error of a subchain P_i^j can be formulated as the maximum of the characteristic of a geometric object that contains P_i^j. Moreover, the geometric object must satisfy some conditions on its location relative to the vertices p_i and p_j. For the **minimum height** criterion, the geometric object corresponds to a rectangle such that p_i and p_j lie on two opposite sides orthogonal to the segment p_ip_j. The length of these sides defines the approximation error. For the **minimum width** criterion, the geometric object corresponds to a strip which contains the segment p_ip_j and whose width determines the approximation error. These last two criteria associate two vertices p_i and p_j with a set of tolerance regions whose axis are not forced to match the direction of the straight-line p_ip_j unlike in the previous criteria. In Fig. 3, we present different tolerance regions for each of these criteria.

2.3. Strong and weak criteria

Heuristics are not guaranteed to produce an optimal solution. These methods try to construct a good quality solution and achieve a better run time. Thus, the non-optimal character of a min-# heuristic can be expressed by a non-optimal simplification criterion whose simplification may not perceptually match the input chain. For the approximation criteria, the main problem is to control the behavior of the approximation in the neighborhood of p_i and p_j. When we allow the subchain P_i^j to go farther than these two endpoints, we may loose one part of the original chain and obtain a peculiar simplification (see Fig. 4). Neither the parallel-strip criterion nor the minimum width criterion preserve the features of the original chain. Thus, we call them *weak criteria*. On the contrary, the segment distance criterion and the minimum height criterion retain the shape of the original chain and so we describe them as *strong criteria*.

3. History and classification

Many algorithms were created in order to solve the min-# problem under the previous error criteria. We can classify these methods according to their ability to find an ϵ-approximation with or without a minimum number of vertices. In the following, we first present the class of *optimal algorithms* that compute a simplification with an optimal number of vertices. Then, we focus on the other class of the *heuristic methods*.

3.1. Optimal algorithms

Early results for the min-# problem, under various error criteria and under various metrics were presented by Agarwal and Varadarajan [1], Chan and Chin [8], Chen and Daescu [9], Eu and Toussaint [15], Hakimi and Schmeichel [16], Imai and Iri [20] and Melkman and O'Rourke in [24]. The class of the optimal algorithms is based on the general approach set up by Imai and Iri [20]. All the algorithms derived from this technique build an approximation with an optimal number of vertices. Imai and Iri formulated the min-# problem in terms of graph theory. They defined an unweighted directed acyclic

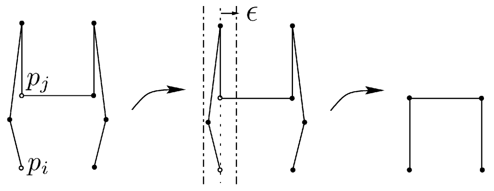

Fig. 4. Typical inconsistency produced by the infinite beam criterion.

graph $G_\epsilon = (V, E_\epsilon)$, where $V = \{p_1, \ldots, p_n\}$ and $E_\epsilon = \{\ p_i p_j \mid \exists U \in \Upsilon_\epsilon(p_i, p_j), P_i^j \subset U\}$. Their method consists of a two-step algorithm. In the first step, the graph G_ϵ, that represents the subchains that can be simplified by a line-segment, is built. Then, in the second step, a shortest path is computed in G_ϵ in order to obtain an approximation with a minimum number of segments. The shortest path computation is processed in time linear in the number of G_ϵ edges using dynamic programming. Nevertheless, when ϵ is large enough, the number of edges becomes quadratic. In this paper, all the methods we cite with a quadratic or superquadratic time complexity are derived from the Imai and Iri approach. They all suffer from the bottleneck induced by the processing of all the G_ϵ edges. In the following, we recall the different methods in the class of optimal algorithms. We separate them depending on whether the method uses a strong criterion or a weak criterion.

3.1.1. *Strong criteria*

Imai and Iri [20] proposed an algorithm under the minimum height criterion that runs in $O(n^2 \log n)$ time. The use of the L_2 metric with this criterion is implicit. The min-# problem under the uniform measure criterion was solved by Hakimi and Schmeichel [16] in quadratic time. Using the tolerance zone criterion, Melkman and O'Rourke [24] proposed an algorithm that takes $O(n^2 \log n)$ time and $O(n^2)$ space. Later, Chan and Chin in [8] reduce the time complexity to $O(n^2)$. Their method, even if they do not mention this remark, can certainly be extended to the L_1 and L_∞ metrics. Finally, Chen and Daescu [9] proposed the first algorithm under the L_1, L_2 and L_∞ metrics that achieves a linear space complexity while preserving the quadratic time performance. Using a theoretical approach based on a graph compression technique, Agarwal and Varadarajan [1] present a new method for the uniform measure criterion and the segment distance criterion under the L_1 metric. Their algorithm achieves an $O(n^{\frac{4}{3}+\gamma})$ time complexity for any $\gamma > 0$ where the constant of proportionality in the running time depends on γ.

3.1.2. *Weak criteria*

For the minimum width criterion, which implicitly uses the L_2 metric, Imai and Iri [20] proposed an algorithm with an $O(n \log n)$ time complexity. This algorithm is derived from the general approach of Imai and Iri but it succeeds in bypassing the quadratic time bottleneck. Indeed, we can notice that, under this criterion, when a subchain P_i^j can be simplified, all the subchains $P_i^k, i \leqslant k \leqslant j$ can also be simplified. Thus, the graph G_ϵ can be represented in only $O(n)$ space. A greedy algorithm, where the farthest reachable vertex is chosen at each iteration, is sufficient to compute the shortest path in G_ϵ. Obviously, this greedy approach has a linear time complexity in the number of vertices. For the infinite beam criterion, Toussaint [29] and Imai and Iri [20], proposed a version under the L_2 metric that has an $O(n^2 \log n)$ time complexity and an $O(n^2)$ space complexity. Then, Eu and Toussaint [15] claimed to improve this result under the L_1, L_2 and L_∞ metrics to a quadratic time complexity. However, they seem to miss an additional logarithmic factor [9]. Finally, Chen and Daescu [9] presented a method with an $O(n^2 \log n)$ time complexity under the same metrics that achieves a linear space complexity. More recently, when the vertices of the input chain satisfy the following condition: $d_{L_2}(p_i, p_j) \notin [\epsilon, \epsilon\sqrt{2}], 1 \leqslant i < j \leqslant n$, Daescu and Mi [11] notice that the min-# problem can be solved in $O(n^{4/3+\gamma})$ time with the infinite beam criterion under the L_2 metric.

3.2. *Heuristic methods*

One of the oldest and most popular heuristic [13] is the Douglas–Peucker algorithm. This heuristic is based on the infinite beam criterion under the L_2 metric, but any other metric can be used. This method starts by approximating the subchain P_1^n by the line-segment $p_1 p_n$. If the distance of the farthest vertex p_k from $p_1 p_n$ is greater than the tolerance threshold, we recursively approximate the two polygonal chains $p_1 p_k$ and $p_k p_n$. Observing that the farthest vertex is always located on the convex hull, Hershberger and Snoeyink [17] proved that the Douglas–Peucker algorithm can be solved in $O(n \log n)$ time. Extending this study, Hershberger and Snoeyink in [18] showed that this algorithm can be implemented in $O(n \log^* n)$ expected time, where $\log^*$ denotes the iterated logarithm function. The Douglas–Peucker algorithm provides a convenient algorithmic solution to solve topological inconsistencies. In fact, when a self-intersection is detected in the simplified chain between the vertices p_i and p_j, we simply refine the simplification process between these two vertices. This method always suppresses topological inconsistencies because, in the worst case, it converges to the original subchain. Consult [27] for further information. Faster heuristics have been developed, but all of these are generally believed to have inferior quality [22]. The simplest algorithm called the 'nth-point algorithm' consists of sub-sampling the chain by keeping every kth point of the input. This algorithm is extremely fast but yields poor quality approximations.

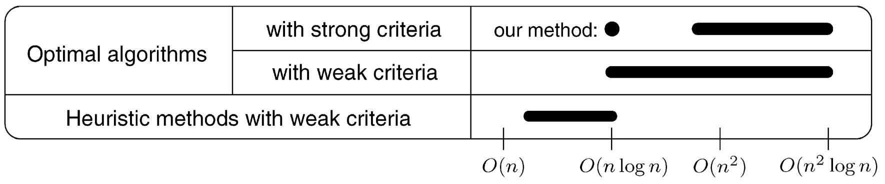

Fig. 5. Complexity of the different classes of algorithms.

3.3. Conclusion

More recently, Daescu [10] and Buzer [5] noticed that we can create algorithms whose running time depends on the size of the output. For this, a breadth first traversal can be used to compute the shortest path in G_ϵ. Indeed, if m denotes the number of the vertices of the approximation, only the edges in G_ϵ that can be reached in at most $m-1$ BFT iterations need to be generated. Later, Daescu et al. [11,12] proposed output sensitive algorithms for the tolerance zone criterion and the infinite beam criterion under the L_2 metric. Nevertheless, their algorithms fail to improve the worst-case time complexity. In conclusion, when we want to use an optimal algorithm with a strong criterion, we only have at our disposal quadratic or superquadratic time algorithms. We can choose to use a weak criterion, but the corresponding algorithms fail to preserve the features of the original polygonal chain. Nevertheless, for some of them, we can achieve a quasi linear time complexity. For a simple implementation, we can choose heuristic methods. As these methods implicitly use weak criteria, they do not retain the shape of the input chain. Nevertheless, they achieve a quasi linear time complexity but they fail to produce a simplification with a minimum number of vertices. We recall all these considerations in Fig. 5. Our work focuses on the class of the optimal algorithms based on strong criteria. To our knowledge, we operate a breakthrough in the time complexity by offering the first method that achieves a quasi linear time complexity for this class of methods.

4. Introducing the digital zone criterion

In [20], Imai and Iri develop an algorithm under the minimum width criterion that is the only optimal algorithm with near-linear time performance. We show how to improve this weak criterion to a strong criterion while preserving the near-linear time performance. In this section, we describe our new criterion and its connection with computer graphics. In our context, a screen is described as a regular grid of pixels. For simplicity, we suppose that the length of a pixel pitch is fixed at 1. We call *Pixel*(u) the square region $\{v \in \mathbb{R}^2 \mid L_\infty(u, v) \leqslant 1/2\}$ whose *center* is an integer point u. To represent a pixel, we generally refer to its center and so, a sequence of pixels is described by the sequence of its centers.

4.1. Digital line-segment

As most of the simplified polygonal chains are screen-rendered, we think about defining our tolerance regions in such a way that they match the digitizations of the line-segments. Let us recall the definition of a digital line-segment.

Definition 1. We call a horizontal digital line-segment (HDLS) any sequence of pixels $((x_i, y_i))_{0 \leqslant i \leqslant k}$ such that $x_i = x_0 + i$, $i = 0, \dots, k$ with $x_0 \in \mathbb{Z}$ and such that $y_i = \lceil m \cdot x_i + b \rceil$, $i = 0, \dots, k$ with $m \in [-1, 1]$ and with $b \in \mathbb{R}$.

Notice that a HDLS corresponds to the digitization of a straight-line of the form $y = mx + b$ with $m \in [-1, 1]$ and with $b \in \mathbb{R}$. Symmetrically, we can define the vertical digital line-segments (VDLS) which are mappings from y into x. The next property plays an important role in the definition of our approximation criterion.

Property 1. *A sequence of pixels $S = ((x_i, y_i))_{0 \leqslant i \leqslant k}$ with $x_i = x_0 + i$, $i = 0, \dots, k$ is a HDLS iff there exists a strip, defined by $\{(x, y) \mid mx + b \leqslant y < mx + b + 1\}$ with $m \in [-1, 1]$ and with $b \in \mathbb{R}$, that contains all the pixels' centers.*

The HDLS and the VDLS describe all the digital line-segments (DLS) that can be rendered on a graphic screen. Let us examine the DLS generated by the well-known Bresenham line-drawing algorithm. This method takes as input two pixels u and v with integer coordinates. Then, it forces the central axis of the strip that contains all the pixels' centers to correspond to the straight-line uv. Thus, there exists only one Bresenham's DLS whose extremities correspond to u and v contrary to our family of HDLS and VDLS where there exist $gcd(\max(|u.x - v.x|, |u.y - v.y|))$ DLS in the same situation. We present an example in Fig. 6. Consult [26] for further explanations. In conclusion, our family of DLS is richer than the family of Bresenham's DLS and, therefore, it is more suitable to define our approximation criterion.

4.2. The digital zone criterion

The *vertical sectional length* (VSL) of a convex body B at the abscissa α, denoted by $v_B(\alpha)$, is the length of the line-segment defined by the intersection between B and the straight-line $x = \alpha$. We now define the error of a subchain P_i^j

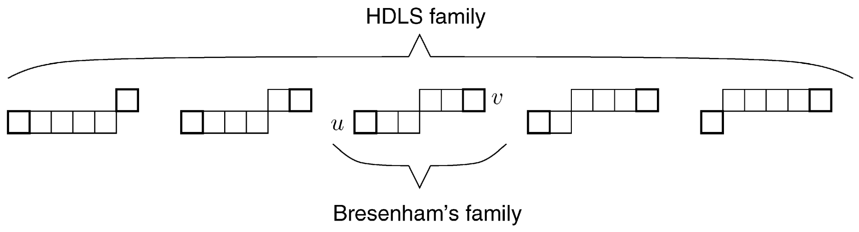

Fig. 6. For the same endpoints, our family of horizontal digital line-segments offers several possibilities compared with the family of Bresenham's digital line-segments.

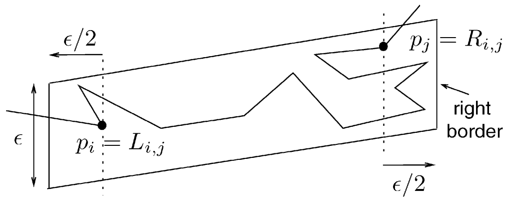

Fig. 7. Tolerance region associated with δ_H.

relative to our new approximation criterion called the *digital zone criterion*. Let S denote the set of the strips of the form $\{(x, y) \mid mx + b \leqslant y \leqslant mx + c\}$ with $m \in [-1, 1]$ and with $b, c \in \mathbb{R}$. The VSL of such a strip r, denoted by v_r is equal to $c - b$ for any abscissa. Let S_i^j denote the strips of S that contain the subchain P_i^j. Let $L_{i,j}$ and $R_{i,j}$ respectively denote the leftmost and rightmost vertices of the two vertices p_i and p_j. The error δ_H of a subchain P_i^j is defined as follows:

$$\delta_H(P_i^j) = \max\Bigl(\min_{s \in S_i^j} v_s,\ 2 \max_{i \leqslant k \leqslant j}(L_{i,j} - p_k.x),\ 2 \max_{i \leqslant k \leqslant j}(p_k.x - R_{i,j}.x)\Bigr) \tag{1}$$

Symmetrically, we define the error δ_V. Finally, the error of a subchain P_i^j relative to the digital zone criterion is:

$$\delta_{\textit{digital zone criterion}}(P_i^j) = \min\bigl(\delta_H(P_i^j), \delta_V(P_i^j)\bigr) \tag{2}$$

We give an example of a tolerance region associated with δ_H and with two vertices p_i and p_j such that p_i lies on the left of p_j (Fig. 7). Such a tolerance region is linked to a strip of the form $\{(x, y) \mid mx+b \leqslant y \leqslant mx+b+\epsilon\}$ with $m \in [-1, 1]$. On the left of p_i, the tolerance region corresponds to the intersection between this trip and the constraint $x \geqslant p_i.x - \epsilon/2$. We obtain a similar condition on the right of p_j. The two vertical sides of the tolerance regions are called the *borders*. Symmetrically, we obtain the family of tolerance regions associated with δ_V. From the definition of the digital zone criterion, a subchain P_i^j can be simplified if it is included either in a tolerance region of δ_H or in a tolerance region of δ_V.

Remark 1. We observe that the digital zone criterion generates hybrid tolerance regions between the ones produced by the segment distance criterion under the L_∞ metric and the ones coming from the minimum width criterion.

4.3. Approximation quality

When ϵ corresponds to the pixel pitch, our approximation criterion guarantees that the original chain lies at most half a pixel away from the rendering of the simplified chain relative to the L_∞ metric. Thus, we can optimally choose ϵ to achieve such a level of detail while obtaining a minimum number of segments. Only our approximation criterion makes this functionality available.

4.3.1. Trust-region and rendering

The distance between a point $q \in \mathbb{R}^2$ and a pixel g is defined as: $d_\infty(q, g) = \min_{u \in \textit{Pixel}(g)} L_\infty(q, u)$. We define the *trust-region* of a sequence of pixels $U = (u_i)_{0 \leqslant i \leqslant k}$ as follows:

$$\textit{Trust Region}(U) = \bigl\{v \in \mathbb{R}^2 \mid \exists u \in U, d_\infty(v, u) \leqslant 1/2\bigr\}$$

We present an example in Fig. 8.

Property 2. *If a subchain P_i^j is covered by a strip s of the form $\{(x, y) \mid mx + b \leqslant y \leqslant mx + 1\}$ with $m \in [-1, 1]$ and with $b \in \mathbb{R}$, then there exists a HDLS whose trust region contains the subchain P_i^j.*

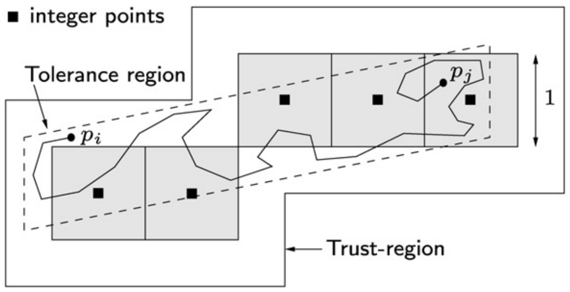

Fig. 8. Example of a trust-region.

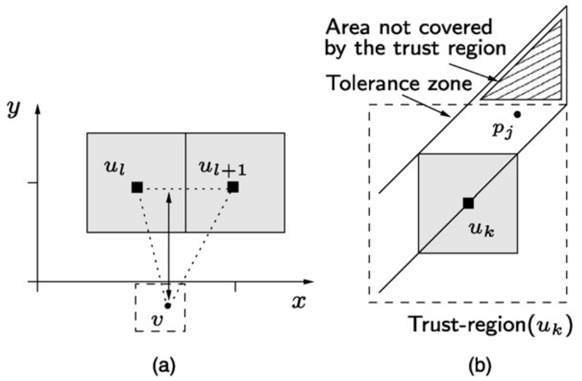

Fig. 9. Pixel selection.

Proof. For convenience, we assume that p_i lies on the left of p_j. Let $(u_q)_{0\leqslant q\leqslant k}$ denote the sequence of the pixels lying in s such that $u_0.x = \lceil p_i.x\rceil$, $u_k.x = \lfloor p_j.x\rfloor$ and such that $u_{q+1}.x = u_q.x + 1, 0 \leqslant q < k$. Let v denote a point lying on the subchain P_i^j. When the abscissa of v is between $u_0.x$ and $u_k.x$, there exists an index m such that *Pixel*(u_m) lies above or under the point v. We focus on the case where v lies outside of the trust region of u_m. Suppose that $v.x > u_m.x$. If $u_{m+1}.y = u_m.y$, the vertical distance between the point v and the line-segment $u_{m+1}u_m$ is greater than one. As these three points belong to the strip s whose VSL is less than one, this configuration can not happen and so u_{m+1} must be located in such a way that its trust region covers the point v (see Fig. 9(a) for an example). The same remark holds when $v.x < u_m.x$. When $v.x = u_m.x$, as the vertical line-segment vu_m lies in s, its length is less than one. Thus, in this case the point v lies in the trust region of v_m. When the abscissa of v is greater than $u_k.x$, v can be located outside of the trust region of the current sequence of pixels (see Fig. 9(b) for an example). As $|u_k.x - p_j.x| \leqslant 1$ and as $|p_j.x - v.x| \leqslant 1/2$, we may have to insert one or two more pixels on the right of u_k. Thus, we render a HDLS with a minimum number of pixels whose trust region covers P_i^j. □

This way, we obtain a method to render the simplified polygonal chain. The key property of our work follows:

Property 3. *Under the digital zone criterion, the original chain lies in the trust region of the pixels coming from the rendering of the simplified chain.*

Remark 2. When we find a tolerance region that contains a subchain P_i^j such that $\delta_H(P_i^j) \leqslant \epsilon$, we indirectly know a strip of the form $\{(x, y) \mid mx + b \leqslant y \leqslant mx + b + 1\}$ containing P_i^j. Thus, during the rendering stage, we select the pixels whose centers lie in the strip defined by $\{(x, y) \mid mx + b \leqslant y < mx + b + 1\}$.

4.3.2. Simplicity

The Imai and Iri approach does not guarantee that the resulting chain is simple. This graph-based approach only optimizes the number of segments and it is blind relative to topological concerns. The problem of building a simplification that selects vertices among the vertices of the input chain and that preserves simplicity is known to be NP-complete [14]. Building an optimal simplification without topological inconsistencies [2] seems to be an intractable problem that is beyond the scope of this paper. Nevertheless, when we set ϵ equal to the pixel pitch, the rendering of the simplification lies at most half a pixel away from the original chain. Thus, in such a case, we can claim that this method does not produce visible topological inconsistencies. When only the visual aspect of the simplification is important, we can consider this statement as sufficient to handle topological inconsistencies.

4.3.3. Connexity

Definition 2. Two pixels P and Q are 8-connected if they either share an edge or a vertex. A sequence of pixels is 8-connected if any pair of two successive pixels is 8-connected.

When ϵ corresponds to the pixel pitch, we show that the previous method of rendering can build a sequence of 8-connected pixels after a minor modification. Let $p_{i_{k-1}}$, p_{i_k} and $p_{i_{k+1}}$ denote three successive vertices of the simplification. Let A and B denote two pixels whose center is the nearest from p_k among all the pixels of the DLS associated to the line-segments $p_{k-1}p_k$ and p_kp_{k+1}, respectively. From the property of the trust region, when $d(p_k, A) + d(p_k, B) < 1$, we can easily see that these two vertices are 8-connected and so the two successive DLS create a sequence of 8-connected pixels. Nevertheless, when $d(p_k, A) = 1/2$ and when $d(p_k, B) = 1/2$, the two DLS may be disconnected. Such a case can only happen when the abscissa and the ordinate of p_k are integer values. See an example in Fig. 10. To correct this small defect,

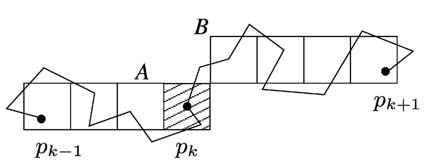

Fig. 10. Ensuring the 8-connexity of the rendering.

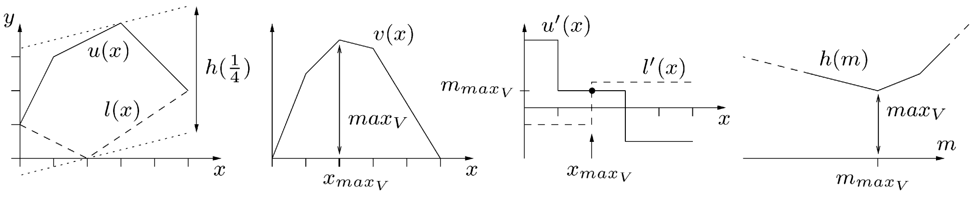

Fig. 11. Relation between the minimum vertical distance of the strips containing a convex polygon and the maximum vertical sectional length of this convex polygon.

we automatically render the vertex of the simplified polygonal chain with integer coordinates. Such a choice is sufficient to ensure the 8-connexity of the final rendering.

5. Determining the minimum vertical sectional length

The first term in the definition (1) of the error δ_H of a subchain P_i^j corresponds to the minimum VSL of the strips of the form $\{(x,y) \mid mx+b \leqslant y \leqslant mx+b+c\}$ with $m \in [-1,1]$ and with $b, c \in \mathbb{R}$, that contain P_i^j. We call this term the *minimum VSL* of the *strips covering* P_i^j. In this section, we first explain how to compute the minimum VSL associated with a given subchain. Then, by using a preprocessing step, we show how to efficiently process this query for any subchain.

5.1. Maximum VSL of a convex hull

We show that the minimum VSL of the strips covering P_i^j is linked to the maximum VSL of the convex hull of P_i^j. Fig. 11 illustrates our presentation. Let C_i^j denote the convex hull of the subchain P_i^j. We explain how we can determine the maximum VSL, denoted max_V, of C_i^j. Let $u(x)$ and $l(x)$ denote, respectively, the upper and lower borders of a convex body B. Thus, we have $v_B(x) = u(x) - l(x)$. Notice that this function is a concave function. Actually, $u(x)$ and $-l(x)$ are concave functions and so their sum is still concave. Therefore, the function v_B reaches a unique maximum value denoted by max_V. For simplicity, consider that $u(x)$ and $v(x)$ are two differentiable functions on the domain $\{x \in \mathbb{R} \mid v_B(x) > 0\}$. Then, the value max_V is reached at the abscissa x where $u'(x) = v'(x)$. When B is a convex polygon, $u(x)$ and $-l(x)$ are two concave piecewise functions, so we have to extend the notion of derivative to the notion of subgradient. Thus, the notation $u'(x)$ corresponds to the set of the slopes of the straight-lines tangent to the function u at the abscissa x. This time, $u'(x)$ and $v'(x)$ are two piecewise constant functions where $u'(x)$ is a non-increasing function and where $v'(x)$ is a non-decreasing function. In this configuration, we want to determine an abscissa x such that $u'(x) \cap l'(x) \neq \emptyset$. This problem was solved in [4,21] in $O(\log k)$ time where k represents the number of the vertices of the convex polygon.

To compute the minimum VSL of the covering strips, we first determine the maximum VSL of the convex hull. Doing this, we indirectly know the abscissa, denoted x_{maxV}, where the VSL of C_i^j reaches its maximum and a slope, denoted m_{maxV}, that belongs to $u'(x_{maxV}) \cap l'(x_{maxV})$. By definition of the subgradient, we have two parallel straight-lines whose slope is equal to m_{maxV} in $[-1,1]$ and that enclose P_i^j. Thus, there exists a strip covering P_i^j whose VSL is equal to max_V.

Let $h(m)$ denote the vertical distance between the two parallel straight-lines tangent to C_i^j, whose slope is equal to m. It can be seen easily that this function is defined as follows:

$$h(m) = \max_{i \leqslant k \leqslant j} (p_k.y - m \cdot p_k.x) - \min_{i \leqslant k \leqslant j} (p_k.y - m \cdot p_k.x)$$

The function $h_k(m) = p_k.y - m \cdot p_k.x$ with $i \leqslant k \leqslant j$ corresponds to the ordinate at the origin of the straight-line of slope m that passes through the vertex p_k. As this function is convex, the function h is also convex. By definition of h, we already know that $h(m_{maxV})$ is equal to max_V. Moreover, the function h is always greater than max_V because any strip that contains C_i^j must contain the vertical line-segment $[(x_{maxV}, l(x_{maxV}), (x_{maxV}, u(x_{maxV})]$. Thus, $h(m)$ reaches its minimum when m is equal to m_{maxV}. If the value m_{maxV} satisfies $-1 \leqslant m_{maxV} \leqslant 1$, the problem is solved. Otherwise, when $m_{maxV} < -1$, as $h(m)$ is a convex function, we know that the minimum value reached by $h(m)$ on the domain $[-1,1]$ is equal to $h(-1)$. Thus, in logarithmic time, we look for two extremal vertices of the convex hull C_i^j in the direction $(1,1)$ and thus we know the value of $h(-1)$. In the same way, when $m_{maxV} > 1$, we compute the value of $h(1)$. As a conclusion, we formulate the following lemma:

Property 4. *We can determine the minimum vertical sectional length of the strips of the form* $\{(x, y) \mid mx + b \leqslant y \leqslant mx + c\}$ *with* $m \in [-1, 1]$ *that contain the subchain* P_i^j *in* $O(\log(j - i))$ *time.*

5.2. Managing the constraint on the VSL

We show that by using some information preprocessed in $O(n \log n)$ time, we are able to determine in constant time whether the minimum VSL of the strips covering P_i^j is less than ϵ. Let Ω_i denote the maximum index for which there exists a covering strip s associated with P_i^j. For any k such that $i < k \leqslant \Omega_i$, the strip s contains the subchain P_i^k and so we know that the minimum VSL associated with P_i^k is also less than ϵ. By definition of Ω_i, the minimum VSL associated with a subchain P_i^k, with $k > \Omega_i$, is greater than ϵ. Thus, the knowledge of Ω_i is sufficient to determine in constant time whether the minimum VSL associated with a subchain P_i^j is less than ϵ.

We now explain how to preprocess this sequence of values. A natural way to tackle this question is to incrementally insert vertices in the current convex hull and to check whether the minimum VSL of the associated subchain is still valid. We recall that the minimum VSL can be computed in logarithmic time from Property 4. Suppose we know the convex hull of the subchain P_i^k. Then, we successively insert the vertices $p_{k+1}, p_{k+2}, \ldots$ When the minimum VSL associated with the current subchain is greater than ϵ, we know that $\Omega_i = k - 1$. We move to the next value Ω_{i+1}. We could proceed in the same way, but it is useless to determine the minimum VSL associated with the subchains P_{i+1}^m, $i + 1 < m \leqslant \Omega_i$, because we already know that these values are less than ϵ. Thus, the next convex hull we need is associated with the subchain $P_{i+1}^{\Omega_i+1}$. It corresponds to last convex hull we process where we would have removed the vertex p_i. Thus, the need for a dynamic convex hull approach is apparent.

Dynamic convex hull is an active field in computational geometry. The best known lower bounds for the online version seem to be stated by Brodal and Jacob in [3] and by Chan in [7]. With these methods, each query on the convex hull takes a logarithmic time. Insertions and deletions are supported in quasi-logarithmic or in logarithmic amortized time. Nevertheless, their approaches incorporate complex data structures. The former result of Overmars and van Leeuwen's [25] is more suitable for implementation. But, their method supports insertions and deletions in only $O(\log^2 n)$ time.

Recently, the author in [6] presents an online method that updates the convex hull in $O(1)$ amortized time. The vertices are required to lie on a simple polygonal chain and some conditions on the order of insertion and deletion must be fulfilled. This method manages one decremental and one incremental convex hull at the same time. The incremental convex hull is processed by using Melkman's algorithm [23] that is linear in the number of inserted vertices. A simple variant of Melkman's method is sufficient to convert it to a decremental version. Thus, by maintaining at each iteration the merging between these two convex hulls, we obtain an algorithm that can compute in linear time all the convex hulls of the subchain $(P_{u_i}^{v_j})_{1 \leqslant i \leqslant n}$ where $(u_i)_{1 \leqslant i \leqslant n}$ and $(v_i)_{1 \leqslant i \leqslant n}$ are two non-decreasing sequences. The proof of this algorithm mainly relies on the fact that the convex hulls of two successive subchains of a simple polygonal chain have only two intersection points. Consult [6] for further explanation. This method, which is but an extension of Melkman's algorithm, uses only two double-ended queues and a simple routine to update the merging between the two hulls. Thus, this approach leads to a simple and efficient implementation. We conclude this section by the following property:

Property 5. *The values* Ω_k, $1 \leqslant k \leqslant n$ *associated with a simple polygonal chain of n vertices can be processed in* $O(n \log n)$ *time.*

Remark 3. If the input polygonal chain is not simple, we cannot use Melkman's approach. If we use any of the two previously cited optimal dynamic algorithms, each query on the convex hull has a logarithmic time complexity. As we perform $O(\log n)$ queries to determine the maximal VSL, the resulting time complexity would be $O(n \log^2 n)$. So, the method of Overmars and van Leeuwen's can be used because their approach yields to the same complexity.

6. The monotonicity tree

The definition of the error δ_H of a subchain P_i^j is described by three terms. The first comes from the minimum VSL of the strips covering P_i^j. We show in the previous section how we efficiently compute this value. The two other terms depend on the gap existing between the subchain and the two vertices p_i and p_j around the borders of the tolerance region. In this section, we present a convenient data structure useful to manage these constraints.

6.1. Definition

We define a *candidate point* of a vertex p_i to be any vertex p_k such that $k > i$ and such that p_k is the leftmost vertex of the subchain P_i^k. A natural order of these candidate points emerges if we store them by increasing indices or similarly

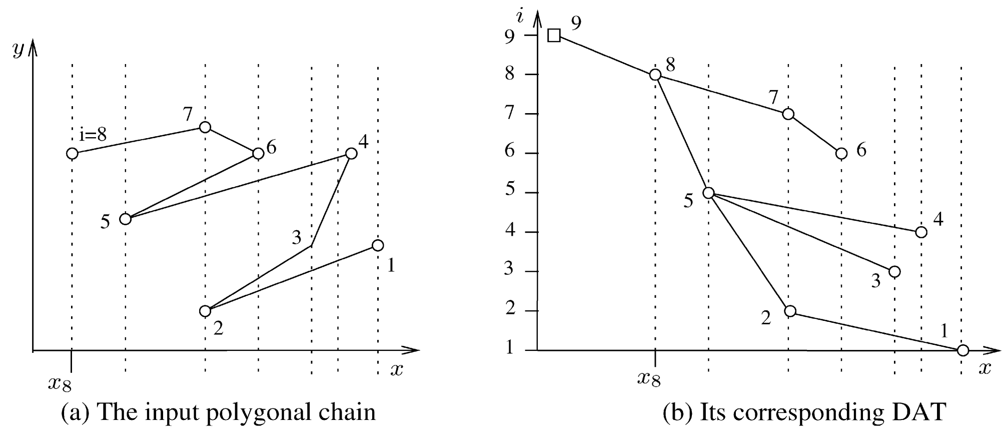

Fig. 12. The decreasing abscissae tree.

```
DAT Construction($(p_i)_{1\leqslant i\leqslant n}$)
    Stack $S$;   Tree $T$;              // Local data structures
    For  $i = 1$ to $n$ $T.CreateNode(i)$;
    For  $i = 1$ to $n$
      If  $S.NonEmpty()$
        While  $p_i.x \leqslant p_{S.top}().x$   // Compare $p_i.x$ with the abscissa
        $k = S.pop()$;                  // of the vertex on top of the stack.
        $T.AddEdge(k, i)$;              // Create and edge whose node $i$
      $S.Push(i)$;                      // is the parent of node $k$.
    While  $S.NonEmpty()$
        $k = S.pop()$;
        $T.AddEdge(k, n + 1)$;
    Return $T$;
```

Fig. 13. Pseudo-code of the decreasing abscissae tree construction.

by decreasing abscissae. We can create a forest where each node is associated with exactly one vertex of the chain. The parent–child relationship is defined as follows: the node p_k is the parent of the node p_i if p_k is the candidate point of p_i with the smallest index (see Fig. 12 for an example). For convenience, we add a fictitious node to obtain a rooted tree. This node corresponds to a vertex of index $n+1$ whose abscissa would be equal to $-\infty$. We call such a tree the *decreasing abscissae tree* (DAT).

6.2. Building the DAT

We show that we can build the DAT in linear time by traversing the vertices of the input chain. When we process a vertex v, we compare its abscissa with the abscissae of the vertices stored in a stack. By definition of the DAT, when we find a vertex w whose abscissa is greater than the abscissa of v, then v becomes the parent of the vertex w and w is removed from the stack. After this step, v is pushed onto the stack. Thus, v has the greatest abscissa of the vertices stored in the stack. As a consequence, the vertices are pushed onto the stack by decreasing abscissae. So, when we want to find the vertices in the stack whose abscissae is greater than a given value, it is sufficient to test the vertices by beginning at the top of the stack. As each vertex enters and leaves the stack only once, the tree is built in linear time. This yields to the algorithm presented in Fig. 13.

6.3. Managing the constraints on the borders

We present an application of this tree that consists of checking if the vertices of the subchain P_i^j lie at most $\epsilon/2$ away from the vertices p_i and p_j. Let L_i denote the smallest index greater than i satisfying $p_{L_i}.x < p_i.x - \epsilon/2$. By definition of L_i, the following property holds:

Property 6. *The subchain P_i^j satisfies $p_k.x \geqslant p_i.x + \epsilon/2$, for all $i \leqslant k \leqslant j$ iff we have $j \leqslant L_i$.*

We now focus on the problem of computing the sequence $(L_k)_{1\leqslant k\leqslant n}$. If a point $p_k, k > i$ lying on the left of a point p_i is not a candidate point of p_i in the DAT, this means that there exists an index k' with $i < k' < k$ such that $p_{k'}$ lies on the left of p_k. Thus, this point $p_{k'}$ prevents the point p_k from defining the value L_i. So, we only need to look for the candidate points of p_i to determine the value L_i. By operating a depth-first traversal in the DAT, when we visit a node p_i, we have in the current stack all its candidate points stored by decreasing abscissae. By applying a binary search, we determine the value L_i in logarithmic time. If no vertex of the polygonal chain lies on the left of p_i, the binary search will point out the fictitious node of the DAT. In this case, L_i equals $n+1$ and this value is valid relative to Property 6. As a conclusion, we state the final result of this section:

Property 7. *We can compute in $O(n \log n)$ time the sequence $(L_k)_{1 \leqslant k \leqslant n}$ where each value L_k represents the smallest index greater than k satisfying $p_{L_k}.x < p_k.x - \epsilon/2$.*

7. Algorithm design

7.1. A new approach

We recall that the class of the optimal algorithms is based on the general approach set up by Imai and Iri [20]. Their method consists of a two-step algorithm. In the first step, we build the graph G_ϵ that represents the subchains that can be simplified. Then, in the second step, a shortest path is computed to obtain an approximation with a minimum number of segments. We know that the optimal algorithms based on strong criteria require to build all the edges of G_ϵ in the worst-case. So, they inevitably suffer from a quadratic bottleneck. Our work belongs to this class of algorithms. We succeed in improving the approach of Imai Iri under the digital zone criterion and thus we present a new method that achieves a quasi linear time complexity.

The first key improvement of the Imai Iri approach is proposed by Chen and Daescu in [9]. They present a technique for finding the shortest path in G_ϵ incrementally and so the edges are computed only when needed. Thus, they avoid maintaining the graph G_ϵ explicitly in memory by combining the two distinct steps of Imai and Iri. Like this, they succeed in reducing the space complexity to $O(n)$. Nevertheless, their method needs to compute all the edges of G_ϵ in the worst-case, so their approach do not bypass the quadratic bottleneck.

The second key improvement appears when algorithms whose running time depends on the size of the output (see Section 3.3) were considered. For this, a breadth first traversal (BFT) is used to compute the shortest path in G_ϵ. Thus, we only generate the edges that can be reached in at most m iterations where m denotes the number of the vertices of the resulting approximation. Using this technique, the number of created edges is reduced, but remains quadratic in the worst case.

We introduce a new method for solving the min-# problem. As in the two previous improvements, we combine the two successive steps of Imai and Iri approach and we perform a breadth-first traversal of the graph. Our key improvement follows. Instead of checking all the emerging edges from the node we visit in order to find its unvisited neighboring nodes, we prefer to call a *guide* that only considers the emerging edges that lead to unvisited nodes. This is the crucial point of our method. Thus, by performing at most $n-1$ calls to the guide, we can operate a BFT in the graph without knowing all its edges. Each call generates an edge, and so at most $n-1$ edges of G_ϵ are built. The knowledge of these edges is sufficient to determine a shortest path. This approach can be reused for any criterion where we can set up such a guide. Thus, the following assertion holds:

Observation 1. *For a given approximation criterion, if we can set up a* guide *that reveals an edge between a visited node and an unvisited node in $\theta(n)$ time, then we can solve the min-# problem in $O(n \cdot \theta(n))$ time by performing a breadth-first traversal of G_ϵ.*

7.2. Introducing the guide

When we visit a node p_i during the BFT, we want to efficiently detect an unvisited node p_j such that the subchain P_i^j is included either in a tolerance region of δ_H or in a tolerance region of δ_V. These two cases can be processed separately. Thus, we only present the guide, denoted *guide*$_H$, used for finding the unvisited vertices p_j, $j > i$ from a node p_i such that the subchain P_i^j is included in a tolerance region of δ_H (see Fig. 7 for an example). The other guide, denoted *guide*$_V$ is built analogously. We recall the conditions that must satisfy a subchain P_i^j in order to lie in a tolerance region of δ_H:

(1) P_i^j is included in a strip with a valid VSL and with a valid slope;
(2) For all $k, i \leqslant k \leqslant j$ $p_k.x \geqslant \min(p_i.x, p_j.x) - \epsilon/2$;
(3) For all $k, i \leqslant k \leqslant j$ $p_k.x \leqslant max(p_i.x, p_j.x) + \epsilon/2$.

In Section 5.2, we define the notation Ω_i that denotes the maximum index for which there exists a valid strip containing P_i^j. In this way, the first condition can be rewritten as $j \leqslant \Omega_i$. The second and the third conditions are a bit more difficult to transform. We are forced to consider two different cases depending on the location of p_j relative to p_i. Let us suppose that p_i lies on the left of p_j. We can define the two sequences $(L_k)_{1 \leqslant k \leqslant n}$ and $(R_k)_{1 \leqslant k \leqslant n}$ where each value L_k and R_k represent the smallest index greater than k that respectively satisfies $p_{L_k}.x < p_k.x - \epsilon/2$ and $p_{R_k}.x > p_k.x + \epsilon/2$. In this way, the second condition can be transformed into $j < L_i$. Similarly, we define the two sequences $(\overline{L}_k)_{1 \leqslant k \leqslant n}$ and $(\overline{R}_k)_{1 \leqslant k \leqslant n}$

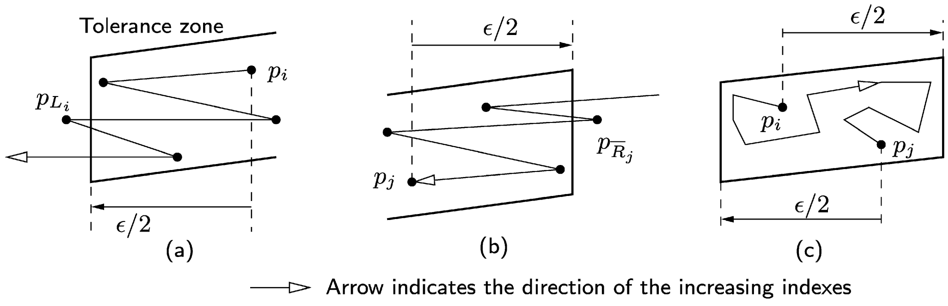

Fig. 14. Rewriting the conditions that define the approximation error δ_H.

where each value $\overline{L}_k$ and $\overline{R}_k$ represent the greatest index smaller than k that respectively satisfies $p_{\overline{L}_k}.x < p_k.x - \epsilon/2$ and $p_{\overline{R}_k}.x > p_k.x + \epsilon/2$. The third condition can be rewritten as: $i > \overline{R}_j$. See an example in Figs. 14(a) and 14(b). Finally, we obtain the following property that allows to determine whether a subchain P_i^j with $p_i.x \leqslant p_j.x$ will satisfy $\delta_H(P_i^j) \leqslant \epsilon$:

Property 8. *The subchain P_i^j with $p_i.x \leqslant p_j.x$ satisfies $\delta_H(P_i^j) \leqslant \epsilon$ iff we have $i < j$, $p_i.x \leqslant p_j.x$, $j \leqslant \Omega_i$, $j < L_i$ and $i > \overline{R}_j$.*

We extend this property to the general case where no assumption is required on the location of p_j relative to p_i. The next property is the keystone of the function *guide*$_H$.

Property 9. *The subchain P_i^j satisfies $\delta_H(P_i^j) \leqslant \epsilon$ iff at least one of the two following conditions is fulfilled*:

(1) $i < j \leqslant \min(\Omega_i, L_i - 1)$ *and* $\overline{R}_j < i$.
(2) $i < j \leqslant \min(\Omega_i, R_i - 1)$ *and* $\overline{L}_j < i$.

Proof. From Property 8, we know that if $\delta_H(P_i^j) \leqslant \epsilon$ and if $p_i.x \leqslant p_j.x$ then the first condition is fulfilled. Symmetrically, when $p_i.x \geqslant p_j.x$, the second condition holds. The converse statement is not immediate. Let us suppose that the subchain P_i^j satisfy $p_i.x \leqslant p_j.x$. From Property 8, we known that if the first condition is fulfilled we have $\delta_H(P_i^j) \leqslant \epsilon$. We examine what happens when the second condition only is satisfied, see an example in Fig. 14(c). As $i < j < R_i$, this means that $p_k.x \leqslant p_i.x + \epsilon/2$ for all $k, i \leqslant k \leqslant j$. As p_i lies on the left of p_j, this implies that $p_k.x \leqslant p_j.x + \epsilon/2$ for all $k, i \leqslant k \leqslant j$. Analogously, we deduce that $p_k.x \geqslant p_i.x - \epsilon/2$ for all $k, i \leqslant k \leqslant j$. As a conclusion, even if the constraint on the location of p_j relative to p_i disappears from the two stated conditions, this does not impact on the process of detecting the vertices p_j satisfying $\delta_H(P_i^j) \leqslant \epsilon$. □

7.3. Setting up the guide and complexity analysis

From Section 6.3, we know that we can preprocess the sequence $(L_k)_{1\leqslant k\leqslant n}$ in $O(n \log n)$ time. Similarly, we precompute the three other sequences $(\overline{L}_k)_{1\leqslant k\leqslant n}$, $(R_k)_{1\leqslant k\leqslant n}$ and $(\overline{R}_k)_{1\leqslant k\leqslant n}$ with the same performance. In Section 5.2, we show that the sequence $(\Omega_k)_{1\leqslant k\leqslant n}$ can be created in $O(n \log n)$ time. Thus, the preprocessing stage has an $O(n \log n)$ time complexity.

The function *guide*$_H$ follows from Property 9. By using two priority search trees PST_1^H and PST_2^H, we can easily process each of the two conditions. We recall that a PST is a specialized data structure that can be used for two-dimensional rectangular range queries of the form $[a_1, a_2]\times] - \infty, b_2]$. For this, we represent each vertex p_k, $1 \leqslant k \leqslant n$, by the pair of values $(k, \overline{R}_k)$ in PST_1^H and by the pair $(k, \overline{L}_k)$ in PST_2^H. Thus, we are now able to set up the function *guide*$_H$. We present the main loop of our algorithm in Fig. 15.

The PST can be constructed in $O(n \log n)$ time. Moreover, this data structure supports queries and deletions in $O(\log n)$ time. Each time an unvisited node p_j is found by a query in a PST, we remove it from all the PSTs of the different guides. Thus, we can guarantee that we visit each node only once during the BFT. As a conclusion, we state the following property:

Property 10. *The min-# problem can be solved under the digital zone criterion in $O(n \log n)$ time using a preprocessing step that achieves the same time complexity.*

8. Experiments

In this section, we present some results of our experiments. We decide to compare the digital zone criterion (DZC) with the tolerance zone criterion (TZC) and the Douglas–Peucker heuristic (DPH). As input, we use a polygonal chain of 209 regularly distributed vertices that represent the Italian coast border (see Fig. 17).

```
// $PST_1^H$, $PST_2^H$, $PST_1^V$ and $PST_2^V$ correspond to priority search trees.
// $(\Omega_i)_{1\leqslant i\leqslant n}$, $(L_i)_{1\leqslant i\leqslant n}$, $(\bar{L}_i)_{1\leqslant i\leqslant n}$, $(R_i)_{1\leqslant i\leqslant n}$ and $(\bar{R}_i)_{1\leqslant i\leqslant n}$ correspond to
// arrays. These preprocessed data structures are used as global variables.

Function guide_H(i)
  $S_1 = PST_1^H([i+1, \min(\Omega_i, L_i - 1)], ]-\infty, i-1])$;
  For each $k \in S_1$
     Remove $k$ from $PST_1^H$, $PST_2^H$, $PST_1^V$ and $PST_2^V$;
  $S_2 = PST_2^H([i+1, \min(\Omega_i, R_i - 1)], ]-\infty, i-1])$;
  For each $k \in S_2$
     Remove $k$ from $PST_1^H$, $PST_2^H$, $PST_1^V$ and $PST_2^V$;
  Return $S_1 \cup S_2$;

Function BFS Spanning Tree Construction($(p_i)_{1\leqslant i\leqslant n}$)
  Queue $Q$;
  Graph $ST$;                // spanning tree
  $Q.Enqueue(0)$;            // store the vertex $p_0$ in the queue
  While $Q.nonEmpty()$       // main loop of the breadth-first traversal
     $i = Q.Dequeue()$;
     $S = guide_H(i)$;
     For each $k \in S$
        $Q.Enqueue(k)$;    $ST.AddEdge(p_i p_k)$;
     $S = guide_V(i)$;
     For each $k \in S$
        $Q.Enqueue(k)$;    $ST.AddEdge(p_i p_k)$;
  Return $ST$;
```

Fig. 15. Core functions of the simplification algorithm.

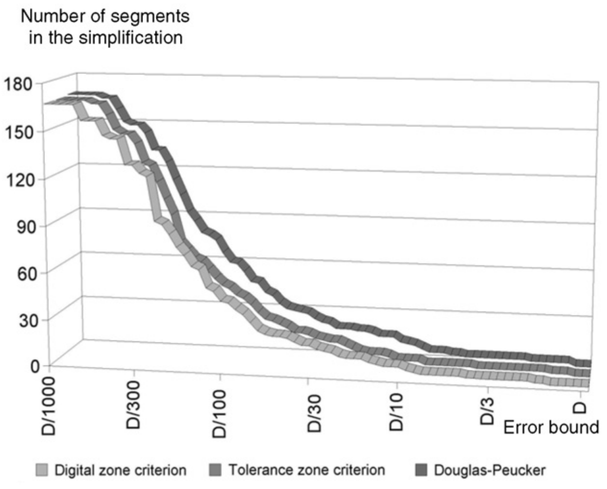

Fig. 16. Simplification algorithms' performance.

8.1. *Efficiency*

We study the variation of the number of segments in the simplified polygonal chains relative to the error bound value (see Fig. 16). Let ϵ_{DZC}, ϵ_{TZC} and ϵ_{DPH} denote, respectively, the error bound of the DZC, the TZC and the DPH. To compare the three methods fairly, we try to choose some values for the error bounds that lead to geometrically equivalent tolerance regions. In fact, when a subchain is covered by a parallelogram of thickness $\epsilon_{DZC} = 1$, this parallelogram is approximatively equivalent to the tolerance region of the TZC when $1/\sqrt{2} \leqslant \epsilon_{TZC} \leqslant 1/2$. This variation depends on the direction of the line-segment chosen by the TZC. This remark holds for the DPH because this method is also based on an error measure associated with the Euclidean distance. Thus, to compare these criteria fairly, we choose, when $\epsilon_{DZC} = 1$, to set ϵ_{TZC} and ϵ_{DPH} equal to $1/\sqrt{2}$. On one hand, the tolerance regions associated with the DZC are always thinner than the tolerance regions generated by the TZC and the DPH. On the other hand, the TZC and the DPH force the line-segments of the simplified chain to lie in the middle of their tolerance regions. Thus, the settings chosen for the simplification process balance advantages and disadvantages between each criterion.

We compare the number of generated segments in the simplifications when ϵ_{DZC} varies from $D/1000$ to D, where D represents the size of the image. For the TZC and the DPH, we apply the correction factor of $1/\sqrt{2}$. See results in Fig. 16. Experiments show that the performance of the DZC and the TZC are very similar. Their two curves are quasi superimposed.

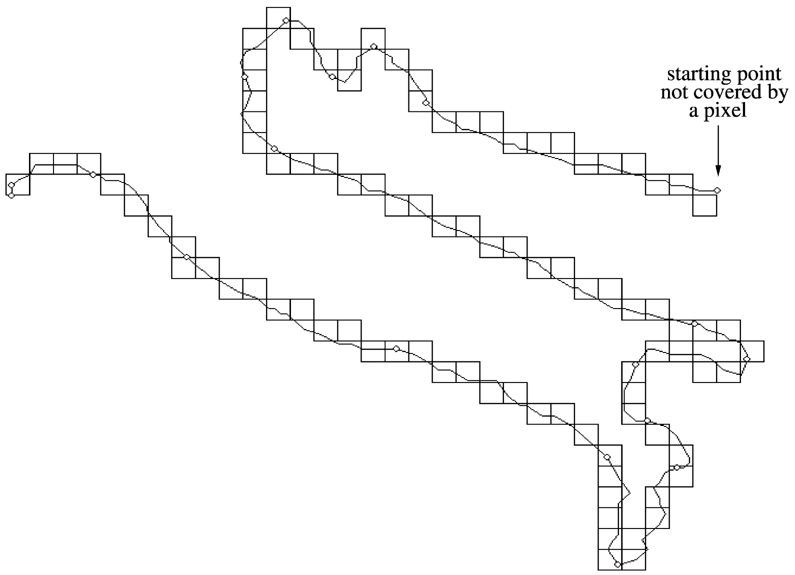

Fig. 17. Rendering of digital zone criterion when ϵ is equal to the pixel pitch.

When ϵ is between $D/300$ and $D/10$, the DPH produces simplifications that have between 30% and 50% more vertices than the two other methods. This graph confirms that the optimality reached by our method is equivalent to the TZC optimality. We recall that our algorithm bypasses the quadratic time bottleneck contrary to the TZC.

8.2. Rendering and algorithm weakness

By definition, the tolerance regions of the DZC produce an image in such a way that the original chain lies at most half a pixel away from the drawn pixels. This produces a pleasant simplification. See an example in Fig. 17. Moreover, we know the optimal value of ϵ_{DZC} in order to obtain this property. This is possible neither with the TZC nor with the DPH.

Unlike other simplification criteria, the DZC does not guarantee that the first and the last vertices of the polygonal chain lie in the pixels generated by the rendering. The shape of the simplification is preserved, but, the fact that these two endpoints lie outside of the pixels of the image can produce an uncomfortable feeling. Fig. 17 shows such a case. This behavior is nonetheless unavoidable, because it is valid with respect to the definition of our approximation criterion.

9. Conclusion

Our algorithm solves the min-# problem under a new criterion called the digital zone criterion. It produces an approximation with a minimum number of vertices. When the maximal error of the simplification is fixed to the pixel pitch, the rendering is guaranteed to lie at most half a pixel away from the original chain. Our method retains the shape of the input polygonal chain. Moreover, our method achieves an $O(n\log n)$ time complexity. Thus, this is the first near-linear time algorithm for the min-# problem that ensures optimality and that preserves the features of the original chain at the same time. This method is based on simple data structures such as stacks, queues and priority search trees. Its inner machinery uses binary search and a simple variant of Melkman's convex hull algorithm. Therefore, it leads to a simple and efficient implementation. This algorithm is well suited for peripherals with a low screen resolution like mobile phones, handled video game consoles and automotive navigation systems. Its main application consists of producing the most accurate rendering with a minimum number of segments.

Acknowledgements

The author would like to thank the reviewers for their helpful suggestions.

References

[1] P.K. Agarwal, K.R. Varadarajan, Efficient algorithms for approximating polygonal chains, Discrete Comput. Geom. 23 (2000) 273–291.
[2] M. de Berg, M. van Kreveld, S. Schirra, A new approach to subdivision simplification, in: Proc. Twelfth International Symposium on Computer-Assisted Cartography, 1995, pp. 79–88.
[3] G.S. Brodal, R. Jacob, Dynamic planar convex hull, in: Proc. 43rd Annual Symp. on Foundations of Computer Science, 2002, pp. 617–626.
[4] L. Buzer, Digital line recognition, convex hull, thickness, a unified and logarithmic technique, in: Proc. IWCIA, 2006, pp. 189–198.
[5] L. Buzer, A survey and a new competitive method for the planar min-# problem, arXiv: cs.CG/0209007, 2002.

[6] L. Buzer, Computing multiple convex hulls of a simple polygonal chain in linear time, in: Proc. EWCG, 2007, pp. 114–117.
[7] T.M. Chan, Dynamic planar convex hull operations in near-logarithmic amortized time, J. ACM 48 (1) (2001) 1–12.
[8] W.S. Chan, F. Chin, Approximation of polygonal curves with minimum number of line segments or minimum error, Internat. J. Comput. Geom. Appl. 6 (1) (1996) 59–77.
[9] D.Z. Chen, O. Daescu, Space-efficient algorithms for approximating polygonal curves in two dimensional space, Int. J. Comput. Geom. Appl. 13 (2) (2003) 95–112.
[10] O. Daescu, New results on path approximation, Algorithmica 38 (2) (2003) 131–143.
[11] O. Daescu, N. Mi, Polygonal chain approximation: A query based approach, Comput. Geom. Theory Appl. 30 (1) (2005) 41–58.
[12] O. Daescu, N. Mi, C.-S. Shin, A. Wolff, Farthest-point queries with geometric and combinatorial constraints, Comput. Geom. Theory Appl. 33 (3) (2006) 174–185.
[13] D. Douglas, T. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, Canadian Cartographer 10 (2) (1973) 112–122.
[14] R. Estkowski, No Steiner point subdivision simplification is NP-complete, in: Proc. of the 10th Canadian Conference on Computational Geometry, 1998, pp. 76-77.
[15] D. Eu, G. Toussaint, On approximation polygonal curves in two and three dimensions, Graph. Models Image Process. 56 (3) (1994) 231–246.
[16] S.L. Hakimi, E.F. Schmeichel, Fitting polygonal functions to a set of points in the plane, Graph. Models Image Process. 53 (2) (1991) 132–136.
[17] J. Hershberger, J. Snoeyink, Speeding up the Douglas–Peucker line simplification algorithm, in: Proc. 5th Intl. Symp. Spatial Data Handling, IGU Commission on G.I.S., 1992, pp. 134–143.
[18] J. Hershberger, J. Snoeyink, Cartographic line simplification and polygon CSG formulae in $O(n\log^* n)$ time, in: 5th International Workshop on Algorithms and Data Structures, 1997, pp. 93–103.
[19] X. Hilaire, K. Tombre, Robust and accurate vectorization of line drawings, IEEE Trans. Pattern Anal. Machine Intell. 28 (6) (2006) 890–904.
[20] H. Imai, M. Iri, Polygonal Approximations of a Curve-Formulations and Algorithms in Computational Morphology, North-Holland, Amsterdam, 1988.
[21] D. Kirpatrick, J. Snoeyink, Tentative prune-and-search for computing fixed-points with applications to geometric computation, Fund. Inform. 22 (4) (1995) 353–370.
[22] R.B. McMaster, Automated line generalization, Cartographica 24 (2) (1987) 74–111.
[23] A. Melkman, On-line construction of the convex hull of a simple polyline, Inf. Proc. Lett. 25 (1987) 11–12.
[24] A. Melkman, J. O'Rourke, On Polygonal Chain Approximation in Computational Morphology, North-Holland, Amsterdam, 1988.
[25] M.H. Overmars, J. van Leeuwen, Maintenance of configurations in the plane, J. Comput. Syst. Sci. 23 (1981) 166–204.
[26] J.P. Reveillès, Géometrie discrète, calculs en nombre entiers et algorithmique, Thèse d'état, Université Louis Pasteur, Strasbourg, France, 1991.
[27] A. Saalfeld, Topologically consistent line simplification with the Douglas–Peucker algorithm, Cartography Geographic Inform. Sci. 26 (1) (1999) 7–18.
[28] D. Thalmann, A key-posture extraction out of human motion data, in: Proc. IEEE Conf. Engineering in Medicine and Biology Society (EMBC), vol. 2, 2001, pp. 1167–1169.
[29] G.T. Toussaint, On the complexity of approximating polygonal curves in the plane, in: Proc. IASTED, International Symp. on Robotics and Automation, 1985.
[30] L. Wenyin, D. Dori, A protocol for performance evaluation of line detection algorithms, Machine Vision Appl. 9 (1997) 240–250.