Average Curve of n Digital Curves

Isabelle Sivignon

Univ. Grenoble Alpes, CNRS, Grenoble INP, GIPSA-Lab, 38000 Grenoble, France
isabelle.sivignon@gipsa-lab.grenoble-inp.fr

Abstract. Building on [23], we investigate the problem of defining and computing the average of digital curves. Given n digital curves that satisfy compatibility conditions, a set - called the *gap* - in which the average curve is looked for, is defined. The proposed definition rewrites the classical arithmetic mean for curves by (i) defining the distance between each point of the gap and its projection on each curve and (ii) computing the points that minimize the sum of squared deviations. We show that, algorithmically speaking, computing such projections comes down to computing a distance transform with visibility constraints. We propose a fast algorithm to compute a good approximation of these distance maps and finally show that the average curve can be obtained using classical watershed algorithm.

Keywords: Digital curves, Arithmetic mean, Distance Transform, Visibility

1 Introduction

Context Shape averaging is an important task in many applications in order to take into account fluctuation of data, sensibility to the choice of parameters or to compute a morphing between several shapes. To cite only a few examples, it may be used in geological studies to define a shoreline despite short-lived deformations, or as a post-treatment in image segmentation to average the effect of the choice of algorithm parameters. Many works have been conducted on this subject, especially when input data is a set of n polygonal curves. When $n = 2$, medial axis provides a solution. When $n > 2$ the problem is much more complex, and a two-steps process is usually followed: first establish a correspondence between input curves, and then construct a "centroid" from this correspondence. Correspondence between input curves can be achieved using for instance the Fréchet distance [5], or normal map computation [15] for instance. However, in the examples cited above, shapes are defined as objects in digital images, so that defining the average of digital shapes is particularly relevant for many applications. On this matter, most previous works focused on morphing-based approaches [3].

Arithmetic mean Going back to basics, the arithmetic mean $\bar{x}$ is the best way to select a "typical" value for a set of input values x_i, in the sense that it minimizes the sum of squared deviations. Indeed, if we denote $f(x) = \sum_i (x - x_i)^2$, $\bar{x} =$

$argmin_x f(x)$. Minimizing f can be done either by looking for local minima of f, or by looking for the zero set of its derivative $f'(x) = \sum_i (x - x_i)$. Anyway, the arithmetic mean $\bar{x}$ lies in the interval $[x_m, x_M]$, where x_m and x_M denote the minimal and maximal value of the x_i respectively.

Outline In [23], the authors present a framework to extend the notion of arithmetic mean to "values" that are not numbers but smooth curves. In this article, we investigate the extension and relevance of this definition of average curve in the case of digital curves. We also discuss efficient algorithmic solutions to compute it. In Section 2 we recall the main results of [23] and revisit them for digital curves, which reveals a new constrained distance transform problem. Section 3 focuses on algorithmic solutions to solve this problem. Then, in Section 4 we show how to obtain an average curve from the constrained distance map and give some experimental results. Last, Section 5 raises the question of necessary conditions on the input curves for this approach to be valid, and provides tentative answers.

2 Which definition of Average Curve ?

2.1 "Arithmetic" mean curve

Given a set of n smooth Jordan curves $\{\mathscr{C}_i\}$, the authors of [23] define the Valley Average Curve as the valley of the scalar height field $Q(p) = \sum_i d(p, \pi_i(p))^2$, and the Zero Average Curve as the zero set of the scalar height field $D(p) = \sum_i d(p, \pi_i(p))$, where d denotes the Euclidean distance and $\pi_i(p)$ the "projection" of p onto $\mathscr{C}_i$. Since all curves are Jordan, each $\mathscr{C}_i$ is the boundary of a bounded shape $\mathscr{S}_i$. As the arithmetic mean of a set of numbers lies somewhere in between the minimum and maximum values, the average curve is looked for somewhere in the symmetric difference of the $\mathscr{S}_i$: no point of the average curve may lie in the interior or in the exterior of all the $\mathscr{C}_i$. Formally, in [23] the authors define the *gap* of $\{\mathscr{C}_i\}$ as $\Delta(\{\mathscr{C}_i\}) = (\bigcup \overline{\mathscr{S}_i}) \backslash (\bigcap \mathring{\mathscr{S}_i})$ (see Figure 1 - we use the notation Δ for short when there is no ambiguity). Note that, from this definition, an interesting property is that the average curve goes through all the points shared by all input curves.

The authors remarked that if the projection $\pi_i(p)$ of a point p is defined as the point of $\mathscr{C}_i$ closest to p, some parts of the average curve may be missed in the case of the Valley Average Curve, or

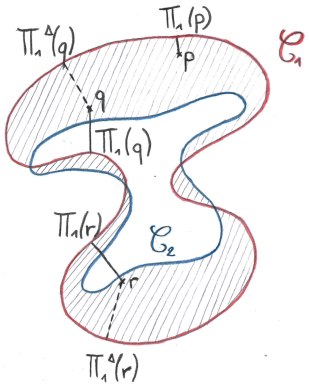


Fig. 1. Gap (dashed) of two curves $\mathscr{C}_1$ and $\mathscr{C}_2$. The closest projections $\pi_1(q)$ and $\pi_1(r)$ of q and r on $\mathscr{C}_1$ yield erroneous mean curve. Gap relative projections π_1^Δ are defined instead.

misplaced for the Zero Average Curve (see Figure 1). Indeed, the idea behind the definition of average curve is to suppose that projections of each point of the average curve $\mathscr{C}$ on each $\mathscr{C}_i$ provide a continuous and bijective mapping between $\mathscr{C}$ and each $\mathscr{C}_i$: if the segment $[p, \pi_i(p)]$ crosses Δ, this property fails. Thus they introduce the *gap relative projection* $\pi_i^{\Delta}(p)$ as the closest point on $\mathscr{C}_i$ such that $[p, \pi_i^{\Delta}(p)] \subseteq \Delta$.

2.2 Arithmetic mean digital curve

Consider a set of n digital 4-connected simple and closed curves $\{C_i, i = 1 \dots n\}$. Each C_i being a Jordan curve, $\mathbb{Z}^2 \backslash C_i$ has exactly two 8-connected components, the unbounded one being called its exterior, and the other its interior. We denote by O_i the digital object defined as the union of C_i and its interior, that we denote $\mathring{O}_i$. Similarly to [23] we define the *gap* of $\{C_i\}$ as $\Delta(\{C_i\}) = (\bigcup O_i) \backslash (\bigcap \mathring{O}_i)$ (or Δ for short). Note that the Zero Average Curve definition involves non-integer computation, while the Valley Average Curve can be computed using integers when all points are in $\mathbb{Z}^2$. That is why in the following, we focus on the Valley Average Curve definition, that we call arithmetic mean digital curve.

For any point $p \in \mathbb{Z}^2$, we can define $H(p) = \sum_i d(p, \pi_i^{\Delta}(p))^2$ (H is called height map) as before. However, the gap relative projection π_i^{Δ} has to be redefined for digital points and objects. Given a model of digital straight segment (DSS for short), we say that two points p and q are *visible* in a digital set A and denote $Vis_A(p, q)$ if there exists a DSS $s \subseteq A$ such that $p, q \in s$. We also define $V_A(p)$ as the visibility region of p in $A : V_A(p) = \{q \in A | Vis_A(p, q)\}$. Thus we have $\pi_i^{\Delta}(p) = \arg\min_{q \in V_{\Delta}(p) \cap C_i} \{d(p, q)\}$.

Hence, a main issue in the computation of the arithmetic mean digital curve is to compute gap relative projections efficiently for all points in the gap, which is the topic of the next section.

3 Computation of Gap Relative Projections

We consider here the most general problem of computing the visible projection of any point of a set A with respect to a subset $B \subseteq A$. In the following, A^c is the complement of A in $\mathbb{Z}^2$, and points of A^c are called *obstacles*.

Problem 1 (Visible Voronoi Map). Given two digital sets A and B with $B \subseteq A$, compute for each point $p \in A$ the projection $q \in B$ of p such that $Vis_A(p, q)$.

3.1 Brute force algorithm and discussion

Algorithm Efficient separable algorithms to compute distance transforms and Voronoi Maps have been proposed [8], and can be used to design a first straightforward brute-force algorithm. Given two digital sets A and B such that $B \subseteq A$, Algorithm 1 solves Problem 1. Function VISIBLE is the implementation of the Vis predicate defined in Section 2.2.

Algorithm 1: BruteForceVisibleVoronoiMap(DIGITAL SET A, DIGITAL SET B, $B \subseteq A$)

```
1 Initialization: VisiVoroMap(p) = VoronoiMap(p) = arg min_{r∈B} d(p, r);
2 for p ∈ A do
3     if ¬Visible(A, p,VisiVoroMap(p)) then
4         Loop on points of B to find the visible projection q of p in A;
5         VisiVoroMap(p) = q
6 return VisiVoroMap
```

Proposition 1. *If n and m are the cardinality of A and B respectively ($n > m$), if V denotes the computational cost of* VISIBLE*, and k is the number of points of A for which the Voronoi Map differs from the Visible Voronoi Map then Algorithm 1 is in $O(n + V.(n + km))$.*

The complexity V of the VISIBLE predicate is discussed in Section 3.3. However, even if V is neglected, this algorithm is not satisfying since, when k and m are in the order of n, the complexity becomes quadratic in n.

Discussion Several options may be considered to design a more efficient algorithm. The first one would be to adapt the optimal separable distance transform algorithm of [8] to take into account obstacles. Following the rewriting proposed in [7] using two predicates (CLOSEST and HIDDENBY), it would be a matter of modifying the HIDDENBY predicate since a site hidden by others in the Voronoi Map may very well be the projection of a point if other sites are not visible from this point. It appears that, with no further assumption on obstacles, the list of sites cannot be shortened as efficiently as for the Voronoi Map.

Another option would be to first compute the visibility region of each point $p \in A$ before computing the distance transform of p with respect to $V_A(p) \cap B$. Computing the visibility region of a point has ben widely studied in computational geometry [16], where obstacles are polygons. In the case of digital sets, the problem has been tackled in [6] but the difficulty here is to efficiently compute the visibility region for all points of A.

Yet another alternative would be to build a tailored data structure to perform efficient visible nearest neighour queries, as proposed in [19] for instance. Here again, obstacles are supposed to be represented as polygons, and a fast computation of a predicate returning the minimum visible distance from a point to an obstacle is key. In our context, an important point is the definition of obstacles, which could range from an obstacle per point in B (a lot of obstacles with a very fast predicate) to connected components of B (few obstacles but more convoluted predicate) for instance. The best solution would highly depend on the geometry of B with respect to A.

The last option we discuss here is to propagate projections locally, from point to point. Even though it has been reported in the literature that such local approaches fail to compute exact Voronoi Maps [12], in Section 3.2 we provide

Algorithm 2: FMMLoop(Accepted Point Set Λ, Candidate list Γ)

```
1 Extract from Γ the point p with smallest distance to its projection π^A(p);
2 if p ∉ Λ then
3 |   Λ = Λ ∪ {p, π^A(p)};
4 |   foreach q ∈ N(p) ∩ A do addCandidatePoint(Λ,Γ,A,q) ;
```

new results on local approaches accuracy, and show that a strong point of this approach is that visibility can be checked on the fly. In the next section, we propose to adapt the Fast-Marching Method [24] (FMM for short) algorithm to compute an approximation of the solution to Problem 1, but other local schemes may also be considered.

3.2 Fast Marching Method-like algorithm using projections

Basic loop The algorithm follows the propagation principle of the Fast Marching Method (which in turns is very similar to Dijkstra's shortest path algorithm). It was designed primarily to describe the evolution of an interface as a function of time and at a given speed in the normal direction to the surface. But more generally, given a domain A and a set $B \subseteq A$, the algorithm computes for each point p of A an approximation of the length of the shortest geodesic from B to p in A. From the list of points for which the distance is known (called *accepted points*), a list of *candidate points* that lie in the neighbourhood of accepted points is maintained. For each candidate point, a tentative value of distance is computed from its neighbours belonging to the accepted point set. Each step of the algorithm consists in adding the candidate point with the minimum distance. We adopt the same simplified FMM loop as presented in [17], which is sketched in Algorithm 2, and implemented in DGtal library [1]. Initially, accepted point set Λ is set to B, candidate list Γ is empty.

Algorithm 3 details the addCandidatePoint function. Parts in orange are to be disregarded at this point and will be explained later in this section. In order to be able to include a visibility constraint in the course of the algorithm, instead of propagating distance information from point to point, visible projection point is propagated. Note that a point may appear several times in the candidate list, with different projections, and thus distance values.

About the accuracy of the algorithm Propagating closest point information in FMM, without the visibility constraint, was already done in [4]. Accuracy relies on the assumption that for any point $p \in A$, its visible projection in B is also the visible projection of one of its neighbours. We formalize this condition in Proposition 2 when $\mathcal{N}$ is the classical 8-neighbourhood $\mathcal{N}_8$.

Proposition 2. *Let $p \in A$, and $\pi^A(p)$ its ground truth visible projection on B. If there exists an 8-connected path of points $\{p_i\}_{1..k}$ in A such that (i) $p_1 = p$,*

Algorithm 3: addCandidatePoint(Accepted Point Set Λ, Candidate list Γ, Digital Set A, Digital Set B, Point p)

```
1 L = ∅;
2 foreach q ∈ N(p) ∩ Λ do
3     if Visible(A, p, π^A(q)) then L = L ∪ π^A(q);
4     else
5         τ(p) = computeProjTangent(p, q, A, B);
6         if τ(p) exists then L = L ∪ τ(p);
7     π^A(p) = arg min_{x∈L}{d(p, x)}; Γ = Γ ∪ {p, π^A(p)};
```

$p_k = \pi^A(p)$, for all $i \in \{1 \dots k\}$ $\pi^A(p_i) = \pi^A(p)$ (ii) the distance to $\pi^A(p)$ is decreasing, then Algorithm 2 computes the exact visible projection $\pi^A(p)$ of p.

Proof. We proceed by induction on the length of the path.

[Initialization] If $k = 2$, then p is 8-neighbour to $\pi^A(p) \in B$. If $\pi^A(p)$ is the only point of B in $\mathcal{N}_8(p)$, we are done. Otherwise, any point of B wad added to Λ before p, and p was added in the candidate point set Γ, using its 8-neighbours, in particular $\pi^A(p)$. Then p may have been added several times in Γ with different tentative projections, but only the one with the closest tentative projection, here $\pi^A(p)$, is actually added in Λ thanks to line 2 of Algorithm 2.

[Induction step] Consider a point $p \in A$ for which there exists a path $\{p_i\}$ of length $j + 1$, with $p_1 = p$ and such that the two conditions of Proposition 2 are satisfied. Point p_2 is such that $\pi^A(p_2) = \pi^A(p)$ and there exists a path of length j between p_2 and $\pi^A(p_2)$. By induction hypothesis, Algorithm 2 correctly computes $\pi^A(p_2)$. Consider now the moment when p_2 is included in the accepted point set Λ (Algorithm 2 line 4). Several cases may arise:

- if p_1 is not in Λ, and whether it is already in the list of candidates Γ or not, it is added to the list, with the smallest distance possible since $\pi^A(p_2)$ is equal to the true visible projection $\pi^A(p)$ of p ;
- if p_1 is already in Λ with an erroneous projection, this means that p_1 was added before p_2 (otherwise p_1, as a 8-neighbour of p_2, would have been added to Γ line 7 of Algorithm 3, since $\pi^A(p_2) = \pi^A(p)$). Let q be the projection associated to p when it was added to Λ. We have $d(p, q) \leq d(p_2, \pi^A(p_2))$. But by hypothesis we have $d(p, \pi^A(p)) \geq d(p_2, \pi^A(p)) = d(p_2, \pi^A(p_2)) \geq d(p, q)$, thus contradicting the fact that $\pi^A(p)$ is the ground truth projection.

The sufficient conditions of Proposition 2 may be broken in at least two situations. Firstly, even if there is no obstacle around p and $\pi^A(p)$, $\pi^A(p)$ may not be the projection of any of p's neighbours. This issue is inherent in any approach based on local propagation because digital Voronoi cells may be not connected. This has been discussed in the literature, and we extend here the results of [12] by showing that erroneous values cannot occur close to B.

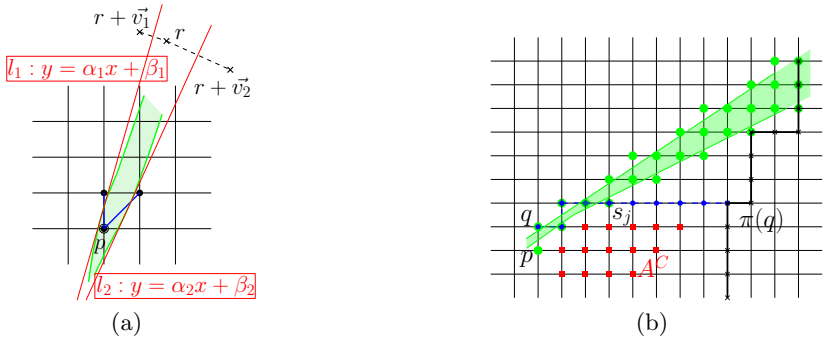


Fig. 2. (a) Illustration of the proof of Proposition 3. V is (incompletely) depicted in green. r is the site associated to V. (b) Computation of a tangent from a point p using the DSS from one of its neighbours q to $\pi^A(q)$ (in blue). Red points are obstacles. Points in B are marked by black crosses.

Proposition 3. *For any digital set B, for all p such that $d(p, B) < 13$, there exists a point $q \in \mathcal{N}_8(p)$ such that $\pi(q) = \pi(p)$, where $\mathcal{N}_8$ denotes the 8 neighbourhood.*

Proof. Suppose wlog that $p(0, 0)$. See Figure 2(a) for an illustration of the notations. If there is no point in $\mathcal{N}_8(p)$ such that $\pi(q) = \pi(p)$, this implies that the Voronoi cell V of $\pi(p)$ does not contain any point of $\mathcal{N}_8(p)$. Up to symmetries, this means that the boundary of V contains two segments, one cutting the segment $[p, p + (0, 1)]$ and the other cutting the segment $[p, p + (1, 1)]$. If we call $l1$ and $l2$ respectively the two lines (of positive slopes) supporting these two segments, this translates into the following constraints: $\beta_1 < 1$, $\beta_1 > 0$, $\beta_2 + \alpha_2 > 1$ and $\beta 2 < 0$. Let r be the site associated to p (its closest point in B). The two lines l_1 and l_2 are bisectors of r with points $r + \boldsymbol{v_1} \in B$ and $r + \boldsymbol{v_2} \in B$ respectively. Since B is a digital set, r, $r + \boldsymbol{v_1}$ and $r + \boldsymbol{v_2}$ are points of $\mathbb{Z}^2$. The values β_1, β_2, $\beta_2 + \alpha_2$ can be written as functions of r, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$. Combining the constraints stated above with a study of the variations of these functions to derive necessary conditions on r, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ is quite tedious. Experimentally, it is however pretty straightforward to find that the triplet of points defined by $r(5, 12)$, $\boldsymbol{v_1} = (-4, 1)$ and $\boldsymbol{v_2} = (2, -1)$ fulfills the constraints. This gives an upper bound on the minimal distance ($d(p, r) = 13$) for which there exists a point fulfilling the constraints. We verify that this is also a lower bound by exhaustively checking the constraints for all digital points at distance less than 13[1].

Secondly, visible projections of p's neighbours may not be visible from p, and p's projection may be somewhere else. In the next paragraph, we propose

[1] Python code to make this test is available at `www.gipsa-lab.grenoble-inp.fr/~isabelle.sivignon/Code`

an improved version of the function addCandidatePoint, using the parts in orange, in order to take into account such cases.

Algorithm using tangents If, for a given neighbour q of p, $\pi^A(q)$ is not visible from p, there is a connected component C of A^C between p and $\pi^A(q)$. The idea is to compute a tangent to C passing through p. Such a construction echoes the computation of visibility regions in polygons [16]. The returned point, if there is one, is the intersection between this tangent and B. Many tangent estimators for digital curves have been proposed in the literature [26, 22], but the problem we are facing here is slightly different. Indeed, we need to compute the tangent to an obstacle through a given point (not on the obstacle's boundary) with no apriori information on the obstacle. In order to solve this problem, we present here a fast heuristic. Figure 2(b) illustrates how computeProjTangent is implemented. Let us suppose that a DSS $S(q) = \{s_1 = q \dots s_k = \pi^A(q)\}$ between q and $\pi^A(q)$ is known (in blue on Figure 2(b) and see Section 3.3 for a discussion). Since $\pi^A(q)$ is not visible from p, $p \cup S(q)$ is not a DSS. Let $j \in [2, k-1]$ be the index such that $p \cup \{s_1, \dots s_j\}$ is a DSS while $p \cup \{s_1, \dots s_{j+1}\}$ is not. Part $\{s_1, \dots s_j\}$ is depicted as blue and green dots in Figure 2(b). The set $\mathcal{L}$ of all digital straight lines that contain the DSS $p \cup \{s_1, \dots s_j\}$ defines a family of lines close to C through p. In Figure 2(b), the green region represents the parameters of this set of digital straight lines. Thus, there is not one unique intersection point between B and a line of $\mathcal{L}$, but a set of points $P = \bigcup_{l \in \mathcal{L}, l \cap A^c = \emptyset} l \cap B$ (green points overlayed by black crosses in Figure 2(b)). Computing P to find the point of P closest to p may be computationally expensive in general. If only one intersection point between B and a specific digital straight line $l \in \mathcal{L}$, $l \cap A^C = \emptyset$ is computed, then committed error (in comparison with the exhaustive computation of P) is upper bounded by $\max_{r,s \in P} |d(p,r) - d(p,s)|$.

Computational Complexity

Proposition 4. *If A is a digital set of n points, computational complexity of Algorithm 2 is $O(n.|\mathcal{N}|.(|\mathcal{N}|.V + \log(|\Gamma|)))$ when Algorithm 3 is called without the tangent computation. $|\Gamma|$ is the maximal size of the list of candidates, and V is the computational complexity of the predicate* Visible.

Complexity of Algorithm 2 is $O(n.|\mathcal{N}|.(|\mathcal{N}|.(V + T) + \log(|\Gamma|)))$ when Algorithm 3 with tangent computation is used instead. T is the computational complexity of the function computeProjTangent.

Proof. The complexity analysis is pretty straightforward. The $\log(|\Gamma|)$ term comes from the insertion of a new candidate in Γ line 7 of Algorithm 3. This complexity is achieved by using an ordered set, where the elements $\{p, \pi^A(p)\}$ are ordered with respect to the value of $d(p, \pi^A(p))$.

3.3 Visibility test

The Visible predicate is called many times in the algorithm, so that its implementation is key to the algorithm efficiency. We propose below a visibility

test in three steps. Each step consists in checking sufficient visibility conditions of increasing precision and complexity: VISIBLE(A,p,q)=(VISIBLEADD(A,p,q) or VISIBLEUPLOW(A,p,q) or VISIBLEPREIMAGE(A,p,q)). Note that the *or* short-circuiting ensures that the least computationally expensive tests handle most common cases.

VISIBLEADD For each accepted point $q \in \Lambda$, we also store a DSS $S(q) \subseteq A$ such that $q, \pi^A(q) \in S(q)$. Maintaining and propagating this information is actually very easy using a classical incremental DSS recognition algorithm [13] and the fact that Algorithm 2 computes the projection of a point from the projections of its neighbours. The VISIBLEADD function returns true if $p \cup S(q)$ is still a DSS, false otherwise. If true, we set $S(p) = p \cup S(q)$ and store it with p and $\pi^A(q)$ in the list of candidates Γ. This test runs in constant time.

VISIBLEUPLOW This predicates verifies whether the DSS having p and q as upper (resp. lower) leaning points is included in A. This test runs in $O(\max(abs(x_p - x_q), abs(y_p - y_q)))$, which is upper bounded by the maximal distance between two points of A.

VISIBLEPREIMAGE The last test finally enables to thoroughly check whether there exists a DSS included in A among all possible DSSs between p and q. This problem was tackled in [6] and the solution proposed can be implemented using the linear-in-time stabbing lines algorithm of [20] for which an implementation is available in DGtal [1]. The general idea of the algorithm is the following : given a set of digital points that must belong to the DSS, and another set of digital points that must not, saying that a given digital point must or must not belong to a DSS is translated into a couple of constraints on the DSS parameters. When all the points have been added, it is enough to check whether the set of possible DSS parameters is empty or not. In our test, the only two points we want in the DSS are p and q. The points that must not belong to the DSS are obstacles points, *i.e.* points of A^C. However, in general, few points of A^C actually interfere with a DSS between p and q, and such points can be detected while tracking the upper and lower DSS in function VISIBLEUPLOW.

3.4 Experimental comparison of brute force and FMM-like algorithms

Algorithm 1 and Algorithm 2 were implemented using DGtal library [1]. Code is available at `www.gipsa-lab.grenoble-inp.fr/~isabelle.sivignon/Code`. Figure 3 gives elements of comparison in terms of computational speed and accuracy. The result of Algorithm 1 is considered as the ground truth since an exhaustive search of visible projections is performed when necessary. We test our algorithms on three different sets A (one obstacle in (d), two large obstacles in (e), many obstacles in (f)), with the same set B. Figures 3(a-c) show experimental computation times in the three cases, at different resolutions and for each algorithm. Unsurprisingly, Algorithm 1 is much faster when $\pi(p) =$

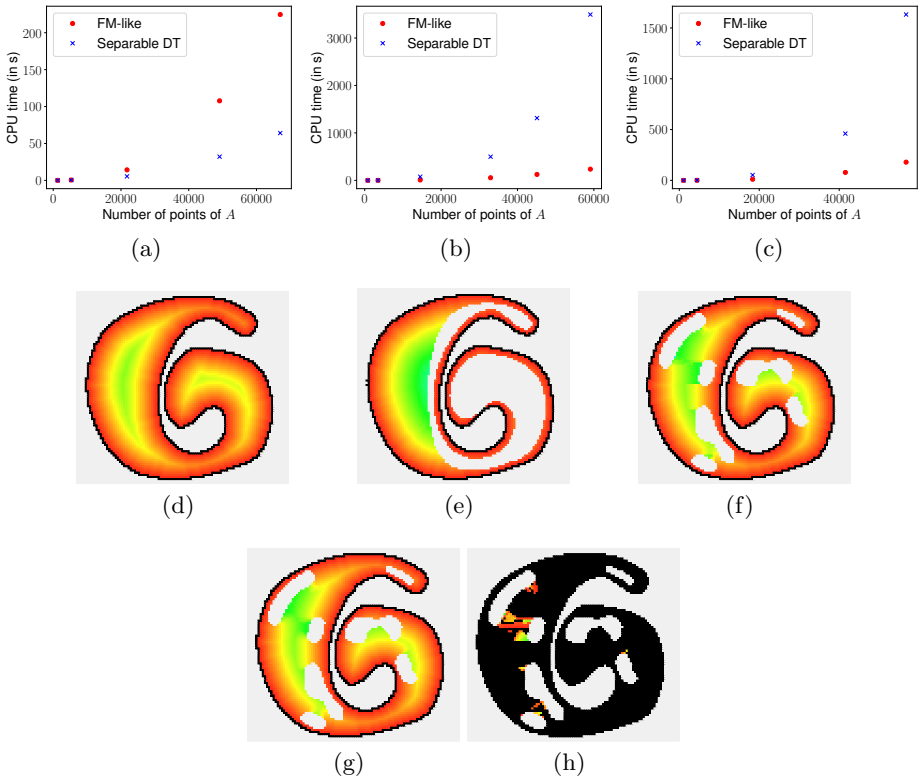
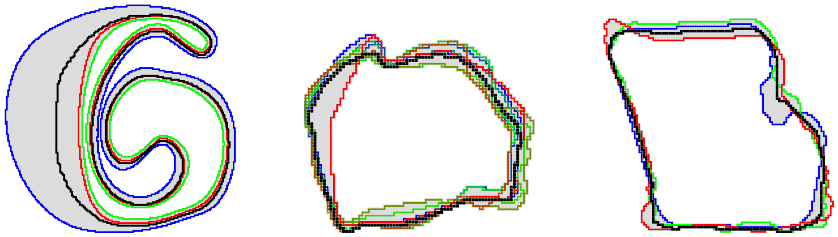


Fig. 3. In (d-g), A^C is depicted in light grey, B in black. Figures (a) to (c) report computation times when running Algorithms 1 and 2 on input data depicted in (d) to (f) respectively. In (d-g) a color map from red (low values) to green (large values) is used to represent the distance between a point and its visible projection computed thanks to Algorithm 2, except for (g) where Algorithm 1 is used. In (h), the same color map is used to depict the difference between (f) and (g).

$\pi^A(p)$ for all p as in case (d). However, when visibility issues arise for many points p, the trend is reversed. Figures (d-h) provide elements to evaluate the quality of the approximation computed by Algorithm 2. The results of this algorithm are depicted in (d) and (e): we do not show the results of Algorithm 1 since it is the same apart for very few points where the error is very small (equal to 1 on these examples, see Proposition 3). However, the results tend to differ more for the shape with many small obstacles (see (f), (g) and (h)), especially in parts where the heuristic tangent estimator was called.



(a) (b) (c)

Fig. 4. Results of arithmetic mean digital curve computation : input curves are in colors, the result mean curve is in black, and the gap in gray: (a) three curves issued from [23]; in (b) and (c), input curves are the results of a morphological snake image segmentation algorithm [2] with different parameters - eight curves in (b), three in (c).

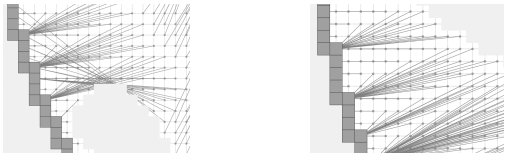
4 Computation of the Arithmetic Mean Digital Curve

Coming back to the initial problem, recall that the goal is the following: given a set of digital curves $\{C_i\}$, (i) compute the height map $H : \Delta(\{C_i\}) \to \mathbb{R}$, $p \mapsto \sum_i d(p, \pi_i^{\Delta}(p))^2$ (ii) compute the valley of the height field H. Section 3 was dedicated to the design of an algorithm to solve step (i). The concepts of ridges and valleys involved in step (ii) have been widely studied in computational imaging, with applications in image segmentation or shape recognition (see [14] for an overview). As stated in [21], there are two main approaches to define these concepts. The first one is to define ridges as solution manifolds of algebraic equations using the height field and its derivatives (see [14, 21]). The second approach is the well-known concept of watershed, for which many efficient algorithms have been designed for the last 30 years [25, 9, 11]. Experimental results presented in Figure 4 use the implementation of [25] in the ImageJ open source software [18], which is widely used in biomedical image analysis for instance.

5 About compatibility conditions of input curves

In this last part, we give tentative elements about conditions input curves must fulfill for this approach to be correct, and leave some open questions.

Conditions of [23] To define compatibility conditions of input curves, the authors of [23] define *spikes* for each point $p \in \mathscr{C}_i$: if l is the line l normal to $\mathscr{C}_i$ at p, the spike of $\mathscr{C}_i$ at p is the segment of $l \cap \Delta$ containing p. The assumption they make on the set of input curves is that for any curve $\mathscr{C}_i$, the spikes of $\mathscr{C}_i$ are pairwise distinct and their union is the gap. Under this assumption, they conjecture that the height map f has exactly one local minimum on each spike, for any curve $\mathscr{C}_i$. They also provide a sufficient condition to ensure that the assumption is true: all input curves should be pairwise normal-compatible [15] (meaning that the closest point maps between any pair of curves must be an homeomorphism).



(a) (b)

Fig. 5. C_i is in dark grey, Δ^C in light grey and arrows are depicted between points $p \in \Delta$ and $\pi_i^\Delta(p)$. (a) Intersecting spikes, (b) no spikes intersect.

Finally, they also enounce a necessary condition (called gap compatibility), used to fastly detect families of curves for which the approach fails: for any point $p \in \Delta$ and any curve $\mathscr{C}_i$, $\pi_i^\Delta(p)$ must be unique. This can be rewritten as: if $p \in \Delta$ belongs to the medial axis of $\mathscr{S}_i$, then only one of its closest points is visible in Δ.

Which conditions for a set of digital curves ? The properties and conditions above do not translate straightforwardly for digital curves. First, the gap compatibility condition is not relevant in our context since many points that do not belong to the medial axis (which is the set of maximal balls) may have several closest points. We propose to rewrite this condition as follows: if for a point p, $\pi_i^\Delta(p)$ is not unique, say $\pi_i^\Delta(p) = \{s_j\}$, if (i) the digital ball $b(p, d(p, s_j))$ is not maximal in O_i and (ii) let s_1 and s_k be the projections such that $C_i(s_0..s_k)$ (the part of C_i between s_0 and s_k) is of minimal length and for all j, $s_j \in C_i(s_0..s_k)$; then any point in $C_i(s_0..s_k)$ is visible from p. Verifying that this condition is fulfilled requires to compute all closest points instead of one, and a solution was proposed in [10]. Last, the definition of spike is key to ensure that the minima of the height field define a curve. Again, the definition does not extend to digital curves. The spike of a point $p \in \mathscr{C}_i$ is actually the Voronoi Cell of p in Δ. Thus we propose to define the spike of a point $p \in C_i$ as $\text{spike}(p, C_i) = \text{ConvexHull}(VoronoiCell(p) \cap \Delta)$. The assumption on spikes translates as follows: for any point x in the interior of $\text{spike}(p, C_i)$, $p = \pi_i^\Delta(x)$. This assumption is violated is many cases in the example of the shape with many holes of Figure 3(f), while it is always fulfilled for shape (e). Figure 5 shows a close-up on spikes in that two cases. We end up with an open question: what are the properties of the height field minima with respect to spikes when they are pairwise distinct and their union is the gap ? May we prove that minima define a curve ?

References

1. DGtal: Digital Geometry Tools and Algorithms Library. `http://dgtal.org`
2. Alvarez, L., Baumela, L., Márquez-Neila, P., Henríquez, P.: A Real Time Morphological Snakes Algorithm. Image Processing On Line 2, 1–7 (2012)
3. Boukhriss, I., Miguet, S., Tougne, L.: Discrete average of two-dimensional shapes. In: Springer (ed.) CAIP. vol. LNCS 3691, pp. 145–152 (2005)

4. Breen, D.E., Mauch, S., Whitaker, R.T.: 3d scan conversion of csg models into distance volumes. In: Proceedings of the 1998 IEEE Symposium on Volume Visualization. pp. 7–14. ACM (1998)
5. Buchin, K., Buchin, M., van Kreveld, M., Löffler, M., Silveira, R.I., Wenk, C., Wiratma, L.: Median trajectories. Algorithmica 66(3), 595–614 (2013)
6. Coeurjolly, D., Miguet, S., Tougne, L.: 2d and 3d visibility in discrete geometry: an application to discrete geodesic paths. Patt. Recog. Lett. 25(5), 561–570 (2004)
7. Coeurjolly, D.: 2d subquadratic separable distance transformation for path-based norms. In: Discrete Geometry for Computer Imagery, DGCI. Lecture Notes in Computer Science, vol. 8668, pp. 75–87. Springer (2014)
8. Coeurjolly, D., Montanvert, A.: Optimal separable algorithms to compute the reverse euclidean distance transformation and discrete medial axis in arbitrary dimension. IEEE Trans. Pattern Anal. Mach. Intell. 29(3), 437–448 (2007)
9. Couprie, M., Bertrand, G.: Topological gray-scale watershed transformation (1997)
10. Couprie, M., Coeurjolly, D., Zrour, R.: Discrete bisector function and euclidean skeleton in 2d and 3d. Image and Vision Computing 25(10), 1543 – 1556 (2007)
11. Couprie, M., Najman, L., Bertrand, G.: Quasi-linear algorithms for the topological watershed. Journal of Mathematical Imaging and Vision 22(2), 231–249 (2005)
12. Danielsson, P.E.: Euclidean distance mapping. Computer Graphics and Image Processing 14, 227–248 (1980)
13. Debled-Rennesson, I., reveillès, J.P.: A linear algorithm for segmentation of digital curves. Int. Journal of Pattern Recog. and Art. Intell. 09(04), 635–662 (1995)
14. Eberly, D.: Ridges in image and data analysis. Springer (1996)
15. F. Chazal, A.L., Rossignac, J.: Normal-map between normal-compatible manifolds. Int. Journal of Computational Geometry and Applications 17(5), 403–421 (2007)
16. Goodman, J.E., O'Rourke, J. (eds.): Handbook of Discrete and Computational Geometry. CRC Press, Inc. (1997)
17. Jones, M.W., Baerentzen, J.A., Sramek, M.: 3d distance fields: a survey of techniques and applications. IEEE Trans. Vis. and Comp. Graph. 12(4), 581–599 (2006)
18. Legland, D., Arganda-Carreras, I.: ImageJ-MorphoLibJ. `imagej.net/MorphoLibJ`
19. Nutanong, S., Tanin, E., Zhang, R.: Incremental evaluation of visible nearest neighbor queries. IEEE Trans. on Knowledge and Data Eng. 22(5), 665–681 (2010)
20. O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. Commun. ACM 24(9), 574–578 (Sep 1981)
21. Peikert, R., Sadlo, F.: Height ridge computation and filtering for visualization. In: 2008 IEEE Pacific Visualization Symposium. pp. 119–126 (2008)
22. Prasad, D.K., Leung, M.K.H., Quek, C., Brown, M.S.: Deb: Definite error bounded tangent estimator for digital curves. IEEE Transactions on Image Processing 23(10), 4297–4310 (2014)
23. Sati, M., Rossignac, J., Seidel, R., Wyvill, B., Musuvathy, S.: Average curve of n smooth planar curves. Computer-Aided Design 70, 46 – 55 (2016)
24. 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)
25. Soille, P., Vincent, L.: Determining watersheds in digital pictures via flooding simulations (1990)
26. de Vieilleville, F., Lachaud, J.O.: Comparison and improvement of tangent estimators on digital curves. Pattern Recognition 42(8), 1693 – 1707 (2009), advances in combinatorial image analysis