Published in Image Processing On Line on 2014–05–22.
Submitted on 2013–02–12, accepted on 2013–05–02.
ISSN 2105–1232 © 2014 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials, software, datasets and online demo at
http://dx.doi.org/10.5201/ipol.2014.70

A Near-Linear Time Guaranteed Algorithm for Digital Curve Simplification under the Fréchet Distance

Isabelle Sivignon
GIPSA-lab
http://www.gipsa-lab.inpg.fr/~isabelle.sivignon
isabelle.sivignon@gipsa-lab.grenoble-inp.fr

Abstract

In this paper, we propose an algorithm that, from a maximum error and a digital curve (4- or 8-connected), computes a simplification of the curve (a polygonal curve) such that the Fréchet distance between the original and the simplified curve is less than the error. The Fréchet distance is known to nicely measure the similarity between two curves. The algorithm we propose uses an approximation of the Fréchet distance, but a guarantee over the quality of the simplification is proved. Moreover, even if the theoretical complexity of the algorithm is in O(n log(n)), experiments show a linear behavior in practice.

Source Code

The source code of the algorithm and an online demonstration are accessible at the IPOL web page of this article[1].

Keywords: digital curve simplification; Fréchet distance; approximation

1 Overview

Given a polygonal curve, the curve simplification problem consists in computing another polygonal curve that (i) approximates the original curve, (ii) satisfies a given error criterion, (iii) with as few vertices as possible. This problem arises in a wide range of applications, such as geographic information systems (GIS), computer graphics or computer vision, where the management of the level of details is of crucial importance to save memory space or to speed-up analysis algorithms.

Given a 4- or 8-connected digital curve and a maximum error, we propose an algorithm that computes a simplification of the curve (a polygonal curve) such that the Fréchet distance between the original and the simplified curve is less than the error. The Fréchet distance is known to nicely measure the similarity between two curves. It can be intuitively defined considering a man walking his dog. Each protagonist walks along a path, and controls its speed independently, but cannot go backwards. The Fréchet distance between the two paths is the minimal length of the leash required.

The algorithm we propose (see [4] for the related publication) uses an approximation of the Fréchet distance, but a guarantee over the quality of the simplification is proved. Moreover, even if the theoretical complexity of the algorithm is in $\mathcal{O}(n \log(n))$, experiments show a linear behavior in practice.

[1]http://dx.doi.org/10.5201/ipol.2014.70

2 Algorithm

2.1 Prerequisite

2.1.1 Fréchet Distance

Given two curves f and g specified by functions $f : [0,1] \to \mathbb{R}^2$ and $g : [0,1] \to \mathbb{R}^2$, and two non-decreasing continuous functions $\alpha : [0,1] \to [0,1]$ and $\beta : [0,1] \to [0,1]$ with $\alpha(0) = 0, \alpha(1) = 1, \beta(0) = 0, \beta(1) = 1$, the **Fréchet distance** $\delta_F(f,g)$ between two curves f and g is defined as

$$\delta_F(f,g) = \inf_{\alpha,\beta} \max_{0 \le t \le 1} d(f(\alpha(t)), g(\beta(t))).$$

As illustrated in Figure 1, contrary to the Hausdorff distance denoted by $\delta_H(f,g)$, the Fréchet distance takes into account the course of the curves.

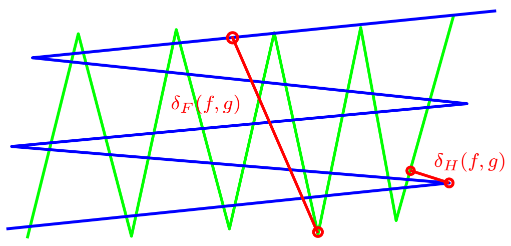


Figure 1: Difference between the Fréchet distance and the Hausdorff distance: red segments indicate the pair of points minimizing either the Fréchet or the Hausdorff distance.

2.1.2 Curve Simplification Problem

Given a polygonal curve $P = \langle p_1, \dots p_n \rangle$, a curve $P' = \langle p_{i_1}, \dots p_{i_k} \rangle$ with $1 = i_1 < \dots < i_k = n$ is said to **simplify** the curve P (see Figure 2). $P(i,j)$ denotes the sub-path from p_i to p_j. Given a pair of indices $1 \le i \le j \le n$, $\delta_F(p_ip_j, P)$ denotes the Fréchet distance between the segment p_ip_j and the part $P(i,j)$ of the curve. For the sake of clarity, the simplified notation $error(i,j) = \delta_F(p_ip_j, P)$ will sometimes be used. We also say that p_ip_j is a **shortcut**. In other words, the vertices of P' form a subset of the vertices of P, and the computation of P' comes down to the selection of "shortcuts" p_ip_j.

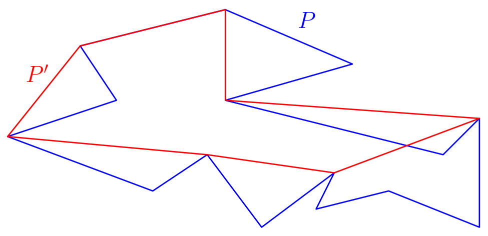


Figure 2: The red curve P' is a simplification of the blue curve P.

All in all, to **find** P' **an** ε**-simplification of** P we have to:

1. **find shortcuts** p_ip_j **such that** $error(i,j) = \delta_F(p_ip_j, P) \le \varepsilon$;

2. **minimize the number of vertices of P'.**

The following nice local property of the Fréchet distance proved in [2] will be very useful to prove a guarantee on the quality of the result produced by our algorithm (see Figure 3): let $P = \{p_1, p_2, \dots, p_n\}$ be a polygonal curve. For all $i, j, l, r, 1 \leq i \leq l \leq r \leq j \leq n$, $error(l, r) \leq 2 \times error(i, j)$. In other words, the shortcuts of any $\frac{\varepsilon}{2}$-simplification cannot strictly enclose the shortcuts of an ε-simplification.

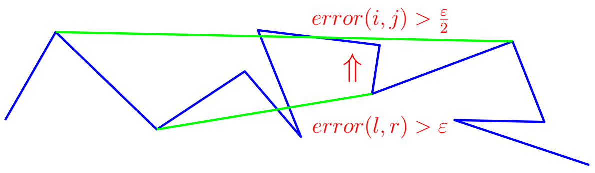

$error(i,j) > \frac{\varepsilon}{2}$
$\Uparrow$
$error(l,r) > \varepsilon$

Figure 3: Illustration of the local property.

2.2 Guaranteed Algorithm Using an Approximated Distance

2.2.1 Definitions and Overall Algorithm

Using the exact Fréchet distance appears to be too expensive to design an efficient algorithm. Instead, we use an approximation of the Fréchet distance proposed in [1]. More precisely, the authors of [1] show that $error(i, j)$ can be upper and lower bounded by functions of two values, namely $\omega(i, j)$ and $b(i, j)$. $\omega(i, j)$ is the width of the points of $P(i, j)$ in the direction $\overrightarrow{p_i p_j}$. $b(i, j)$ is the length of the longest backpath in the direction $\overrightarrow{p_i p_j}$ (see Figure 4). We have the following property [1], which leads to Algorithm 1. The Fréchet error of a shortcut $p_i p_j$ satisfies:

$$\max\left(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}\right) \leq error(i,j) \leq 2\sqrt{2}\max\left(\frac{w(i,j)}{2}, \frac{b(i,j)}{2}\right).$$

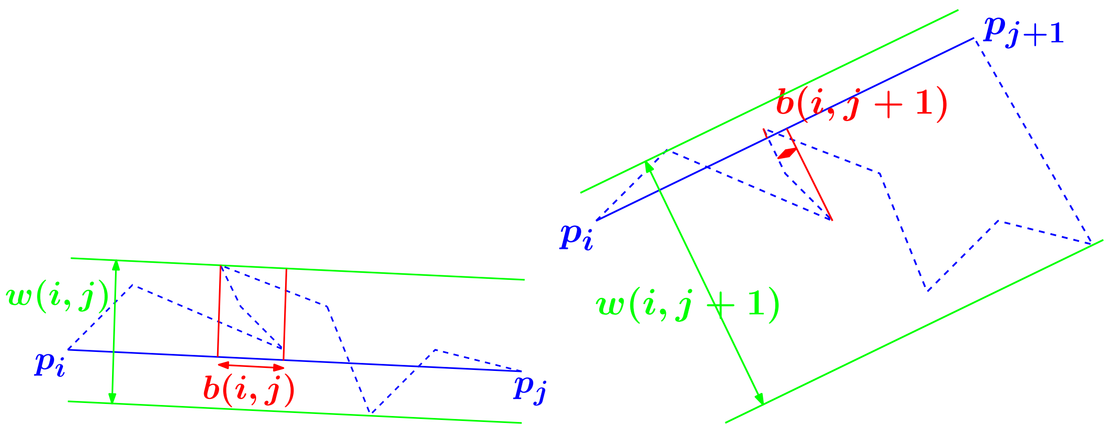


Figure 4: (a) Illustration of the definition of the width and the backpath length. (b) When a new point is considered, the width and backpath lengths may totally change.

Algorithm 1: Greedy Fréchet simplification algorithm

```
1 $i = 1$, $j = 2$
2 while $i < n$ do
3     while $j < n$ and $\max(w(i,j), b(i,j)) \leq \frac{\varepsilon}{\sqrt{2}}$ do
4         j=j+1
5     end
6     create a new shortcut $p_i p_{j-1}$
7     $i = j - 1$,$j = i + 1$
8 end
```

2.2.2 Updating the Approximated Distance Efficiently

The difficulty of Algorithm 1 lies in the update of the values of $\omega(i,j)$ and $b(i,j)$ when a new point is taken into account as illustrated in Figure 4 (b).

Updating $\omega(i,j)$ Instead of updating $\omega(i,j)$, it is enough to consider the maximal distance between any point of $P(i,j)$ and the vector $\overrightarrow{p_ip_j}$. This is done using the algorithm of Chan and Chin [3] illustrated in Figure 5. Given an origin point p_i and a set of points $P(i,j)$ we construct in constant time the set S_{ij} of straight lines l going through p_i such that $\max_{p \in P(i,j)} d(p,l) \leq r$. As a result, deciding whether $d_{max}(i,j)$ is lower than r or not is equivalent to checking whether the straight line (p_i, p_j) belongs to S_{ij} or not, which is also done in constant time.

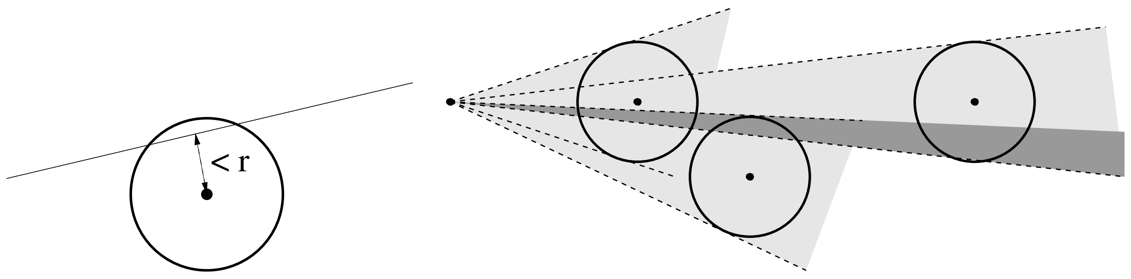


Figure 5: The cone (dark gray) is computed incrementally as the intersection of the light gray cones.

Updating $b(i,j)$ This update is trickier. When a new point p_{j+1} is considered, we want to check if there exists a backpath longer than a threshold in the direction $\overrightarrow{p_ip_j}$. Let us first give some definitions.

Definition 1 *Let l be a straight line of directional vector $\overrightarrow{d}$. α is the angle between l and the abscissa axis. We denote by $proj_\alpha(p)$ the orthogonal projection of p onto the line of angle α. If $\overrightarrow{p_lp_m} . \overrightarrow{d} < 0$, then $proj_\alpha(p_l)$ is "after" $proj_\alpha(p_m)$ in direction α, and we write $proj_\alpha(p_l) >> proj_\alpha(p_m)$.*

Then we can define some points named **occulters** (see Figure 6) which are the furthest points for a given direction.

Definition 2 *An* occulter *for the direction $\overrightarrow{d}$ is a point p_k such that for all $l < k$, $proj_\alpha(p_k) >> proj_\alpha(p_l)$. Moreover, an occulter is said to be* active *if there is no occulter $p_{k'}$ with $k' > k$.*

We can easily prove (see [4]) that the origin of the longest backpath is an occulter. Considering whether the last movement $\overrightarrow{p_jp_{j+1}}$ is forward or backward in the direction $\overrightarrow{p_ip_j}$, we can decide if there

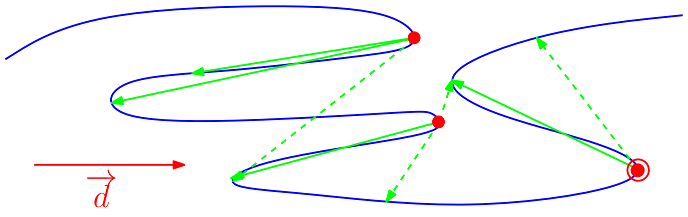


Figure 6: Occulters for the direction d in red. Green arrows represent backpaths: the length of the plain arrows is to be checked, whereas we know that the backpaths represented by dashed arrows are not the longest ones.

Algorithm 2: Active occulter update and backpath computation for a direction $\overrightarrow{d}$

```
1 Let $occ_{max}$ be the active occulter for $P(i,k)$ in direction $\overrightarrow{d}$.
2 if $\overrightarrow{p_kp_{k+1}}$ is negative then
3     if $\overrightarrow{p_{k-1}p_k}$ is positive then
4         if $proj_\alpha(p_k) >> proj_\alpha(occ_{max})$ then
5             $occ_{max} = p_k$
6         end
7     end
8     The vector $\overrightarrow{occ_{max}p_{k+1}}$ may be a backpath.
9 end
```

is a new backpath possibly longer than the threshold or not. This is done according to Algorithm 2 below.

According to Algorithm 1 we see that Algorithm 2 must be applied for any possible direction for a given curve P, which is computationally expensive. However, the computation of backpaths can be mutualized in the case of digital curves. Indeed for a digital curve, the set of elementary shifts $\overrightarrow{p_jp_{j+1}}$ is well defined and it is actually possible to cluster the set of directions such that given an elementary shift e, this shift is either forward (positive) or backward (negative) for all the directions of the cluster.

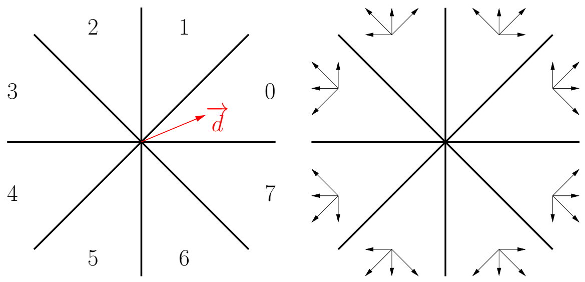


Figure 7: Left: the directions of the plane are clustered into 8 octants. For instance, direction d is in octant 0. Right: illustration of the positive (or forward) elementary shifts for each octant.

If we now go back to Algorithm 2, we see that the result of the test lines 2-3 is the same for all the directions of a given octant. This test can thus be done jointly for all the directions of an octant. Nevertheless, to determine if a new point p_k is the new active occulter (the furthest point for the direction studied), the projection of the current active occulter and the point p_k on a direction are compared: the furthest of the two points is the active occulter. Therefore, for any two points p_k and q the result of this comparison is not the same for all the directions of a given octant. This fact is illustrated in Figures 7 and 8 for the octant 0:

- for any point q in the gray area, and for any direction in octant 0, q is after p;
- for any point q in the dashed area, and for any direction in octant 0, p is after q;
- in the white area, the order changes, as illustrated in Figure 8(a) and (b).

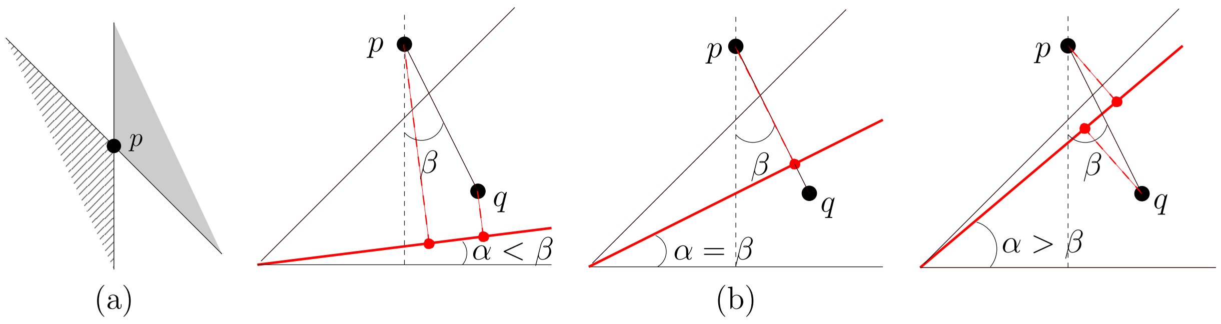


Figure 8: (a) For two points p and q and a direction α, the order of their projections changes when q is in the white area: (b) on the left, q is *after* p, and is the active occulter for direction α, whereas on the right, p is after q and is the active occulter.

Algorithm 3 puts together these observations to update the list of active occulters for one octant.

Memorizing the directions for which there exists a too long backpath From Algorithm 1, we see that the length of the longest backpath is tested for each new point, which defines a new direction. Moreover, we see from Algorithm 2 line 8 that for each negative shift, we can have as many backpaths as active occulters. All in all, testing individually all the possible backpaths when a new point is added is too expensive. To solve this problem, we propose to maintain a "set" of the directions for which there exists a backpath of length greater than the error ε. This set actually consists in a list of intervals: for a given backpath of length $\mathbf{l}$ and a given error ε, the interval of directions for which the projection of the backpath is longer than ε is computed easily. The union of all these intervals is stored.

3 Quality of the Result and Complexity Analysis

3.1 A Guaranteed Algorithm

An important issue when designing an algorithm that is known not to be optimal is to prove that the result of this algorithm is not so far from the optimal. In this work, the optimal solution is to compute the ε-simplification of a digital curve $\mathbf{P}$ according to the Fréchet distance with a minimum number of vertices. The algorithm we propose here is not optimal for two reasons:

- it is greedy: the simplification is computed from a given starting point, in a given scanning order.
- the distance used is an approximation of the Fréchet distance.

However, we prove that the number of vertices of the simplified curve computed by our algorithm is upper bounded by a function of the optimal solution:

Lemma 1 *Algorithm 1 computes an ε-simplification P' of a polygonal curve P such that $|P'|$ is lower than the number of vertices of an optimal $\frac{\varepsilon}{4\sqrt{2}}$-simplification of P.*

For more details about this proof, please refer to [4].

Algorithm 3: Update of the list of active occulters for the octant 0

```
Let $p$ be the last point added, we want to check if $p$ is an active occulter.
forall the active occulters $p_i(\alpha_{i_{min}}, \alpha_{i_{max}})$ do
    $v = \overrightarrow{p_i p}$
    if $\overrightarrow{v}.(1,0) < 0$ and $\overrightarrow{v}.(1,1) < 0$ then
        p is not an active occulter
    end
    if $\overrightarrow{v}.(1,0) > 0$ and $\overrightarrow{v}.(1,1) > 0$ then
        $p_i$ is not an active occulter anymore
        $p$ is an active occulter on $[0, ?]$ with $\alpha_{i_{min}} <? \leq \frac{\pi}{4}$
    end
    if $\overrightarrow{v}.(1,0) > 0$ and $\overrightarrow{v}.(1,1) < 0$ then
        compute the angle $\beta$ ;                                /* see Figure 8 */
        if $\alpha_{i_{min}} \leq \beta < \alpha_{i_{max}}$ then
            $p$ is an active occulter on $[0, \beta]$
            $p_i$ is an active occulter on $[\beta, \alpha_{i_{max}}]$
        end
        if $\beta < \alpha_{i_{min}}$ then
            $p_i$ is still an active occulter
        end
        if $\beta \geq \alpha_{i_{max}}$ then
            $p_i$ is not an active occulter anymore
            $p$ is an active occulter on $[0, ?]$ with $\alpha_{i_{max}} \leq? \leq \frac{\pi}{4}$
        end
    end
    +similar process for symmetrical cases (roles of $p_i$ and $p$ inverted)
end
```

3.2 Complexity

The theoretical complexity of this algorithm is $\mathcal{O}(n \log(n_i))$, for a digital curve of n points. n_i is the number of intervals used to store the directions for which there exists a backpath of length greater than the error. n_i is upper bounded by n. Nevertheless, experiments on noisy shapes show that the general behavior of the algorithm in linear in time.

4 Source Code

4.1 Download and Installation

A C++ implementation is provided. It is part of the open source library DGtal[2] (Digital Geometry Tools and Algorithms). It should compile on any linux, Windows or MacOS system. The installation is done through cmake[3] (version >= 2.8).

1. Download the source code on theIPOL web page of this article[4]. Untar and unzip the archive.

[2] http://libdgtal.org
[3] http://www.cmake.org/
[4] http://dx.doi.org/10.5201/ipol.2014.70

2. create a `build` directory;

3. in this directory, run `cmake ..` Some directories are created.

4. in the `demoIPOL_FrechetSimplification` directory, run `make frechetSimplification`.

4.2 Usage

The executable file is named `frechetSimplification`.

Input A 2D digital curve (4 or 8 connected), given as a list of points:

```
x0 y0 #coordinates of the first point
x1 y1 #coordinates of the second point
...
xn yn
```

Such digital curves are obtained after a contour extraction from a binary image. Note that the input file should contain one contour only. Syntax for running the simplification algorithm on a `contour.sdp` file is:

```
./frechetSimplification -sdp contour.sdp
```

Options To get the list of options, type

```
./frechetSimplification
```

The main parameter is the error value used for the simplification. It is given with the option `-error` (default value is 2). Two simplifications are available:

- the first one is the simplification according to the Fréchet distance as described above;

- the other one is a simplification where only the width of the shortcut is taken into account: in Algorithm 1, the test on line 3 is replaced by $w(i,j) \leq \frac{\varepsilon}{\sqrt{2}}$.

Default is the computation of the Fréchet simplification. If the width simplification is preferred, the `-w` option should be used.

Output The output consists in:

- the display of the number of points of the curve, the error given as input, the number of vertices of the simplified polygon and the cpu time (in ms) for the simplification performed;

- the list of points of the simplified polygon in a file named `output.vertices`;

- an eps file named `output.eps` with both the original digital curve and the simplified curve. Other output formats are available using DGtal (svg, fig, tikz for instance). See the DGtal documentation for more information.

5 Implementation Details

Here are some useful hints about the implementation choices. Some examples can be consulted in the tests (testFrechetShortcut.cpp) and examples (exampleFrechetShortcut.cpp) directories. The FrechetShortcut class (FrechetShortcut.h and FrechetShortcut.ih files) contains all the elements used for the simplification computation. It is a templated class with two parameters: the first one is an iterator on the list of points to process; the second one is the type of integer used for the computations. Here is a very simple example of use of this class:

```
typedef PointVector<2,int> Point;
typedef std::vector<Point>::iterator Iterator; // the curve is stored as↩
    a 2D vector of integer points
typedef FrechetShortcut<Iterator,int> Shortcut;

std::vector<Point> contour;
contour.push_back(Point(0,0)); // add points
contour.push_back(Point(1,0));
...

Shortcut s(5); // define a shortcut with an error of 5
s.init(contour.begin()); // initialize the shortcut with the first point↩
    of the contour

while ( (s.end() != contour.end()) && (s.extendForward()) ) {} // add ↩
   points to the shortcut while possible
```

This setting is similar to other classes of the Geometry Package of the DGtal library. Consequently, the FrechetShortcut can be used to compute greedy segmentations of a contour using the GreedySegmentation class. Here is an example:

```
typedef Curve::PointsRange::ConstIterator Iterator; // use the DGtal ↩
    Curve to store the input contour
typedef FrechetShortcut<Iterator,int> SegmentComputer;

Curve aCurve;
aCurve.initFromVector(contour); // the contour is the same as before

typedef Curve::PointsRange Range;
Range r = aCurve.getPointsRange();

typedef GreedySegmentation<SegmentComputer> Segmentation; // define a ↩
    segmentation
Segmentation theSegmentation( r.begin(), r.end(), SegmentComputer(↩
    error) ); // compute the greedy segmentation between the points ↩
    defined by r.begin() and r.end()
```

More precisely, the FrechetShortcut class includes two nested classes called Backpath and Cone. The first one is used to check the backpath lengths, while the second one is used to check the width.

The extendForward() method tests is the next point on the curve can be added to the current shortcut. Its implementation follows the inner while loop of Algorithm 1 and mainly consists in calling the functions updateBackPath() and updateWidth().

The updateWidth() function updates the current cone according to the new point (see description of the general algorithm above). It returns true if the shortcut defined by the first point of the curve and the next point still complies with the error parameter given.

For a given elementary shift, the `updateBackPath()` method updates the backpath structure for the eight octants. This is done through a call to `updateBackPathFirstQuad()` after the appropriate rotation of the elementary shift. This function implements Algorithm 2, with a call to `addPositivePoint()` in the case of a positive shift, or `addNegativePoint()` otherwise. In the latter case, the `backPath` structure is updated with a call to `updateOcculters()`: this method implements Algorithm 3. Last, the list of directions for which there exists a backpath of length greater than the error is updated in `updateIntervals()`.

6 Examples

```
        ./frechetSimplification -error 5 -sdp Data/Plant046.sdp
```

The result of this command is (number of points of input curve, error, number of points of simplified curve, cpu time):

```
    2402              5                   76           0.025
```

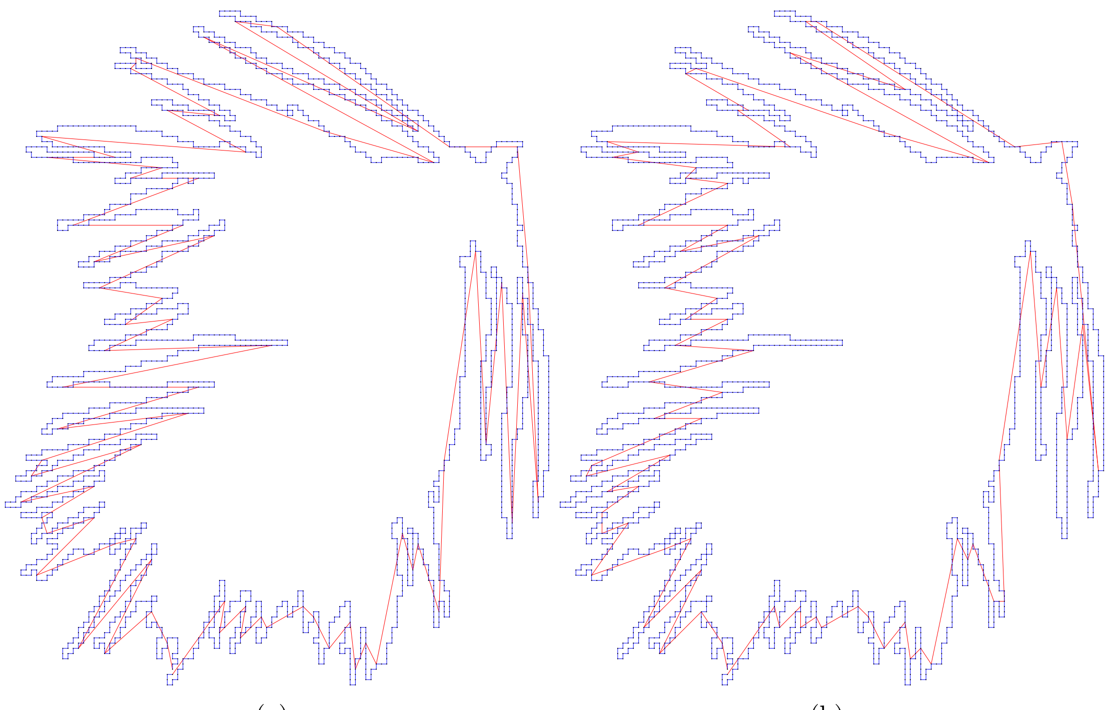
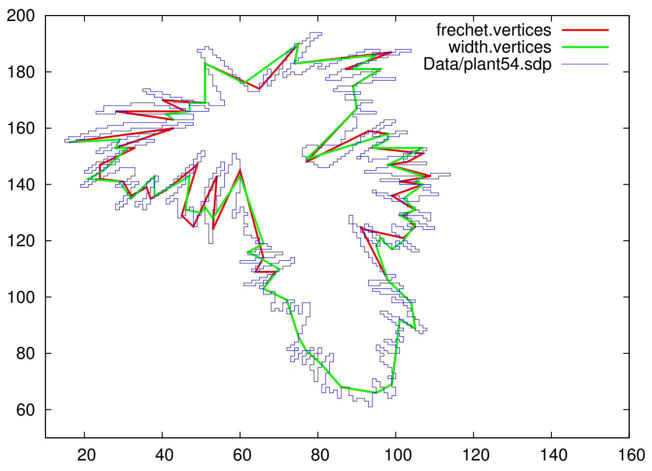
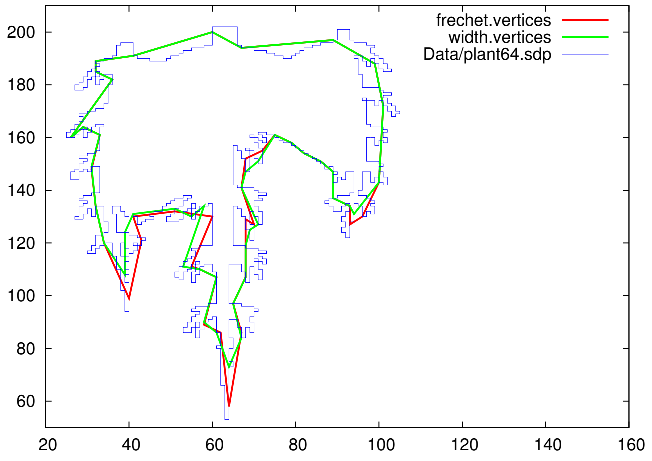

(a) (b)

Figure 9: Simplification with the Fréchet distance (a), and with the width only (b) with the maximal error set to 5.

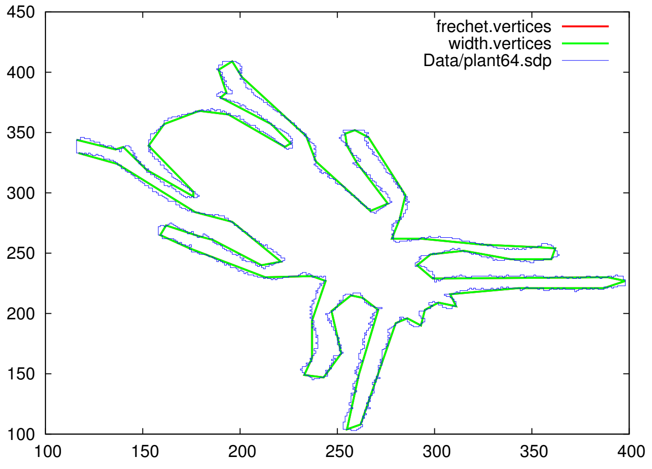
Figure 9 illustrates the eps files output for the simplification using the Fréchet distance (a) or with the width criterion only (option `-w`, (b)). Note that the simplification using the Fréchet distance better retrieves the sharp features of the contour for a similar number of points (76 in both cases). Actually, the difference between the Fréchet simplification and the width simplification is generally very light. The Fréchet distance is particularly efficient for contours with long and narrow features like the ones presented here. In Figure 10, the input data file and output files `output.vertices` successively computed for the Fréchet simplification and the width simplification are displayed with gnuplot.



maximal error = 8



maximal error = 8



maximal error = 5

Figure 10: Some other examples of digital curve simplification with different maximal errors.

Image Credits

All images are given by the authors. Contours come from the large binary image database http://www.lems.brown.edu/~dmc/

References

[1] Mohammad Ali Abam, Mark de Berg, Peter Hachenberger, and Alireza Zarei, *Streaming algorithms for line simplification*, in Proceedings of the 23rd annual Symposium on Computational Geometry, ACM, 2007, pp. 175–183. http://dx.doi.org/10.1145/1247069.1247103.

[2] P.K. Agarwal, S. Har-Peled, N.H. Mustafa, and Y. Wang, *Near-linear time approximation algorithms for curve simplification*, Algorithmica, 42 (2005), pp. 203–219. http://dx.doi.org/10.1007/s00453-005-1165-y.

[3] W. S. Chan and F. Chin, *Approximation of polygonal curves with minimum number of line segments*, in Proceedings of 3rd International Symposium on Algorithms and Computation, Springer-Verlag, 1992, pp. 378–387. http://dx.doi.org/10.1007/3-540-56279-6_90.

[4] Isabelle Sivignon, *A near-linear time guaranteed algorithm for digital curve simplification under the Fréchet distance*, in Proceedings of the 16th IAPR International Discrete Geometry for Computer Imagery, vol. 6607 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2011, pp. 333–345. ISBN 978-3-642-19866-3.