Comparison Between Space-Efficient Algorithms for Approximating Polygonal Curves in 2-D Spaces

Victor Cuadrado Juan

January 17, 2013

Abstract

This document contains a comparison between two algorithms for approximating polygonal curves in two dimensional spaces, present in *Space-efficient algorithms for approximating polygonal curves in two dimensional space* [1] and *On approximating polygonal curves in two and three dimensions* [2], focusing on the space cost of them. First, the problem of efficiently approximating polygonal curves in 2-D (line simplification) is presented, focusing in understandability, and last the algorithms are discussed.

Contents

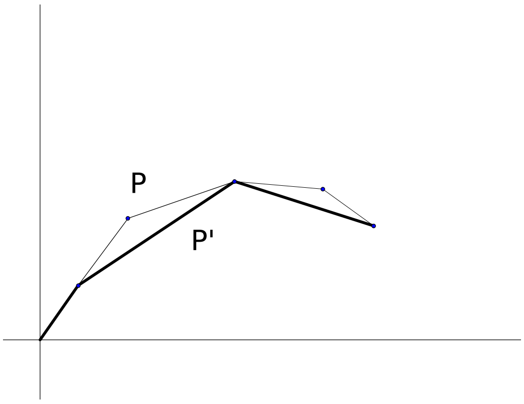

Figure 1: A polygonal curve P and it's simplification, P' (with thick lines).

1 Introduction

The problem of line simplification covered in this paper is widely present in computer applications: graphics, cartography and image processing are all populated with data in the form of polygonal curves. A polygonal curve in R^2 is specified by an ordered set $[p_1, p_2, ..., p_n]$ of vertex points in R^2 such that any two consecutive vertices p_i, p_{i+1} are connected by the line segment $\overline{p_i, p_{i+1}}$, $1 \leq i < n$. It is possible that the polygonal curve has self-intersections (figure 1). This representation allows for an efficient description of the boundaries of shapes.

The problem

Specifically, the problem of approximating a polygonal curve is:

given a polygonal curve $P = [p_1, p_2, ..., p_n]$ in R^2, to determine another polygonal curve $P' = [p_1, p_2, .., p_m]$ of m vertices in R^2 such that:

1. $m \leq n$ desirably, m is significantly smaller than n,
2. the vertex sequence of P' is a subsequence of the vertex sequence of P , with $p'_1 = p_1$ and $p'_m = p_n$ (first and last are the same between P and P'), and
3. each edge $\overline{p_i, p_{i+1}}$ of P is an approximating line segment of the subcurve $[p_j, p_{j+1}, .., p_k]$ of P , where $p'_i = p_j$ and $p'_{i+1} = p_k$ and $j < k$. That is, for every point p of the subcurve $[p_j, p_{j+1}, ..., p_k]$ of P , the error incurred by using $\overline{p'_i, p'_{i+1}} = \overline{p_j, p_k}$ to approximate p, based on a given error criterion, is no bigger than a specified error tolerance . Such a line segment $\overline{p'_i, p'_{i+1}}$is called the *approximating line segment* of the corresponding subcurve $[p_j, p_{j+1}, ..., p_k]$ of P(figure 2).

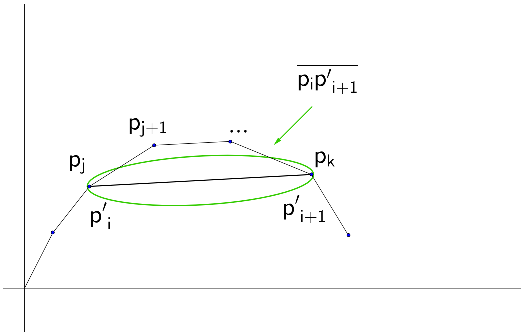

Figure 2: Aprox. line segment of the subcurve $[p_j, p_{j+1}, ..., p_k]$ of P

The third point of the constraints can be said also as follows:

$$\forall p \in [p_j, p_{j+1}, ..., p_k] \text{, the approximating line segment error of } \overline{p'_i, p'_{i+1}} = \overline{p_j, p_k} \text{ is } \leq \varepsilon.$$

In this constrains, the parameter m specifies the size (i.e., the number of vertices) of the "compressed" version P' of P , and the parameter ε controls the "closeness" of P' to P (under a certain error criterion which we shall see now). There is a trade-off between the two parameters m and $\leq \varepsilon$: The smaller ε is, the larger m tends to be, and vice versa. Based on this relation between m and ε, two versions of optimization problems on approximating polygonal curves in the plane can be considered:

- given $\leq \varepsilon$, minimize m (called the min-# problem),
- given m, minimize $\leq \varepsilon$ (called the min-$\leq \varepsilon$ problem).

If this constraints are achieved, we end with another curve that resembles the original one, but has fewer vertices. Obviously, it is desirable to obtain a curve that does not get distorted, with independency of the method of approximation we use. The role of the third constraint presented in the formalisation of the approximation method is to specify which vertices of P should be deleted. So, it's the primary parameter for controlling to output of the shape of the approximating curve P', and so, is the error criterion which defines the goodness of fit between the two curves. There are multiple approaches to measure the error of approximation of a vertex, but the choice of a method depends primary on how we answer the following question: how do we mathematically define a "good approximation"?. There are several definitions of error tolerance (that can be found in [3], [4], [5] and more). In this paper, as we review the algorithms presented in [1] and [2], we are going to see only the error criterion for those algorithms: the *tolerance zone* *criterion* and the *infinite beam criterion* (also called the parallel-strip criterion in [2]).In this paper we study both the approximation problems only in 2-D space. Although we are not explicity choosing a metric now, the more used distance metrics are L_1, L_2 and L_∞, and we will stick the discussion to them. We will present the algorithms with the L_2 metric, and highlight the differences for L_1 and L_∞ later in the paper.

2 The error criterions

Tolerance zone criterion

Under it, the approximation error between a segment $\overline{p_j, p_k}$ and the corresponding subcurve $S = [p_j, p_{j+1}, .., p_k]$ of P is defined as the maximum distance in an L_h metric between $\overline{p_j, p_k}$ and each point on the subcurve S. we consider $h \in \{1, 2, \infty\}$.

Thanks that P is a polygonal curve, the maximum L_h distance between $\overline{p_j, p_k}$ and the points of S can be computed by simply finding the maximum L_h distance between $\overline{p_j, p_k}$ and each vertex p_l of S (with $j \leq l \leq k$). We denote by $dist_h(\overline{p_j, p_k}, p_l)$ the L_h distance between $\overline{p_j, p_k}$ and p_l. As it can be seen in the example at figure 3, the tolerance zone has a distinct shape.

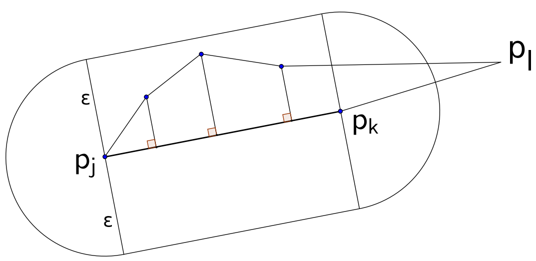

Figure 3: the point p_l is out of the tolerance zone. Note the semicircles with radius equal $\leq \varepsilon$ that are formed by the $\leq \varepsilon$ distances on p_j and p_k

The tolerance zone criterion can also be explained as follows: Based on the tolerance zone criterion, a vertex p_k of P is within distance ε from a line segment $\overline{p_i, p_j}$, where $i \leq k \leq j$, if the following conditions are all satisfied [6]:

1. Condition 1: $dist(L(\overline{p_i, p_j}), p_k) \leq \varepsilon$. (see figure 4)

2. Condition 2: If the convex angle defined by $\overline{p_i, p_k}$ and $\overline{p_i, p_j}$ is greater than $\varpi/2$, then $d(p_k, p_i) \leq \varepsilon$, where $d(p_k, p_i)$ denotes the L_2 distance between p_k and p_i , and an angle defined by two line segments is said to be convex if the angle is no larger than ϖ. (see figure 5)

3. Condition 3: If the convex angle defined by $\overline{p_j, p_k}$ and $\overline{p_j, p_i}$ is greater than $\varpi/2$ then $d(p_k, p_j) \leq \varepsilon$. (see figure 6)

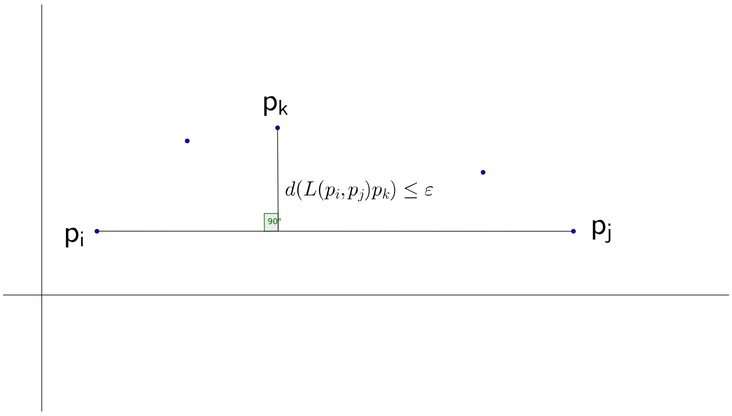

Figure 4: Example of a distance within condition 1.

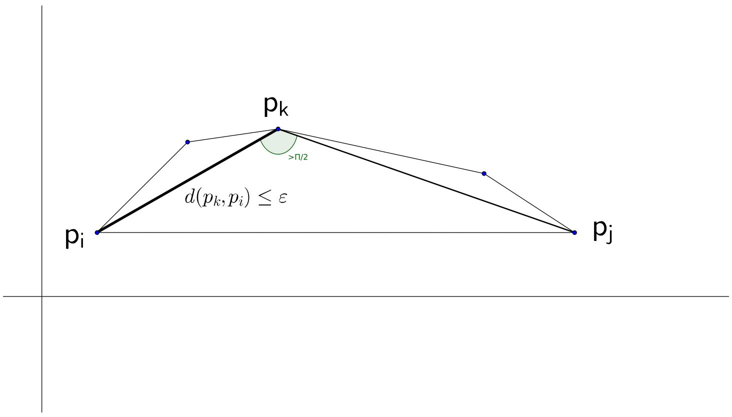

Figure 5: Example of a distance within condition 2.

Infinite beam criterion

Under the infinite beam criterion, the approximation error between a segment $\overline{p_j, p_k}$ and the corresponding subcurve $S = [p_j, p_{j+1}, .., p_k]$ of P is defined as the maximum L_h distance between the line (and not the segment)

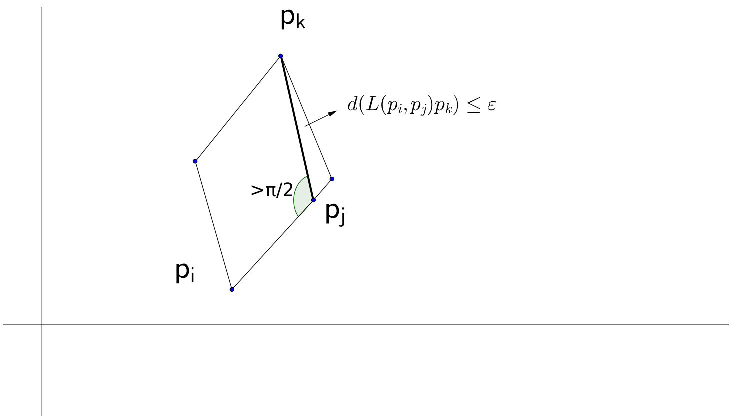

Figure 6: Example of a distance within condition 3.

$L(\overline{p_j, p_k})$ that contains $\overline{p_j, p_k}$ and each point of the subcurve S. We denote by $dist_h(L(\overline{p_j, p_k}), p_l)$ the L_h distance between the line $L(\overline{p_j, p_k})$ and the vertex p_l of S. As it can be seen in the example at 7, this creates an infinite parallel strip region.

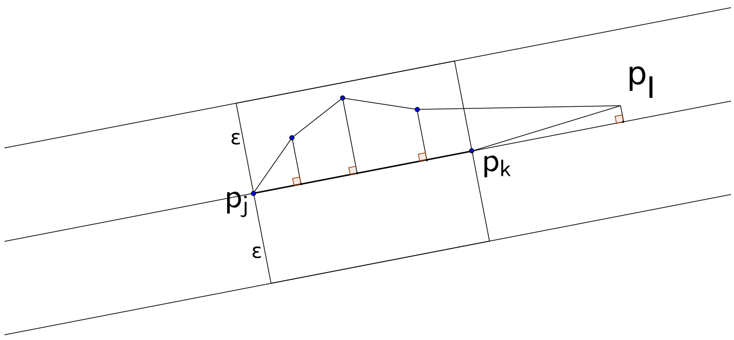

Figure 7: the point p_l is inside of the tolerance zone. Note the infinite region created by the two lines parallel to the line $L(\overline{p_j, p_k})$, each of them at a distance of $\leq \varepsilon$ from $L(\overline{p_j, p_k})$.

The *infinite beam criterion* is a popular criterion: experiments and studies (as in [7] and [8]) show that this criterion bounded with a proper method of approximation, is capable of approximations that are among the most perceptually pleasing. Also, within this criterion, as the input curve P is approximated by the curve P' within a tolerance ε if and only if the vertices of P, it is sufficient to consider only the vertices of the input curve, and that allows for simple algorithms which are easy to implement.

Approximation error According to these error criteria, the approximation error incurred by using a curve $P' = [p_1, p_2, .., p_m]$ to approximate P is defined as the maximum error among those of the edges of P with respect to their corresponding subcurves of P:

$$max_{i=1}^{m-1}\{max\{dist_h(\overline{p'_i, p'_{i+1}}, p_l)|p'_i = p'_j, p'_{i+1} = p'_k, andj \leq l \leq k\}\}$$

The parameter ε specifies the upper limit of the approximation error of P' with respect to P. (see figure 8)

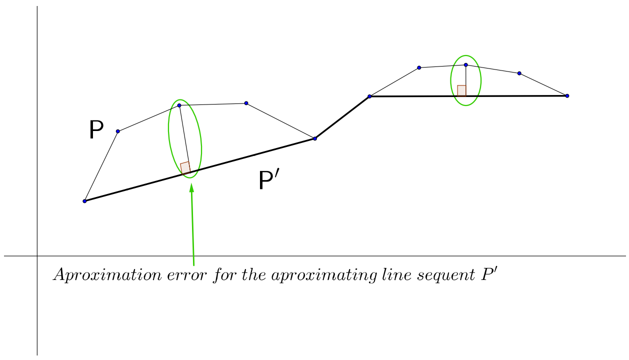

Figure 8: Example of the approximation error for the aproximating line segment P'. Notice is the max between all posibles.

3 Concepts and definitions for the algorithms

Error tolerance region

The three conditions of the tolerance zone criterion (in section 2) together define a region called the *error tolerance region* [1] of the line segment $\overline{p_i, p_j}$. Let $ray(p_i, p_j)$ denote the ray emanating from p_i and passing through p_j . Since the line segment $\overline{p_i, p_j} = ray(p_i, p_j) \cap ray(_pj, p_i)$, the error tolerance region of $\overline{p_i, p_j}$ is the intersection of the error tolerance regions of $ray(p_i, p_j)$ and $ray(p_j, p_i)$.

Let p_i , p_j , and p_k be vertices of P with $i < k \leq j$.
If $dist(ray(p_i, p_j), p_k) \leq \varepsilon$, then $ray(p_i, p_j)$ is said to be an approximating ray

[1] In comparison with the conditions listed in the tolerance zone criterion (see section 2), only Condition 1 needs to be satisfied for the infinite beam criterion, and hence the shape of the error tolerance region of a segment $\overline{p_i, p_j}$ is an infinite "strip" of width 2ε in 2-D.

of p_k . If $dist(ray(p_i, p_j), p_k) \leq \varepsilon$ for each k with $i < k \leq j$, then $ray(p_i, p_j)$ is an approximating ray of the chain $[p_i, p_{i+1}, ..., p_j]$ of P (an approximating ray, for short). Thus, under the tolerance zone criterion, $dist(\overline{p_i p_j}, p_k) \leq \varepsilon$iff $dist(\overline{p_i p_j}, p_k) \leq \varepsilon$ and $dist(\overline{p_j p_i}, p_k) \leq \varepsilon$. In other words, $\overline{p_i p_j}$ is an approximating line segment of p_k iff $ray(p_i, p_j)$ and $ray(p_j, p_i)$ are both approximating rays of p_k . Therefore, for a given , one can first compute all approximating rays $ray(p_i, p_j)$, then all approximating rays $ray(p_j, p_i)$, with $1 \leq i < j \leq n$, and finally find the set of approximating line segments from the set of approximating rays.

Expressing the error tolerance region

In 2-D, for two vertices p_i and p_k , let r_a and r_b be two rays emanating from p_i such that the distance between p_k and each of r_a and r_b is exactly ε. Let D_{ik} be the whole plane if $d(p_i, p_k) \leq \varepsilon$ (see figure 10), and let D_{ik} be the convex region bounded by r_a and r_b otherwise (see figure 9). Then, by Conditions 1–3:

$$dist(ray(p_i, p_j), p_k) \leq \leq > p_j \in D_{ik}.$$

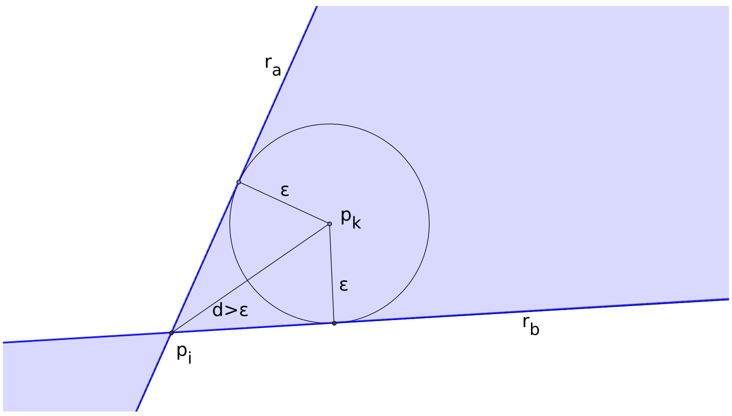

Figure 9: Example of a plane formed by vertices p_i and p_k. Note that r_a and r_b create a double cone. Every point situated on the coloured area is inside the error tolerance region.

Lets see now when a segment is a valid approximating line segment: For $1 \leq i < j \leq n$ let $F_{ij} = \cap_{k=i+1}^{j} D_{ik}$ and let $B_{ij} = \cap_{k=i}^{j-1} D_{ik}$. Observe that, if F_{ij} (resp., B_{ij}) is not empty, then every ray $r \in F_{ij}$ (resp., B_{ij}) that starts from p_i (resp., p_j) has non empty intersection with the disc of radius centered at p_k , for each $i \leq j \leq k$. Thus, under the tolerance zone criterion, $\overline{p_i p_j}$ is an approximating line segment of P (and hence $e_{ij} \in E$) iff $pj \in F_{ij}$

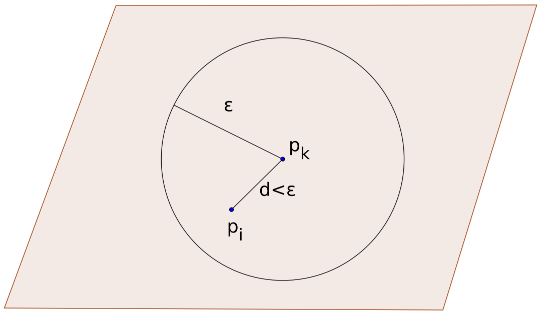

Figure 10: Example of a plane formed by vertices p_i and p_k: note that because of their closeness and being at a distance $\leq \varepsilon$ between them, D_{ik} is the whole plane.

and $pi \in B_{ij}$. Note that F_{ij} (resp., B_{ij}) always consists of one (possibly empty) cone. Hence each F_{ij} (resp., B_{ij}) takes O(1) space to store. So: if $\overline{p_i p_j}$ is an approximating line segment of P , then $\overline{p_i p_j} \in F_{ij} \cap B_{ij}$.

Example of a valid approximating line segment

Supose that we have $i \in 1, 2, 3, 4, 5$ and $j \in 2, 2, 3, 4, 6$. Is $\overline{p_2 p_5}$ a valid approximating line segment? For $i = 2$ and $j = 5$, we need to compute:

$$F_{ij} = F_{25} = D_{23} \cap D_{24} \cap D_{25}. \; B_{ij} = B_{25} \cap D_{52} \cap D_{53} \cap D_{54}.$$

First, we compute F_{ij}, as seen in figure 11. Notice how the D_{ik} intersect between them, and form the plane F_{25}, with blue color. The segment $\overline{p_2 p_5}$, in red color, is inside that plane.

Second, we compute B_{ij}, as seen in figure 12.Notice how the D_{ik} intersect between them, and form the plane B_{25}, with green color. The segment $\overline{p_2 p_5}$, in red color, is inside that plane. Because that the segment $\overline{p_2 p_5}$ is in F_{ij} and B_{ij}, we can say that it is an approximating line segment for the points p_2 and p_5.

Shortest path

Now we define the notion of shortest path prom one point p_i to another point p_j: For $1 \leq i < j \leq n$, let $SD(p_i, p_j)$ denote the length of a shortest path from p_i to p_j in G, and let $w(e_{ij})$ denote the weight of an edge $e_{ij} \in E$. Since the graph G is directed acyclic, it is clear that the inductive relation $SD(p_1, p_j) = minSD(p_1, p_k) + w(e_{kj})|1 \leq k < j \leq nande_{kj} \in E$, where

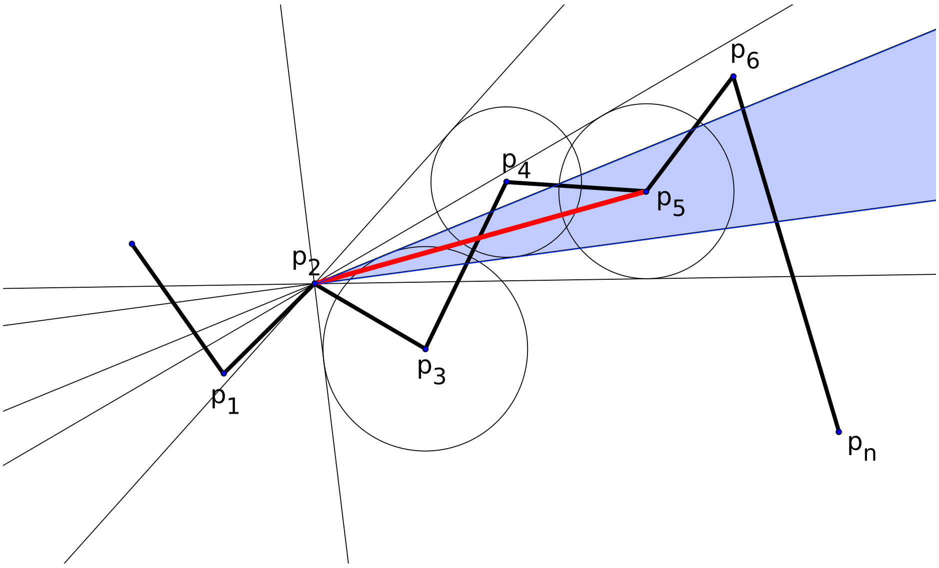

Figure 11: The blued plane is F_{ij}. Only the tangent lines of each of the D_{ik} for calculating F_{ij} have been drawn, and the polygonal line is represented as a thick line.

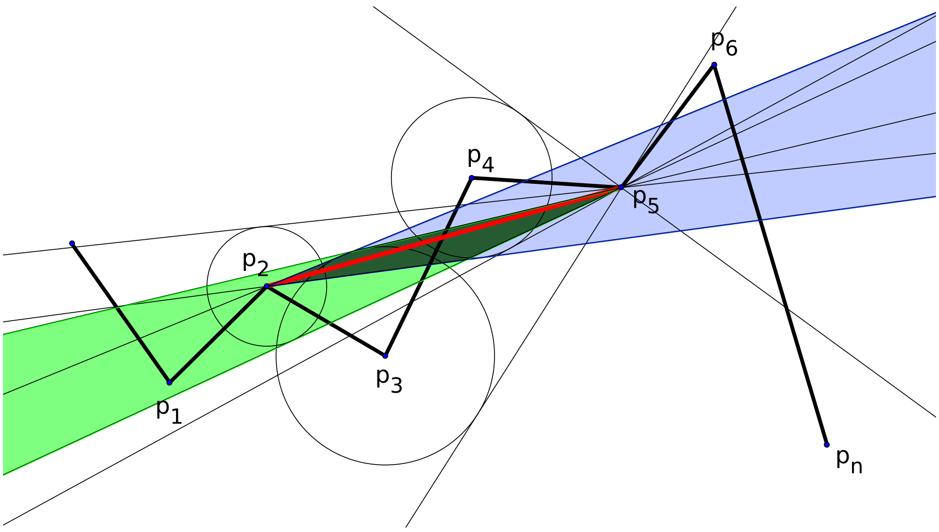

Figure 12: The blued plane is F_{ij}, and the greened plane is The blued plane is B_{ij}. The intersecction is in a darker color. Only the tangent lines of each of the D_{ik} for calculating B_{ij} have been drawn.

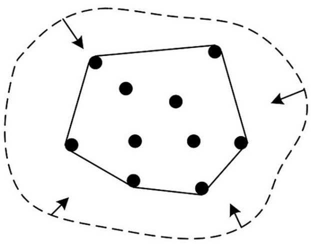

Figure 13: Example of the intuitive definition of a convex hull

$SD(p_1, p_j) = 0$, holds. This immediately suggests an incremental algorithm for computing $SD(p_1, p_j)$ (from the $SD(p_1, p_k)$'s, with $1 \leq k < j$). Without loss of generality, we describe only an algorithm for computing the length $SD(p_1, p_n)$ of a shortest p_1-to-p_n path in G (the algorithm can be easily modified to produce, in addition to $SD(p_1, p_n)$, a shortest path tree of G rooted at p_1).

Convex hull of a set of points

Let S be a set of points in the plane. Formal definition: the *convex hull* of S is the smallest convex polygon that contains all the points of S. A polygon P is said to be convex if P is non-intersecting, and for any two points p and q on the boundary of P, segment $\overline{pq}$ lies entirely inside P. And intuitive definition of a convex hull could be as follows: Imagine the points of S as being pegs; then the convex hull of S is the shape of a rubber band stretched around the pepgs(see figure 13).

4 Algorithms

The algorithms here explained are renamed for an easy reading of this document. The two algorithms are:

- The *approximating algorithm with $O(n^2)$ space complexity*[2]. It uses a directed graph for the L_2 min-# problem, and a binary search that invokes the L_2 min-# line segment algorithm for the min-$\leq \varepsilon$ problem. It constructs the graph in $O(n^2)$ and gets $O(n^2 logn)$ time. It's complexities are then as shown in table 1.
- the *approximating algorithm with $O(n)$ space complexity*[3]. Differently to the algorithm with $O(n^2)$ space complexity, it does not need to

[2]present in *On approximating polygonal curves in two and three dimensions* [2]

[3]present in *Space-efficient algorithms for approximating polygonal curves in two dimensional space* [1]

maintain a directed graph for the L_2 min-# problem, and only stores a fraction of the $O(n^2)$ approximation errors for the the binary search process for the min-$\leq \varepsilon$ problem. It's complexities are then as shown in table 1.

	infinite beam crit.		tolerance zone crit.	
	time	space	time	space
min-#	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
min-ϵ	$O(n^2 logn)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

Table 1: The complexities of the approximating algorithm on $O(n^2)$ space.

	infinite beam crit.		tolerance zone crit.	
	time	space	time	space
min-#	$O(n^2 logn)$	$O(n)$	$O(n^2)$	$O(n)$
min-ϵ	$O(n^2 logn)$	$O(n)$	$O(n^2 logn)$	$O(n)$

Table 2: The complexities of the approximating algorithm on $O(n)$ space.

5 The approximating algorithm on $O(n^2)$ space

For the min-# problem

Under the tolerance zone criterion

It is solved by first constructing a directed acyclic graph $G = (V, E)$ for the curve approximation (where V is the vertex set of the curve P), and then finding a p_1-to-p_n shortest path in G. G has an arc $e_{ij} = (p_i, p_j)$, $i < j$, if and only if (iff) $\overline{p_i, p_j}$ is the approximating line segment for the chain $S = [p_i, p_{i+1}, .., p_i]$ of P . For the 2-D case, the number of edges in G, |E|, is $O(n^2)$, and the time complexity of the min-# algorithm is dominated by the time for constructing G.

The construction of the graph G(V,E) is as follows:

```
algorithm build_graph
{input: a polygonal curve P = [p_1,p_2,..,p_n] in R^2, with ↩
   error tolerances for each point: [ε_1,ε_2,..,ε_n]}
{output: a directed graph G(V,E) of the O(n^2) candidate ↩
   approximating line segments of P}

for i=1 to n do v_i ← v_p (set vertices of V to correspond↩
    with those on P)
for i=1 to n-1 do
begin
  incrementalPlane ← R^2 (set to an open cone)
  for j=i+1 to n do and meanwhile incrementalPlane ≠ ∅ (↩
     meanwhile [p_i,p_j] continues to be an approximating ↩
     segment)
  begin
    if p_j ∈ incrementalPlane then e_ij<-[p_i,p_j] with weight (↩
       j-i-1)
    incrementalPlane ← incrementalPlane ∩ D_ij
  end
end
```

Note the 2 anidated for. Since we need at most two lines to represent a D_{ik} double cone plane (an open cone doesn't require anything, is just the whole plane), the seen algorithm build_graph only requires constant storage to maintain and update the incrementalPlane variable. Therefore,

$$max\{\sum_{i=1}^{n-1} O(1), \sum_{i=1}^{n-1}(\sum_{j=i+1}^{n} O(1))\} = \sum_{i=1}^{n-1}(\sum_{j=i+1}^{n} O(1)) \approx O(n)^2$$

This gives $O(n^2)$ time and $O(n^2)$ space completixies for building the graph.

For finding the required path P' from the graph G, a fordward dynamic programming technique [9] is used. The time complexity of this technique is proportional to the number of edges in G ($O(n^2)$). **the space complexity for maintaining G is also $O(n^2)$: this gives $O(n^2)$ time and $O(n^2)$ space complexities for the whole algorithm**.

Under the infinite beam criterion

If wee let the error tolerances of the vertices in P be uniform (e.g. $\forall\ p_i$ in P, $\varepsilon_i = \varepsilon$), then **the algorithm build_graph solves the problem in $O(n^2)$ time and also $O(n^2)$ space for the infinite beam criterion**.

For the min-ε problem

Under the tolerance zone criterion and infinite beam criterion

It first computes and stores the $O(n^2)$ approximation errors for all the segments $\overline{p_i, p_j}$ defined on P , and then performs a binary search on these errors for the sought error , at each step of the search applying a min-# algorithm. let ε_A, $\varepsilon_B > 0$ be two error tolerances (see figure 15 for an example). Suppose that we solve the min-# problem twice: once with ε_A to obtain a curve P_A' of m_A vertices, and a second time for ε_B to obtain a curve P_B' of m_B vertices. Then

1. if $\varepsilon_A < \varepsilon_B$ then $m_A \geq m_B$
2. if $m_A < m_B$ then $\varepsilon_A > \varepsilon_B$

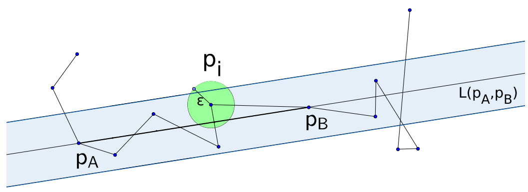

Figure 14: Example of a point p_i with it's error tolerance region created by its ε.

Let store in a sorted array Q the maximum error for each of the $O(n^2)$ segments of pairs of vertices of P. We then apply a binary search on Q, at each step invoking build_graph (from the min-# problem) with a uniform error tolerance that is the median of Q. If:

A $|P'| < m$: all elements in Q above the median element ε allow no more line segments than m. We recurre to the lower half to find a P' with more vertices and lower ε's.

B $|P'| > m$: all elements in Q below the median element ε require no less vertices than m. We recurre to the upper half of Q.

C $|P'| = m$: it is possible that an elemnt $\varepsilon' < \varepsilon$ exists in the lower half and yields $|P'| = m$ as well. We recurre to the lower half. Otherwise, we return with the curve of error ε.

The algorithm associated with this binary search is as follows:

```
algorithm minimize_error
{input: a polygonal curve P = [p_1, p_2, .., p_n] in R^2}
{output: a curve P' = [p_1, p_2, .., p_n] satisfing the min-\↩
   varepsilon problem}

(1) compute the maximum approx. error (err_ab) for all ↩
   overline{p_A, p_B} where p_A, p_B ∈ P,  A < B,  ↩
   err_ab = max_{A<i<B} D(p_i, L(p_A, p_B)). Store all the err_ab in an ↩
   array Q.
(2) Sort Q by increasing order.
(3) i <- 1, j <- |Q| = n(n-1)/2
(4) ε <- Q[(i+j)/2], the median element in {Q[i], ... ,↩
   Q[j]}
(5) solve the min-# problem using build_graph with the ↩
   uniform error ε of (4) to obtain a curve P' = ↩
   [p'_1, p'_2, .., p'_m*] of m* vertices.

(6)
  if i = j then (corresponds to [A])
    begin
    if (i+j)/2 -1 < 1
      return P' with error ε (the element to the left of↩
           our median < 1)
    else
      begin (recurre the lower half)
      ε' <- Q[(i+j)/2 -1]
      invoke build_graph with ε' to obtain P'' with m** ↩
         vertices
      if m** = m* then return P'' with error ε'
      else return P' with error ε
      end
    end
  else if m* ≤ m then j <- (i+j)/ 2 (corresponds to [B↩
     ]), recurre to the lower half
  else if  m* > m then i <- (i+j)/ 2 (corresponds to [↩
     C]), recurre to the upper half

(7) repeat from step (4)
```

To complete step (1), an on-line convex hull algorithm is used. It is called on-line because it computes the vertices one by one, at the pace they are available. As seen in [2], for two vertices p_A,p_B of P, $A < B$, and given the convex hull $CH[p_A, ..., p_B]$ of the subchain $[p_A, .., p_B]$ of P, it is possible to determine the vertex p with the maximum error vertex of the subchain in $O(logn)$ time (see figure 15),and **the algorithm has** $O(n^2 log^2 n)$ **time complexity**. **As Q requires** $O(n^2)$**, the algorithm build_graph has**

$O(n^2)$ **space complexity, and thus, the whole problem has** $O(n^2)$ **space complexity.**

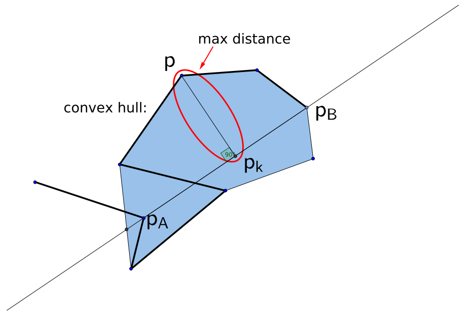

Figure 15: Example of the max error vertex p, using a convex hull.

6 The approximating algorithm on $O(n)$ space

For the min-# problem

Under the tolerance zone criterion

Start with $SD(p_1, p_n) = 0$. Suppose that for a vertex p_j, $2 \leq j \leq n$, we have obtained $SD(p_1, p_k)$ for each p_k $1 \leq k < j$. We must obtain $SD(p_1, p_j)$ from the $SD(p_1, p_k)$'s. We must identify all edges $e_{kj} \in E$ with $1 \leq k < j$ (O(j) time). To know if and edge $e_{kj} \in E$ we need to check if

1. $p_j \in F_{kj}$ (O(1) time: it consists in 4 comparisons, 2 for the x coordinates of r_a and 2 for the y coordinates of r_b), and
2. $p_k \in B_{kj}$ (O(1) time, idem).

For doing those "$\in$" checks, we need the F_{kj}'s and B_{kj}'s. Lets assume we have mantained F_{kj_1} for every k with $1 \leq k \leq j-1$. By definition:
$F_{kj} = F_{kj_1} \cap D_{kj}$. (this takes O(1) time). $B_{kj} = B_{kj_1} \cap D_{kj}$. (O(1) time, idem). **So we don't need to maintain** B_{kj} **and** F_{kj}**. And then, Maintaining** $SD(p_1, p_k)$**'s and** $F_{k,j-1}$**'s uses** $O(n)$ **space.**

Under the infinite beam criterion

The difference with the tolerance zone criterion is that computing $F_{ij} = \cap_{k=i+1}^{j} D_{ik}$ costs $O(n^2)$ space, instead of $O(n^2)$ space. Under the infinite beam criterion, the algorithm cannot maintain $F_{k,j-1}$ for all k with $1 \leq k \leq j-1$ as for the tolerance zone criterion (this would take $O(n^2)$ space). Instead, this algorithm performs this computation:

$$SD(p_1,p_k) = min\{SD(p_1,p_k), SD(p_1,p_i) + w(e_{ik})\} \text{ for each } k \in i < k \leq n \text{ and } e_{ik} \in E$$

(note that $SD(p_1, p_k)$ may not containt the correct shortest path p_1 to p_k). At the beginning of the i-th iteration, the value of SD_{p_1,p_k} is correct. So, if the edges $e_{ik} \in E$ are available for all k, $1 < k \leq n$, then the values of SD_{p_1,p_k}'s can be maintained with the relation with $SD(p_1, p_i)$'s shown. So, to process the i-th iteration is to identify all edges e_{ik}, and this is to test, for each k, $1 < k \leq n$, if $p_k \in F_{ik}$. This testing can be done by a binary tree-guided scheme [10], where each leaf of a complete binary tree T_i is associated with a D_{ik} and each internal node is associated with the $\cap_{k=i+1}^{j} D_{ik}$. **This tree-guided scheme finds all the edges in $e_{ik} \in E$ in $O(nlogn)$ time (then $O(n^2logn)$ time for the whole algorithm) and $O(n)$ space**.

For the min-ε problem

Under the tolerance zone criterion and infinite beam criterion

The approximating algorithm on $O(n^2)$ space stores the $O(n^2)$ approximation errors of P in a sorted array for the binary search. But binary search on set A can be performed in 3 stages, only storing $O(n)$ well chosen samples, meanwhile performing the binary search of all the $O(n^2)$ without increasing it's time bound.

The stages technique consist of $O(1)$ stages, each of which perform:

1. Select $O(n)$ samples from the set S of all the elements that are currently active for the binary search; these samples are such that between any two consecutive samples, there is a probably "small" subset of S
2. Perform binary search on the $O(n)$ samples.
3. At the end of this binary search, reduce the problem to one such "small" subset of the set S (thus only the elements in this subset continue to be currently active).

the (3) step is as follows:

First, we need to organize the $O(n^2)$ approximation errors into n sets, each of which is of size $O(n)$. Let
$err(\overline{p_ip_j}) = max_{ki}^{j}\{dist(\overline{p_ip_j}, pk)\}$ (resp., $err(L(\overline{p_ip_j})) = max_{ki}^{j}\{dist(L(\overline{p_ip_j}), pk)\}$) denote the approximation error of the segment (resp., line) $\overline{p_ip_j}$ (resp., $L(\overline{p_ip_j})$). Since $err(\overline{p_ip_j}) = max\{err(L(\overline{p_ip_j})), max_{ki}^{j}\{d(\overline{p_ip_j}), d(pj, pk)\}\}$, for all the segments $\overline{p_ip_j}$ such that $1 \leq i < j \leq n$, we have
$err(\overline{p_ip_j})|1 \leq i < j \leq n \subseteq err(L(\overline{p_ip_j}))|1 \leq i < j \leq n \cup d(p_ip_j)|1 \leq i \leq j \leq n.$

Let $ERR(P) = err(\overline{p_i p_j})|1 \leq i < j \leq n$,
$L - ERR(P) = err(L(\overline{p_i p_j}))|1 \leq i < j \leq n$, and
$V - ERR(P) = d(p_i p_j)|1 \leq i \leq j \leq n$.
Since $L - ERR(P) \cup V - ERR(P)$ is a superset of $ERR(P)$, the sought error $\varepsilon ERR(P)$ that is the solution for the 2-D min-ε problem is contained in $L - ERR(P) \cup V - ERR(P)$. Hence we search for the $\varepsilon ERR(P)$ by performing binary search on $L - ERR(P) \cup V - ERR(P)$. Note that $|L - ERR(P) \cup V - ERR(P)| = O(n^2)$. For every i with $1 \leq i < n$, let $L - ERR_i$ be the set of the approximation errors $err(L(\overline{p_i p_j}))$ for the lines$L(\overline{p_i p_j})$, and $V - ERR_i$ be the set of the $d(p_i, p_j)$'s, for all j such that $i \leq j \leq n$ (assume that $err(L(p_i p_i)) = 0$). Note that all the errors in $L - ERR_i$ can be computed in $O(nlogn)$ time and $O(n)$ space by using Toussaint's approach based on maintaining on-line convex hulls of planar points [11]. Also, it is easy to compute $V - ERR_i$ in $O(n)$ time and space. Let $ERR_i = L - ERR_i \cup V - ERR_i$. Then $|ERR_i| \leq 2n$ and ERR_i can be obtained in $O(nlogn)$ time and $O(n)$ space. Since $L - ERR(P) \cup V - ERR(P) = \cup_{i=1}^{n-1} ERR_i$, we organize the $O(n^2)$ approximation errors of$L - ERR(P) \cup V - ERR(P)$ into the sets ERR_1 , ERR_2 , ... , ERR_{n-1} . Without loss of generality, we assume that the errors in $L - ERR(P) \cup V - ERR(P)$ are distinct (ties can be easily broken in a systematic way). The following lemma, which has been used in various selection algorithms before (see [12]), is a key to our $O(n)$ space binary search algorithm.

Lemma 2. Suppose that a set S of r distinct elements is organized as m sorted sets C_i of size $O(r/m)$ each. For every $i = 1, 2, ..., m$, let $C'i$ be the subset of Ci that consists of every s-th element of Ci (i.e., the s-th, (2s)-th, (3s)-th, . . ., elements of Ci). Let $S' = \cup_{i=1}^{m} C_i$. If w (resp., z) is the α-th (resp., β-th) i=1 smallest element of S , with $w < z$, then there are at most $s(\beta - \alpha + m - 1)$ elements of S that are between w and z (i.e., these elements are $\geq w$but $\leq z$).

Now from the n - 1 sets ERR_1 , ERR_2 , . . . , ERR_{n-1} of size $O(n)$ each, we need to choose $O(n)$ sample elements for the binary search process. Our basic idea for the sampling is as follows: Partition the n - 1 sets ERR_k into $\sqrt{n}$ groups G_i of (roughly) $\sqrt{n}$ sets each. Treat each group G_i (of size $O(n^{1.5})$) as one single sorted set and select $O(\sqrt{n})$ sample elements from G_i , such that there are $O(n)$ elements of G_i between every two consecutive samples from G_i . The total number of samples so selected from all the $\sqrt{n}$ groups is $O(n)$.

Note that, since we use only O(n) space, we cannot explicitly store a group G_i for the sampling process. Instead, we sample from the $\sqrt{n}$ sets of $_Gi$ with the following procedure:

producere sampling(G_i)

1. For every set ERR_k of G_i, first compute ERR_k and sort it; then select every ($\sqrt{n}$)-th element from the sorted set ERR_k, and put the selected elements in the set S_i for G_i.

2. Sort the set S_i, and choose every ($\sqrt{n}$)-th element from S_i. These chosen elements form the samples of G_i, and are put into the set Sample(G_i).

The total time of the procedure is $O(n^{1.5}logn)$, since we need to compute and sort each of the n sets ERR_k of G_i . The space is $O(n)$, because we only need to store each ERR_k once in Step 1, and store the set S_i of size $O(n)$. The size of Sample(G_i) is clearly $O(\sqrt{n})$. The quality of the samples in Sample(G_i) is ensured because there are $O(n)$ elements of G_i between every two consecutive samples in Sample(G_i).
Now, we apply this algorithm, consisting in three stages:

first stage

We perform procedure sampling on each of the $\sqrt{n}$ groups G_i and obtain $O(n)$ samples (in altogether $O(n^2logn)$ time and $O(\sqrt{n})$ space). Let $Sample = \cup_{i=1}^{\sqrt{n}} Sample(G_i)$. We then perform binary search on the sorted set *Sample*, at each step of the search using the min-# algorithm and an approximation error ε'. This binary search takes $O(T_{min-\#}logn)$ time. Therefore, this stage uses $O(n^2logn+T_{min-\#}logn)$ time and $O(n)$ **space**. At the end of the binary search of the first stage on the set Sample, we obtain two values a and b that are two consecutive errors in Sample, such that the sought error which is the solution to the min- problem satisfies $a \leq \varepsilon \leq b$. Then, the elements of $L - ERR(P) \cup V - ERR(P)$ between a and b are regarded as currently active, and all other elements are not. So, there are $O(n^{1.5})$ elements of $L - ERR(P) \cup V - ERR(P)$ between a and p.

second stage

We first we first partition the $O(n^{1.5})$ currently active elements into $\sqrt{n}$ subsets G'_i of size $O(n)$ each, by the following steps:

1. Compute ERR_k , for every k with1 $\leq k < n$, and find the number of the currently active elements in ERR_k (by comparing the elements of ERR_k with a and b).

2. (2) Partition the currently active elements of the $ERR-k$'s into n subsets G_i , by associating each G_i with several appropriate $ERR-k$'s; this is done by a simple prefix sum computation on the numbers of the currently active elements in ERR_1 , ERR_2 , . . . , ERR_{n-1} . This partitioning process takes $O(n_2 logn)$ time (on re-computing the $ERR-k$'s) and $O(n)$ space

Next, we compute each set G_i from its associated $ERR-k$'s, sort G_i , and select as a sample every $(\sqrt{n})$-th element from G_i. Note that $|G'_i| = O(n)$ and there are $\sqrt{n}+1$ elements of G_i between every two consecutive samples from G'_i. Let Sample' be the set of such selected samples from all the $\sqrt{n}G'_i$'s. Then $|Sample'| = O(n)$. Performing binary search on Sample' again gives two consecutive errors a and b in Sample' such that $a \leq \varepsilon \leq b$. The binary search of the second stage has the same complexity bounds as the first stage. Between a and b , there are $O(n)$ (currently active) elements of $L-ERR(P) \cup V-ERR(P)$.

third stage

Here we compute the $O(n)$ currently active elements from the $ERR-k$'s, and simply perform binary search on these $O(n)$ elements. This binary search obtains the sought error ε. The third stage is again carried out in the same complexity bounds as the first stage. **This algorithm has $O(n^2 logn + T_{min-\#} logn)$ time and $O(n)$ space complexities**.

7 On using L_1 and L_∞ metrics

The definitions of the L_1 and L_∞ metrics are in the form:

$$L_1\text{-norm}: \ \|x\|_1 = (|x|'_1 + |x|'_2 + ... + |x|'_n)^1$$
$$L_\infty\text{-norm}: \ \|x\|_\infty = max(|x|_1, |x|_2, ..., |x|_n)$$

If we use those definitions for computing the error tolerance region of a point

- L_1-norm : p_i: error tol. region$(p_i, \varepsilon_i) = \{q|D(p, q_i) \leq \varepsilon_i\} = \{q|(|X(q) - X(p_i)| + |Y(q) - Y(p_i)|) \leq \varepsilon_i\}$, which has the form of a diamond.
- L_∞-norm : p_i: error tol. region$(p_i, \varepsilon_i) = \{q|D(p, q_i) \leq \varepsilon_i\} = \{q|max(|X(q) - X(p_i)|, |Y(q) - Y(p_i)|) \leq \varepsilon_i\}$, which has the form of a square, with sides parallels to the axis.

For the min-# problem as well as the min-ε problem, and using the error tolerance region as a square(L_∞) or diamond(L_1), the planes D_{ik}(double cones) idea is still valid and the basic shape of the Dik planes are the same. So the computation of a D_{ik}, the intersection between D_{ik}'s, and testing whether a vertex lies inside of a D_{ik} is unaffected, and the algorithm ideas are still valid.

References

[1] D. Chen and O. Daescu, "Space-efficient algorithms for approximating polygonal curves in two dimensional space," *Computing and Combinatorics*, pp. 45–55, 1998.

[2] D. Eu and G. Toussaint, "On approximating polygonal curves in two and three dimensions," *CVGIP: Graphical Models and Image Processing*, vol. 56, no. 3, pp. 231–246, 1994.

[3] L. Cordella and G. Dettori, "An o (n) algorithm for polygonal approximation," *Pattern recognition letters*, vol. 3, no. 2, pp. 93–97, 1985.

[4] G. Dettori, "An on-line algorithm for polygonal approximation of digitized plane curves," in *Proceedings of the 6th International Joint Conference on Pattern Recognition*, vol. 2, pp. 739–741, 1982.

[5] J. Sklansky and V. Gonzalez, "Fast polygonal approximation of digitized curves," *Pattern Recognition*, vol. 12, no. 5, pp. 327–331, 1980.

[6] W. Chan and F. Chin, "Approximation of polygonal curves with minimum number of line segments," *Algorithms and Computation*, pp. 378–387, 1992.

[7] E. White, "Assessment of line-generalization algorithms using characteristic points," *Cartography and Geographic Information Science*, vol. 12, no. 1, pp. 17–28, 1985.

[8] R. McMaster, "A statistical analysis of mathematical measures for linear simplification," *Cartography and Geographic Information Science*, vol. 13, no. 2, pp. 103–116, 1986.

[9] C. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity.* Dover publications, 1998.

[10] G. Barequet, D. Chen, O. Daescu, M. Goodrich, J. Snoeyink, *et al.*, "Efficiently approximating polygonal paths in three and higher dimensions," *Algorithmica*, vol. 33, no. 2, pp. 150–167, 2002.

[11] G. Toussaint, "On the complexity of approximating polygonal curves in the plane," in *Proceedings of the IASTED International Symposium on Robotics and Automation*, p. 59, 1985.

[12] D. Chen, W. Chen, K. Wada, and K. Kawaguchi, "Parallel algorithms for partitioning sorted sets and related problems," *Algorithms—ESA'96*, pp. 234–245, 1996.

Miscelanea

- Almost all illustrations have been done using GeoGebra 4.0 , a free and open source dynamic mathematics software (more info in `http://www.geogebra.org/`) The illustrations can be downloaded from `https://github.com/viccuad/papers/approx_polygonal_curves/images/geogebras/` in .ggb (geogebra's file extension). Geogebra allows to see the functions associated with all the curves, lines and planar regions, and allows also to move on the fly the points and lines. It has proven a valuable tool to understand all the geometric cases of the definitions that appear when trying to approximate polygonal curves.