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

Victor Cuadrado Juan

January 24, 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

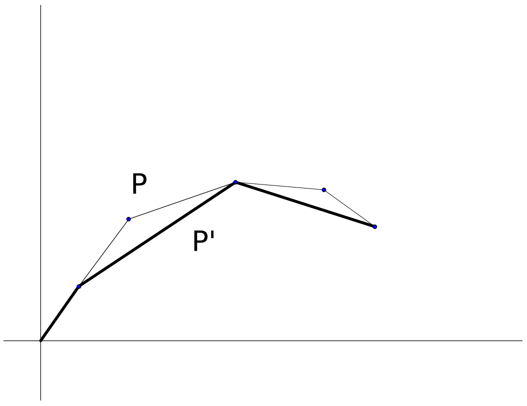

Figure 1: A polygonal curve P and it's simplification, P' (with thick lines).

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, as said in [1]:

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).

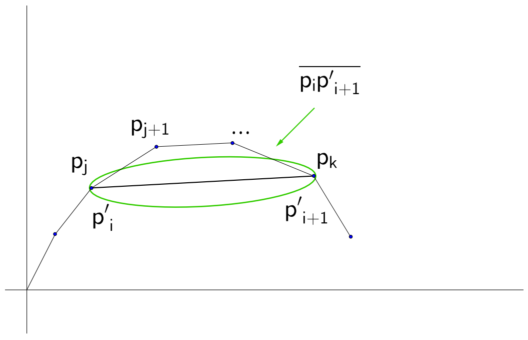

Figure 2: Aprox. line segment of the subcurve $[p_j, p_{j+1}, ..., p_k]$ of P

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

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

With:

- m being the number of vertices of the approximation of P, P',
- ε being the allowed error when creating the approximation P' of P. ε will define the "closeness" of P' to P when is used by an error criterion (we will see some later).

m and ε are related between them: the smaller ε is, the larger m tends to be (if we allow less error when approximating, intuitively we will need more points for creating the approximation), and vice versa (if we allow more error, we can roughly approximate P when less points).

Based on the 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-ε 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

As said in [1], under it, we define 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 as the maximum distance in an L_h metric[1] between $\overline{p_j, p_k}$ and each point on the subcurve S.

because of P being a polygonal curve, the maximum distance between $\overline{p_j, p_k}$ and the points of S can be computed iteratively finding the maximum L_h distance between $\overline{p_j, p_k}$ and each vertex $p_l \in$ S (with $j \leq l \leq k$): we name that distance $dist_h(\overline{p_j, p_k}, p_l)$, and it shall be the L_h distance between $\overline{p_j, p_k}$ and p_l. Using this distance, the tolerance zone has a distinct shape, caused by the semi-circumferences created by ε from p_j and p_k (example at figure 3).

The tolerance zone criterion is also explained in [6], with an equivalent definition, as follows: Based on the tolerance zone criterion, a vertex p_k of P is within distance ε (and then inside the toleranze zone) from a segment $\overline{p_i, p_j}$, with $i \leq k \leq j$, if this three conditions are proven true:

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

2. Condition 2: If the convex angle ($\leq \pi$) defined by $\overline{p_i, p_k}$ and $\overline{p_i, p_j}$ is greater than $\pi/2$, then $d(p_k, p_i) \leq \varepsilon$. $d(p_k, p_i)$ is the L_2 distance between p_k and p_i (intuitively, when defining the segments, we then define a polygon. We always choose the angle between the two segments that falls inside the polygon). (see figure 5)

[1]we consider $h \in \{1, 2, \infty\}$

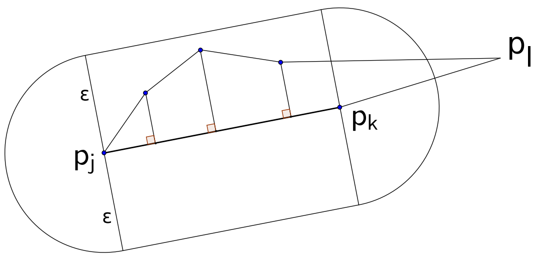

Figure 3: the point p_l is out of the tolerance zone. Note the semicircles with radius equal $\leq \varepsilon$ that are formed by the $\leq \varepsilon$ distances on p_j and p_k

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

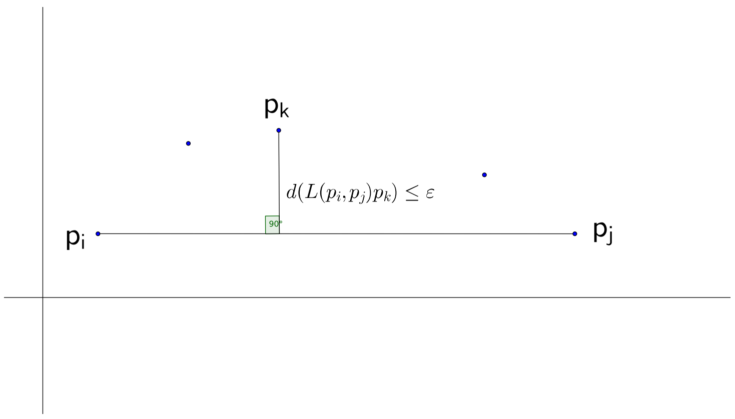

Figure 4: Example of a distance within condition 1.

Infinite beam criterion

As said in [1], under it, we define 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 as the maximum L_h distance between the line (and not the segment, as in the tolerance zone criterion) $L(\overline{p_j, p_k})$ that contains $\overline{p_j, p_k}$ and each point of the subcurve S. $dist_h(L(\overline{p_j, p_k}), p_l)$ shall be 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.

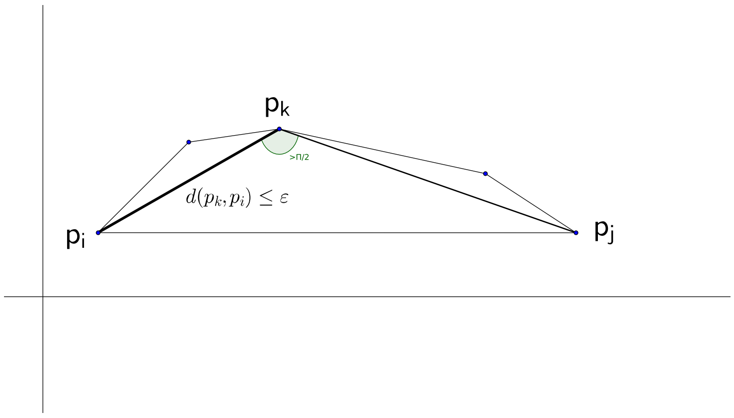

Figure 5: Example of a distance within condition 2.

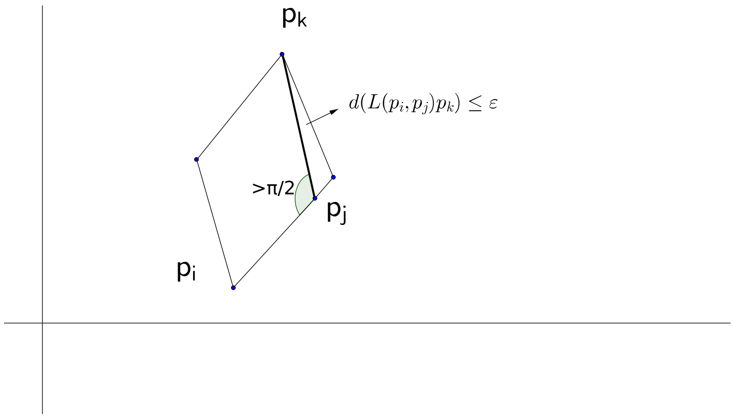

Figure 6: Example of a distance within condition 3.

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 After seeing these error criteria, we can define the approximation error generated when using a curve $P' = [p_1, p_2, .., p_m]$ to approximate P as: ([1])

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

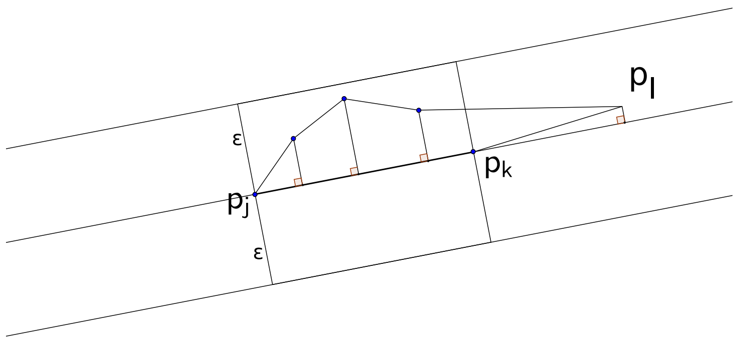

Figure 7: the point p_l is inside of the tolerance zone. Note the infinite region created by the two lines parallel to the line $L(\overline{p_j, p_k})$, each of them at a distance of $\leq \varepsilon$ from $L(\overline{p_j, p_k})$.

This means is the maximum error between the vertixes of P' and P (see figure 8). ε gives the upper limit of the approximation error.

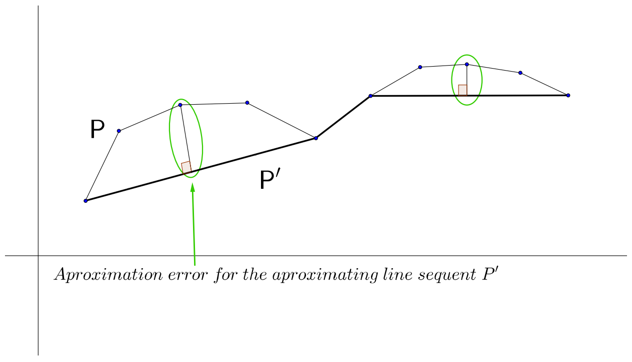

Figure 8: Example of the approximation error for the aproximating line segment P'. Notice is the max between all posibles.

3 Concepts and definitions for the algorithms

Error tolerance region

When a point p_k abides to the three conditions of the tolerance zone criterion (in section 2), it belongs to a region called the *error tolerance region* [2] of the line segment $\overline{p_i, p_j}$. Lets name the line from p_i to p_j $ray(p_i, p_j)$. Because the line segment $\overline{p_i, p_j} = ray(p_i, p_j) \cap ray(p_j, p_i)$, 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 $ray(p_j, p_i)$.

Lets choose three points from P, p_i , p_j , and p_k, $i < k \leq j$.
If $dist(ray(p_i, p_j), p_k) \leq \varepsilon$, that gives us $ray(p_i, p_j)$, which is an approximating ray 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. Therefore, under the tolerance zone criterion for example:
$dist(\overline{p_ip_j}, p_k) \leq \varepsilon \Leftrightarrow dist(\overline{p_ip_j}, p_k) \leq \varepsilon$ and $dist(\overline{p_jp_i}, p_k) \leq \varepsilon$.
($\overline{p_ip_j}$ is an approximating line segment of $p_k \Leftrightarrow ray(p_i, p_j)$ and $ray(p_j, p_i)$ are both approximating rays of p_k) . Therefore, the process to find a set of approximating line segments (and find the error tolerance region, then) is to first compute all approximating rays $ray(p_i, p_j)$, and last, all approximating rays $ray(p_j, p_i)$, $1 \leq i < j \leq n$.

Expressing the error tolerance region

Lets take two vertices p_i and p_k from P. We can build a circumference with radius $= \varepsilon$ and centered in p_k. There appear two tangent lines of the circumference from p_i, which we call r_a and r_b. r_a and r_b form a plane D_{ik}: it will be the whole plane if $d(p_i, p_k) \leq \varepsilon$ (see figure 10), or the convex region between by r_a and r_b (see figure 9). Then, by the conditions of the tolerance region:

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

Lets see now when a segment is a valid approximating line segment. As said in [1]:
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

[2]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.

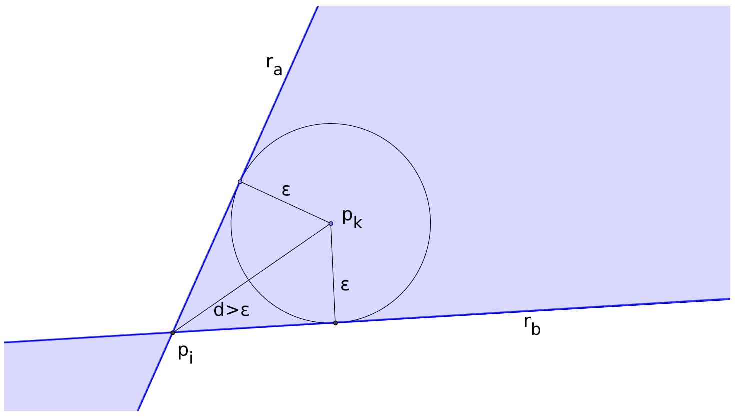

Figure 9: Example of a plane formed by vertices p_i and p_k. Note that r_a and r_b create a double cone. Every point situated on the coloured area is inside the error tolerance region.

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 $e_{ij} \in E$) $\Leftrightarrow pj \in F_{ij}$ and $pi \in B_{ij}$. Note that F_{ij} (resp., B_{ij}) always consists of one (possibly empty) cone. Each F_{ij} or B_{ij} takes only O(1) space to store (3 points per plane, or 1 point, and two inecuations). We can conclude that a $\overline{p_i p_j}$ segment is an approximating line segment of P when $\overline{p_i p_j} \in F_{ij} \cap B_{ij}$.

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 colour. 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 colour. 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

Lets build a directed acyclic graph $G = (V, E)$ for the curve approximation, where:

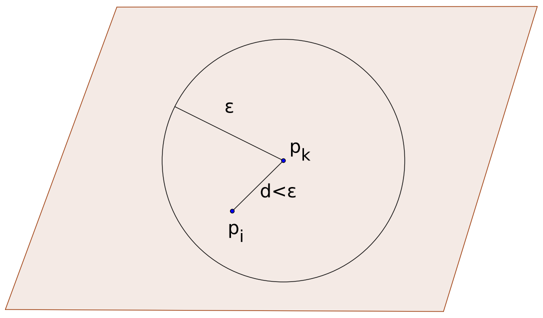

Figure 10: Example of a plane formed by vertices p_i and p_k: note that because of their closeness and being at a distance $\leq \varepsilon$ between them, D_{ik} is the whole plane.

- V is the vertex set of the curve P,
- E is the set of all arcs $e_{ij} = (p_i, p_j)$, $i < j$, $\leftrightarrow$ $\overline{p_i, p_j}$ is the approximating line segment for the chain $S = [p_i, p_{i+1}, .., p_i]$ of P.

Now we define the lenght of the shortest path $SD(p_i, p_j)$, from p_i to p_j, $1 \leq i < j \leq n$, in G. Let $w(e_{ij})$ denote the weight of an edge $e_{ij} \in E$. Because of the graph G is directed acyclic, the following inductive relation holds: $SD(p_1, p_j) = minSD(p_1, p_k) + w(e_{kj})|1 \leq k < j \leq n \; and \; e_{kj} \in E, SD(p_1, p_j) = 0$
We can use an incremental algorithm for computing $SD(p_1, p_j)$ (using the $SD(p_1, p_k)$'s, with $1 \leq k < j$). We will describe only an algorithm for computing the length $SD(p_1, p_n)$ of a shortest p_1-to-p_n path in G, but the algorithm can be easily modified to record, in addition to $SD(p_1, p_n)$, the shortest path tree of G rooted at p_1, which we could use to obtain the shortest path for a curve approximation.

Convex hull of a set of points

Let S be a set of points in the plane. A *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 pegs(see figure 13).

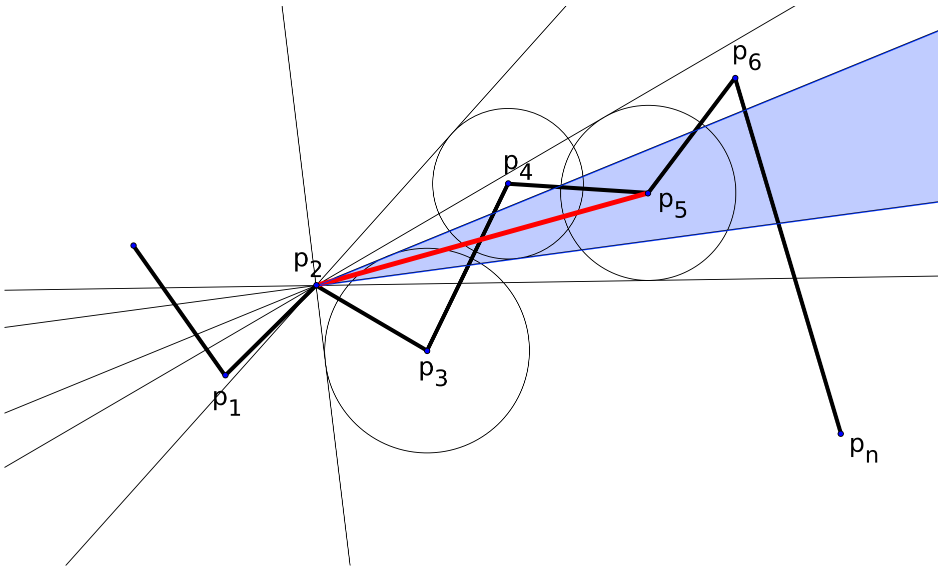

Figure 11: The blued plane is F_{ij}. Only the tangent lines of each of the D_{ik} for calculating F_{ij} have been drawn, and the polygonal line is represented as a thick line.

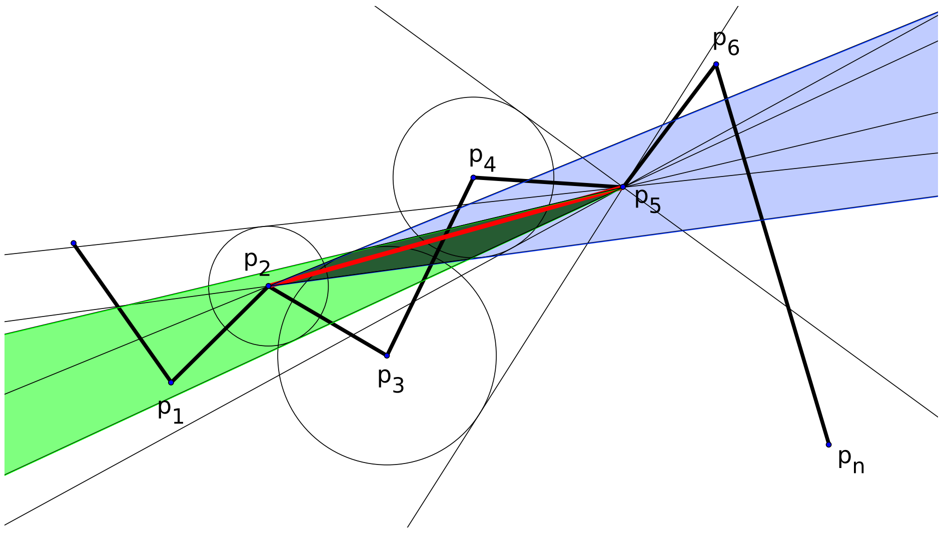

Figure 12: The blued plane is F_{ij}, and the greened plane is B_{ij}. The intersection is in a darker colour. Only the tangent lines of each of the D_{ik} for calculating B_{ij} have been drawn.

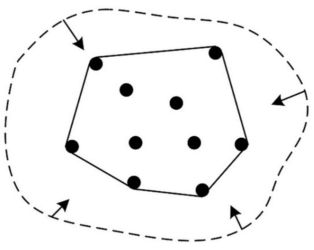

Figure 13: Example of the intuitive definition of a convex hull

4 Algorithms

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

- The *approximating algorithm with $O(n^2)$ space complexity*[3]. 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-ε 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*[4]. Differently to the algorithm with $O(n^2)$ space complexity, it does not need to 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-ε 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.

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

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

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

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 $G = (V, E)$, as seen in 3. 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: $[\varepsilon_1, \varepsilon_2, .., \varepsilon_n]$}
{output: a directed graph G(V,E) of the $O(n^2)$ candidate ↩
    approximating line segments of P}

for i=1 to n do $v_i \leftarrow v_p$ (set vertices of V to correspond↩
     with those on P)
for i=1 to n-1 do
begin
  incrementalPlane $\leftarrow R^2$ (set to an open cone)
  for j=i+1 to n do and meanwhile incrementalPlane $\neq \varnothing$ (↩
      meanwhile $[p_i, p_j]$ continues to be an approximating ↩
      segment)
  begin
    if $p_j \in$ incrementalPlane then $e_{ij}$<-$[p_i, p_j]$ with weight (↩
       j-i-1)
    incrementalPlane $\leftarrow$ incrementalPlane $\cap D_{ij}$
  end
end
```

Note the 2 nested 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 forward dynamic programming technique [9] is used. The time complexity is proportional to |E|: $(O(n^2))$. **Because we need each e_ik, the space complexity for maintaining G is $O(n^2)$: this gives $O(n^2)$ time and $O(n^2)$ space complexities for the whole algorithm**.

Under the infinite beam criterion

With uniform error tolerances for the vertices in P ($\forall$ p_i in P, $\varepsilon_i = \varepsilon$), **the build_graph algorithm resolves 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.

Lemma, in [2] Let ε_A, $\varepsilon_B > 0$ be two error tolerances (see figure 14 for an example of an error tol. zone inside a infinite beam). 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 $\varepsilon_A > \varepsilon_B$

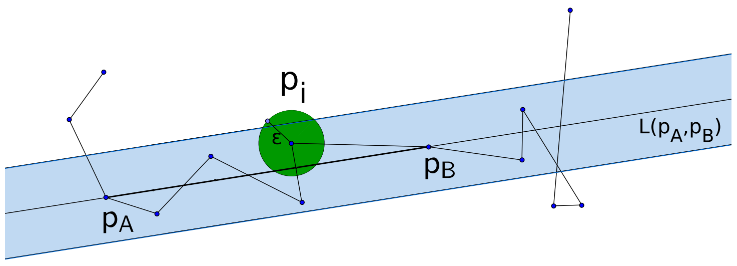

Figure 14: Example of a point p_i with it's error tolerance region created by its ε, and the infinite beam region created by p_A and p_B.

If we store in a sorted array Q the maximum error for each of the $O(n^2)$ segments of pairs of vertices of P, then we can apply a binary search on Q (because of this lemma), 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 recur 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 recur to the upper half of Q.

(C) $|P'| = m$: it is possible that an element $\varepsilon' < \varepsilon$ exists in the lower half and yields $|P'| = m$ as well (as said in the previous lemma,5). We recur to the lower half. Otherwise, we return with the curve of error ε.

The algorithm associated with this binary search, as seen in [2], 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 $p_A, p_B \in P$, $A < B$, ↩
    $err_{ab} = max_{A<i<B} D(p_i, L(p_A, p_B))$. 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) $\varepsilon$ $<-$ Q[(i+j)/2], the median element in {Q[i], ... ,↩
    Q[j]}
(5) solve the min-# problem using build_graph with the ↩
    uniform error $\varepsilon$ 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 $\varepsilon$ (the element to the left of↩
           our median < 1)
    else
      begin (recurre the lower half)
      $\varepsilon$' $<-$ Q[(i+j)/2 -1]
      invoke build_graph with $\varepsilon$' to obtain P'' with m** ↩
          vertices
      if m** = m* then return P'' with error $\varepsilon$'
      else return P' with error $\varepsilon$
      end
    end
  else if m* $\leq$ 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.**

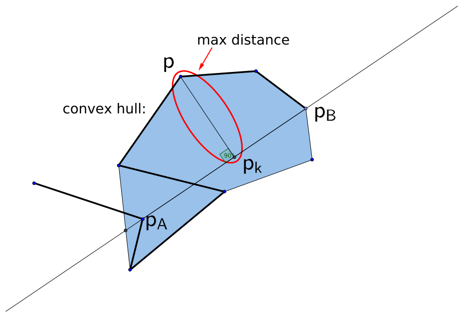

Figure 15: Example of the max error vertex p, using a convex hull.

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, as is said in [1].

That technique consists of $O(1)$ stages, each of which perform:

1. Take $O(n)$ samples from the set S of all the elements that are currently being used for the current call of the binary search, so that between any two of them there is a probably "small" subset of S.
2. Make the binary search call 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 (so only the elements of that subset are going to be active).

For (1), we need to sort all the $O(n^2)$ approximation errors into n sets (each will be of size $O(n)$). For the tolerance zone criterion, let

$$err(\overline{p_ip_j}) = max_{ki}^{j}\{dist(\overline{p_ip_j}, pk)\}$$

denote the approximation error of the segment $\overline{p_ip_j}$. Or, if we are working with lines (infinite beam criterion), let

$$err(L(\overline{p_ip_j})) = max_{ki}^{j}\{dist(L(\overline{p_ip_j}), pk)\}$$

denote the approximation error of the line $L(\overline{p_ip_j})$.

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
$err(\overline{p_ip_j})|1 \leq i < j \leq n \subseteq err(L(\overline{p_ip_j}))|1 \leq i < j \leq n \cup d(p_ip_j)|1 \leq i \leq j \leq n$.
We can call:

- $ERR(P) = err(\overline{p_ip_j})|1 \leq i < j \leq n$,
- "line error" $L\text{-}ERR(P) = err(L(\overline{p_ip_j}))|1 \leq i < j \leq n$, and
- "vertex error" $V\text{-}ERR(P) = d(p_ip_j)|1 \leq i \leq j \leq n$.

Because $L\text{-}ERR(P) \cup V\text{-}ERR(P)$ is a superset of $ERR(P)$, the error $\varepsilon \in ERR(P)$ that is the solution for the 2-D min-ε problem is contained in $L\text{-}ERR(P) \cup V\text{-}ERR(P)$. So, we search for the $\varepsilon \in ERR(P)$ by performing binary search on $L\text{-}ERR(P) \cup V\text{-}ERR(P)$. Note that $|L\text{-}ERR(P) \cup V\text{-}ERR(P)| = O(n^2)$. For all i such $1 \leq i < n$, let:

- $L\text{-}ERR_i$ be the set of the approximation errors $err(L(\overline{p_ip_j}))$ for all the lines$L(\overline{p_ip_j})$, and
- $V\text{-}ERR_i$ be the set of the $d(p_i, p_j)$'s.

For all j with $i \leq j \leq n$ (assume that $err(L(p_ip_i)) = 0$ there is no distance to compute there). Note that all the errors in $L\text{-}ERR_i$ can be computed in $O(nlogn)$ time and $O(n)$ space by maintaining on-line convex hulls of planar points (Toussaint's approach [11]). Also, $V\text{-}ERR_i$ can be computed in $O(n)$ time and space. Let $ERR_i = L\text{-}ERR_i \cup V\text{-}ERR_i$. Then we can obtain $|ERR_i| \leq 2n$ and ERR_i in $O(nlogn)$ time and $O(n)$ space. Since $L\text{-}ERR(P) \cup V\text{-}ERR(P) = \cup_{i=1}^{n-1} ERR_i$, the $O(n^2)$ approximation errors of $L\text{-}ERR(P) \cup V\text{-}ERR(P)$ are put into the sets ERR_1 , ERR_2 , ... , ERR_{n-1} . Without loss of generality, its assumed that errors in $L\text{-}ERR(P) \cup V\text{-}ERR(P)$ are distinct and ties can be easily undone.

The following lemma, used in many selection algorithms before (see [12]), is crucial to this $O(n)$ space binary search algorithm [1].

Lemma, in [1] 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 $\leq z$). (see the proof in [13], and 16)

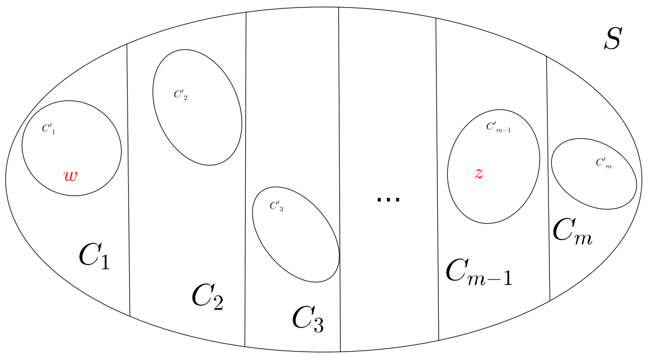

Figure 16: Construction of sets of lemma 6. If: $w < z \rightarrow$ nº of elements k, $w \leq k \leq z$ is $s(\beta - \alpha + m - 1)$. Note that C_i sets are sorted sets.

We are at the beginning of our first phase: from the sets ERR_1 , ERR_2 , . . . , ERR_{n-1}, each of size $O(n)$, lets pick the $O(n)$ sample elements needed for the binary search process. The basic idea for the sampling, as in [1], is: Partition the n - 1 sets of approximation errors ERR_k into $\sqrt{n}$ groups G_i of (more or less) $\sqrt{n}$ sets each. Treat each group G_i (of size $O(n^{1.5})$) as one single sorted set and select $O(\sqrt{n}) = O(n^{0.5})$ sample elements from G_i. Now there are $O(n)$ elements of G_i between every two consecutive samples from G_i and the total number of selected samples from all the $\sqrt{n}$ groups is $O(n)$ $(O(n^{1.5}) - O(n^{0.5}) = O(n))$. but because the purpose of this method is to only use O(n) space, its not possible to explicitly store a group G_i for the sampling process. Instead, its needed to sample from the $\sqrt{n}$ sets of $_Gi$ with the following procedure, as seen in [1]:

procedure sampling(G_i)

1. As seen, for each set of approximation errors ERR_k of G_i, first compute ERR_k and sort it. After that, lemma 6 is applied: every $(\sqrt{n})$-th element from the sorted set ERR_k is selected, and the selected elements are put in the set S_i for G_i.

2. At the end, S_i is sorted. Then every $(\sqrt{n})$-th element from S_i is chosen. These chosen elements are the ones wanted, and form the samples of G_i, which are stored into the wanted set Sample(G_i) (of size $O(\sqrt{n})$). The quality of those samples obtained in Sample(G_i) is ensured because there are $O(n)$ elements of G_i between every two consecutive samples in Sample(G_i).

The cost in time of the procedure is $O(n^{1.5} logn)$: we need to compute and sort each of the n sets ERR_k of G_i.The cost in space is $O(n)$: there is only need to store once every ERR_k in Step 1, and store the set S_i of size $O(n)$.

Algorithm

Now, after knowing how the sample is made, we can apply and this algorithm, consisting in three stages, as seen in [1]:

first stage

We perform the said procedure sampling on each of the $\sqrt{n}$ groups G_i and obtain $O(n)$ samples (with $O(n^2 logn)$ time and $O(\sqrt{n})$ space boundaries). We construct our sample set as $Sample = \cup_{i=1}^{\sqrt{n}} Sample(G_i)$. 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' from *Sample.* This binary search takes $O(T_{min-\#} logn)$ time[5], so this stage uses $O(n^2 logn + T_{min-\#} logn)$ time and $O(n)$ **space**.

After performing the binary search on *Sample*, we have two values, ε_a and ε_b, which are consecutive errors, $\in$ *Sample*, such that the sought error which is the solution to the min-ε problem satisfies $\varepsilon_a \leq \varepsilon \leq \varepsilon_b$. We call now the elements between ε_a and ε_b and in $L\text{-}ERR(P) \cup V\text{-}ERR(P)$ *currently active.* Every other element is called not active. The number of *currently active* elements is in $O(n^{1.5})$.

second stage

We arrive here with $O(n^{1.5})$ currently active elements. We partition that set into $\sqrt{n}$ subsets G'_i of size $O(n)$ each, by the following steps:

1. Calculate ERR_k, $\forall k \in 1 \leq k < n$, and then, comparing the elements of ERR_k with ε_a and ε_b, find the number of *currently active* elements in ERR_k.

[5] $T_{min-\#}$ being the time cost of a min-# algorithm

2. Once we have the *currently active* elements of ERR-k's, separate them into n subsets G'_i, by associating each G'_i with multiple appropriate ERR-k's. This can be achieved by a simple prefix sum computation on the numbers of the *currently active* elements $\in ERR_1$, ERR_2 , . . . , ERR_{n-1}. Because of the recomputing of the ERR-k's, this second step takes $O(n_2 logn)$ time, and **we mantain $O(n)$ space**.

After those steps, we then find each set G'_i from its associated ERR-k's. Then we sort every G'_i, and select as a sample every $(\sqrt{n})$-th element from G'_i (as in lemma 6). $|G'_i| = O(n)$ and between every two consecutive samples in G'_i, there are $\sqrt{n} + 1$ elements. Lets call *Sample'* the set of the selected samples from all the $\sqrt{n}G'_i$'s elements (therefore $|Sample'| = O(n)$). If we apply a binary search on *Sample'* again, we find two consecutive errors ε'_a and ε'_b in Sample' such that $\varepsilon'_a \le \varepsilon \le \varepsilon'_b$ (this binary search is within the same complexity bounds as the one on the first stage). In the outcome of the binary search, ε'_a and ε'_b, as seen in lemma 6, **there are $O(n)$ (currently active) elements** of L-$ERR(P) \cup V$-$ERR(P)$, and thus, **the binary search is performed on $O(n)$ samples**.

third stage

Finally, we use the currently active elements from the ERR-k's (in $O(n)$), and perform a binary search. This binary search obtains the error ε we want to find. This stage has the same complexity bounds as stage 1.

So, at the end **this algorithm has $O(n^2 logn + T_{min-\#} logn)$ time and $O(n)$ space complexities**, $T_{min-\#}$ being the time complexity of a min-# algorithm.

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$(p_i, \varepsilon_i) = \{q | D(p, q_i) \le \varepsilon_i\} = \{q | (|X(q) - X(p_i)| + |Y(q) - Y(p_i)|) \le \varepsilon_i\}$, 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 D_{ik} 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.

[13] H. Imai and M. Iri, "Computational-geometric methods for polygonal approximations of a curve," *Computer Vision, Graphics, and Image Processing*, vol. 36, no. 1, pp. 31–41, 1986.

Miscelanea

- All illustrations but figure 13 have been done using GeoGebra 4.0 (`http://www.geogebra.org/`), a free and open source dynamic mathematics software (thanks to Miguel González Pérez for pointing me to it and teaching me how to use it). 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.