Application of Polygons Intersection to Safe Communication in Networks

Moustafa Nakechbandi and M.A. Aziz-Alaoui
LIH and LMAH - Université du Havre
25 rue Ph. Lebon - BP 540
76058 Le Havre Cedex - France
E-mail : {moustafa.nakechbandi,aziz.alaoui}@univ-lehavre.fr

Keywords

Polygon intersection, Chaotic system, Synchronization, Computational Geometry, Graphics algorithms.

Abstract

The possibility to master and even to exploit chaotic systems has been for some years, very established, see for instance [7, 17, 18, 3, 6, 4, 23, 11] and references therein. As an example of application, one can mention the use of the chaos in the growth of the power of a laser, the control of the oscillations in chemical reactions, the synchronization between electric circuits used for coding of electronic messages for a better security in the communications networks, etc.... For more details on chaos and synchronization, see for example [9, 23].
The first part of this work consists in making the link between the *cryptographic method* of the useful messages that the user wishes to secure and the chaotic Synchronization. The second part presents a method of encryption based on the intersection of the polygons. In the third part, a state of the art of polygons intersection algorithms is discussed [19].

1 INTRODUCTION

The rapid increase of information flow which circulates inside transmission networks has made classical communication tools less sure. Indeed, classical encoding methods have become ineffectual because of their relative slowness under a high flow of information. By cracking information codes, the secret message can be detected.
That was why, over the last few years, there has been a growing interest in developing communications using chaotic signals. Such a problem generates many **mathematical** problems which are, in particular, connected with the problems of recovering trajectories of chaotic dynamical systems from their $\varepsilon-$perturbed trajectories, (this is connected to the theory of *shadowing* which is important from the numerical point of view). Here, we have used a new notion of synchronization which is different from the classical one, see [3]. Indeed, we have no subsytems to synchronize, but we construct a sequence of intervals, or polygons, containing a sequence of points which tends towards the unknown true trajectory. This kind of mathematical problem appears to be equivalent to the extraction of an information bearing signal from an input signal in order to reduce the noise in an input signal, to encode/decode electronic messages for secure communications or to increase and stabilize the power of lasers.
For a chaotic dynamical system, where there is a sensitive dependence on initial data and every trajectory on the strange attractor is unstable, the problem of extraction of efficient signal from input signal in receiving system is difficult, it is unlike of the case where almost each trajectory of a dynamical system is Lyapunov stable, for which detailed methods of solutions of this problem are available.
The elaboration of methods of extraction of efficient signal from input signal for chaotic dynamical system is still a very attractive problem. It leads, among other, to the problem of polygons intersection computation. Note also, that there are different approaches to investigate such a type of task (for instance, [2, 3, 7, 14, 16, 17, 18, 20, 21]).
This is the purpose of our paper in which present some results on recovering the trajectories of chaotic dynamical systems (in particular, systems given by piecewise linear maps) from their ε-perturbed trajectories. Moreover, we give some algorithms for computing polygons intersection, to carry on the work given in [3, 4].

2 DEFINITIONS AND ARGUMENTS

In this section we will recall the notion of the recovering trajectories property in dynamical systems and will give some relevant results, see [3].
Let $\{X, f\}$ be a dynamical system defined by the map $f : X \to X$, where $X \subset R^k \quad (k \geq 1)$, and let $\|\cdot\|$ be a metric in R^k.

Definition 1 *A sequence* $\{x_i\}_{i=0}^{n}, n \leq \infty$, *is called a true trajectory of the dynamical system if* $x_{i+1} = f(x_i)$. *A se-*

quence $\{x_i\}_{i=0}^n, n \leq \infty$, is called an ε-trajectory of the dynamical system if $\|x_{i+1} - f(x_i)\| < \varepsilon$.

The notion "ε-trajectory" (or "pseudotrajectory", which is the same) is an object of modern theory of dynamical systems [1],[15]. Because numerical trajectories are always only ε-trajectories (where ε is calculation precision), the question "Do ε-trajectories represent true trajectories?" [12], [13] is pertinent.

Definition 2 *A sequence $\{\overline{x}_i\}_{i=0}^n$ is called an ε-perturbed trajectory of the dynamical system, if there exists a true trajectory $\{x_i\}_{i=0}^n$ such that $\|\overline{x}_i - x_i\| < \varepsilon$ for all i.*

According to these definitions every ε-perturbed trajectory is an ε-trajectory too. But the converse is not true (an ε-trajectory is not always an ε'-perturbed trajectory of some true trajectory with $\varepsilon' \leq \varepsilon$).
Nevertheless, there are classes of dynamical systems (for instance, structurally stable systems), for which every ε-trajectory with $\varepsilon < \varepsilon_0$ (which depends on the dynamical system) is an ε'-perturbed trajectory and, moreover, it is possible to take ε' as small as desired. Such a property of ε-trajectories of dynamical system is called shadowing property, see [1].
Our purpose now is to recover trajectories of a chaotic dynamical system from their ε-perturbed trajectories.

Definition 3 *We will say that a ε-perturbed trajectory $\{\overline{x}_i\}_{i=0}^n$ of the dynamical system determines a true trajectory of this system on some time interval $[m_-, n - m_+] \subset [0, n]$ with the accuracy δ, if*
a) there is a trajectory $\{x_i\}_{i=0}^n$ such that $\|\overline{x}_i - x_i\| < \delta$ for $m_- \leq i \leq n - m_+$ and $\|\overline{x}_i - x_i\| < \varepsilon$ for the others i;
b) if $\{x_i'\}_{i=m_-}^{m_+}$ and $\{x_i''\}_{i=m_-}^{m_+}$ are two such trajectories, then $\|x_i' - x_i''\| < \delta$ for all $m_- \leq i \leq n - m_+$.

Of course, it is desirable to deal with dynamical systems, for which every ε-perturbed trajectory determines a true trajectory with an accuracy given in advance.

Definition 4 *We will say that a dynamical system has the recovering trajectories property, if for this dynamical system, there exists $\varepsilon_0 > 0$, such that for every $\varepsilon < \varepsilon_0$ and any δ there are integers $m_-, m_+ \geq 0$ such that every ε-perturbed trajectory, given on the time interval $[0, n]$ with $n > m_- + m_+$, defines a true trajectory with accuracy δ on the time interval $[m_-, n - m_+]$.*

For one-dimensional piecewise discontinuous maps, which have uniformly separated preimages of any point, the problem of recovering the trajectories is solved identically (the preimages are separated uniformly, if there is δ such that, if $f(x') = f(x'')$, then either $x' = x''$, or $\|x' - x''\| > \delta$).
Both integers m_- and m_+ determine the speed of recovering the true trajectory; m_- corresponds to "the synchronization time" for one part of (relative) coordinates, m_+ corresponds to "the antisynchronization time" for the other part of coordinates. It is expected naturally, that they depend on Lyapunov exponents of the "strange" attractor. It is not difficult to see, that for structurally stable systems, m_-, m_+ can be estimated from the expressions

$$\varepsilon \lambda_-^{m_-} \approx \delta, \quad \varepsilon \lambda_+^{-m_+} \approx \delta$$

where

$$\lambda_- = \max_{|\lambda_j|<1} |\lambda_j|, \quad \lambda_+ = \min_{|\lambda_j|>1} |\lambda_j|,$$

and $\lambda_j, j = 1, 2, \ldots, k$, are the multipliers of the attractor of the dynamical system considered.
In this paper we consider dynamical systems with only discrete time given by maps.

3 ALGORITHM OF RECOVERING THE TRAJECTORIES

3.1 Description of the algorithm, see [3].

Let a dynamical system given by the map $f : X \to X$, $X \subset R^k, k \geq 1$, having the recovering trajectories property with some $\varepsilon_0 > 0$. As mentioned, the map f must either be a one-to-one map, or possess the property of δ_0-separation of preimages with some $\delta_0 \geq \varepsilon_0$ (so that the map f is not a continuous 1-D map).
Let us suppose that the dynamical system has a hyperbolic chaotic attractor, so that both trajectories on the attractor, and these attracted by the attractor, have a sensitive dependence on initial conditions. Moreover, the absolute value of the multipliers $\lambda_1, \ldots, \lambda_k$ of this attractor (and, consequently, the absolute value of the multipliers for almost all of such trajectories) is not equal to 1.
For a given δ-accuracy, which is necessary for the reconstruction of the "true" trajectory, if we know $\lambda_- = \max_{|\lambda_s|<1} |\lambda_s|$, and $\lambda_+ = \min_{|\lambda_s|>1} |\lambda_s|$, we can evaluate the "recovering times" m_-, m_+.
Let $\{\overline{x}_i\}_{i=0}^n, n \leq \infty$, be an ε-perturbed trajectory of this dynamical system with some $\varepsilon \leq \varepsilon_0$. How to compute a "true" trajectory $\{x_i\}_{i=m_-}^{n-m_+}$?
This question, in the case of piecwise linear dynamical systems, directly induces the hard problem of intersection of polygons, as we will see below.

Let $B_\varepsilon(\overline{x}) = \{x \in X : \|x - \overline{x}\| \leq \varepsilon\}$. As far the point $\overline{x}_i$ of the ε-perturbed trajectory is distant from a point x_i we have to find (not farther then of ε), then $x_i \in B_\varepsilon(\overline{x})$. Due to the hyperbolicity of the dynamical system, for ε small enough the intersection of the ball $B_\varepsilon(\overline{x})$ with the hyperplane corresponding to the "stable manifold" of the trajectory through the point x, is contracted under the map f on the average in λ_- times and, simultaneously, the intersection with the hyperplane corresponding to the "unstable manifold" of the trajectory through the x, is expanded on average in λ_+ times, at least.

That is why, the recovering trajectory algorithm can be the following:
1) We construct the sets

$$B_\varepsilon(\overline{x}_0), B_\varepsilon(\overline{x}_1), \ldots, B_\varepsilon(\overline{x}_i), B_\varepsilon(\overline{x}_{i+1}), \ldots,$$

each of the balls $B_\varepsilon(\overline{x}_i)$, as we know, contains an unknown point x_i.
2) If both "times" m_- and m_+ differ a little, using the sets $S_i^0 = B_\varepsilon(\overline{x}_i), i = 0, 1, 2, \ldots$, of "level" 0, we sequently construct sets of "levels" $j = 1, 2, \ldots, m$, where $m = \max\{m_-, m_+\}$, by the formula

$$S_i^j = f(S_{i-1}^{j-1}) \cap S_i^{j-1} \cap f^{-1}(S_{i+1}^{j-1}), \quad i = j, j+1, j+2, \ldots,$$

where f^{-1} is the inverse function of f (f^{-1} may be multivalued, it is always multivalued when $k = 1$).
3) At the typical situation we get that diam $S_i^m < \delta$, $i = m, m+1, \ldots$, and we can take any point from the set S_i^m as x_i.
If both "times" m_- and m_+ differ drastically it is better to modify a few the algorithm and to define the sets S_i^j starting with the "level" $j = m_* = \min\{m_-, m_+\} + 1$ by the formula

$$S_i^j = S_i^{j-1} \cap f^{\mp 1}(S_{i\pm 1}^{j-1}), \quad i = m_*, m_* + 1, \ldots$$

with f^{-1} if $m_- < m_+$ and $f^{+1} (= f)$ if $m_- > m_+$.
In the case, where all multipliers are larger than 1 in absolute value (this is always the case when $k = 1$), the algorithm is somewhat simplified and the formula for S_i^j is transformed into the form

$$S_i^j = S_i^{j-1} \cap f^{-1}(S_{i+1}^{j-1}), \quad i = 0, 1, 2, \ldots.$$

In this case it is possible to reconstruct the trajectory, starting from the first point.

3.2 Algorithm for one-dimensional dynamical system

Since the inversed map f^{-1} is multivalued, then computing the true trajectory $\{x_i\}_{i=0}^{n-m_+}$, it is necessary to walk on "nonregular" (with variable branching) tree defined by intersections of preimages of the interval with the intervals themselves. For carrying out efficient computations (cutting of the empty intersections and doing a rational passage along the tree) it is reasonable to use the recursion for the construction of the algorithm.
We will explain the algorithm using the operators of the pseudo-C language.
Input:
– the map (for simplification we will restrict ourselves by using a prime model)

$$x_{i+1} = f(x_i) = \begin{cases} f_l(x_i) = ax + b, & \text{if } 0 \le x_i \le p; \\ f_r(x_i) = cx + d, & \text{if } p < x_i \le 1; \end{cases}$$

– ε-perturbed trajectory $\{\overline{x}_i\}_{i=0}^n$;
– ε is an evaluation of perturbations;
– δ is the accuracy used in recovering the true trajectory.
Output:
the true trajectory $\{\tilde{x}_i\}_{i=0}^{n-m_+}, m_+ = m(\delta)$, recovered with given accuracy, where $m_+ = m(\delta)$ is computated in the algorithm.
Algorithm:

```
proc recovering(f, {x̄_i}_{i=0}^n, ε, δ)
    λ = Lyap(f); // estimate the multiplier λ = λ_+
    // for the map
    m_+ = m(δ, λ); // define an evaluation m_+ – a
    // number of steps, which are necessary for
    // recovering the true trajectory
    for j = 0 to n − m_+ do
        B(x̄_j) = [x̄_j − ε; x̄ +_j +ε];
        inv_recover(f, {x̄_i}_{i=j}^{j+m_+}, B(x̄_j), m_+);
    end // for
end // proc recovering
proc inv_recover(f, {x̄_i}_{i=0}^{m_+}, B(x̄_i), m)
    if m = 0 then
        output x̃ ∈ B(x̄_i);
    B^l(x̄_{i−1}) = f_l^{−1}(B(x̄_i)) ∪ B(x̄_{i−1});
    B^r(x̄_{i−1}) = f_r^{−1}(B(x̄_i)) ∪ B(x̄_{i−1});
    m = m − 1;
    if B^l(x̄_{i−1}) ≠ ∅ then
        inv_recover(f, {x̄_i}_{i=0}^{m_+}, B^l(x̄_{i−1}))
    if B^r(x̄_{i−1}) ≠ ∅ then
        inv_recover(f, {x̄_i}_{i=0}^{m_+}, B^r(x̄_{i−1}))
end // proc inv_recover
```

3.3 Algorithm for dynamical system with 2-D piecewise map

In contrast to this one-dimensional case, due to the complexity of solving the problem, we will use here a piecewise linearity of a nonlinear map as maximum, as is possible and as a basic model let us take the dynamical system defined by the Lozi map,

$$\begin{cases} x_{i+1} = -a|x_i| + y_i + 1; \\ y_{i+1} = bx_i. \end{cases}$$

This system present a classical chaotic attractor given by figure 1.
In the dynamical system with the Lozi map for any convex polygon V_r, which completely lies in the right (V_l – in the left) half-plane R^2, its image/preimage is also a convex polygon. It is necessary to preserve such a property during the computations in order to use efficient algorithms of determination of the polygon [10]. Therefore, it is necessary to divide S into two polygons S_r and S_l, belonging to right and left half-planes R^2 respectively, if image/preimage S of the convex polygon $\overline{S}_i, m_- \le i \le n$ (or $\tilde{S}_i, 0 \le i \le n - m_+$) intersects the line $x = 0$.
Now the other concept of application of the recursion to create a recovering algorithm, precisely, computations, by

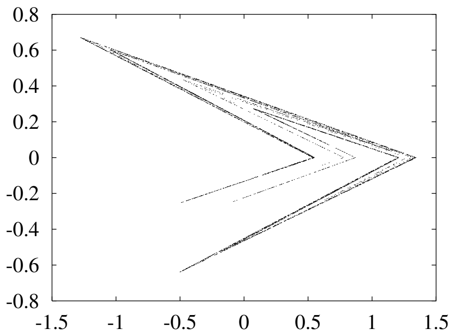

Figure 1: Phase portrait of the Lozi map for the parameters $a = 1.7$ and $b = 0.5$.

preservation convexity of all polygons $\overline{S}_i, \tilde{S}_i$, is fulfilled on some "nonregular" trees and, therefore, it is necessary to simplify the circuit along the tree as much, as possible, at the expense of cutting-off "empty" branches, which will appear on the intersection of polygons.

The structure of the algorithm of recovering trajectory of the dynamical system with Lozi map is developped using the same method, which has been illustrated in the construction of our algorithm for one-dimension case and with the use of all needed methods of computing geometry.

4 INTERSECTION OF POLYGONS

In general, the intersection of a polygon with n vertices with another polygon having m vertices will have O(nm) vertices. However, if we intersect two convex polygons, the intersection will have O(n+m). In other words, the intersection of two convex polygons is linear in the size of the inputs (figure 2).

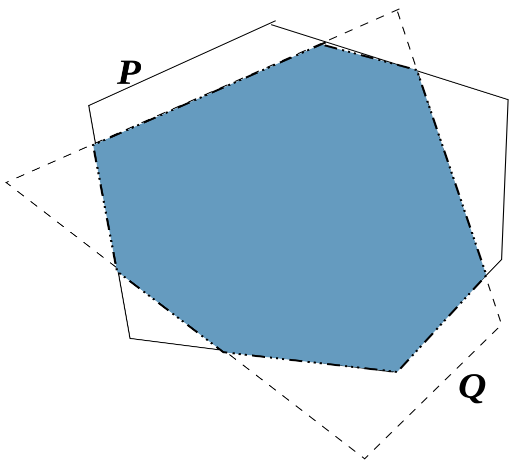

Figure 2: Intersection of polygons

4.1 Intersection of convex Polygons

Several algorithms that intersect two convex polygons in linear time have been invented. Here, we present tree algorithms, one is a trivial algorithm the others given by O'rourke [19], and by Toussain [22].

Trivial algorithm
Make the intersction of all edges by synchronized scan of edges of P and Q, so that all intersection points are eventually found and *inner* intersection boundary is known at each step.

O'rourke Algorithm
The central point of O'rourke's algorithm is to keep a *current* edge on each polygon, and to march both edges counter-clockwise around both polygons. The idea is to have the edges *chase* each other in such a way that the intersection points will all be found. This means that neither edge can get too far ahead of the other. As O'rourke puts it, the basic idea of the algorithm is straightforward, but the translation of the idea into code is delicate (as is often the case).

Toussain Algorithm
This is a simple O (n + m) algorithm for computing the intersection of tow convex polygons.

Unlike previous algorithms, the new algorithm consists of a two-step combination of two simple algorithms for finding convex hulls and triangulations of polygons.

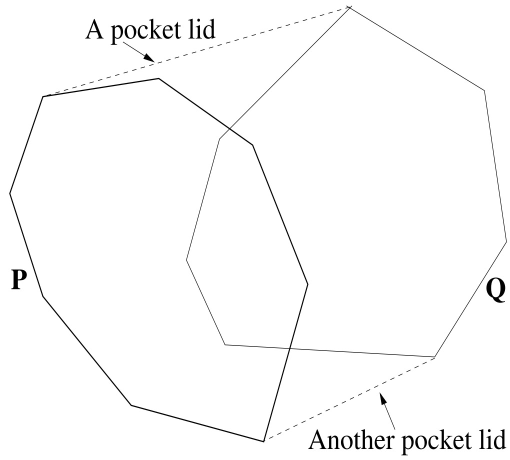

Figure 3: Pocket lids

Let P and Q be two convex polygons whose intersection is a convex polygon. The algorithm for finding this convex intersection polygon can be described by these three steps:

1. Construct the convex hull of the union of P and Q;
2. For each pocket lid L_i (see figure 3) of the convex hull, find the intersection of P and Q that lies in the pocket;

3. Merge together the polygonal chains between the intersection points found in 2.

notes

- A pocket lid is a line segment belonging to the convex hull of the union of P and Q, but which belongs t o neither P nor Q.
- A pocket lid connects a vertex of P with a vertex of Q; if it were to connect two vertices of P, then P would not be convex, since the lid lies on the convex hull and is not a segment of P.

4.2 Intersection of non convex Polygons

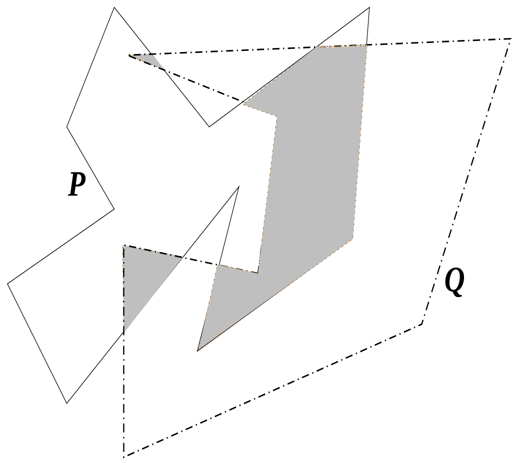

Figure 4: Intersection of non convex polygons

The solution of the intersection of polygons in the general case (figure 4) is based on the computing of all intersections of a set of line segments (figure 5). So this is the object of the following paragraph.

4.2.1 Find all intersections line segments

We are going to present some methods say naive, then the method of Bentley and Ottman [5]. An algorithm, optimal, has been proposed in the community of the algorithmic geometry, by Chazelle and Edelsbrunner [8], but, to our knowledge, it is not used in images synthesis because it is very difficult to programing.

Naive algorithm

The first idea is based on the testing of the intersection for all pairs of segments. This naive algorithm can be improved by classifying the segments from the left to the right and only comparing the segments which the projections on the Ox axis are superimposed (because

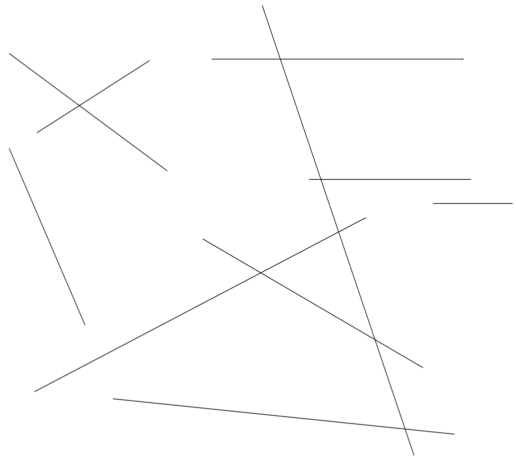

Figure 5: intersections of a set of line segments

otherwise, the cutting is impossible). More precisely, let S_l, S_2.... S_n the segments sorted from the right to the left, with $S.x_1$, $S.y_1$ the coordinates of the left summit of S, $S.x_2$, $S.y_2$ the coordinates of the right summit of S. One has therefore $S_1.x_1 \leq S_2.x_1 \leq \ldots \leq S_n.x_1$. We Have now :

For $i = 1$ to $n-1$ do
  $j = i + 1$;
  while $S_j.x_1 \leq S_i.x_2$ do
    intersect S_i and S_j ;
    j = j + 1;
  end while;
end for;

This method is easy to program and sufficient for not too big n.

The Bentley-Ottmann Algorithm

The input for the Bentley-Ottmann algorithm is a collection $\Omega = \{L_i\}$ of line segments L_i, and its output will be a set $\Lambda = \{I_j\}$ of intersection points. This algorithm is referred to as a "sweep line algorithm" because it's operation can be visualized as having another "sweep line" SL sweeping over the collection Ω and collecting information as it passes over the individual segments L_i. The information collected is basically an ordered list of all segments in Ω that are currently intersected by SL. The data structure maintaining this information is often also called the "sweep line". It also detects and outputs intersections as it discovers them. The process by which it discovers intersections is the heart of the algorithm and its efficiency.
To implement the sweep logic, we must first linearly order the segments of Ω to determine the sequence in which SL encounters them. Actually, we need to order the endpoints $\{E_{i0}, E_{i1}\}$ of all segments L_i so we can detect when SL starts and stops to intersecting each L_i. Traditionally, endpoints are ordered by increasing x and then increasing

y-coordinate values, but any linear order will do (some authors prefer a decreasing y and then increasing x value). With the traditional ordering, the sweep line is vertical and moves from left to right as it encounters each segment, as shown in the diagram of the figure 6.

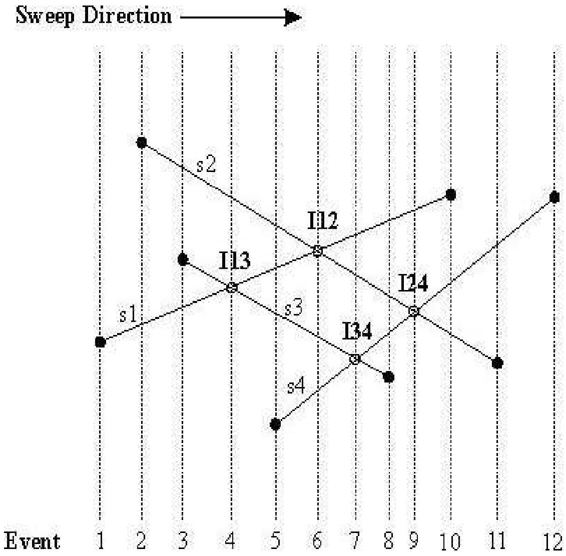

Figure 6: Sweeping diagram

At any point in the algorithm, the sweep line SL intersects those segments with one endpoint to the left of it and the other endpoint to the right. The SL data structure keeps track of these segments by: (1) adding a segment when its left endpoint is encountered, and (2) deleting a segment when its right endpoint is encountered. The SL also maintains the segments in a list ordered by an "above-below" relation. So, to add or delete a segment, its position in the list must be determined, and this can be done by a worst-case O(log n) binary search of the current SL segments. But, besides adding or deleting segments, there is another event that changes the structure of the SL; namely, whenever two current segments intersect, then their positions in the ordered list are swapped. Given the two segments, which must be neighbors in the list, this swap is an O(log n) operation.

To organize all this, the algorithm has an ordered *event queue* ξ whose elements cause a change in the SL segment list. Initially, ξ is set to the sweep-ordered list of all segment endpoints. But as intersections between segments are found, then they are also added to ξ in the same sweep-order as used for the endpoints One must test, though, to avoid inserting duplicate intersections onto the event queue. The example in the above diagram shows how this can happen. At event 2, segments $s1$ and $s2$ cause intersection $I12$ to be computed and put on the queue. Then, at event 3, segment $s3$ comes between and separates $s1$ and $s2$. Next, at event 4, $s1$ and $s3$ swap places on the sweep line, and s1 is brought next to s2 again causing $I12$ to be computed again. But, there can only be one event for each intersection, and $I12$ cannot be put on the queue again. So, when an intersection is being put on the queue, we must find its potential x-sorted location in the queue, and check that it is not already there. Since there is at most one intersect point for any two segments, labeling an intersection with identifiers for the segments is sufficient to uniquely identify it. As a result of all this, the maximum size of the event queue = $2n + k \leq 2n + n^2$, and an insertion into it can be done with at worst an $O(log(2n + n^2)) = O(log(n))$ binary search.

But, what does all this have to do with efficiently finding the complete set of segment intersections? Well, as segments are sequentially updated on the SL list, their possible intersection with other eligible segments is determined. When a valid intersection is found, then it is inserted into the event queue. Further, when an intersection-event on ξ is processed during the sweep, then it causes a re-ordering of the SL list, and is also added to the output list Λ. In the end, when all events have been processed, Λ will contain the complete set of all intersections.

However, there is one critical detail, the heart of the algorithm, that we still need to describe; namely, how does one compute a valid intersection? Clearly, two segments can only intersect if they occur simultaneously on the sweep-line at some time. But this by itself is not enough to make the algorithm efficient. The important observation is that two intersecting segments must be immediate above-below neighbors on the sweep-line. Thus, there are just a few restricted cases for which possible intersections must be computed:

- When a segment is added to the SL, determine if it intersects with its above and below neighbors.
- When a segment is deleted from the SL, its previous above and below neighbors are brought together as new neighbors. So, their possible intersection needs to be determined.

At an intersection event, two segments switch positions in the SL, and their intersection with their new neighbors must be determined.

This means that for the processing of any one event (end point or intersection) of ξ , there are at most 2 intersection determinations that need to be made. One detail remains, namely the time needed to add, find, swap, and remove segments from the SL structure. To do this, implement SL as a balanced binary tree (such as an AVL, a 2-3, or a red-black tree) which guarantees that these operations take at most $O(log(n))$ time since n is the maximum size of SL. Thus, each of the $(2n + k)$ events has at worst $O(log(n))$ processing to do. Adding everything up, the efficiency of the algorithm $= O(initial - sort) + O(event - processing) =$

$O(nlogn) + O((2n + k)log(n)) = O((n + k)log(n))$.

Pseudo-Code: Bentley-Ottmann Algorithm

Putting all of this together, the top-level logic for an implementation of the Bentley-Ottmann algorithm is given by the following pseudo-code:

```
{   Initialize event queue ξ = all segment endpoints;
  Sort ξ by increasing x and y;
  Initialize sweep line SL to be empty;
  Initialize output intersection list Λ to be empty;
  While (ξ is nonempty) {
     Let E = the next event from ξ;
     If (E is a left endpoint) {
       Let segE = E's segment;
       Add segE to SL;
       Let segA = the segment above segE in SL;
       Let segB = the segment below segE in SL;
       If (I = Intersect( segE with segA) exists)
       Insert I into ξ;
       If (I = Intersect( segE with segB) exists)
            Insert I into ξ;
     }
     Else If (E is a right endpoint) {
       Let segE = E's segment;
       Let segA = the segment above segE in SL;
       Let segB = the segment below segE in SL;
       Remove segE from SL;
       If (I = Intersect( segA with segB) exists)
            If (I is not in ξ already) Insert I into ξ;
       }
       Else { // E is an intersection even
            Add E to the output list Λ;
             Let segE1 above segE2 be E's intersecting segments in SL;
             Swap their positions so that segE2 is now above segE1;
            Let segA = the segment above segE2 in SL;
            Let segB = the segment below segE1 in SL;
            If (I = Intersect(segE2 with segA) exists)
              If (I is not in ξ already) Insert I into ξ;
            If (I = Intersect(segE1 with segB) exists)
              If (I is not in ξ already)
              Insert I into ξ;
       }
       remove E from ξ;
     }
     return Λ;
  }
```

This routine outputs the complete list of all intersection points.

4.2.2 Using The Bentley-Ottmann Algorithm to obtain simples polygons intersection

We present here a simplified version, for more detail see [10]

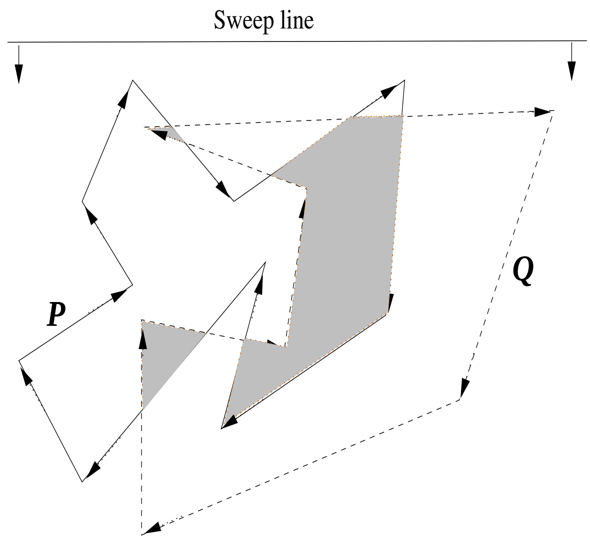

Figure 7: Sweep line

Here we suppose that the polygons P, Q are represented by a set of oriented edges (figure 7). To achieve the intersection $P \cap Q$, we are going to use the algorithm of Bentley and Ottman with some adaptation; one starts with calculating all the points of intersection between the edges of P and Q. For every interval (between the adjoining active edges) of the sweep line, one manages two booleans x, y; x to indicate if we are inside P, the other y to indicate if we are inside Q. Initially the sweep line is empty (see figure 7),and we are outside P and outside Q. When an edge α of P is inserted in the sweep line, its orientation permits to know if P is under or over α. Furthermore two other Booleans are therefore necessary : w underside α, z overside. If α is inserted in an interval of the sweep line that is inside (respectively outside) of Q, then the underside and the overside of α are inside (respectively outside) of Q and the Booleans below(w) and above(z) are initialized consequently. Symmetrically one achieves the same process when an edge of Q is inserted in the sweep line. When the edge α is exhausted (or is swept), the part of α that is in P and in Q is added to the list of oriented edges of $P \cap Q$ when one of the two following conditions is verified

- if underside of α but not overside, one is inside P and inside Q,
- or, if overside of α but not underside, one is inside P and inside Q.

So with these four booleans we know at each tim if the edge α separates the inside of the outside of the searched polygon $P \cap Q$.

Because the set of vertex of $P \cap Q$ is a subset of all intersection points of the edges of P and the edges of Q, the running time of this algorithm is $O(nlog(n) + klog(n))$ where n is the total number of vertices in P and Q ($n = n_1 + n_2$; n_1 is the number of vertices of P, n_2 is the number of vertices of Q), and k is the complexity $P \cap Q$.

5 CONCLUDING REMARKS

We have given a method that are able to extract information from a chaotic signal. This method is iterative and generate at every iteration a set of polygons to intersected, what brought us to make a presentation of the polygons intersection geometry.
We are at the beginning of this work, and a forthcoming project will be put under way to program and to test the efficiency of our approach.

References

[1] Anosov, D.V. (Ed.) 1995. *Dynamical Systems IX* (Encyclopaedia of Mathematical Sciences, v.66), Springer.

[2] Aziz-Alaoui,M.A. and Lozi,R. 1995. *Numerical result on nonlinear filtering for coupled Chua oscillators*, Proc. 2nd Int. Conf. ODE, Marrakech.

[3] Aziz-Alaoui M.A., Fedorenko A.D., Lozi R. and Sharkovsky A.N. 1997. "Recovering trajectories of chaotic piecewise linear dynamical systems", *Proc. IEEE, Control Oscillations and chaos*, eds. F.L. Chernousko, A.L. Fradkov, Vol. **2**, 230-233.

[4] Aziz-Alaoui M.A., Fedorenko A.D., Lozi R. and Sharkovsky A.N. 2000. "Recovering Parameters of Chaotic Piecewise Linear Dynamical Systems", *Proc. IEEE, Control of Oscillations and Chaos*, Vol. 2, pp. 380-383, eds. F.L. Chernousko, A.L. Fradkov.

[5] Bentley, J.L. and Ottmann, T.A. 1979. "Algorithms for reporting and counting geometric intersections", *IEEE Transaction on Computer*, Vol C-28, N 9, 643-647.

[6] Bose R. and Banerjee A. 1999. "Implementing Symmetric Key Cryptography using Chaos Function", 318 - 321, *7th International Conference on Advanced Communications and Computing*, December , University of Roorkee, Roorkee, India.

[7] Carroll, T. and Pecora, L. 1992. "A circuit for studying the synchronization of chaotic systems", *Int. J. Bifurcation and Chaos*, **2** (2), , 659–667.

[8] Chazalle B., Edelsbrunner H. 1992. "An Optimal Algorithm for Intersecting Line Segments in the Plane", *Journal of the ACM*, 39(1), pp. 1-54.

[9] Chen G. and Dong X. 1998. *From chaos to order* (Singapore: World Scientific).

[10] De Berg M., Van Kreveld M., Overmars M. and Schwarzkopf O. 1998. *Computational Geometry, Algorithm and Application*, Springer.

[11] Guglielmi V., Besnard P-Y., Fournier-Prunaret D., Pinel P., Taha A.K., Beneteau L. 2003. "Un systme numrique de cryptographie bas sur les proprits des signaux chaotiques discrets", Colloque IEEE GRETSI'03 sur le Traitement du Signal et des Images, Paris, Septembre.

[12] Hammel, S.M., Yorke, J.A. and Grebogi, C. 1987. "Do numerical orbits of chaotic dynamical processes represent true orbits?", *J. Complexity*, **3**, 136–145.

[13] Hammel, S.M., Yorke, J.A. and Grebogi, C. 1988. "Numerical orbits of chaotic dynamical processes represent true orbits", *Bull. Amer. Math. Soc.*, **19**, 465–469.

[14] Hasler M. 1998. "Synchronization of chaotic systems and transmission of information". *International Journal of Bifurcation and Chaos* 8(4): 647-659.

[15] Katok, A. and Hasselblatt, B., *Introduction to the modern theory of dynamical systems*, Cambridge University Press, 1995.

[16] Kocarev, Lj., Halle, K.S., Eckert, K., Chua, L.O. and Parlitz, U. 1992. "Experemential demonstration of secure communication via chaotic synchronization", *Int. J. Bifurcation and Chaos*, **2** (3), 709–713.

[17] Lozi,R. and Chua,L.O., "Secure communications via chaotic synchronization II: Noise reduction by cascading two identical receivers", *Int. J. Bifurcation and Chaos*, **3** (5), 1993, 1319–1325.

[18] Lozi, R. and Aziz-Alaoui, M.A., 1994. "Secure communication via chaotic synchronization in Chua's circuit: Numerical analysis of the errors of the recovered signal", *Proc. Symposium on Nonlinear Theory and its Application, Kagoshima*, 145–148.

[19] O'Rourke J. 1982. "A new linear algorithm for intersecting convex polygon". *Comput. Graph Image Processing* 19:384-391.

[20] Pecora, L.M. and Carroll, T.,1990. "Synchronization in chaotic systems", *Phys. Rev. Lett.*, **64**, 821–824.

[21] Sharkovsky, A.N. 1995. "From one-dimensional to infinite-dimensional dynamical systems", *Proc. 3th Int. Specialist Workshop on Nonlinear Dynamics of Electronic Systems*, Dublin, 28–29 July .

[22] Toussain, Godfried T., 1985. "A simple linear algorithm for intersecting convex polygon", *Visual Computer*, vol. 1, , pp. 118-123.

[23] Wu C.W. 2002. *Synchronization in Coupled Chaotic Circuits and Systems* (World Scientific).