Snap Rounding with Restore: an Algorithm for Producing Robust Geometric Datasets

ALBERTO BELUSSI, University of Verona (Italy)
SARA MIGLIORINI, University of Verona (Italy)
MAURO NEGRI, Politecnico of Milano (Italy)
GIUSEPPE PELAGATTI, Politecnico of Milano (Italy)

This paper presents a new algorithm called Snap Rounding with Restore (SRR), which aims to make geometric datasets robust and to increase the quality of geometric approximation and the preservation of topological structure. It is based on the well-known Snap Rounding algorithm, but improves it by eliminating from the snap rounded arrangement the configurations in which the distance between a vertex and a non-incident edge is smaller than half-the-width of a pixel of the rounding grid. Therefore, the goal of SRR is exactly the same as the goal of another algorithm, Iterated Snap Rounding (ISR), and of its evolution, Iterated Snap Rounding with Bounded Drift (ISRBD). However, SRR produces an output with a quality of approximation that is on average better than ISRBD, both under the viewpoint of the distance from the original segments and of the conservation of their topological structure. The paper also reports some cases where ISRBD, notwithstanding the bounded drift, produces strong topological modifications while SRR does not. A statistical analysis on a large collection of input datasets confirms these differences. It follows that the proposed Snap Rounding with Restore algorithm is suitable for applications that require both robustness, a guaranteed geometric approximation and a good topological approximation.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Spatial databases and GIS

General Terms: Design, Algorithms

Additional Key Words and Phrases: Geometric robustness, Rounding algorithms, Snap rounding

ACM Reference Format:
Alberto Belussi, Sara Migliorini, Mauro Negri, and Giuseppe Pelagatti, 2013. Snap Rounding with Restore: An Algorithm for Producing Robust Geometric Datasets *ACM Trans. Embedd. Comput. Syst.* 0, 0, Article 0 (March 2014), 36 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

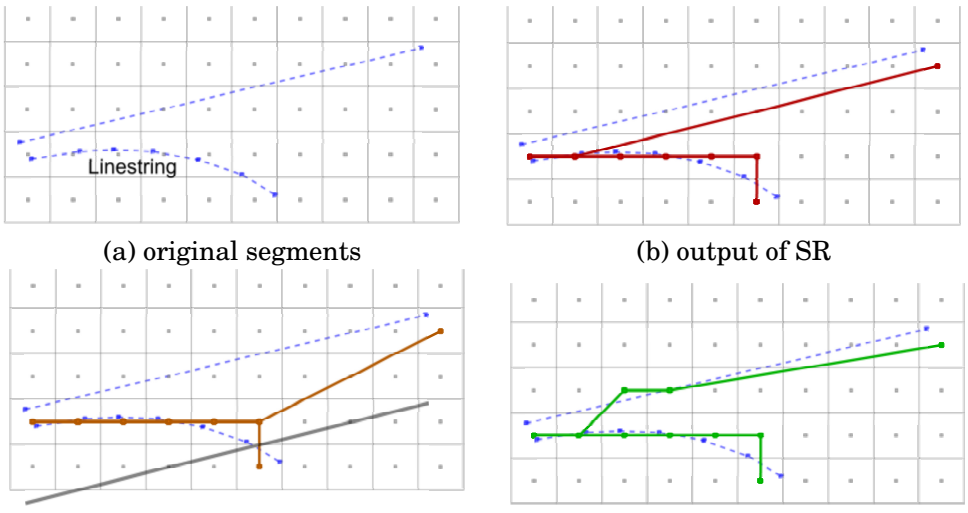
The importance of processing geometric datasets in order to make them robust is widely recognized and is continually increasing, since the availability of geometric data is steadily growing, and new applications are developed which use geometric data. The impact of a lack of robustness in some applications has been analyzed for instance in [Belussi et al. 2012; Thompson and van Oosterom 2006].

A geometric dataset is *robust* if it can be processed by the same algorithm in different environments always producing the same result. This condition can be achieved by guaranteeing that the distance between any vertex and non-incident edge is greater

Author's addresses: A. Belussi and S. Migliorini, Computer Science Department, University of Verona, Strada Le Grazie, 15, 37134 Verona (Italy); M. Negri and G. Pelagatti, Department of Electronics, Information and Bioengineering, Politecnico of Milano, Via Ponzio, 34/35, 20133 Milan (Italy).
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2014 ACM 1539-9087/2014/03-ART0 $15.00
DOI: http://dx.doi.org/10.1145/0000000.0000000

than a given minimum value. In the literature several techniques and algorithms have been proposed in order to make a dataset robust, some of them are treated in Section 2. One of these techniques, which is also the starting point of this work, is represented by the *Snap Rounding* (SR) algorithm, firstly defined in [Hobby 1993], [Hobby 1999] and further optimized in [Goodrich et al. 1997] and [de Berg et al. 2007]. SR takes a set of segments as input and transforms it into an arrangement such that: (i) all segments have been split at their intersection points obtaining two or more segments for each original one; (ii) all segments that traverse a hot pixel have been split at the hot pixel centre; (iii) segment endpoints (even those created by the previous steps) have been rounded to a given finite precision grid. Halperin and Packer [Halperin and Packer 2002] showed an important drawback of SR: the dataset produced by this algorithm may still be not robust because the distance between a vertex and a non-incident edge can be extremely small.

In order to overcome this problem, in [Halperin and Packer 2002] the authors propose a new algorithm, called *Iterated Snap Rounding* (ISR), which performs additional snapping operations for eliminating such non-robust configurations. However, while ISR solves the robustness problem of SR, it introduces a new problem: in some configurations the distance between the produced segments and the original ones can be very large, thus degrading the guaranteed quality of the approximation with respect to SR. In [Packer 2008] the authors try to solve this defect by proposing an evolution of ISR, called *ISR with Bounded Drift* (ISRBD). ISRBD guarantees that the distance between the output segments and their corresponding original ones is less than a user-specified parameter δ; the minimum value of δ is about 3 times the maximum deviation of SR.



(a) original segments (b) output of SR

(c) output of ISRBD (d) output of SRR

Fig. 1. Example of linear collapsing.

The potential lack of robustness of the original SR algorithm can produce another important drawback: the SR execution may be not stable, namely repeating SR on the result of a previous execution may produce a different dataset. In [Hershberger 2011] an algorithm is defined, called *Stable Snap Rounding*, which produces an arrangement that is very similar to the one produced by SR, but is stable. This is achieved by modifying SR so that it does not use the rounded pixels for snapping. Stable SR

is not considered further in this paper, since it does not preserve the property for which ISR (ISRBD) has been introduced (elimination of non-robust configurations, i.e. near-degenerate output), although it eliminates one of the negative aspects of non-robustness.

This paper proposes an algorithm, called *Snap Rounding with Restore* (SRR), which is based, like ISR and ISRBD, on a first SR step followed by the elimination of the non-robust configurations produced by this step. However, this elimination is performed without degrading the quality of the approximation; indeed, as simulations have shown, on average the quality of the geometric approximation is even improved, while an effort is made to avoid as much as possible topological degradation. An intuitive idea of this fact can be gained by observing Fig. 1 and Fig. 2, which will be discussed in more detail in Section 5. The main problem of ISRBD is due to the behaviour of the algorithm in presence of small angles between segments, as in Fig. 1: while ISRBD produces a linear collapse of the original segments, SRR tries to maintain them well separated. This kind of geometric configurations occurs rather frequently in vector data representing geographical information. For example, geometric realizations of network graphs (frequently used in the representation of transport or service networks) might present cases in which many chains with a curvilinear shape converge in the same node. A linear collapse in this kind of geometric configurations should be avoided in order to preserve the topological properties of the graph in its vector realization.

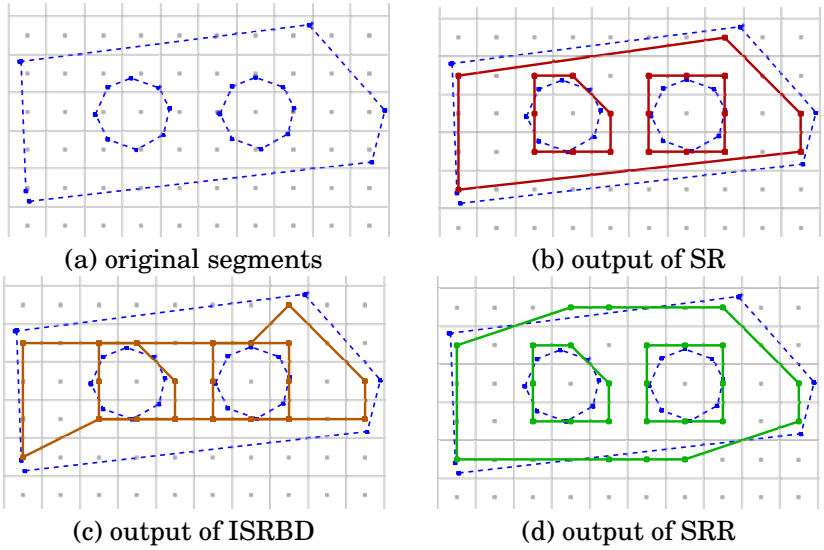


Fig. 2. Example of linear collapsing of polygon boundaries.

Moreover, Fig. 2 exemplifies the case where segments are used to represent polygon boundaries. In this case, ISRBD has a strong distortion effect on the polygonal topology, and this effect could have worse consequences if the original dataset was intended to represent a polygon with two holes. Of course this example assumes that the grid of cells does not correspond to the grid of the floating point representation of numbers, but to a coarser grid applied to a scene that contains two small holes in a small polygon. Finally, notice that the linear collapsing produced by SR algorithms can have a negative impact on the correctness of geometries with respect to their geometric type.

Sometimes the consequence of the collapse is the violation of a constraint that is required by the geometric type of the geometry. For example, a constraint of the type *Surface* of OGC standards [OGC 2011] requires that the outer boundary of a surface must not have self-overlapping: by allowing linear collapsing this constraint might be violated. Therefore, it is important to reduce linear collapsing as much as possible.

The fundamental idea applied by SRR is as follows: in order to eliminate a situation where a vertex and a non-incident edge are too near to each other, it is not always convenient to perform a snapping operation. Instead, it may be convenient to spread them apart. This idea is very general and can be applied in several ways; in SRR the assumption is that if an edge has been moved by the SR algorithm too close to a vertex, then it is convenient to move it in the opposite direction, thus bringing it back towards its original position; this operation is called *Restore*.

The improvement of output quality obtained by SRR is paid in terms of complexity. The complexity of the proposed algorithm in the worst case is $O(n^4 \cdot log(n))$ (where n is the number of edges generated after the first SR iteration). However, in all practical cases, for example in applications dealing with geographical datasets, where a real scene cannot consist of a set of segments in which each segment intersects all the others, the complexity is reduced to $O(n^2 \cdot log(n))$ as discussed in more details in Section 4.6. Finally, by means of experiments on synthetic and real datasets, we compare SRR only with ISR and ISRBD since also they ensure the elimination of the non-robust configurations.

The paper is organized as follows: Section 2 presents several related contributions about the discrete representation of geometric datasets. Section 3 summarizes some background information about the SR algorithm; since SRR is based on many concepts defined by SR, this background is necessary to understand and evaluate it. Section 4 presents the SRR algorithm and the main properties of its output. Section 5 discusses the approximation induced by SRR and compares the algorithm with ISRBD, since the two algorithms pursue the same goals. Section 6 reports the experimental results obtained on a large number of segment arrangements, and the behaviour of SRR is compared with respect to the other existing algorithms considering a few quantitative parameters.

2. RELATED WORK

Geometric algorithms are usually defined under the simplified assumption that computations are performed with an infinite-precision arithmetic which cannot be actually provided by the adopted computer representations. This assumption raises great difficulties in implementing robust geometric algorithms. In the literature, several techniques have been proposed in order to overcome this problem. Some of them analyze a group of algorithms and propose an approach that ensures the exact computation by means of a greater precision in number representation. For instance, in [Boissonnat and Preparata 2000] a robust implementation of the well known plane sweep approach for intersecting segments has been proposed that requires a precision, which is triple of the input. More recently, the Exact Geometric Computation (EGC) model [Chen 2001] has been proposed; it provides a general approach for making robust the evaluation of geometric algorithms. This can be achieved by representing the underlying mathematical objects in an *exact* manner through the use of algebraic numbers which allow to perform computations without errors. By definition, an algebraic number is the root of an univariate polynomial with integer coefficients. For instance, the number $\sqrt{5}$ has no finite representation, but it can be represented exactly as the pair $(X^2 - 5, [1, 4])$, interpreted as the root of the polynomial $X^2 - 5$ lying in interval $[1, 4]$. These techniques have made many progresses so that for certain problems

the introduced performance penalty is acceptable. However, when the computation is performed on curved objects (they can be approximated in linear geometry thought linestrings having many vertices) or in three-dimensional space, the overhead is still large. An alternative approach has been proposed which is called Controlled Perturbation (CP) [Halperin and Shelton 1998; Halperin 2010; Packer 2011] and belongs to the family of Finite-Precision Approximation techniques. This method proceeds by perturbing the input slightly but in a controlled manner such that all predicates used by the algorithm are guaranteed to be evaluated correctly with floating-point arithmetic of a given precision, and the degeneracies are removed. The algorithms of the Snap Rounding approach belong to the same family of Finite-Precision Approximation techniques where you find CP. However, they are mainly based on the application of some rounding algorithms that convert an arbitrary-precision arrangement of segments into a fixed-precision representation.

In [Greene and Yao 1986] the authors propose a first rounding scheme for polygonal subdivisions that precedes the Snap Rounding idea. In particular, they discuss a technique for transforming geometric concepts and algorithms from the continuous domain to the discrete one, by defining an interface between these two domains which satisfies certain invariants. The first Snap Rounding algorithm (SR) for an arrangement of segments has been given in [Hobby 1999]. After this work many alternatives have been proposed with the aim to improve the asymptotic time complexity of the process, to provide on-line algorithms, or to support 3D extensions. In particular, in [Guibas and Marimont 1995] the authors define a dynamic algorithm for snap rounding an arrangement of segments, as well as providing elementary proofs of the topological properties maintained by SR. Their algorithm is dynamic because it uses a randomized hierarchy of vertical cell decompositions in order to make efficient the localization of segment intersections and the deletion of segments. Conversely, [Goodrich et al. 1997] improves SR when many segments intersect a pixel by proposing an efficient algorithm that does not require to firstly determine all segment intersections, but it is based on an incremental construction. In [de Berg et al. 2007] the authors propose an SR variant whose complexity depends upon the number of intersections between the input segments and guarantees the additional property that no degree-2 vertices are present in the interior of the produced edges. A first method for rounding curvilinear arrangements is presented in [Eigenwillig et al. 2007], where the authors extend the SR algorithm from straight-line segments to Bézier curves of arbitrary degree. The extension of the SR algorithm to the 3D space is considered in [Goodrich et al. 1997; Fortune 1998].

Two important variants of SR are considered in this paper: the Iterated Snap Rounding (ISR) and the Iterated Snap Rounding with Bounded Drift (ISRBD). ISR has been presented in [Halperin and Packer 2002] and is equivalent to iteratively perform a SR step until all vertices and non-incident edges in the rounded arrangement become well separated. ISRBD [Packer 2008] augments ISR with simple and efficient procedures that guarantee the quality of the geometric approximation of the original segments, while still maintaining the property that vertices and non-incident edges are well separated. Another interesting variant of SR has been presented in [Hershberger 2011] which is called Stable Snap Rounding (StableSR). It has all advantages of SR and is also idempotent (i.e., stable). In other words, if StableSR is applied to the output of either StableSR or SR, it produces no effect. However, as it is stated in the paper, StableSR may produce rounded segments that intersects hot pixels, passing arbitrarily close to the pixel center. For this reason, StableSR is not further considered in this paper, since the aim of SRR is to ensure the same properties of ISR (or ISRBD) plus enhancing the quality of the approximation both under the viewpoint of the distance from the original segment and the conservation of the topological structure.

3. BACKGROUND

This section presents a set of basic concepts which are fundamental for understanding the remainder of this paper and in particular the SRR behaviour. Every snap rounding algorithm is based on the concept of *arrangement of segments*.

Definition 3.1 (*Arrangement*). Let $\mathcal{S}$ be a collection of line segments in the plane. The *arrangement* $\mathcal{A}$ of $\mathcal{S}$ is the decomposition of the plane into *vertices*, *edges*, and *faces* induced by $\mathcal{S}$. In particular:

(1) A *vertex* of $\mathcal{A}$ is either a segment endpoint or the intersection between two segments of $\mathcal{S}$.
(2) An *edge* of $\mathcal{A}$ is a connected set of points belonging to one segment of $\mathcal{S}$ that do not belong to any other segment of $\mathcal{S}$. Each edge starts and ends at vertices of the arrangement. The set of edges corresponding to a single original segment forms a *polygonal chain*.
(3) A *face* of $\mathcal{A}$ is a subset of the plane not contained in any segment of $\mathcal{S}$. □

Definition 3.2 (*Snap Rounding*). *Snap Rounding* (SR) is a method for converting arbitrary precision arrangements of segments into fixed-precision representations [Hobby 1999]. □

Definition 3.3 (*Hot pixel*). Given an arrangement of segments and a subdivision of the plane into a grid of unitary cells, called pixels, a pixel is said to be *hot* if it contains a vertex of the arrangement. □

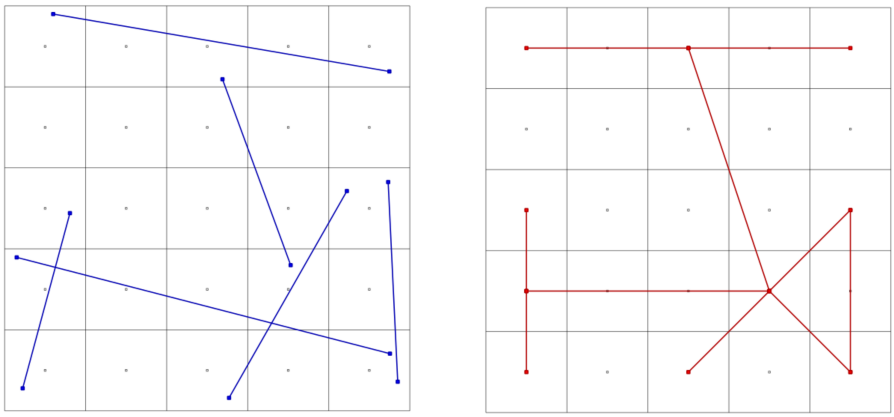


Fig. 3. Example of application of the SR algorithm. (a) Initial set of segments. (b) Arrangement after the SR execution: each segment endpoint and each intersection point has been snapped to the pixel center.

Algorithm 1 (*Snap Rounding (SR)*). Given a set of segments $\mathcal{S}$ whose end-points are represented using finite precision coordinates, and a grid of pixels having integer coordinates, a possible SR algorithm proceeds as follows:

— It firstly computes the initial arrangement $\mathcal{A}$, corresponding to the set $\mathcal{S}$, splitting intersecting segments into several edges.
— Subsequently, each vertex of the arrangement $\mathcal{A}$ is replaced by the center of the hot pixel containing it, and each polygonal chain c of the arrangement is replaced by a

polygonal chain d whose vertices are the centers of the hot pixels met by c in the same order as they are met by c. [1]

Example 3.4 (*SR application*). Fig. 3 illustrates an example of SR application. Given the configuration in Fig. 3.a, the SR algorithm produces the result shown in Fig. 3.b. Notice that after the execution of SR some vertices and edges of the original setting may be collapsed, while some segments have been split into two or more parts in correspondence to intersections.

Property 3.1 (*SR features*). As proved in [Hobby 1999; de Berg et al. 2007], the SR algorithm has the following properties:

(1) *Fixed-precision representation*: Given a set of segments $\mathcal{S}$ and a grid of unitary square pixels, all vertices of the arrangement $\mathcal{A}$ produced from $\mathcal{S}$ by SR are located at the pixel centers.
(2) *Geometric similarity*: For each segment s of the original set $\mathcal{S}$, its approximation in the resulting arrangement $\mathcal{A}$ is a chain $c \in \mathcal{A}$ which lies within the Minkowski sum of s and a cell of unit square (i.e. a cell with side length equal to the chosen pixel size) centered at the origin. Fig. 4 illustrates an example of Minkowski sum of a segment s and a cell of unitary square.
(3) *Topological similarity*: The original set of segments $\mathcal{S}$ and the resulting arrangement $\mathcal{A}$ are topologically equivalent up to the collapsing of features. More specifically there is a continuous deformation of the original segments to their snap-rounded counterparts such that no segment ever passes completely over a vertex of the arrangement. □

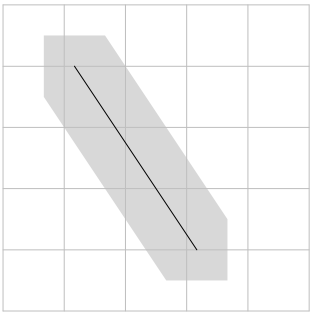
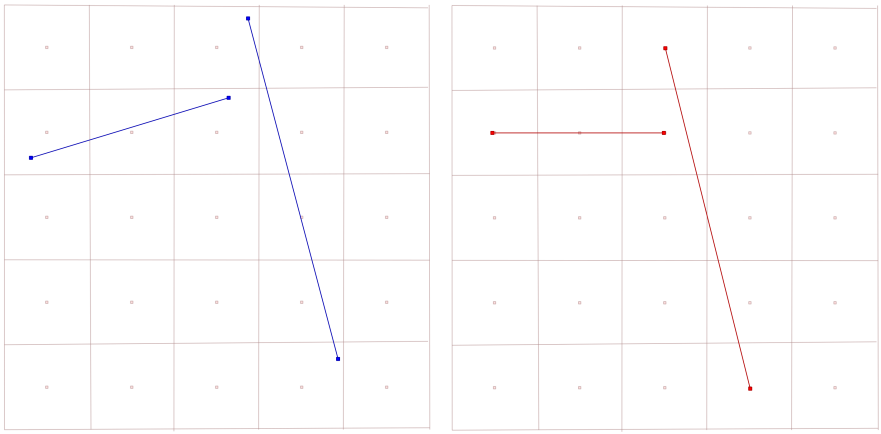


Fig. 4. In the figure the gray polygon represents the Minkowski sum of the black segment and a cell of unitary square centered at the origin.

SR makes the vertices of the original segments well separated when they are not snapped together. However, this is not true when vertices and non-incident edges are considered. In other words, after the execution of the SR algorithm, the distance between a vertex and a non-incident edge can be extremely small w.r.t. the width of a pixel in the used grid. For instance, Fig. 5.a shows a set of original segments and Fig. 5.b the arrangement produced by SR: after the application of the SR algorithm the distance between an endpoint of the horizontal segment and the oblique segment is very small, namely they share the pixel highlighted in gray in Fig. 5.b. Notice that

[1]The notion of tolerance square is considered during the segment intersection: whenever a hot pixel is crossed by one or more segments, the presence of an intersection is assumed and they are all split and snapped to the corresponding hot pixel center.



(a) (b)

Fig. 5. In (a) and (b) an example of the robustness problem of the result produced by SR is shown. In (a) the initial arrangement is shown; in (b) the SR result. Notice that in (b) the distance between the vertex and the edge crossing the gray pixel is less than $\sqrt{2}\,ps/2$, where ps is the chosen pixel size.

they are not snapped together by the SR algorithm, because the input right segment does not pass through the corresponding hot pixel before the rounding took place.

Definition 3.5 (*Critical Pixel*). A *critical pixel* is a hot pixel which is crossed by an edge whose start and end nodes are outside the pixel. [2] □

Using the notion of critical pixel, the *robustness condition* for a dataset can be reformulated as follows: a rounded arrangement is robust if it does not contain critical pixels. The elimination of critical pixels from the result of SR is the goal of SRR.

4. SNAP ROUNDING WITH RESTORE (SRR)

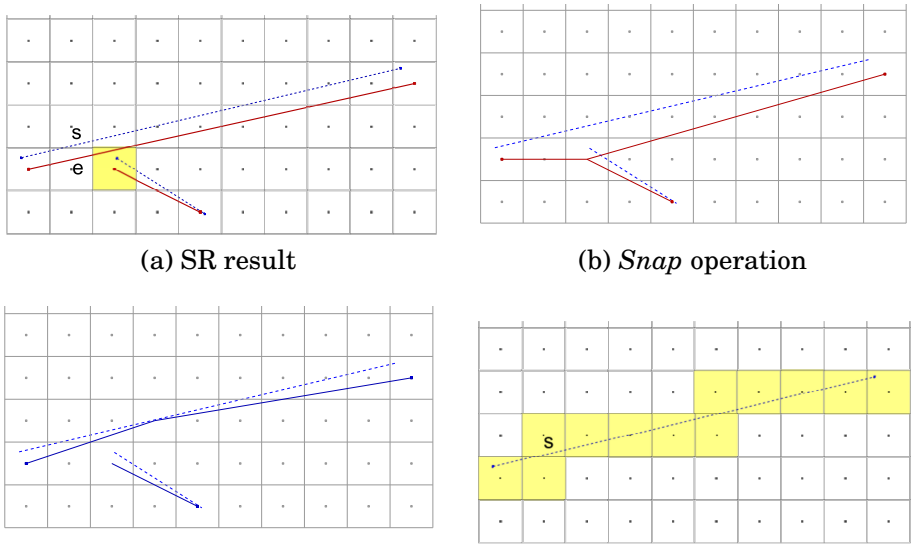
This section presents the SRR algorithm. First the key ideas are presented, then the algorithm is explained, some important properties are stated, and finally an example of its behaviour is illustrated.

4.1. Key Ideas of SRR

SRR performs an initial SR step which snaps each vertex of the arrangement to the center of the corresponding hot pixel. In SRR an additional property is added to each edge e of the arrangement produced by SR: the property $OS(e)$ which represents the set of *original segments* from which the edge has been generated. This SR step is followed by a **Restore** procedure which solves the possibly generated critical pixels by adding a vertex to the edges crossing each one of them. In order to understand the basic idea applied by the **Restore** procedure, we start with a simple example.

Fig. 6.a shows the output of an SR step (solid lines) together with the two segments which constitute the input dataset (dashed lines). The pixel $\langle 3, 2\rangle$ (highlighted[3]) is a critical pixel, because it contains a vertex and a crossing edge, e. In order to eliminate this critical pixel there are fundamentally two options, which constitute the two basic operations in SRR:

[2] An edge crosses a pixel, if it passes through its interior, while if it only intersects the pixel boundary, it touches the pixel but does not cross it.
[3] By default in all the figures the lower left pixel has coordinates $\langle 1, 1\rangle$.



(a) SR result (b) *Snap* operation

(c) *Spread* operation (d) *PixelEnvelope* of a segment

Fig. 6. A set consisting of two original segments (dashed lines), the result of SR (a) and of the Snap (b) and Spread (c) operations (continuous lines). In (d) the *PixelEnvelope* of a segment s is shown.

— *Snap*: this is the classical approach and consists of adding the critical pixel to the vertices of the crossing edge, thus producing a chain which contains the center of the critical pixel as a vertex, as shown in Fig. 6.b.
— *Spread*: this operation consists in adding to the crossing edge a new vertex, such that the edge becomes a chain which does not cross the pixel that was critical, as shown in Fig. 6.c.

Notice that while snapping can be performed in a unique way, there are several pixels which can be chosen for performing the *Spread* operation. Therefore, the core of the **Restore** procedure consists in the **Choose-Pixel** function, which determines the pixel to use for rerouting the edge that crosses a critical pixel. For instance, in Fig. 6.c the pixel $\langle 4, 3 \rangle$ has been chosen, thus becoming a new hot pixel of the arrangement.

4.2. The SRR Algorithm

Listing 1 illustrates the SRR algorithm. The algorithm assumes the existence of a grid of unitary pixels; in the sequel the existence of the grid is implicitly assumed and not explicitly recalled. Moreover, the algorithm starts from a set of segments $\mathcal{S}$, produces the corresponding arrangement $\mathcal{A}$ and works on $\mathcal{A}$ in order to resolve critical configurations by splitting the edges of the arrangement. The edges obtained after a split are collected in the set of edges of the arrangement $\mathcal{A}$, since they can be iteratively split by the algorithm.

The algorithm performs firstly an SR operation producing the arrangement $\mathcal{A}$; after the SR step the function **CriticalPixels** creates the critical pixel set *CPS*. In the main cycle, **Restore** is the fundamental procedure. The **Restore** procedure processes all critical pixels which were created by the previous iteration, but, since the restore operation may produce new critical pixels, at the end of the while cycle the function **CriticalPixels** is invoked on the modified arrangement and produces a new critical pixel set *CPS*, which can be non-empty and therefore a new iteration may be required. Notice that the necessary edge splitting is performed at the end of the **foreach** cycle so

LISTING 1: Snap-Rounding with Restore.

Input: A finite-precision set of segments $\mathcal{S}$ and a grid g with unitary pixels.
Output: A fixed-precision arrangement $\mathcal{A}$ of $\mathcal{S}$.

```
begin
    A ← ∅; CPS ← ∅;
    Snap-Rounding(S, g, *A);
    CriticalPixels(A, *CPS);
    while CPS ≠ ∅ do
        splitEdges ← ∅;
        foreach cp ∈ CPS do
            Restore(cp, A, *splitEdges);
        end
        UpdateArrangement(splitEdges, *A);
        CriticalPixels(A, *CPS);
    end
    return A;
end
```

In the pseudo-code: (i) the notation $x \leftarrow expr$ is used for the assignment operation; (ii) the procedure parameters with the prefix "" are passed by reference and are modified by the procedure, while the other ones are passed by value and are not modified.*

that, during an iteration, the **Restore** procedure always works on the same arrangement $\mathcal{A}$, while *splitEdges* collects the edges to be split together with the corresponding vertices. They will be used for updating the arrangement in the **UpdateArrangement** procedure. The whole algorithm terminates when an iteration does not produce any new critical pixels.

In order to illustrate the logic of the **Restore** procedure presented in Listing 2 the functions **Pe**(), **Cvs**() and **Fcvs**() are specified in the following Definitions 4.1, 4.4 and 4.5.

Definition 4.1 (*Pixel Envelope (**Pe**)*). Given a segment s and a grid g of unitary square pixels, the *pixel envelope* of s, denoted as **Pe**(s), returns the set of pixels of g crossed by the segment itself. □

An example of pixel envelope for a segment s is shown in Fig. 6.d.

Definition 4.2 (*Pixel Frame (**Pf**)*). Given a critical pixel *cp* in a grid g of unitary square pixels, the *pixel frame* of *cp*, denoted **Pf**(*cp*), is the set of pixels of g surrounding *cp*. □

Definition 4.3 (*Useful Vertex*). Given a critical pixel *cp* in a grid g of unitary square pixels and an edge e crossing it, called *critical edge*, the predicate **Useful-Vertex**(p, cp, e) is true if by splitting e at the center of p the resulting chain does not cross *cp* any more. □

Definition 4.4 (*Candidate Vertex Set (**Cvs**)*). Given a critical pixel *cp* in a grid g of unitary square pixels and an edge e crossing it, the function *CandidateVertexSet*, denoted as **Cvs**(cp, e, os), returns the set of pixels p such that:

$$\textbf{Cvs}(cp, e, os) = \{p \mid p \in \textbf{Pe}(os) \wedge p \in \textbf{Pf}(cp) \wedge \textbf{Useful-Vertex}(p, cp, e)\}$$

where $os \in OS(e)$ and $OS(e)$ represents the set of original segments corresponding to the edge e. □

In Fig. 6.a, SR generates a critical pixel $\langle 3, 2\rangle$ with a critical edge e; by applying the above definitions it follows that **Cvs**$(\langle 3, 2\rangle, e, s) = \{\langle 2; 3\rangle, \langle 3; 3\rangle, \langle 4; 3\rangle\}$. Notice that the

pixel $\langle 4, 3\rangle$, which has been chosen by the algorithm SRR, produces a chain that does not cross *cp* although it touches it.

Definition 4.5 (*Free Candidate Vertex Set (**Fcvs**)*). Given an arrangement $\mathcal{A}$ and a set of pixels C produced by the function **Cvs**(), the function *FreeCandidateVertexSet*, denoted as **Fcvs**$(\mathcal{A}, C)$ returns the set obtained from C as follows: **Fcvs**$(\mathcal{A}, C) = \{p | p \in C \wedge \neg(\exists e \in \mathcal{A} : e.\mathit{crosses}(p))\}$ □

In Fig. 6.a the set returned by **Cvs**$(\langle 3, 2\rangle, e, s)$ is equal to the set returned by **Fcvs**$(a,$ **Cvs**$(\langle 3, 2\rangle, e, s))$, because there are no other edges crossing the pixels in **Cvs**$(\langle 3, 2\rangle, e, s)$. For simplicity, in the sequel the shorter term "set **Cvs**()" is used instead of "the set returned by the function **Cvs**()", and similarly for the function **Fcvs**(). The other functions used by the **Restore**/**ChoosePixel** procedures are explained in Tab. I.

In the sequel we refer to the execution block, which contains the call to the SR algorithm and the first call of the **CriticalPixel** function (see Listing 1), as the preliminary step of SRR; therefore, the **Restore** procedure is executed for the first time during the first iteration of the main cycle of SRR. In order to simplify the discussion of the properties of the algorithm, the rules applied by **Restore**/**ChoosePixel** are restated in Tab. II, which represents exactly the same control flow presented in Listing 2 and identifies three possible situations that are labelled as rules: R1, R2 and R3. In Tab. II columns 2 and 3 express the conditions on the critical pixel and critical edge being evaluated, column 4 identifies the pixel which is returned by the function, column 5 interprets the choice in terms of the spread or snap operations and column 6 states whether the choice implies the generation of a new critical pixel, thus causing a further iteration of the algorithm. In the last column "YES" means that a new critical pixel is explicitly added by the rule. Also the other rules can produce critical pixels, but only indirectly, since they move some edges.

Listing 1 and 2 define an abstract implementation of the SRR approach. This implementation is not optimized, however it is suitable for analyzing the fundamental properties of the output produced by SRR. In section 4.6 a more efficient implementation of the core procedures of the SRR algorithm is proposed and its computational complexity is discussed in detail.

4.3. Properties of SRR

The main properties of SRR are stated by the following propositions. We first deal with SRR termination, then with preservation of SR properties: fixed precision representation, geometric similarity and topological similarity.

4.3.1. Termination of the Algorithm. In order to prove SRR termination some preliminary results are presented regarding the properties of the arrangement $\mathcal{A}$ produced after each SRR iteration (**while** cycle in Listing 1).

PROPOSITION 4.6 (FIXED-PRECISION REPRESENTATION IN SRR ITERATIONS). *Given an arrangement $\mathcal{A}$ and a grid of pixels, each chain of the arrangement, which is produced after an iteration of SRR, is composed of edges that start/end in hot pixel centers.*

PROOF. This proposition is a direct consequence of the algorithm: indeed, after the preliminary step of SRR, from Property 3.1.1 of SR the proposition is true; at each successive iteration, if an edge is split into two or more edges with the **Restore** procedure, then the resulting edges start and end in hot pixel centers. Indeed, in the **Restore** procedure the vertices for splitting an edge are produced by the **ChoosePixel** procedure, which always returns hot pixel centers. Therefore, also the chains containing the split edge are updated and pass through hot pixels centers. □

LISTING 2: Restore & ChoosePixel & UpdateArrangement.

Restore procedure
Input: A critical pixel *cp*, the current arrangement $\mathcal{A}$ and the set of edge to be split *splitEdges*.
Result: The modified set *splitEdges* containing the edges to be split.

```
begin
    foreach e ∈ A do
        if e.crosses(cp) then
            foreach os ∈ OS(e) do
                newVertex ← ChoosePixel(cp, e, os, A);
                splitEdges.addEdge(e, os, newVertex);
            end
        end
    end
end
```

ChoosePixel function
Input: A critical pixel *cp*, a critical edge e and an original segment os of e, the current arrangement $\mathcal{A}$.
Output: The center c of the chosen pixel.

```
begin
    cvs ← Cvs(cp, e, os);
    if Fcvs(A, cvs) ≠ ∅ then
        return the center of the pixel p ∈ Fcvs(A, cvs) such that p.distance(os) is minimal.
    else
        if cp ∈ Pe(os) then
            return the center of cp
        else
            return the center of the pixel p ∈ cvs such that p.distance(os) is minimal.
        end
    end
end
```

UpdateArrangement procedure
Input: An arrangement $\mathcal{A}$ and a set set *splitEdges* containing the edges to be split.
Result: The modified arrangement $\mathcal{A}$.

```
begin
    foreach (e, os) ∈ splitEdges.getEdges() do
        S ← e.split(splitEdges.getVertices(e, os));
        A.addEdges(S, os);
        A.dropEdge(e);
    end
end
```

PROPOSITION 4.7. *Given an arrangement $\mathcal{A}$ and a grid of pixels, every new hot pixel generated at each iteration of SRR belongs to the pixel envelope of one original segment of $\mathcal{A}$.*

PROOF. New hot pixels are generated starting from the first iteration of SRR. Given a chain c representing an original segment *os*, new vertices can be generated in c when the **Restore** procedure is applied to a critical pixel *cp* crossed by c. This procedure generates new hot pixels by calling the function **ChoosePixel**, which chooses them from the set **Cvs**(cp, e, os), being e the edge of c crossing *cp*. Since by Def. 4.4 we have

Table I. The functions used by the **Restore** and **ChoosePixel** procedures.

Functions	Description
$splitEdges.addEdge(e, os, v)$	If the pair (e, os) is already stored in *splitEdges* the vertex v is added to its ordered list of split vertices, otherwise the edge e is added together with its original segment os to the set of edges to be split and the list of split vertices is set to (v).
$splitEdges.getEdges()$	It returns a set containing pairs (e, os) each one representing an edge e that has to be split together with its original segment.
$splitEdges.getVertices(e, os)$	It returns a list of vertices where the edge e associated to os has to be split.
$e.split(L_v)$	It returns the list of edges obtained by splitting the edge e at each vertex v of the list L_v.
$\mathcal{A}.addEdges(S_e, os)$	It adds to the arrangement $\mathcal{A}$ all the edges belonging to S_e and associates the original segment *os* to each added edge.
$\mathcal{A}.dropEdge(e)$	It deletes the edge e from the arrangement $\mathcal{A}$.
$e.crosses(cp)$	It returns true if the edge e crosses the critical pixel cp, i.e. it intersects cp interior, false otherwise.
$p.distance(s)$	It returns the Eucledian distance between the center of the pixel p and the segment s.
Pe(os), **Cvs**(cp, e, os) and **Fcvs**$(\mathcal{A}, cvs)$	Their semantics is specified in Def. 4.1, 4.4 and 4.5 respectively.

Table II. Actions performed by the **ChoosePixel**$(cp, e, os, \mathcal{A})$ function. **Fcvs**$()$ stands for **Fcvs**$(\mathcal{A},$ **Cvs**$(cp, e, os))$ and **Cvs**$()$ stands for **Cvs**(cp, e, os).

Rule	**Fcvs**$() \neq \emptyset$	$cp \in$ **Pe**(os)	Returned Pixel	Operation	New critical pixels
R1	TRUE	not relevant	the pixel $p \in$ **Fcvs**$()$ such that $p.distancs(os)$ is minimal	*Spread*	NO
R2	FALSE	TRUE	cp	*Snap*	NO
R3	FALSE	FALSE	the pixel $p \in$ **Cvs**$()$ such that $p.distance(os)$ is minimal	*Spread*	YES

that:

$$\mathbf{Cvs}(cp, e, os) \subset \mathbf{Pe}(os)$$

then, the new hot pixels are always pixels of **Pe**(os). Therefore, every vertex of any output chain c produced by each iteration of SRR is located at the center of a pixel of **Pe**(os), where *os* is the original segment of c. □

PROPOSITION 4.8 (SRR GEOMETRIC SIMILARITY IN SRR ITERATIONS). *Each output chain produced by each iteration of SRR lies in the Minkowski sum of its original segment os and a unitary cell centered at the origin.*

PROOF. After the preliminary step of SRR (first two statements of Listing 1), each resulting chain has this characteristic from Property 3.1.2 of SR. Any subsequent iteration of SRR could introduce new vertices into the chain representing *os*, which are always located in pixels of **Pe**(os) from Prop. 4.7. Thus, every vertex of any output chain c produced by SRR is located at the center of a pixel of **Pe**(os). Assuming unitary

cells, the center of the pixels of $\mathbf{Pe}(os)$ cannot be located at a distance greater than $\sqrt{2}/2$ from *os*, otherwise they would not have been pixels of $\mathbf{PE}(os)$. Thus, having this maximum distance from *os* they are inside the *Minkowski sum* of *os* and a unitary cell centered at the origin. Finally, as the considered *Minkowski sum* is a convex hull, also the chain c is inside it, since it passes through vertices located at the center of pixels of $\mathbf{Pe}(os)$. □

PROPOSITION 4.9 (CVS FUNCTION RESULT IS NOT EMPTY). *Given a critical pixel cp and a critical edge e crossing it, such that the considered original segment $os \in OS(e)$ does not cross cp, the set returned by $\boldsymbol{Cvs}(cp, e, os)$ contains at least two pixels.*

PROOF. Since *os* does not cross *cp* but e crosses *cp*, and the distance between an edge and its original segment is always less than $\sqrt{2}/2$ according to Prop. 4.8, there always exist some pixels which belong to both $\mathbf{Pe}(os)$ and $\mathbf{Pf}(cp)$. Thus, *os* always intersects some pixels of $\mathbf{Pf}(cp)$, even if it may not intersect *cp*.

Consider first the case where a vertex v of e lies in a pixel which is adjacent to cp as shown in Fig. 7.a, where the pixel touches cp on the left. As shown in Fig. 7.b and Fig. 7.c, in this case there are two pixels of $\mathbf{Pe}(os) \cap \mathbf{Pf}(cp)$ which allow to split e and avoid the crossing. The configuration shown in Fig. 7.a can be generalized to all other configurations having a vertex on top, bottom or right of the critical pixel, by considerations based on symmetry. Every other configuration, where v is located farther from *cp* produces the same result since the pixels that avoid crossing in the previous case do avoid crossing for any v' located farther than v on the left (or top, bottom, right). □

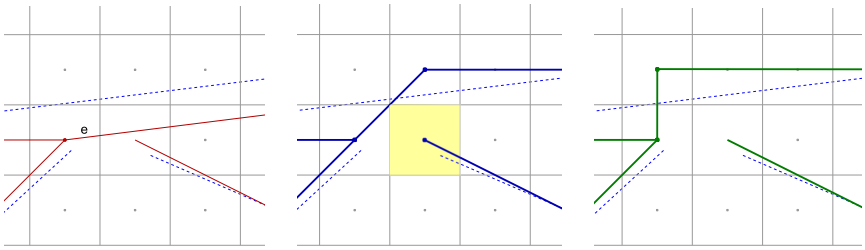


Fig. 7. The worst case configuration from the viewpoint of rerouting a critical edge (see Prop. 4.9)

PROPOSITION 4.10 (CHOOSEPIXEL FUNCTION RESULT IS NOT EMPTY). *The function $\boldsymbol{ChoosePixel}(cp, e, os, \mathcal{A})$ always returns a vertex, and therefore the function $\boldsymbol{Restore}(cp, \mathcal{A}, splitEdges)$ always produces in splitEdges the vertices that are necessary for rerouting the edge of the resulting arrangement so that the critical pixel cp is not critical in the arrangement considered in the next iteration.*

PROOF. As shown in Listing 1, the **ChoosePixel** function has three return statements that correspond to the three possible actions described in Tab. II. The structure of the conditions of **ChoosePixel** ensures that an action is always reached, since: (i) if $\mathbf{Fcvs}(\mathcal{A}, \mathbf{Cvs}(cp, e, os)) \neq \emptyset$, then a pixel of **FCVS** is returned; (ii) otherwise, if $cp \in \mathbf{Pe}(os)$, then the second action can always be performed, i.e. the reroute of e can be performed at least in *cp*, if no other candidates are available or perform better; (iii) finally, if $cp \notin \mathbf{Pe}(os)$, then Prop. 4.9 ensures that **Cvs** contains at least two

pixels, thus, there are useful pixels for performing the reroute of e and the function **ChoosePixel** always returns a not empty result. □

PROPOSITION 4.11 (SRR TERMINATION). *SRR terminates after a finite number of steps.*

PROOF. SRR can generate new hot pixels because it can introduce new vertices also inside pixels that were not hot at the beginning. However, from Prop. 4.7 any new vertex created in a chain by SRR belongs to the pixel envelope of its original segment, and this pixel can become critical only if it contains or will contain the edge of another chain. From this and Prop. 4.8 it follows that a pixel can become critical, namely it may contain a vertex v and an edge e at some SRR iteration, only if v will be a vertex of a chain related to an original segment os_1 and e will be an edge of a chain related to an original segment os_2 such that:

$$\mathbf{Pe}(os_1) \cap \mathbf{Ms}(os_2) \neq \emptyset$$

where $\mathbf{Ms}(\mathbf{os})$ represents the *Minkowski sum* of *os* and a unitary cell centered at the origin. Since the number of pixels in this intersection is finite, it follows that even in the worst case the number of new generated critical pixels is finite and the algorithm terminates. □

4.3.2. Preservation of the Properties of SR (and of ISR). In this section we show that the properties of SR (see Section 3) and the minimum distance property of ISR are preserved also by SRR.

PROPOSITION 4.12 (SRR FIXED-PRECISION REPRESENTATION). *Given an arrangement $\mathcal{A}$ and a grid of pixels, if a chain of the arrangement produced by SRR crosses a hot pixel then it passes through its center.*

PROOF. Obviously from Prop. 4.6 and Prop. 4.11 (termination). □

PROPOSITION 4.13 (SRR GEOMETRIC SIMILARITY). *Each output chain produced by SRR lies in the Minkowski sum of its original segment and a unitary cell centered at the origin.*

PROOF. Obviously from Prop. 4.8 and Prop. 4.11 (termination). □

PROPOSITION 4.14 (SRR REDUCTION TO SR). *The output of SRR is equivalent to the output of SR applied to an arrangement where all new vertices added by SRR have been added to the input chains associated to the initial set of segments $\mathcal{S}$.*

PROOF. The set of hot pixels produced by SRR can be partitioned into two sets: the initial set of hot pixels corresponding to the endpoints of the original segments (H) and the set of hot pixels ($H_{restore}$) created by the successive iterations of SRR as a result of the application of the function **Restore**. The hot pixels of H are those which have been created by SR, therefore the different result produced by SRR is due only to hot pixels of $H_{restore}$. However, from Prop. 4.7 the pixels of $H_{restore}$ always belong to the pixel envelope of an original segment of $\mathcal{S}$. Therefore, each final chain c always has vertices that lie inside $\mathbf{Pe}(os)$, where *os* is the original segment of c. Therefore, the same result produced by SRR can be obtained as follows: (i) starting from $\mathcal{S}$, for each pixel $hp \in H_{restore}$ a new vertex is added to the chain associated to each original segment $os \in \mathcal{S}$, such that $hp \in \mathbf{Pe}(os)$; this vertex has to be located on *os* and inside *hp*. (ii) an iteration of SR is applied. Since $H_{restore}$ contains the hot pixels generated by SRR, no additional critical pixels can be produced by this SR step, which therefore returns the same arrangement produced by SRR, since each chain is snapped to all the hot pixels that SRR have produced ($H \cup H_{restore}$). □

PROPOSITION 4.15 (SRR TOPOLOGICAL SIMILARITY). *SRR preserves the topology of the input arrangement in the same sense that SR does.*

PROOF. SR preserves topological properties since it can be viewed as a continuous deformation of segments into chains such that no vertex of the input arrangement ever crosses through a segment, as shown by Guibas et al. in [Guibas and Marimont 1995]. Thus, the proof follows directly from Prop. 4.14. □

PROPOSITION 4.16 (SRR MINIMUM DISTANCE). *In an arrangement $\mathcal{A}$ produced by SRR each vertex is at least half a unit away from any non-incident edge.*

PROOF. This property follows directly from the algorithm behavior (Listing 1): indeed, the algorithm terminates when the cycle **while** $CPS \neq \emptyset$ **do** ends i.e., when no other critical pixel is produced by the **Restore** procedure. The proof follows from this and Prop. 4.11 about the algorithm termination. □

PROPOSITION 4.17 (SRR RESULT CANONICITY). *The arrangement produced by SRR does not depend on the order in which the critical pixels are considered in the algorithm iterations.*

PROOF. The set of critical pixels is computed by the **CriticalPixel** procedure before starting each iteration of the *while* cycle (see Listing 1), thus it is invariant during each iteration. Moreover, the **Restore** procedure always works on the same arrangement $\mathcal{A}$ during an iteration; indeed, any edge rerouting produced by **Restore** has no effect on the current arrangement, but only on the one that will be available in the next iteration of the *while* cycle. This is shown clearly in Listing 1, since the arrangement $\mathcal{A}$ for the next iteration is computed by the **UpdateArrangement** procedure after the end of the *foreach* cycle. **UpdateArrangement** uses the set *splitEdges* produced by **Restore**, which contains the edges that require a reroute operation. Therefore, the order in which the critical pixels cp of the set CPS are considered during the *foreach* cycle is irrelevant w.r.t. the resulting arrangement of each iteration. Thus, it has no influence on the final result of the SRR algorithm. □

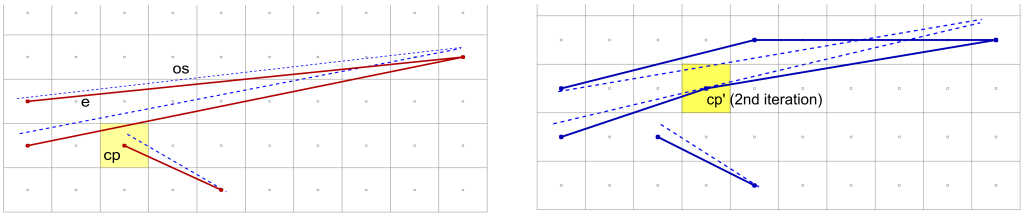
Finally, notice that SRR output of any original segment is always weakly-monotone. This is a direct consequence of the behavior of the **CriticalPixel** procedure (see Table II) that always chooses pixels belonging to the pixel envelope of the original segment, thus in the worst case vertical edges are generated, but never edges that go in an opposite direction w.r.t. the direction of the original segment.

4.4. Optimizing the Quality of the Approximation

A crucial aspect of the SRR algorithm is the effort to produce a good approximation of the original setting. Two approximation criteria are used: geometrical similarity and topological similarity. A detailed discussion of these criteria is performed in Section 5; here it is sufficient to state that the function **ChoosePixel** tries to minimize the distance between each chain and its original segment in order to maximize geometrical similarity, and tries to minimize the number of snapping operations in order to maximize topological similarity. Unfortunately, in some situations there is a conflict between these criteria; in this case the minimization of snapping operations has been considered more important, since the maximum distance between a chain and its original segment is bound anyway, as shown by Prop. 4.13 . The effect of this choice will be analyzed in Section 5.

As stated by Prop. 4.9, the set *CVS* of pixels among which **ChoosePixel** performs a choice has a cardinality greater than 1 and therefore a choice is effectively possible. The fundamental criterion adopted by **ChoosePixel** is: choose the pixel $p \in \mathbf{Cvs}(cp, e, os)$ which is nearest to the original segment *os*. The motivation of this

criterion is obviously to maximize the geometrical approximation of the result to the original arrangement. However, this is not always possible or convenient, because the pixel which is nearest to the original segment may contain an edge, thus giving rise to the creation of a new critical pixel and hence to an iteration of the **Restore** procedure, or may produce new edges that cross another hot pixel, thus eventually giving rise to a snap operation. For these reasons the **ChoosePixel** function must use the more articulated criteria illustrated by Table II than just the minimization of distance. Their application and motivation is explained referring to some examples which are built by adding some segments to the configuration of Fig. 6.a in order to make the added pixel $\langle 4,3\rangle$ critical and cause a second iteration of SRR. Therefore, in all following examples we are considering the rule applied by **ChoosePixel** during the second iteration of SRR in order to eliminate the critical pixel $\langle 4,3\rangle$ produced during the first iteration, by always applying Rule 3. In all figures the result of SR is shown on the left and the result of SRR is shown on the right.



(a) Output of SR. (b) Output of *Restore* after two iterations.

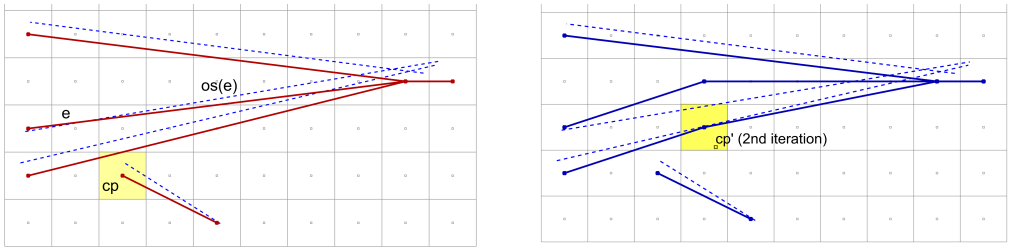
Fig. 8. Example of application of Rule 1 (optimal choice).

Rule 1 – There are two possible cases:

— **Optimal choice (with respect to geometric approximation)** - In Fig. 8 a third segment has been added to the arrangement in Fig. 6.a so that the edge e produced by SR (and also its original segment *os*) crosses pixel $\langle 4,3\rangle$. Since all pixels of $\mathbf{Cvs}(\langle 4,3\rangle, e, os)$ are free, $\mathbf{Fcvs}(a, \mathbf{Cvs}(\langle 4,3\rangle, e, os)) = \mathbf{Cvs}(\langle 4,3\rangle, e, os)$ and Rule 1 chooses the pixel $\langle 5,4\rangle$ having minimum distance from the original segment. This is an example of configuration where the second iteration of SRR produces its optimal result (as shown in Fig. 8.b): the topology of the arrangement produced by SR is completely maintained by SRR and the geometric approximation is the best one in the given scene.

— **Suboptimal choice** - Fig. 9 shows the application of Rule 1 in a case where the pixel of optimal choice ($\langle 5,4\rangle$) is not free. Since the set $\mathbf{Fcvs}(a, \mathbf{Cvs}(\langle 4,3\rangle, e, os))$ is not empty, Rule 1 is still applied (pixel $\langle 4,4\rangle$ is chosen). Notice that, since in this case the critical pixel $\langle 4,3\rangle$ belongs to the envelope of the original segment *os*, a snapping operation of edge e to pixel $\langle 4,3\rangle$ (i.e. an application of Rule 2) would produce a better geometrical approximation. However, the rules are designed to give more importance to the avoidance of snapping operations than to the minimization of the distance from the original segment (notice that the result remains in the pixel envelop anyway).

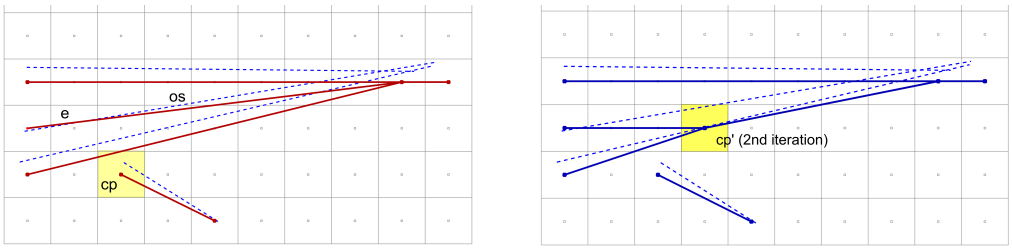
Rule 2 – In Fig. 10 Rule 2 is applied: since there are no pixels in $\mathbf{Fcvs}(a, \mathbf{Cvs}(\langle 4,3\rangle, e, os))$, but the critical pixel $\langle 4,3\rangle$ belongs to $\mathbf{Pe}(os)$, the critical pixel is used for snapping. In this case SRR performs a *Snap* operation since there is no place for spreading the edge e without producing a new critical pixel and consequently an



(a) Output of SR. (b) Output of **Restore** after two iterations.

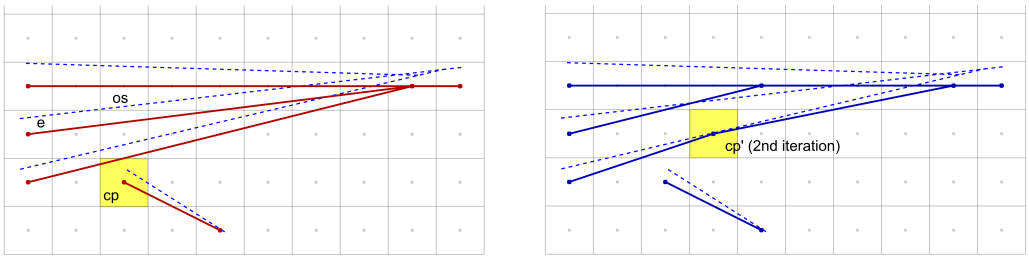
Fig. 9. Example of application of Rule 1 (suboptimal choice).

additional iteration; the avoidance of iteration is considered more important than the avoidance of snapping, since the creation of new critical pixels may lead to snapping anyway.



(a) Output of SR. (b) Output of **Restore** after two iterations.

Fig. 10. Example of application of Rule 2 (snapping).



(a) Output of SR. (b) Output of **Restore** after two iterations.

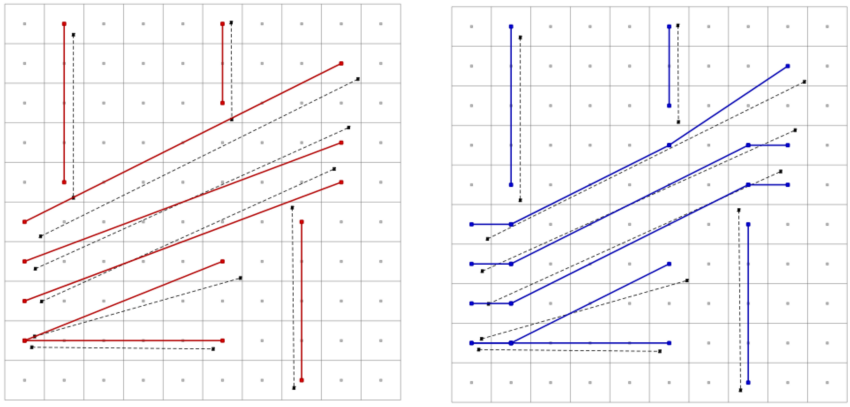
Fig. 11. Example of application of Rule 3 (optimal choice but new critical pixel and new iteration with snap).

Rule 3 – Finally, Fig. 11 shows the application of Rule 3: since there are no pixels in $\mathbf{Fcvs}(a, \mathbf{Cvs}(\langle 4,3\rangle, e, os)$ the configuration is similar to the previous example, but in this case the critical pixel does not belong to $\mathbf{Pe}(os)$ and therefore it is not possible to perform a snapping. In this case during the second iteration the minimum distance pixel ($\langle 5,4\rangle$) is chosen although it is crossed by an edge, and the newly created hot pixel becomes critical causing a third iteration, which leads to a snapping operation. Rule 2 is applied during the third iteration and the final result is shown in Fig. 11.b.

4.5. Examples of the Behavior of SRR

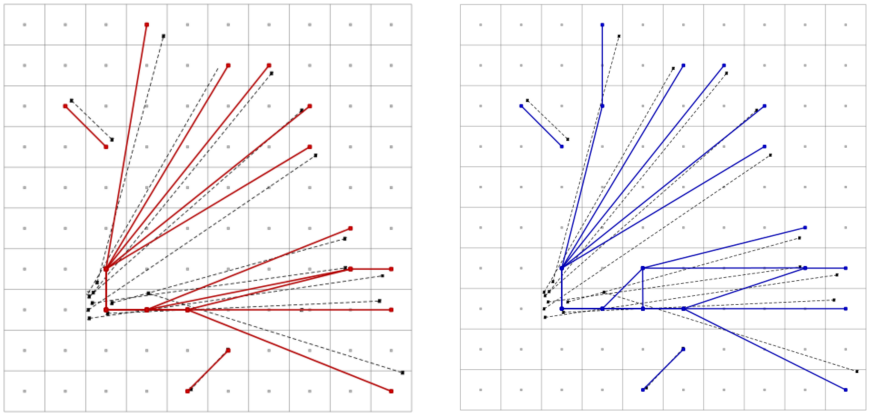
The following two examples illustrate the behavior of SRR compared with SR. The first example in Fig. 12 shows how SRR tries to keep the topological structure of the input arrangement; of course, this is possible only if there is enough space between the segments.

A more complicated case is shown in Fig. 13, where many segments converge in a few pixels. In the upper left half of the picture the segments are relatively less dense, while in the bottom right half the density is higher. While in the less dense portion the topological structure is maintained, in the more dense portion many snapping operations have modified it. Notice that in this area also the SR algorithm has altered the topological structure.



(a) Output of SR. (b) Output of SRR.

Fig. 12. SR vs SRR on a specific dataset, which is representative of configurations where some segments are almost parallel.



(a) Output of SR. (b) Output of SRR.

Fig. 13. SR vs SRR on a specific dataset, which is representative of configurations where many segments converge in the same pixel.

4.6. Implementation of the Functions and Complexity of SRR

This section presents some implementation aspects of the SRR algorithm and discusses its complexity. In particular, Section 4.6.1 proposes some useful data structures that can improve SRR performance, Section 4.6.2 illustrates how these data structures can be computed in the preliminary step of SRR and finally Section 4.6.3 analyses the algorithm complexity.

4.6.1. Data Structures Used by the Implementation of SRR. In the algorithm **SRR** presented in Listings 1 and 2 each iteration of the main loop requires three sequential scans of the arrangement $\mathcal{A}$ in order to find: (I scan) all edges which cross the critical pixel *cp* being considered (in the main loop of **Restore**); (II scan) the new critical pixels that can be produced by the new edges created by **Restore** (in the **CriticalPixels** function) and (III scan) the set of edges which cross the pixel frame **Pf**(*cp*) of a critical pixel cp (in function **Fcsv**$(\mathcal{A}, cvs)$ of the **ChoosePixel** procedure). In the sequel we propose to introduce auxiliary data structures in order to avoid these scans of $\mathcal{A}$:

— **(I scan)** This scan can be avoided by transforming the critical pixel set *CPS* into a map *CPSMap* that associates to each critical pixel the set of the edges which cross it. Whenever a critical pixel is discovered by the **CriticalPixels** function, the edges crossing it are known and can be easily stored in *CPSMap*.
Therefore, the first 2 lines of the **Restore** procedure, i.e. the main for loop (**for** $e \in \mathcal{A}$ **do**) and the if statement (**if** $e.crosses(cp)$ **then**), can be substituted with **for** $e \in CPSMap(cp)$ **do**.
The *CPSMap* structure is computed initially by the first call of the **CriticalPixels** function, then it is updated when a new hot pixel h is created by the **Restore** procedure; new critical pixels can be generated by two different processes:
  (1) h has become critical itself: this is the case when the **Restore** procedure applies Rule 3, choosing a pixel which was not free. In this case the **Restore** procedure adds h to *CPSMap*, which is being prepared for the next iteration together with the set of edges which cross it.
  (2) At least one of the two newly created edges has invaded an existing hot pixel, thus making it critical. In this case the update of *CPSMap* is performed by the next call of the **CriticalPixels** function.

— **(II scan)** In order to reduce the complexity of this operation, the following idea is applied: since (i) any edge e of $\mathcal{A}$ produced by an iteration of SRR is inside the Minkowski Sum of any original segment $os \in OS(e)$ and a unitary cell centered at the origin (Prop. 4.8), and (ii) the **Restore** procedure can generate new hot pixels only inside the pixel envelope **Pe**(os) of the considered original segment os (Prop. 4.7), it is possible to pre-compute some information which can be used for reducing the number of hot pixels that have to be analyzed by the **CriticalPixels** function. A new data structure is created for these pre-computed information and is called *Candidate Critical Edge Map* (*CCEMap*). It stores, for each edge e of $\mathcal{A}$, the set of edges with end-points in hot pixels which could be invaded by the edges produced by splitting edge e. These hot pixels are candidate new critical pixels, when e is split.

— **(III scan)** With the same reasoning we can pre-compute some other information which can be used for reducing the number of edges to be considered for computing the function **Fcsv**$(\mathcal{A}, cvs)$ in **ChoosePixel**. These auxiliary information regard the set of edges which cross a pixel which could be used for splitting edge e. We have chosen to add these critical edges in the *CCEMap* structure, avoiding the introduction of a new one, thus reducing the space requirements of the optimized algorithm.

The map *CCEMap* is created during the first iteration of SRR and updated at the end of each iteration of the main loop of SRR after the call to the **CriticalPixels** procedure.

The rules for these update operations are presented in the next subsection, while the algorithms used by the functions **CriticalPixels** and **Fcvs** are shown in Listing 3. These algorithms use the *CCEMap* and *CPSMap* and *splitEdges* data structures and the following definition.

Definition 4.18 (*Pixel Boundary*). The *pixel boundary* of an edge e, denoted as **Pb**(e), is the pair of pixels which contain the start and end points of e. □

LISTING 3: CriticalPixels procedure & Fcvs function

CriticalPixels procedure
Input: The maps *CPSMap*, *CEEMap* and *splitEdges* containing the set of edges that have been split by *Restore*.
Result: The modified map *CPSMap*.

```
begin
    foreach (e, os) ∈ splitEdges.getEdges() do
        foreach e' ∈ CCEMap(e) do
            foreach e_i ∈ e.split(splitEdges.getVertices(e, os)) do
                (p_start, p_end) = Pb(e');
                if e_i.crosses(p_start) then
                    CPSMap.add(p_start, e_i);
                end
                if e_i.crosses(p_end) then
                    CPSMap.add(p_end, e_i);
                end
            end
        end
    end
end
```

Fcvs function
Input: The current edge e, the set *Cvs* of available pixels for rerouting e and the map *CCEMap*
Output: *Fcvs* representing the set of free candidate pixels for rerouting e.

```
begin
    Fcvs ← Cvs
    foreach e' ∈ CCEMap(e) do
        foreach pixel p ∈ Cvs do
            if e'.crosses(p) then
                Fcvs.drop(p)
            end
        end
    end
    return Fcvs
end
```

In function **CriticalPixels** *CPSMap.add*(p, e) is a procedure that updates *CPSMap* as follows: if p is not already present in *CPSMap* then it adds p to *CPSMap* with the associated edge set equal to $\{e\}$, otherwise it adds e to the associated edge set of p.

The optimized implementation of the algorithm SRR of Listing 1 is presented in Listing 4. Notice that, the function **ChoosePixel** receives in input the map *CEEMap* instead of the arrangement $\mathcal{A}$, moreover it modifies, when necessary, the map *CPSMap** as requested by Rule 3 of the **Restore** procedure. Finally, the function *CCEMap.update* will be defined in the next subsection.

LISTING 4: Snap-Rounding with Restore implementation

```
SRR
Input: An finite-precision set of segments S and a grid g with unitary pixels.
Output: A fixed-precision arrangement A of S.
begin
    A ← emptyArrangement;
    Snap-Rounding(S, g, *A);
    CPSMap ← ∅; CCEMap ← ∅;
    FirstScan(A, *CPSMap, *CCEMap);
    while CPSMap ≠ ∅ do
        CPSMap^0 ← CPSMap; splitEdges ← ∅;
        foreach cp ∈ CPSMap do
            Restore(cp, CPSMap, CEEMap, *CPSMap^0, *splitEdges);
        end
        UpdateArrangement(splitEdges, *A);
        CriticalPixels(CCEMap, splitEdges, *CPSMap^0);
        CPSMap ← CPSMap^0;
        CCEMap.update(splitEdges);
    end
end
–
Restore procedure
Input: A critical pixel cp, the current maps CPSMap and CCEMap, the CPSMap under
       modification, denoted CPSMap^0, and the set under modification containing the edges to
       be split, denoted splitEdges^0.
Result: The modified structures splitEdges^0 and CPSMap^0.
begin
    foreach e ∈ CSPMap(cp) do
        foreach os ∈ OS(e) do
            newVertex ← ChoosePixel(cp, e, os, CEEMap, *CPSMap^0);
            splitEdges^0.addEdge(e, os, newVertex);
        end
    end
    CSPMap^0.drop(cp);
end
```

4.6.2. Pre-computing Interferences between Edges. This section analyzes how the auxiliary data structure Candidate Critical Edge Map *CCEMap* can be pre-computed in order to avoid a complete search over the arrangement at each iteration. In particular, the following propositions state a set of useful condition for performing such computation. Fig. 14 illustrates many of the concepts used in the following definitions and propositions.

PROPOSITION 4.19. *Let cp be a critical pixel, e be an edge crossing it, $s \in OS(e)$ be the original segment of e which is being considered in the inner loop of the **Restore** procedure. If h is a pixel which could be chosen in order to split and reroute e, and e' is an edge which crosses h so that h must be eliminated from the set fcvs of free candidate pixels, then all original segments $s' \in OS(e')$ must satisfy the following condition: $distance(s, s') < (3/2)\sqrt{2}$.*

PROOF. Let p be a point which belongs to the intersection of e' and h. (i) Since by Proposition 4.7, e' belongs to the Minkovski Sum of all original segments $s' \in OS(e')$,

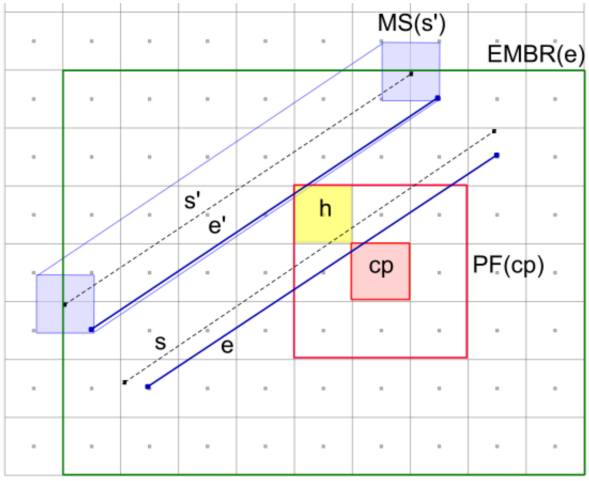


Fig. 14. Examples of pixel frame ($PF(cp)$) of a pixel cp, extended MBR ($EMBR(e)$) of an edge e, Minkowski Sum ($MS(s')$) of an original segment s' and a critical edge e' of a given edge e.

the maximal distance of p from any s' is $\sqrt{2}/2$. Moreover, (ii) since by the rules of the **Restore** procedure, $h \in$ **Pe**(s), the maximal distance of p from s is $\sqrt{2}$. Therefore, since p has to satisfy both conditions (i) and (ii), it follows that the distance between s and s' has to be less than $(3/2)\sqrt{2}$. □

Definition 4.20 (*Pixel and Extended MBR*). The *pixel MBR* and the *extended MBR* of an edge e, denoted as *PMBR*(e) and *EMBR*(e), are set of pixels defined as follows: *PMBR*(e) is the set of pixels which intersect *MBR(e)*, while *EMBR*(e) is the set *PMBR*(e) augmented by the pixels which are adjacent to it. Fig. 14 illustrates the *EMBR* of the edge e. □

PROPOSITION 4.21. *Let cp be a critical pixel, e be an edge crossing it, h be a pixel which could be chosen in order to split and reroute e, and e' be an edge which crosses h so that h must be eliminated from the set fcvs of free candidate pixels; then e' intersects EMBR(e).*

PROOF. By the rules of the **Restore** procedure, $h \in$ **Pf**(cp). Moreover, since e crosses *cp* and h is adjacent to a pixel crossed by e, $h \in$ *EMBR*(e) Therefore, from the fact that e' crosses h, it follows that e' intersects *EMBR*(e). □

PROPOSITION 4.22. *Let e be an edge, $h \in$ Pb(e) be a hot pixel, and e' be an edge which crosses h; the following condition must hold between all pairs of original segments $s \in$ OS(e) and $s' \in$ OS(e'): distance$(s, s') < (3/2)\sqrt{2}$.*

PROOF. Let p be a point which belongs to to the intersection of e' and h. If $h \in$ *PB*(e), then by Proposition 3.1 $\forall s \in$ *OS*$(e)(h \in$ *PE*$(s))$, and therefore the maximum *distance*(p, s) is $\sqrt{2}$. Moreover, since e' is contained in Minkovskj Sum of s' by Proposition 4.7, the maximal distance of p from s' is $\sqrt{2}/2$. Therefore, since p has to satisfy both conditions (i) and (ii), it follows that the distance between s and s' must be less than $(3/2)\sqrt{2}$. □

Proposition 4.19 is very similar to Proposition 4.22, except for the fact that here all original segments must satisfy the condition on the distance. The reason of this

difference is that in Proposition 4.19 the creation of a hot pixel h for splitting an edge is being evaluated with respect to at least one original segment, while in Proposition 4.22 h is an already created hot pixel and therefore all the original segments of edges having this vertex must be considered.

PROPOSITION 4.23. *Let e be an edge, $h \in Pb(e)$ be a hot pixel and e' be an edge which crosses h; the following conditions must hold: (i) e' intersects PMBR(e), and (ii) e intersects PMBR(e').*

PROOF. Condition (i): *Pb*(e) is contained in *PMBR*(e) by definition, thus if $h \in$ *Pb*(e) then $h \in$ *PMBR*(e), and therefore if e' crosses h then e' intersects *PMBR*(e). Condition (ii), e' crosses $h \implies h \in$ *PMBR*(e'), while $h \in$ *PB*$(e) \implies e$ intersects h; therefore, e intersects *PMBR(e')*. □

Given the above proposition, the rules for building the Candidate Critical Edge Map of an edge e (*CCEMap*(e)) are defined by the following two conditions:

C.1 $e' \in$ *CCEMap*(e) only if there exists a pair of original segments $\langle s, s' \rangle, s \in$ *OS*$(e), s' \in$ *OS*(e'), such that *distance*$(s, s') < (3/2)\sqrt{2}$;
C.2 $e' \in$ *CCEMap*(e) only if *EMBR*(e) intersects *EMBR*(e').

PROPOSITION 4.24 (C.1 AND C.2 ARE SUFFICIENT FOR PRUNING EDGES IN SRR). *If two edges $\langle e, e' \rangle$ do not satisfy conditions C1 and C2, then: (i) e' cannot cross the pixel frame of a critical pixel crossed by e and therefore it does not need to be considered by function **Fcvs**; (ii) e' cannot cross a hot pixel $h \in PB(e)$ and therefore it needs not to be considered by function **CriticalPixels**.*

PROOF. The assertion in (i) is a direct consequence of Propositions 4.19 and 4.21, while the assertion in (ii) is a direct consequence of Propositions 4.22 and 4.23. □

Therefore, by Proposition 4.24, *CCEMap* satisfies the requirements for its use in Listing 3. In particular, conditions C1 and C2 are weaker than the conditions stated by the Propositions 4.19-4.23; therefore, the sets contained in *CCEMap* are larger than it could be achieved by a more complex data structure. However, they have the advantage of being symmetric, and thus easier to update. The function *CCEMap.update*(*splitEdges*), introduced at the end of SRR implementation in Listing 4, can now be defined in details as follows. For each entry $(e, os) \in$ *splitEdges*$.getEdges()$ the following steps are executed:

(1) the set of edges $\{e_1, ..., e_n\}$ obtained by splitting e using the expression $e.split($*splitEdges*$.getVertices(e, os))$ substitute e as key entries of *CCEMap*;
(2) for each $e' \in$ *CCEMap*(e), if conditions C1 and C2 hold for e' and $e_i \in \{e_1, ..., e_n\}$ the following update operations are executed:
— e' is inserted in *CCEMap*(e_i)
— e_i is inserted in *CCEMap*(e')

4.6.3. Complexity Analysis. Like SR, SRR is an output-sensitive algorithm since its complexity depends on the initial number n_h^0 of hot pixels in the arrangement produced by SR and on the number n_h^{new} of new hot pixels which are produced by the **Restore** procedure. n_h^0 depends on the number n_s of original segments and the number I of intersections among them. Indeed, trivially in the worst case, $n_h^0 = 2n_s + I$. With a similar reasoning, the initial number of edges n_e^0 in the arrangement produced by SR is equal, in the worst case, to $n_s + 2I$. Finally, let n_h be the number of hot pixels contained in the final arrangement; clearly $n_h = n_h^0 + n_h^{new}$ and let n_e be the number of edges contained in the final arrangement; clearly $n_e = n_e^0 - n_e^{split} + n_e^{new} = n_e^0 - n_h^{new} + 2n_h^{new} = n_e^0 + n_h^{new}$. Thus, n_h^{new} is the fundamental parameter for assessing the output complexity of SRR.

In order to capture the behavior of the implementation of SRR presented in this section, it is useful to consider some parameters which take care of the "density" of the arrangement; let d_{EP} be the maximum number of edges which can intersect a single pixel, and d_{EE} be the maximum number of edges which satisfy the conditions C1 and C2 for an edge e (i.e. the maximum cardinality of the sets returned by *CCEMap*(e)).

The following elements determine the computational complexity of SRR:

(1) The initial SR step – using the algorithm presented in [Hobby 1999] it has a complexity of $O((n_s + I)\log(n_s))$.
(2) The execution of the **FirstScan** procedure – by applying a plane-sweep approach, which has to compute the maps *CPSMap* and *CCEMap*; it has a cost of $O(n_e^0 \log(n_e^0))$, since the initial number of edges of the arrangement $\mathcal{A}$ is n_e^0 and the auxiliary data structures require $\log(n_e^0)$ for access.
(3) The cost of the i-th iteration of the **Restore** procedure (let n_{cp}^i be the number of critical pixels created by iteration i and processed by iteration $i+1$) – **Restore** has to determine, for each critical pixel produced by the previous iteration (in total n_{cp}^{i-1} pixels), all the edges which can interfere with it (the upper bound of this edge number is given by density parameter d_{EP}). Therefore, its cost is $O(n_{cp}^{i-1} * d_{EP} * d_{EE} * log(n_e^i))$, where the logarithmic term is due to the access to the tree map *CCEMap* with n_e^i edges, and d_{EP} takes care of the cycle **for each** $e \in$ *CPSMap*(cp) **do** of **Restore**, and d_{EE} takes care of the scan of interfering edges in the **Fcvs** function.
(4) The cost of the i-th iteration of the **CriticalPixel** function – this function has to process each new edge $e \in$ *SplitEdges* created by **Restore** ($d_{EP} * n_{cp}^i$) and to scan all edges present in *CCEMap*(e) (d_{EE}). Therefore, its cost is $O(n_{cp}^i * d_{EP} * d_{EE} * log(n_e^i))$.
(5) The cost of updating the *CCEMap* at each iteration is $O(n_e^i \log(n_e^i))$. The same cost has the **updateArrangement** procedure at each iteration.

It follows that **Restore** and **CriticalPixel** have the same cost at each iteration and by summing all iterations the total cost is dominated by the term: $O(n_{cp} \cdot d_{EP} \cdot d_{EE} \cdot \log(n_e))$ where n_{cp} is the total number of critical pixels created by **Restore** and n_e is the number of edges contained in the final arrangement. Since n_h^{new} is an upper bound of n_{cp} then the total cost is: $O(n_h^{new} \cdot d_{EP} \cdot d_{EE} \cdot \log(n_e))$.

Clearly, the cost of the algorithm depends on the assumptions which can be done on the "density" parameters d_{EP} and d_{EE}. The worst case assumption is that they grow with the number of original segments n_s, but in many applications it is possible to consider them constants depending on the rounding factor. For example, consider a transport network: the number of roads (edges) which intersect a given area (pixel) does not depend on the total number of roads but on the density of roads and on the size of the area (pixel size, hence rounding factor). Moreover, since the goal of SRR is to preserve topology, it is not meaningful to use it with a rounding factor that collapses too many edges.

In the worst case, the terms d_{EP} and d_{EE} can be both substituted with n_e^0, representing the total number of edges generated after the initial SR iteration, and n_e with $n_e^0 + n_h^{new}$.

Finally, the following proposition states an upper bound to n_h^{new}. In the sequel we denote n_e^0 as n, since it characterizes the input of SRR.

PROPOSITION 4.25 (NEW HOT PIXELS GENERATED BY SRR (UPPERBOUND(n_h^{new}))). *The maximum number of hot pixels that SRR can generate is equal to* $n \cdot (n-1)$.

PROOF. The construction process of the SRR result can be simulated by supposing to start from the arrangement $\mathcal{A}$ produced by the initial SR step and by adding to an

initially empty arrangement one edge of $\mathcal{A}$ at a time. In this process an edge shared by several chains is counted only once.

By adding a new edge e to an arrangement, e generates new hot pixels and thus possibly new critical pixels in two cases: (i) the end points of e becomes critical pixels; (ii) e interior crosses pixels containing a vertex of the arrangement (previously existing hot pixels).

In the first case, by propagating the rerouting procedure, it is possible through spreading propagation to introduce a number of new hot pixels equal to $2(i-1)$, where i is the current number of edges in the arrangement, namely $(i-1)$ hot pixels for each end point of the added edge.

In the second case, the situation corresponds to the propagation of the effects of another previously inserted edge, that propagates its drift also to the new added one, thus for every previously inserted edge, in total $(i-1)$ edges, two new hot pixels can be generated, and overall $2(i-1)$ pixels.

However, the two cases cannot occur together with the maximum cardinality, as shown in Fig. 15. In this figure the worst case configurations (i.e. those that produce the maximum number of hot pixels) between an existing edge e (and its original segment s) and the inserted edge e' (and its original segment s') are shown. Notice that in Fig. 15.a two new hot pixels are created (in the figure the arrows point to them) and they split the edge e. These two new hot pixels can become critical with respect to other edges (not shown in the figure for simplicity) and thus producing a spreading propagation obtaining in the worst case $2(i-1)$ new hot pixels. In Fig. 15.b the produced hot pixels are always two but one pixel splits e and the other one e' and thus again in the worst case $2(i-1)$ new hot pixels are generated. In conclusion, in every possible configurations two non intersecting segments cannot produce more than two hot pixels, thus with propagation in total no more than $2(i-1)$ pixels. There are other two possible configurations for two non intersecting segments (corresponding to orthogonal segments), which are not shown in the figure, since they only create one hot pixel, thus producing in the worst case $(i-1)$ new hot pixels.

Summing up on all edges generated after the initial SR step, the total number of new hot pixels is:

$$\sum_{i=1}^{n} 2(i-1) = 2\sum_{i=1}^{n}(i-1) = 2\frac{n(n-1)}{2} = n(n-1) \quad (1)$$

□

By combining the cost $O(n_h^{new} \cdot d_{EP} \cdot d_{EE} \cdot \log(n_e))$ with the result of Equation 1 about n_h^{new}, it follows that the algorithm complexity in the worst case scenario ($d_{EP} = n$ and $d_{EE} = n$) is $O(n^4 \cdot log(n))$, but by considering the assumptions on the expected values for the density parameters d_{EP} and d_{EE} in real applications, the complexity in many cases is reduced to $O(n^2 \cdot log(n))$.

Finally, the upper bound for the total number of vertices of the arrangement produced by SRR is $2n + n(n-1) = n^2 + n$.

5. SRR APPROXIMATION AND COMPARISON WITH ISRBD

The aim of the rounding algorithm is to approximate the original dataset, but different application domains may give more importance to different approximation criteria. Often there exists a conflict between such criteria and this makes the definition of an algorithm which is the best under all circumstances very difficult.

This section shows why SRR may be more suited for some rounding applications than ISRBD, which is the only known algorithm with the same guaranteed output properties. It is important to distinguish between the guaranteed properties, i.e. worst

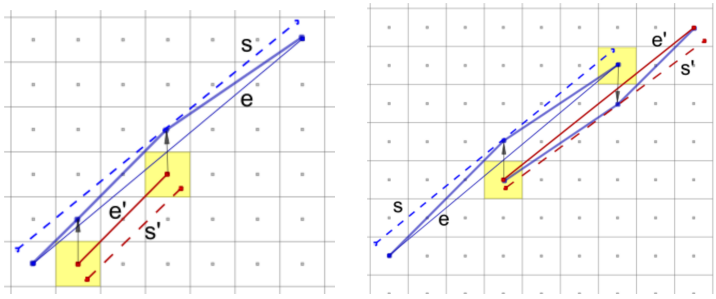


Fig. 15. Two possible configurations of non intersecting segments that produce two new hot pixels when SRR is applied (see, proof of Prop. 4.25). The inserted edge is denoted by e' (and its original segment by s'), while the existing edge is denoted by e (and its original segment by s); the arrows show the additional hot pixels.

case properties, and the average characteristics of the output produced by a rounding algorithm. As proved in the previous section, SRR guarantees, like ISRBD, the same guaranteed properties of SR listed in Section 3, namely *Fixed precision*, *Guaranteed Geometric Similarity* and *Topological Similarity*. However, these are worst case properties; the average quality of the output datasets can be very different among different algorithms. This section compares the average geometric similarity and topological similarity of SRR and ISRBD through a reasoning approach; while the next section shows the result of a statistical analysis.

5.1. Geometrical Similarity

There is a general agreement in literature in considering the Hausdorff distance between the original segments and the corresponding output chains as a measure of the *geometric similarity*. The following arguments suggest that the average geometrical approximation of SRR is better than the one of ISRBD; these arguments are not proofs of an average superiority, and counterexamples can be created where SRR is worse. However, this kind of reasoning is useful to give a foundation to the results of the statistical analysis.

Observation 1. SRR guarantees the same maximum distance as SR, while even if the user-specified threshold δ is taken at its minimum value, ISRBD can guarantee only 3 times the maximum distance granted by SR.

PROOF. The proof can be found in [Packer 2008]. □

Observation 2. The approximation obtained by performing a Spread operation on critical pixels produced by the SR step is always better than the approximation obtained by performing a Snap operation.

PROOF. After the initial SR step, a critical pixel can be produced only because a segment has invaded a hot pixel making it critical: indeed, if the original segment already crossed the pixel, then the SR step would have snapped it. Therefore, the original segment has been moved towards the critical pixel and as a consequence, the *Spread* operation moves the result back towards the original segment (by adding a vertex which belongs to the original segment envelope), thus reducing the distance, while the *Snap* operation moves it in the opposite direction, thus increasing the distance. □

This second observation means that, when a critical pixel allows a spreading operation, the approximation of SRR is better than the one of ISRBD. This should be the case if a dataset is not too dense.

5.2. Topological Similarity

In order to measure the topological similarity between the original arrangement and the produced one the following aspects are considered:

(1) **Number of snapping operations**: this elementary parameter is meaningful, since the snapping operation modifies the topological structure.
(2) **Degree of linear collapsing**: for each edge which is shared by different chains representing different original segments, the length of the shared edge multiplied by the number of corresponding segments gives an indication of the amount of collapsing segments.
(3) **Polygonal similarity**: if the original dataset represents a set of polygons, then also the degree of correspondence between the faces produced by the algorithm and the original arrangement constitutes an indication of the topological similarity.

The following general premise should be considered: there is a tradeoff in making a dataset robust and keeping the level of topological approximation, because in order to make the dataset robust space is needed and sometimes the lack of space leads to snapping and linear collapsing. Since SRR and ISRBD are applied after SR and both algorithms perform additional operations in order to eliminate critical pixels and obtain a robust dataset, in some cases they need to apply additional snapping operations and this might alter existing topological relations among segments. In situations where the density of segments is extremely high, the approximation will degrade, since a high level of snapping is necessary and it is almost impossible to perform any analysis of SRR and ISRBD behavior. However, in many applications, if a reasonable rounding is performed, than there can be dense spots, but they are surrounded by less dense space, where keeping the level of topological approximation is possible. Notice that, SR algorithms are usually applied with very fine grid, thus the above reasoning on data density has to be referred to such situations, i.e. where you have many segments in very small cells (10^{-3} or smaller). In geographical applications for example such kind of density is very rare.

Number of Snapping Operations. SRR is designed to avoid snapping, while ISRBD is based on snapping. In SRR a snapping operation is performed only if a critical pixel produced by SR causes a (sequence of) spreading operations which end up into a snapping operation, due to a set of high density pixels. It is reasonable to assume that at least in low density datasets the number of snapping operations performed by SRR is very low, while in very dense situations the degree of snapping of SRR could tend to the one of ISRBD. In the next section the statistical analysis confirms this assumption.

Linear Collapsing. The principal problem of ISRBD with respect to linear collapsing is due to the behaviour of the algorithm in presence of small angles between segments. In Fig. 1.a an initial setting is shown where a segment is tangent to a curve, which is represented as a sequence of small segments (usually called a linestring). Fig. 1.b shows the result of SR while Fig. 1.c and Fig. 1.d show the result of ISRBD and SRR, respectively. The larger line in Fig. 1.c shows the drift limit imposed by the "drift bound" of ISRBD; the minimum value of the drift bound δ stated in [Packer 2008] has been used, $\sqrt{2} \cdot 3/2$, thus minimizing the drift problem of ISRBD. Fig. 1.c allows one to observe that, although the drift of the segment is bound, this does not avoid a linear collapsing which is due to a different drift: the drift of the topological node

which represents the point of tangency between the segment and the curve. It is easy to construct examples to show that this kind of node drift can be very large if the angle between the segment and the linestring is small.

Polygonal Similarity. The example of Fig. 2.a shows a set of segments which are intended to represent three polygons. After SR the polygons are correctly represented by three faces, as shown in Fig. 2.b. In this case the application of ISRBD has a strong distortion effect on the polygonal topology, as shown in Fig. 2.c; this is measured by the fact that the number of faces has grown to five. This effect can have worse consequences if the original dataset was intended to represent a polygon with two holes, as in many types of applications; in this case the result of ISRBD represents a non valid geometry. Conversely, the result of SRR is topologically correct in this example, as shown in Fig. 2.d.

Notice that the number of faces can grow or shrink in consequence of snapping; this is due to the fact that a linear collapsing can eliminate a face or, like in Fig. 2.c, split one face into several faces. Therefore, a comparison of the number of faces cannot be used as an indication of the degree of polygonal similarity. Clearly, this is just an example; it is easy to build examples where SRR snaps almost as much as ISRBD. However, the goal of the example is to show that the impact of snapping and of linear collapsing on polygonal similarity can be very strong.

6. QUANTITATIVE EVALUATION OF ALGORITHMS

Several experiments have been performed using both randomly generated and real datasets.

6.1. Experiments on Randomly Generated Datasets

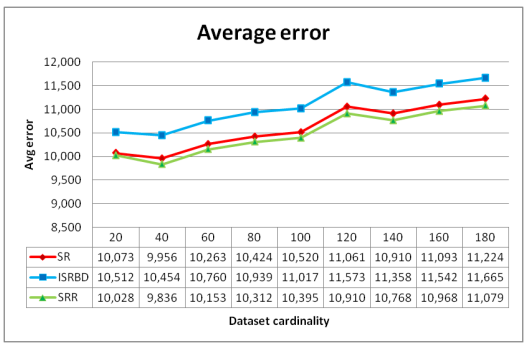
In this set of experiments we have considered datasets of segments between two points uniformly random generated in a square. All these sets have been generated in the same 100×100 pixels grid, so that the cardinality of each set becomes an indicator of the density of the segments. The test considers 10 different datasets for each cardinality.

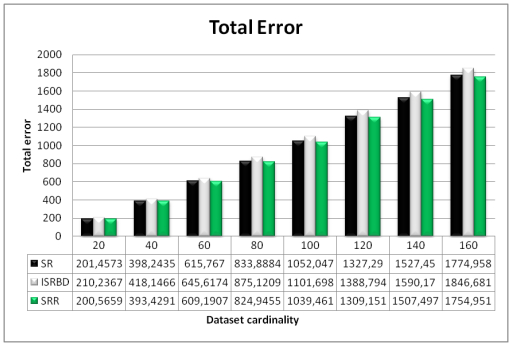
The following statistics have been collected:

— Error (total and average): the error measures the area delimited by each output chain on one side and the corresponding original segment on the other one. This measure gives at the same time an estimate of the Hausdorff distance and a hint on the area variation produced if segments are considered as polygon boundaries.
— Number of vertices in the resulting arrangement: this is an indicator of the result dimension.
— Percentage of hot pixels crossed by an edge that SRR and ISRBD consider w.r.t the number of vertices produced by SR: this is an indicator of the additional cost of ISRBD and SRR algorithms w.r.t. SR.
— Percentage of additional snapping operations performed by ISRBD and SRR w.r.t. SR.
— Percentage of additional collapsing segments (measured by counting the edge length n times if n is the number of chains that share it) performed by ISRBD and SRR w.r.t. SR.
— The execution time required by SRR, ISR and ISRBD with respect to SR.

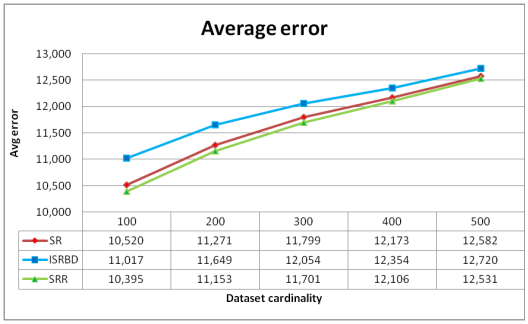
The obtained results are represented considering cardinality as subdivided into two categories: low cardinality up to 160 segments, since some spatial configurations can be obtained only in a non-dense set, and high cardinality up to 500 segments.

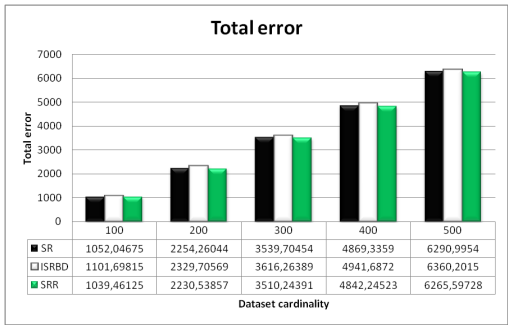
Figures 16(a) and (b) report the plots of the measured average and total error for each of the considered algorithms applied to low and high cardinality datasets, respec-





(a) Low cardinality datasets





(b) High cardinality datasets

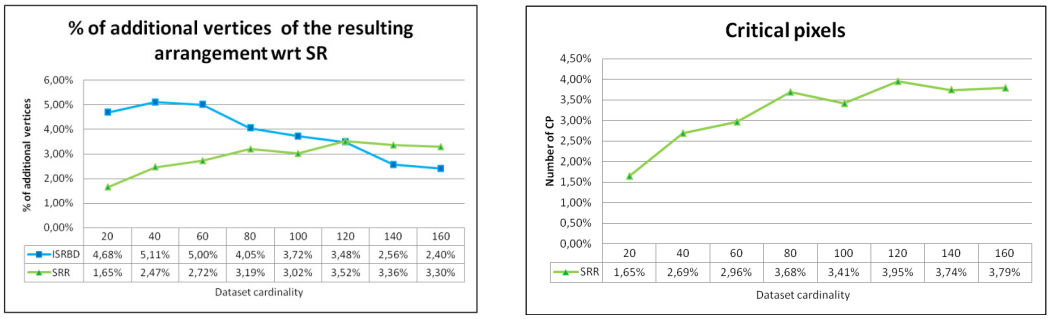
Fig. 16. Experiment results about average and maximum error of SR, ISR and SRR.

tively. Notice that, SRR has an average and total error that is always less than the one produced by the other two techniques.

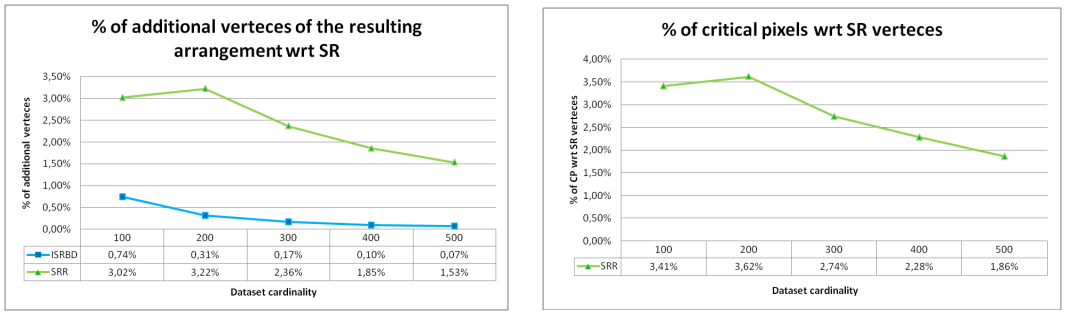
Figure 17 shows the plots of the number of additional vertices in the resulting arrangement and the number of critical pixels. Notice that, the number of additional vertices produced by SRR w.r.t. SR is always less than 4%. Moreover, the number of critical pixels considered by SRR is always less than 4% w.r.t the number of hot pixels of SR, thus the additional effort required by the SRR algorithm is acceptable since it grows linearly w.r.t. dataset cardinality.

Figures 18 (a) and (b) show the plots of the percentage of additional snapping performed by ISRBD and SRR w.r.t. SR. Notice that ISRBD in average performs more snapping than SRR. The difference is more evident considering the length of collapsing segments which is shown in the same figures. Finally, as expected, the difference between the two algorithms reduces when cardinality (i.e., segment density) increases, since snapping operations become dominant w.r.t. spreading ones.

Figures 19 (a) and (b) show the plots of the execution time of all the implemented algorithms w.r.t. the number of edges after the first SR step (n). All the algorithms have been implemented in Java 7u21 using the Java Topology Suite API for performing the required geometric operations, and have been executed on an machine with an Intel i5 2500K processor and 16GB of RAM. The tests have been performed by randomly generating several sets of edges with different cardinalities, starting from 50 to 400 with an interval of 50; for each cardinality 20 different sets have been processed. The ISR and ISRBD implementations use a plane-sweep approach during the search of new critical



(a) Low cardinality datasets



(b) High cardinality datasets

Fig. 17. Experiment results about the percentage of vertices of the produced arrangement and of critical pixels considered for ISR and SRR.

pixels in order to reduce the number of comparisons, while the SRR implementation uses the optimizations discussed in the previous section

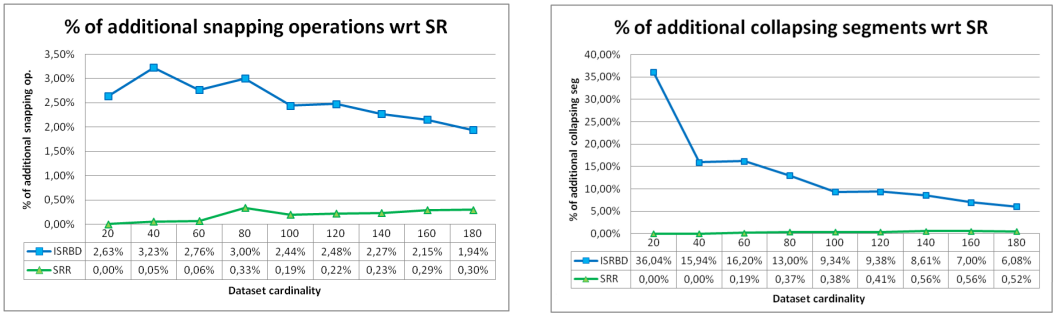
Notice that, as shown by Fig. 19 (a) SRR implementation requires an extra time which is similar to that one required by ISR and ISRBD and when the segment density increases SRR becomes the worst algorithms since the effect of d_{EP} and d_{EE} is highlighted; in Fig. 19 (b) only SRR and SR are shown in order to highlight the extra time required by SRR after the initial SR step. Finally, the plot of SRR confirms the dependency of SRR execution time on the parameter n expressed by $O(n^2 log(n))$.

6.2. Experiments on Real Datasets

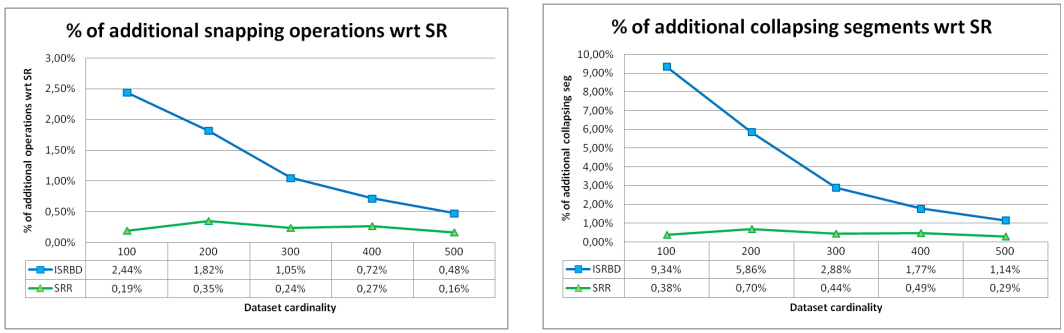
The SRR algorithm has been applied also on some real-world datasets and its output has been compared with the one produced by SR, ISR and ISRBD. In particular, two datasets have been used, which regard the road network of an Italian northern municipality. The two datasets will be denoted in the following as D_1 and D_2, respectively, and have these main characteristics:

— D_1 contains 6,254 MultiLinestrings with a total of 106,860 vertices.
— D_2 contains 3,342 MultiLinestrings with a total of 48,883 vertices.

The experiments have been performed considering for each dataset three different grids with a pixel size of 1.0, 0.1 and 0.01 meters, respectively. The obtained results are shown in Table III where the following information are reported for each algorithm and for each grid: the percentage of additional vertices w.r.t. to SR, the percentage of critical pixels w.r.t. the number of vertices produced by SR, the number of additional iterations performed after the first snapping one, the execution time required by the



(a) Low cardinality datasets



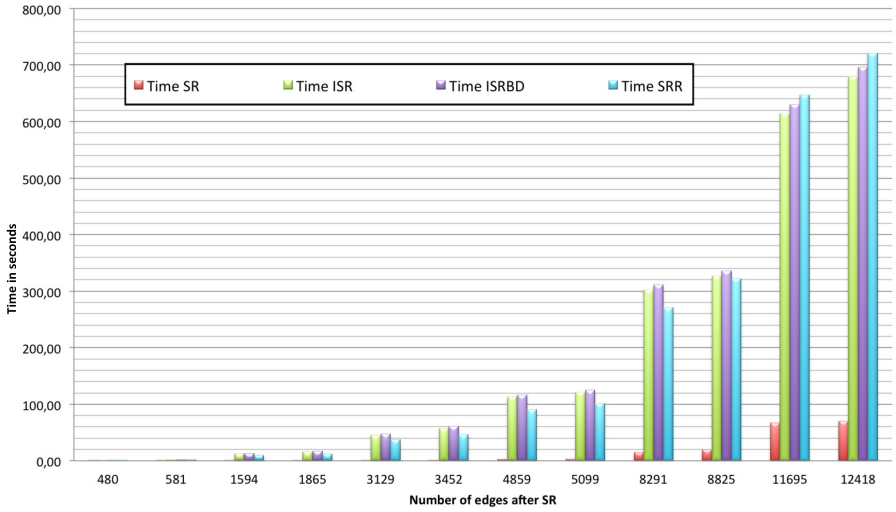
(b) High cardinality datasets

Fig. 18. Experiment results about percentage of additional snapping operations and of additional collapsing segments which have been produced by SRR and ISRBD w.r.t. SR.

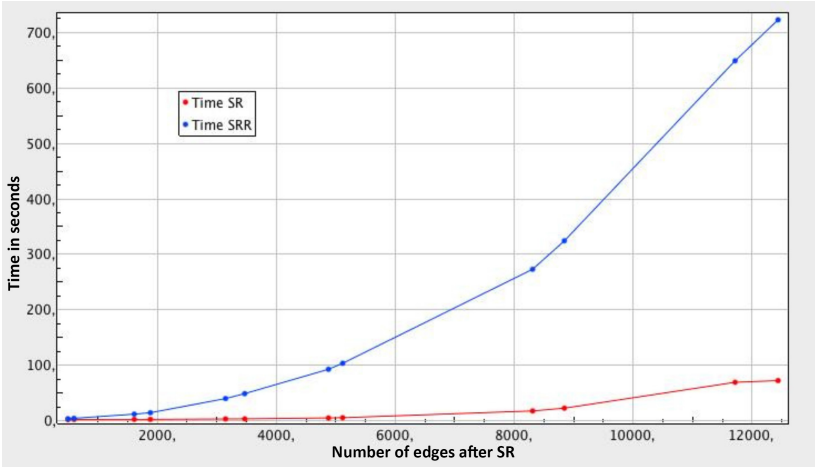
algorithm, the kind of rules applied by SRR, and some information about the geometric and topological similarity of the result. In particular, as regards to the geometric similarity, the table reports the total and average Hausdorff distance of the result w.r.t. to the original arrangement, considering firstly all edges and secondly only the edges treated with the Rule 1 by SRR. Conversely, as regards to the topological similarity, the table shows the additional vertex snapping and the additional edge collapsing performed by each algorithm.

The experiments highlight that the total and average distance from the original arrangement is smaller for the result produced by SRR, than the distance of the result produced by ISR and ISRBD. Similarly, the percentage of additional snapping is smaller for SRR than ISR and ISRBD, and such percentage decreases as the percentage of application of Rule 1 w.r.t. Rule 2 increases. Notice that Rule 3 is never applied, in other words every time Rule 1 cannot be applied, it happens that the critical pixel belongs to the pixel envelop of the original segment. This depends from the fact that, in real-word cases, datasets are not particularly dense, so that if there exists a pixel in the candidate vertex (i.e. a nearby pixel which allows the edge to exit the critical pixel) it is usually free.

Fig. 21 illustrates an example of situation in which SRR applies Rule 1 producing a result which is more similar to the initial situation than the one produced by ISR. In particular, (a) shows the initial situation where five vertices can be identified which are labeled from v_1 to v_5. The application of SR produces the situation in (b) where vertices v_1 and v_2 are collapsed into the center of pixel (2,1), while vertices v_3, v_4 and v_5 are collapsed into the center of pixel (3,1). In (b) pixel (2,1) is critical, since the upper



(a)



(b)

Fig. 19. Experiment results about the time performance of all algorithms and the time required by SRR w.r.t. SR. Both plots show the time in sec w.r.t. the number of edges produced by SR (which is the first step of each algorithm).

segment crosses it without passing through its center. The solution produced by ISR is illustrated in (c) where an additional vertices v_6 is added to the upper edge producing a collapse of the upper and lower segments. Conversely, SRR produces the solution in (d), where the additional vertex added to the upper segment is located at the center of pixel (2,2), maintaining the upper segment separated from the lower one.

7. CONCLUSION

A new algorithm SRR (Snap Rounding with Restore) has been defined for making an arrangement produced by the SR algorithm robust. It has been shown that SRR obtains the same goal as ISRBD, since it makes the arrangement robust, but it does not incur in the possibility of a degradation of the quality of approximation. It has been

Fig. 20. A portion of the real-world datasets considered for the experiments.

Table III. Column names have the following meaning: **PS** is the pixel size in meters, column **M** is the applied algorithm, **V** is the number of additional vertices w.r.t. to the ones produced by SR (reported in row **SR**). **CP** is the number of critical pixels found by the applied algorithm and **IT** is the number of additional iterations performed after SR, **T(s)** is the time in seconds, **THD** is the total Hausdorff distance and **AHD** is the average Hausdorff distance between the original segment and the resulting chain, computed considering only the segments that have been subjected to a spreading during SRR. Finally, **TS** is the number of additional snapping w.r.t. SR (reported in row **SR**) while **EC** is the number of additional edge collapsing w.r.t. SR (reported in row **SR**), and **Rules** contains the type of rule applied by SRR.

PS	M	V	CP	IT	T(s)	THD	AHD	TS	EC	Rules
Dataset 1										
1.0	SR	197,816	-	-	745	14.6644	0.5431	10,755	141	-
	ISR	+70	35	1	752	19.2885	0.7144	+35	+40	-
	ISRBD	+64	35	1	753	19.2885	0.7144	+35	+30	-
	SRR	+70	35	1	751	14.7345	0.5457	+8	+10	78% R1, 22%R2
0.1	SR	200,856	-	-	773	0.7651	0.0500	101,644	41	-
	ISR	+30	15	1	789	1.1064	0.0737	+15	+16	-
	ISRBD	+40	15	1	794	1.1064	0.0737	+13	+10	-
	SRR	+30	15	1	810	0.7976	0.0532	0	+8	99% R1, 1% R2
0.01	SR	201,214	-	-	805	0.0114	0.0057		19	-
	ISR	+2	2	1	807	0.0175	0.0087	+2	+2	-
	ISRBD	+6	2	1	816	0.0175	0.0087	+2	+2	-
	SRR	+2	2	1	814	0.0114	0.0057	+0	+1	100% R1
Dataset 2										
1.0	SR	89,672	-	-	159	5.1674	0.5741	46,111	64	-
	ISR	+32	16	3	164	7.9215	0.8801	+16	+16	-
	ISRBD	+14	17	3	173	7.9216	0.8802	+17	+10	-
	SRR	+28	14	1	163	5.1674	0.5742	+5	+5	65% R1, 35% R2
0.1	SR	90,948	-	-	168	0.1704	0.0569	46,081	37	-
	ISR	+12	6	3	177	0.2757	0.0919	+6	+10	-
	ISRBD	+6	6	3	189	0.2757	0.0919	+4	+7	-
	SRR	+8	4	1	170	0.1705	0.0568	+1	+4	75% R1, 25% R2
0.01	SR	91,080	-	-	156	0.0053	0.0053	46,057	20	-
	ISR	+2	1	1	158	0.0102	0.0102	+1	+2	-
	ISRBD	+8	1	1	164	0.0101	0.0101	+0	+2	-
	SRR	+2	1	1	157	0.0054	0.0054	+0	+1	100% R1

proved that SRR guarantees the same level of approximation of the original SR, and the experimental analysis shows that on average the obtained approximation is better.

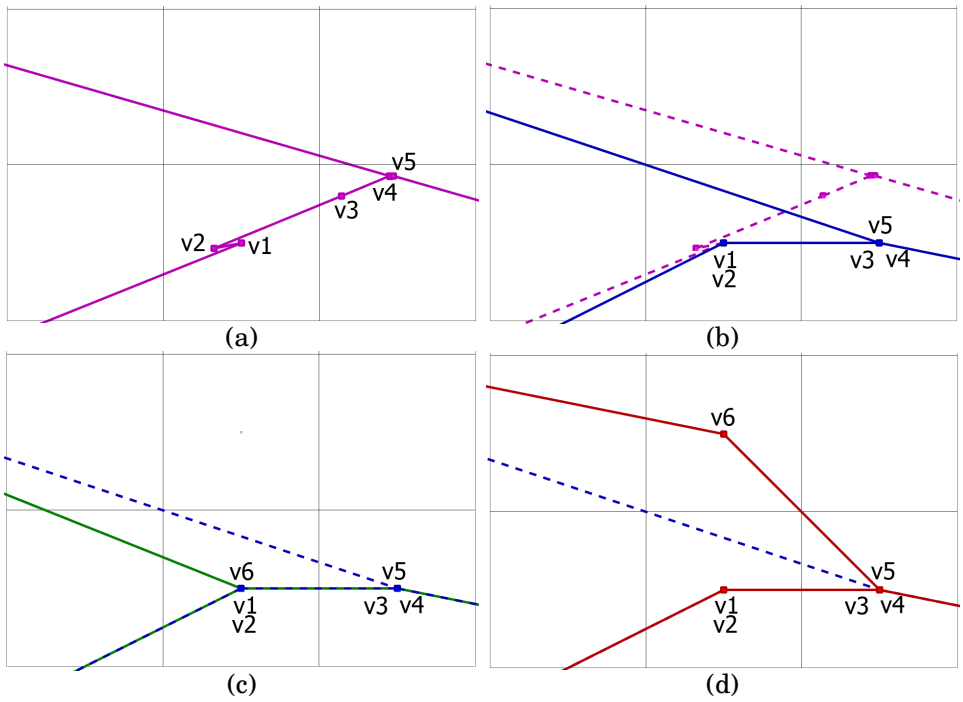


Fig. 21. Example from the experiments on real data: (a) Initial situation in purple, (b) SR result in blue, initial situation in dashed purple. (c) ISR result in green after one iteration, SR result in dashed blue. (d) SRR result in red, SR result in dashed blue.

SRR tries to perform only few additional snapping operations after those performed by SR, thus preserving the topological structure of the input dataset better than ISRBD. In fact, it is impossible to completely avoid to perform snapping in those situations in which the density of the geometries with respect to the rounding dimension makes spreading them unfeasible due to space lack. However, these configurations should be rare in real applications, if the degree of rounding is reasonable with respect to the topological structure and geometrical density of the dataset.

The SRR algorithm is therefore particularly suited for those applications where not only robustness is required, but also maintaining the original topological structure is important. We are convinced that the general idea, which consists in spreading two geometries which are too near to each other instead of snapping them, can be applied to produce robustness also in different contexts. For instance, considering the computation of topological relations in a tolerance model, where the test of equality between two points is based on a tolerance threshold, new algorithms for creating robust datasets might apply the "*spreading instead of snapping*" idea. Moreover, in more specific application contexts with focused requirements, new algorithms can be designed which are able to produce arrangements with specific properties, for example the preservation of topology admitting some kinds of geometric deformations. Further research work can be done in these directions.

REFERENCES

Alberto Belussi, Sara Migliorini, Mauro Negri, and Giuseppe Pelagatti. 2012. Robustness of Spatial Relation Evaluation in a Distributed Environment. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems (ACM SIGSPATIAL GIS 2012)*. ACM, 446–449.

Jean-Daniel Boissonnat and Franco P. Preparata. 2000. Robust Plane Sweep for Intersecting Segments. *SIAM Journal on Computing* 29, 5 (2000), 1401–1421.

Li Chen. 2001. *Exact Geometric Computation: Theory and Applications*. Ph.D. Dissertation. New York University, Department of Computer Science.

Mark de Berg, Dan Halperin, and Mark Overmars. 2007. An Intersection-Sensitive Algorithm for Snap Rounding. *Computational Geometry Theory and Applications* 36, 3 (2007), 159–165.

Arno Eigenwillig, Lutz Kettner, and Nicola Wolpert. 2007. Snap Rounding of Bézier Curves. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry (SCG'07)*. ACM, 158–167.

Steven Fortune. 1998. Vertex-Rounding a Three-Dimensional Polyhedral Subdivision. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry (SCG'98)*. ACM, 116–125.

Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. 1997. Snap Rounding Line Segments Efficiently in Two and Three Dimensions. In *Proceedings of the 13th Annual Symposium on Computational Geometry (SCG'97)*. ACM, 284–293.

Daniel H. Greene and F. Frances Yao. 1986. Finite-Resolution Computational Geometry. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS'86)*. IEEE Computer Society, 143–152.

Leonidas J. Guibas and David H. Marimont. 1995. Rounding Arrangements Dynamically. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry (SCG'95)*. ACM, 190–199.

Dan Halperin. 2010. Controlled Perturbation for Certified Geometric Computing with Fixed-Precision Arithmetic. In *Proceedings of the Third International Congress Conference on Mathematical Software (ICMS'10)*. Springer-Verlag, 92–95.

Dan Halperin and Eli Packer. 2002. Iterated Snap Rounding. *Computational Geometry Theory and Applications* 23, 2 (2002), 209–225.

Dan Halperin and Christian R. Shelton. 1998. A Perturbation Scheme for Spherical Arrangements with Application to Molecular Modeling. *Computational Geometry: Theory and Applications* 10, 4 (1998), 273–288.

John Hershberger. 2011. Stable Snap Rounding. In *Proceedings of the Twenty-seventh Annual Symposium on Computational Geometry (SoCG '11)*. ACM, New York, NY, USA, 197–206.

John D. Hobby. 1993. Practical Segment Intersection with Finite Precision Output. (1993). http://ect.bell-labs.com/who/hobby/93_2-27.pdf

John D. Hobby. 1999. Practical Segment Intersection with Finite Precision Output. *Computational Geometry Theory and Applications* 13, 4 (1999), 199–214.

OGC 2011. *OpenGIS Implementation Standard for Geographic Information – Simple Feature Access – Part 1: Common Architecture*. OGC. version 1.2.1.

Eli Packer. 2008. Iterated Snap Rounding with Bounded Drift. *Computational Geometry* 40, 3 (2008), 231 – 251. DOI:http://dx.doi.org/10.1016/j.comgeo.2007.09.002

Eli Packer. 2011. Controlled Perturbation of Sets of Line Segments in R2 with Smart Processing Order. *Comput. Geom. Theory Appl.* 44, 5 (2011), 265–285.

Rod Thompson and Peter van Oosterom. 2006. Interchange of Spatial Data-Inhibiting Factors. In *Proceedings of the 9th AGILE International Conference on Geographic Information Science*.

Received February 2007; revised March 2009; accepted June 2009