Iterated Snap Rounding*

Dan Halperin Eli Packer

School of Computer Science
Tel Aviv University
{danha,elip}@post.tau.ac.il

Abstract

Snap rounding is a well known method for converting arbitrary-precision arrangements of segments into a fixed-precision representation. We point out that in a snap-rounded arrangement, the distance between a vertex and a non-incident edge can be extremely small compared with the width of a pixel in the grid used for rounding. We propose and analyze an augmented procedure, *iterated snap rounding*, which rounds the arrangement such that each vertex is at least half-the-width-of-a-pixel away from any non-incident edge. Iterated snap rounding preserves the topology of the original arrangement in the same sense that the original scheme does. However, the guaranteed quality of the approximation degrades. Thus each scheme may be suitable in different situations. We describe an implementation of both schemes. In our implementation we substitute an intricate data structure for segment/pixel intersection that is used to obtain good worst-case resource bounds for iterated snap rounding by a simple and effective data structure which is a cluster of kd-trees. Finally, we present rounding examples obtained with the implementation.

1 Introduction

Geometric algorithms are typically described in the infinite-precision "real RAM" model of computation and under the assumption of general position, namely that the input is degeneracy-free. These assumptions raise great difficulties in implementing *robust* geometric algorithms. A variety of techniques have been proposed in recent years to overcome these difficulties [16],[17].

One approach to robust computing produces a finite-precision approximation of the geometric objects in question; for a survey of finite-precision approximation algorithms, see, e.g., [15]. Snap

*This work has been supported in part by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces), by The Israel Science Foundation founded by the Israel Academy of Sciences and Humanities (Center for Geometric Computing and its Applications), and by the Hermann Minkowski – Minerva Center for Geometry at Tel Aviv University.

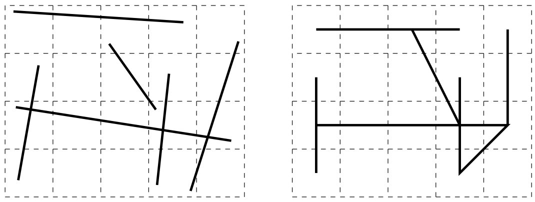


Figure 1: An arrangement of segments before (a) and after (b) snap rounding

rounding is a method of this type for converting an *arrangement* of segments into a low-precision representation.

Given a finite collection $\mathcal{S}$ of segments in the plane, the arrangement of $\mathcal{S}$, denoted $\mathcal{A}(\mathcal{S})$, is the subdivision of the plane into vertices, edges, and faces induced by $\mathcal{S}$. A *vertex* of the arrangement is either a segment endpoint or the intersection of two segments. Given an arrangement of segments whose vertices are represented with arbitrary-precision coordinates, snap rounding (SR, for short) proceeds as follows [9],[12]. We tile the plane with a grid of unit squares, *pixels*, each centered at a point with integer coordinates. A pixel is *hot* if it contains a vertex of the arrangement. Each vertex of the arrangement is replaced by the center of the hot pixel containing it and each edge e is replaced by the polygonal chain through the centers of the hot pixels met by e, in the same order as they are met by e. See Figure 1 for an illustration.

In the process, vertices and edges of the original arrangement may have collapsed. However, the rounded arrangement preserves certain topological properties of the original arrangement: The rounding can be viewed as a continuous process of deforming curves (the original segments into chains) such that no vertex of the arrangement ever crosses through a curve [11]. The rounded version s' of an original segment s approximates s such that s' lies within the Minkowski sum of s and a pixel centered at the origin.

SR makes the vertices of the arrangement well separated. We would expect that in the rounded arrangement a vertex v and an edge e not incident to v will also be well separated, namely, that the minimum separation between a vertex and a non-incident edge will be at roughly the same scale as the minimum separation between vertices. However, as we show in the next section, this is not the case and the distance between a vertex and a non-incident edge can be extremely small compared with the width of a pixel in the grid used for rounding.

We propose an augmented procedure, *iterated snap rounding*, ISR for short, which rounds the arrangement such that each vertex is at least half a unit away from any non-incident edge. ISR preserves the topology of the original arrangement in the same sense as the original scheme does. However, the guaranteed quality of the approximation degrades and the chain may be further away from the segment it approximates than the corresponding chain produced by SR. Thus each scheme may be suitable in different settings. We also show that the maximum combinatorial complexity, namely the maximum overall number of vertices in all the chains as well as the maximum complexity of the rounded arrangements, is the same for SR and ISR.

We present a conceptually simple algorithm for computing ISR (as well as SR), whose only non-trivial component is a data structure to answer segment intersection queries on a given collection of (hot) pixels. To provide asymptotically good worst-case resource bounds we use multi-level partition trees for this structure. In our implementation, however, we use a simple alternative which we call *c-oriented kd-trees*. We present below rounding results obtained with our implementation of SR and ISR.

Throughout the paper we use the following notation and terminology. The input $\mathcal{S}$ consists of n line segments $s_1, \ldots, s_n$. The rounding schemes transform each input segment into a polygonal chain. We call each straight line segment of an output chain between two hot pixels' centers a *link*. The output of SR for an input segment s is denoted by s' and the output of ISR for s is denoted by s^*.

Related Work. Greene and Yao [10] were the first to propose a rounding scheme for polygonal subdivisions. Hobby [12] and Greene [9] give snap rounding algorithms for arrangements of segments—theirs is the SR scheme that we discuss here. Guibas and Marimont [11] give a dynamic algorithm for snap rounding an arrangement of segments, as well as elementary proofs of the topological properties maintained by SR. Goodrich et al. [8] improve the SR algorithms when many segments intersect in a pixel. Milenkovic presents a rounding scheme using shortest paths [14]. Three-dimensional rounding algorithms of a similar nature have also been suggested and studied [7],[8],[13].

The rest of the paper is organized as follows. In the next section we show that in SR a vertex and a non-incident edge of the rounded arrangement can be very close to one another. In Section 3 we describe the augmented procedure ISR, prove its main properties and outline an algorithm for computing it. In Section 4 we fill in the algorithmic details of our algorithm and analyze its complexity. Section 5 is devoted to the implementation of the algorithm using c-oriented kd-trees. Rounding examples obtained with the implementation are given in Section 6. We conclude in Section 7 by pointing out possible directions for future work.

2 The Distance between a Vertex and a Non-Incident Edge

Consider the two segments s, t displayed in Figure 2 before and after SR. We denote the right endpoint of s' by s'_r. (Recall that u' is the rounded version of u.) After rounding, t' penetrates the hot pixel containing s'_r, but it does not pass through its center.

We can modify the input segment t so that t' becomes very close to s'_r: we move the left endpoint of t arbitrarily close to the top right corner of the pixel containing it. We vertically translate the right endpoint of t far downwards—the farther down we translate it, the closer t' will be to s'_r.

We cannot make t' *arbitrarily* close to s'_r. If they are not incident then there is a lower bound on the distance between them. This distance, however, can be rather small. Let b denote the number of bits in the representation of the vertex coordinates of the output chains of SR. We tile a bounding square of the arrangement with $2^b \times 2^b$ unit pixels. In this setting the distance between t' and s'_r can be made as small as $1/\sqrt{(2^b-1)^2+1} \approx 2^{-b}$.

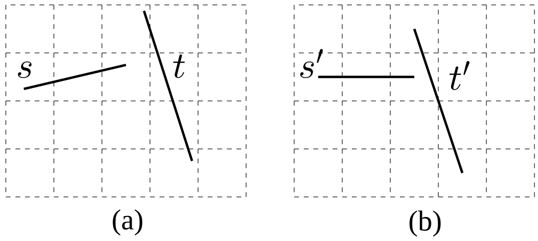


Figure 2: A vertex becomes very close to a non-incident edge after (b) snap rounding

One could argue that although SR produces near-degenerate output, it is still possible, during the rounding process, to determine the correct topology of the rounded arrangement in the hot pixel containing s'_r. However, this requires that the output of SR should include additional information beyond the simple listing of polygonal chains specified by their rounded vertices, making it more cumbersome to use and further manipulate.

3 Iterated Snap Rounding

We augment SR to eliminate the near-degeneracies mentioned above. Our procedure, which we call *iterated snap rounding* (ISR, for short), produces a rounded arrangement where an original segment is substituted by a polygonal chain each vertex of which is at least 1/2 a unit distant from any non-incident edge.

Let $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ be the collection of input segments whose arrangement we wish to round. Recall that a pixel is hot if and only if it contains a vertex of the input arrangement. Let H denote the set of hot pixels induced by $\mathcal{A}(S)$.

Our goal is to create chains out of the input segments such that a chain that passes through a hot pixel is re-routed to pass through the pixel's center. The difficulty is that once we reroute a chain it may have entered other hot pixels and we need to further reroute it, and so on.

Our rounding algorithm consists of two stages. In a preprocessing stage we compute the hot pixels (by finding all the vertices of the arrangement) and prepare a segment intersection search structure D on the hot pixels to answer queries of the following type: Given a segment s, report the hot pixels that s intersects. In the second stage we operate a recursive procedure, Reroute, on each input segment. We postpone the algorithmic details of the preprocessing stage to the next sections and concentrate here on the rerouting stage.

Reroute is a "depth-first" procedure. As we show below, the rerouting that we propose does not add more hot pixels, so whenever we refer to the set of hot pixels we mean H. The input to Reroute is a segment $s \in \mathcal{S}$. The output is a polygonal chain s^* which approximates s. Whenever s^* passes through a hot pixel, it passes through its center. See Figure 3 for an illustration.

We next describe the ISR algorithm. The routine Reroute will produce an output chain s_i^* in the global parameter output_chain as an ordered list of links. If a segment is contained

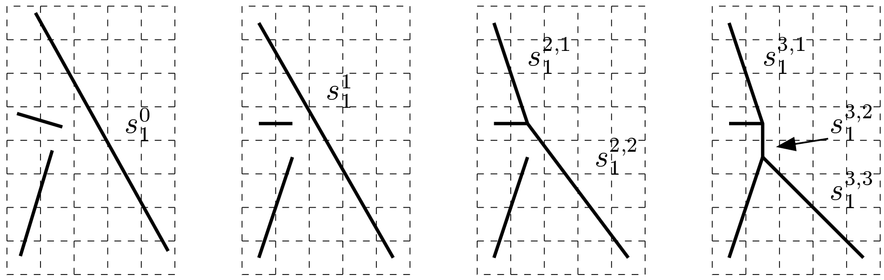


Figure 3: Iterated snap rounding for the input (a) results in (d)

inside a single pixel, the chain degenerates to a single point.

ISR
Input: a set $\mathcal{S}$ of n segments
Output: a set $\mathcal{S}^*$ of n polygonal chains; initially $\mathcal{S}^* = \emptyset$
/* stage 1: preprocessing */
compute the set H of hot pixels
construct a segment intersection search structure D on H
/* stage 2: rerouting */
for each input segment $s \in \mathcal{S}$
    initialize OUTPUT_CHAIN to be empty
    REROUTE(s)
    add OUTPUT_CHAIN to $\mathcal{S}^*$

REROUTE(s)
/* s is the input segment with endpoints p and q */
query D to find H_s, the set of hot pixels intersected by s
if H_s contains a single hot pixel /* s is entirely inside a pixel */
    then add the center of the hot pixel containing s to OUTPUT_CHAIN
    else
        let $m_1, m_2, \ldots, m_r$ be the centers of the r hot pixels in H_s in the order
        of the intersection along s
        if ($r = 2$ **and** p, q are centers of pixels)
            then add the link $\overline{m_1 m_2}$ to OUTPUT_CHAIN
            else
                for $i = 1$ to $r - 1$
                    REROUTE($\overline{m_i m_{i+1}}$)

We next discuss the properties of the procedure.

We fix an orientation for each input segment and its induced chains: it is oriented in lexicographically increasing order of its vertices. Thus, a non-vertical segment for example is oriented

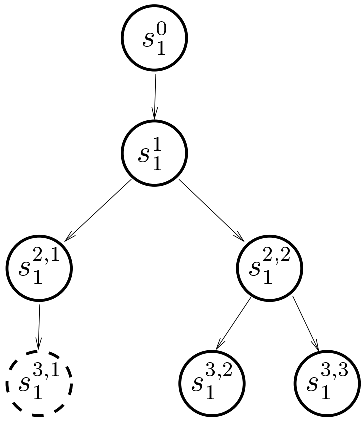


Figure 4: The tree T_1 corresponding to REROUTE(s_1) of Figure 3. Nodes denoted by full-line circles contain segments with which we query the structure D. The dashed-line circle denotes a node containing an exact copy of the segment of its parent.

from its left endpoint to its right endpoint. (The orientation of a chain is well defined since, as is easily verified, a chain is (weakly) x-monotone and (weakly) y-monotone.) We represent the operation of REROUTE on a segment s_i as a tree T_i. The root contains s_i. The leaves of the tree contain the output polygonal chain s_i^*, one link in each node, ordered from left to right where the first link is in the leftmost leaf. Each internal node ν together with its children represent one application of REROUTE (without recurrence): the segment s of ν, which passes through the hot pixels with centers $m_1, m_2, \ldots, m_r$, is transformed into the links $\overline{m_q m_{q+1}}, q = 1, \ldots, r-1$ which are placed in the children of ν ordered from left to right to preserve the orientation of the chain. We denote all the segments in the nodes at the jth level from left to right by $s_i^{j,1}, s_i^{j,2}, \ldots, s_i^{j,l_{i,j}}$, where $l_{i,j}$ denotes the number of nodes at this level. We denote the chain consisting of all the links at level j ordered from left to right by s_i^j. Thus $s_i^0 = s_i$. We denote by k_i the depth of the tree for s_i, and let $k := \max_{i=1}^n k_i$. For notational convenience, if a leaf λ is at level $k_\lambda < k$ then we add a linear path of $k_i - k$ artificial nodes descending from λ and all containing the same link that λ contains (we denote it differently at any level according to the level). See Figure 4 for an illustration of the tree T_1 corresponding to segment s_1 of Figure 3. We denote by $s(\nu)$ the segment (or link) that is contained in the node ν.

The next lemma gives an alternative view of ISR.

Lemma 3.1 *Given a set of segments $\mathcal{S}$, the output of ISR is equivalent to the final output of a finite series of applications of SR starting with $\mathcal{S}$, where the output of one SR is the input to the next SR.*

Proof: Once we determine the hot pixels H, snap rounding an input segment s (i.e., by the standard SR) can be done independently of the other segments. That is, the information necessary for rounding is in H. Notice that the chains $s_i^1, i = 1, \ldots, n$ are the result of applying SR to the original input segments $\mathcal{S}$.

The crucial observation is that SR does not create new hot pixels. It can break a segment into two segments that meet at the center of an existing hot pixel, but it cannot create a new endpoint nor a new intersection point (with another segment) which are not at the center of an existing hot pixel—this would violate the topology preservation properties of SR [11].

It follows that with the same set H of hot pixels, the chains $s_i^{j+1}, i = 1, \ldots, n$ are the result of applying SR to the links in the chains $s_i^j, i = 1, \ldots, n$, and so on.

The process terminates when the link in each leaf of the tree has its endpoints in the center of hot pixels and it does not cross any other hot pixel besides the hot pixels that contain its endpoints.

The tree continues to grow beyond level j only as long as for at least one node ν in level j when we query with $s(\nu)$ we discover a new hot pixel through which $s(\nu)$ passes. We claim that a hot pixel is not discovered more than once per tree. This is so since, as already mentioned, each chain s_i^j is (weakly) x-monotone and (weakly) y-monotone. Since there are at most $O(n^2)$ hot pixels, the process will stop after a finite number of steps. □

The lemma's algorithmic interpretation is inefficient, but it is useful for proving some of the following properties.

Corollary 3.2 *ISR preserves the topology of the arrangement of the input segments in the same sense that SR does.*

Proof: The topological properties that are preserved by SR can be summarized by viewing SR as a continuous process of deforming curves (the original segments into chains) such that no vertex of the arrangement ever crosses through a curve [11]. Since SR does not create new vertices, the assertion follows from Lemma 3.1. □

Lemma 3.3 *(i) If an output chain of ISR passes through a hot pixel then it passes through its center.*
(ii) In the output chains each vertex is at least 1/2 a unit away from any non-incident segment.

Proof: Claim (i) follows from the definition of the procedure Reroute. Since all the vertices of the rounded arrangement are centers of hot pixels, claim (ii) is an immediate consequence of (i). □

A drawback of ISR is that an output chain s_i^* can be farther away from the original segment s_i compared with the chain produced for the same input segment by SR. Recall that k_i denotes the depth of the recursion of Reroute(s_i).

Lemma 3.4 *A final chain s_i^* lies in the Minkowski sum of s_i and a square of side size k_i centered at the origin.*

Proof: In SR, a rounded segment s' lies inside the Minkowski sum of the input segment s and a unit square centered at the origin. Since the ISR is equivalent to k_i applications of SR, the claim follows. $\square$

This deviation may be acceptable in situations where the pixel size is sufficiently small or when $k := \max_{i=1}^{n} k_i$ is small.

4 Algorithmic Details and Complexity Analysis

Let I denote the number of intersection points of segments in the original arrangement $\mathcal{A}(\mathcal{S})$. We first compute the set H of hot pixels. For that we use an algorithm for segment intersection. This could be done with a plane sweep algorithm, or more efficiently in $O(I + n \log n)$ time by more involved algorithms [2],[4]. To compute the hot pixels, the algorithm should also be given a pixel's width w and a point p that will be assigned the coordinates $(0, 0)$. The plane will be tiled with pixels that we will consider to be of unit width, and their centers will have integer coordinates. We denote the number of hot pixels by N. Notice that N is at most $O(n + I)$.

Remark. One could alternatively detect the hot pixels by the SR algorithm of Goodrich et al. [8] and thus get rid of the dependence of the running time of the algorithm on the number of intersections I. Notice however that for this step alone (namely for detecting the hot pixels) and for certain inputs (e.g., the input depicted in Figure 5 and described below) this alternative is costly.

Next we prepare the data structure D on the hot pixels H to answer segment intersection queries. We construct a multi-level partition tree [1] on the vertical boundary segments of the hot pixels, and an analogous tree for the horizontal boundary segments. The partition trees report the segments intersected by a query segment s from which we deduce the hot pixels intersected by s. Each tree requires $O(M^{1+\varepsilon})$ preprocessing time when allowed M units of storage for $N \le M \le N^2$. A query takes $O(N^{1+\varepsilon}/\sqrt{M} + g)$ time, where g is the number of hot pixels found [1].

How many times do we query the structure D for segment intersection?

Lemma 4.1 *If an output chain s_i^* consists of l_i links then during* Reroute*(s_i) the structure D is queried at most $2l_i$ times.*

Proof: During Reroute(s_i) when we query with a link (line 1 of Reroute) either we do not find new hot pixels (new for the rounded version of s_i) in which case we charge the query to the link which is then a link of the final chain, or we charge it to the *first* new hot pixel (recall that we assigned an orientation to each segment and to each link). Each final link is charged exactly once and each vertex of the final chain is charged at most once, besides the last vertex which is never charged. The bound follows. $\square$

Let L denote the overall number of links in all the chains output by ISR. We summarize the performance bounds of ISR in the following theorem.

Theorem 4.2 *Given an arrangement of n segments with I intersection points, the iterated snap rounding algorithm requires $O(n \log n + I + L^{2/3}N^{2/3+\varepsilon} + L)$ time for any $\varepsilon > 0$ and $O(n + N + L^{2/3}N^{2/3+\varepsilon})$ working storage , where N is the number of hot pixels (which is at most $2n + I$) and L is the overall number of links in the chains produced by the algorithm.*

Proof: To find the intersections of the input segments we use Balaban's algorithm which requires $O(n \log n + I)$ time and $O(n)$ working storage. When an intersection is found we simply keep its corresponding hot pixel. For constructing and querying the multi-level partition trees (by Lemma 4.1 we perform at most $2L$ queries overall) we use a standard trick that balances between the preprocessing time and the overall query time, and does not require that we know the number of queries in advance. See, e.g., [5]. □

We conclude this section with combinatorial bounds on the maximum complexity of the rounded arrangements. Interestingly, as shown next, there is no difference between the maximum asymptotic complexity of the rounded arrangements between SR and ISR.

Theorem 4.3[1] *Given an arrangement of n segments in the plane, in its rounded version: (i) the maximum number of hot pixels through which a single output chain passes is $\Theta(n^2)$, and (ii) the maximum overall number of incidences between output chains and hot pixels is $\Theta(n^3)$. (iii) The number of segments in the rounded arrangement (namely without counting multiplicities) is $\Theta(n^2)$, and if the input segments induce N hot pixels then this number is $\Theta(N)$. All these bounds apply both to SR and to ISR.*

Proof: The upper bounds in claims (i) and (ii) are obvious. To see that these bounds are tight consider the following construction (see Figure 5). We take $n/2$ long horizontal segments spanning a row of $n^2/4$ pixels. Next we take $n/2$ short, slightly slanted segments, each spanning $n/2$ pixels such that overall each pixel in the row is intersected by exactly one short segment. The short segments are slanted such that in each pixel that they cross they intersect exactly one of the long segments. Each pixel in the row is now a hot pixel, and each of the long segments crosses all the hot pixels. The rounding obtained with both SR and ISR is the same.

The construction yields a degenerate rounded arrangement. Each of the output chains is in fact a horizontal line segment. This construction can be slanted so that each rounded version of a long segment is a chain with "true" $\Omega(n^2)$ links. In the slanted version we use $n^2/2$ pixels arranged in $n^2/4$ rows. In each row at least one pixel is hot. See Figure 6 for an illustration.

Finally, we ignore the chains, and we ask how complex can the rounded arrangement be, that is, we ignore multiplicities (overlap) of chains. Obviously, the rounded arrangement can have $\Omega(n^2)$ complexity. But this is also an upper bound since the (rounded) arrangement has

[1]The slanted version of our horizontal construction was suggested to us by Olivier Devillers. Claim (iii) is due to Mark de Berg.

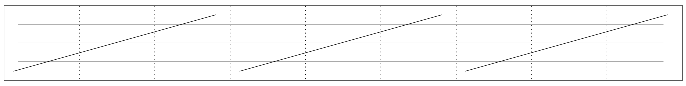


Figure 5: $\Theta(n)$ chains in the rounded arrangement are each incident to $\Theta(n^2)$ hot pixels

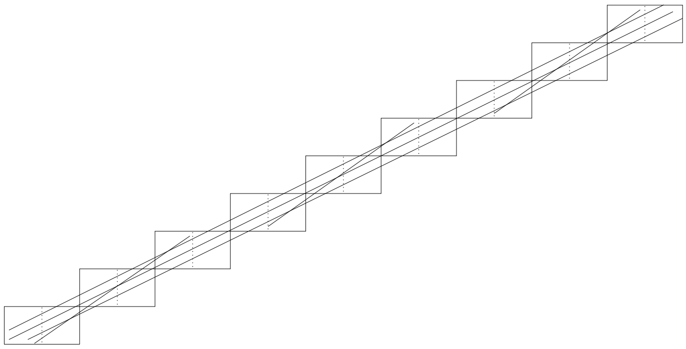


Figure 6: The slanted version yields $\Theta(n)$ rounded segments with $\theta(n^2)$ links each

N vertices and it is a planar graph. Therefore the number of edges can be at most $O(N)$. N can be at most $O(n^2)$. Again, our arguments do not depend on how the rounding was done (by SR or ISR). $\square$

5 c-Oriented kd-Trees

In our implementation we use a plane sweep algorithm to find the intersections between segments in $\mathcal{S}$ and thus we identify the hot pixels. The non-trivial part to implement is the search structure D with which we answer segment/pixel intersection queries. In the theoretical analysis we use partition trees for D, as these lead to asymptotically good worst-case complexity. In practice, (multi-level) partition trees are difficult to implement. Instead, we implemented a data structure consisting of several kd-trees. Next we explain the details.

Observation 5.1 *A segment s intersects a pixel p of width w, if and only if the Minkowski sum of s with a pixel of width w centered at the origin contains the center of p.*

We could use Observation 5.1 in order to answer segment intersection queries in the following way: build a range search structure on the centers of the hot pixels. Let s be the query segment and $M(s)$ be its Minkowski sum with a pixel centered at the origin. Then query the structure with the range $M(s)$. Unfortunately, the known data structures for this type of queries are similar to the multi-level data structures that we have used in the previous section.

Instead we use kd-trees as an approximation of this scheme. A kd-tree answers range queries for axis-parallel rectangles [6]. Its guaranteed worst-case query time is far from optimal but it is practically efficient. A trivial solution would be to query with the axis-parallel bounding box

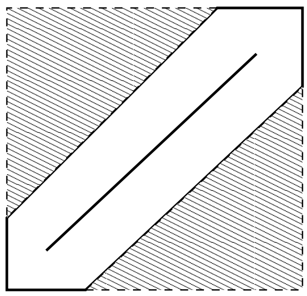


Figure 7: The bounding box of the Minkowski sum of a segment with a pixel centered at the origin. The shaded area is the redundant range

of $M(s)$, which we denote by $B(s)$; see Figure 7. This may not be sufficiently satisfactory since the area of $B(s)$, which we denote by $|B(s)|$, may be much larger than the area of $M(s)$.

If we rotate the plane together with $M(s)$ the (area of the) axis-parallel bounding box changes whereas $M(s)$ remains fixed. The difference between the bounding boxes for two different rotations can be huge. Our goal is to produce a number of rotated copies of the set of centers of hot pixels so that for each query segment s there will be one rotation for which the area of the bounding box is not too much different from the area of $M(s)$. Notice that if a segment s is rotated by $\pi/2$ radians, the size of the relevant bounding box remains the same. Since the determination of which rotation to choose is dependent only on the size of the respective bounding box, the range of rotations should be the half-open interval $[0 : \pi/2)$.

We construct a collection of kd-trees each serving as a range search structure for a rotated copy of the centers of hot pixels. We call this cluster *c-oriented kd-trees.* Let c be a positive integer and let $\alpha_i := (i-1)\frac{\pi}{2c}$ for $1 \le i \le c$. The structure consists of c kd-trees such that the i-th kd-tree, denoted by kd_i, has the input points rotated by α_i. Let $R_i(s)$ be the segment s rotated by α_i. For each query with segment s we do the following: for each $kd_i, 1 \le i \le c$, we compute $|B(R_i(s))|$. Let $1 \le h \le c$ be the serial number of the kd-tree for which $|B(R_h(s))| = \min_{i=1}^{c} |B(R_i(s))|$. Then we use the h-th kd-tree to answer the query with the segment s rotated by α_h. Finally, we discard all the points for which the segment does not intersect the respective hot pixels.

We next discuss a few important issues regarding the implementation and usage of this structure.

Exact rotations. We used exact arithmetic to implement ISR. Unfortunately, the available exact arithmetic data-types do not support the calculations of *sines* and *cosines* which are necessary for calculating rotations. Instead we use only angles for which the sines and cosines can be expressed as rational numbers with small enumerator and denominator [3]. We keep an array $\mathcal{Z}$ of approximations to the *sines* of integer *degree* angles between $0-89$. We emphasize that once we fix an angle β we have the exact *sine* and *cosine* of β. What we cannot do is obtain the exact values of the trigonometric functions of a prescribed arbitrary angle. Since our choice of rotation angles is heuristic to begin with, the precise angle is immaterial, and the angle we use is never more than one degree off the prescribed angle. Moreover, there are techniques to achieve better approximations [3], but we prefer not to use them because of performance reasons.

How big should c be? There are advantages and drawbacks in using few kd-tress, say even one kd-tree compared to using many. When using one kd-tree, we are prone to get many false points in the range queries, resulting in more time to filter out the results. When using many kd-trees, we need to invest time in their construction and a little more time per query to find the best rotation. Our experiments show that in many cases a small number of trees suffices. Consider for example the numerical table "different number of kd-trees" in Figure 8. (The rounding example in this figure as well as the other examples are explained in detail in the next section; here we only refer to the number of kd-trees used in their computation.) The first column shows how many kd-trees were used and the last column shows how much time the overall rerouting stage took compared with the time when using only one kd-tree (the full legend is given in Table 1). The best performance is obtained when we use 7 kd-trees. The time savings in this case is 17% over using a single kd-tree. The analogous table in the next example (Figure 9) shows that in that example there is no benefit in using more than one kd-tree.[2] In the next paragraph we present a heuristic improvement of the number of kd-trees. However, we leave the computation of the best number of kd-trees together with the best rotation angles of each one for further research.

Skipping $\mathbf{kd_i}$'s. Since c should be small, we expect most of the links of a certain input segment to have the same rotation as the input segment, since they should all have nearby slopes. Let J_i be the number of input segments that are rotated by α_i. If J_i is very small, it is not effective to create the respective kd-tree. Thus we fix a lower limit τ, and construct a kd-tree kd_i only if $J_i \geq \tau$. Obviously τ should be a function of c, and be sufficiently small to ensure that at least one kd-tree will be constructed. We chose to use $\tau = \frac{n}{2c}$. In the examples of Figures 8 and 9 τ is always greater than $\frac{n}{2c}$. In other examples, such as geographic data, not all c trees are always constructed—in Figure 10, when the algorithm is given $c > 7$ it chooses to skip some of the kd_i's. In this example, using more than one kd-tree is wasteful since the map is relatively sparse, most of the segments are relatively small compared to the whole map and the bounding box of their Minkowski sum with a unit pixel does not intersect many hot pixels centers.

6 Rounding Examples: SR vs. ISR

To give the flavor of how the output of ISR differs from that of SR we present the rounding results for three input examples; see Figures 8, 9, and 10. For each example we display the input, the SR result and the ISR result. Then we zoom in on a specific area of interest in these three drawings—an area where the rounding schemes differ noticeably. A square near a drawing represents the actual pixel size used for rounding. Then we provide two tables of statistics. The first one refers to the best number of kd-trees as related to the discussion in the previous

[2] The running time indicated in the tables is in seconds while using arbitrary precision rational arithmetic. The pixel size in the first example is 1 and in the second example is 15.

Abbreviation	Explanation
inkd	input number of kd-trees
nkd	actual number of kd-trees created
nfhp	overall number of false hot pixels in all the queries
tt	total time relative to using one tree
md	maximum deviation over all chains
ad	average deviation
mnv	maximum number of vertices in an output chain
anv	average number of vertices in an output chain
mdvs	minimum distance between a vertex and a non-incident edge
ncvs	number of pairs of a vertex and a non-incident edge that are less than half the width of a pixel apart
ps	pixel size
nhp	number of hot pixels

Table 1: Abbreviations

section. The second table summarizes the differences in the rounding for different pixel sizes. The abbreviations we use in these two tables are explained in Table 1. The *deviation* of a chain from its inducing segment s is the maximal distance of a point on the chain from s.

6.1 Congestion Data

The data contains 200 segments with $18,674$ intersections. (For clarity, the pictures in Figure 8 depict a similar example with only 100 segments.) The bottom left part of the arrangement is zoomed in.

Both rounding schemes will collapse thin triangles that have two corners close by. However, not allowing proximity between vertices and non-incident edges, ISR collapses 'skinny' faces of the arrangement that SR does not (see the bottom of the zoomed-in area), for example triangles that have one corner close to the middle of the opposite edge.

For pixel size 1, SR and ISR are very different and the number of vertices that are less than half a unit away from a non-incident edge in the SR output is in the hundreds. The average deviation in ISR in this example is never more than 2.5 times that of the corresponding SR output. For pixel size greater than 1 the average deviation of a chain in ISR is almost the same as in SR. However, for pixel size smaller than 1, the average deviation is larger in the ISR output than in the SR output.

In terms of combinatorial complexity the results are similar and the average number of vertices per chain is roughly the same in both outputs. This is a phenomenon we have observed in all our experiments.

6.2 Triangulation Data

Figure 9 shows a set of input points (courtesy of Jack Snoeyink) and a triangulation of this set. The triangulation consists of 906 segments. The zoomed in pictures show a part of the triangulation for which there is considerable difference between SR and ISR.

Again ISR collapses thin polygons that SR does not collapse. The second table in Figure 9 shows that in this case the average deviation of a chain in both schemes does not differ by much. The maximum deviation in ISR is always less than twice the pixel width. Here also the average number of links per chain is almost the same for the output of SR and ISR.

6.3 Geographic Data

We ran both schemes on several geographic maps of countries and cities which are less cluttered than the examples above. The experiments for this type of data typically show little difference between the SR and ISR results. Figure 10 depicts the result for a map of the USA. The data contains 486 segments intersecting only at endpoints.

The second table in Figure 10 shows the difference of using SR and ISR. In most of the tests, there are occasional cases in which the distance between a vertex and a non-incident segment is shorter than half the size of a pixel. Thus there are differences between the SR and the ISR output. These differences are however minor. In the ISR output the maximum deviation is no more than twice that of the SR output. The average deviation in both the SR and ISR output is similar.

7 Conclusions

We presented an augmented snap rounding procedure which rounds an arbitrary precision arrangement of segments in $\mathbb{R}^2$ with the advantage that each vertex in the rounded arrangement is at least half a unit away from any non-incident edge. The new scheme makes the rounded arrangement more robust for further manipulation with limited precision arithmetic than the output that the standard snap rounding algorithm produces. We implemented ISR using exact arithmetic.

We propose several directions for further research: (1) Can detecting all the hot pixels through which an output chain passes be done more efficiently? (2) Extend the scheme to non-linear curves. (3) The rounded arrangement can have at most $O(n^2)$ segments, whereas our algorithm (as well as the known algorithms for SR) may produce $\Omega(n^3)$ output links. The task here is to devise an output sensitive algorithm where the output size is the size of the rounded arrangement and not the overall complexity of the chains. (4) Improve the heuristics for choosing the directions of the kd-trees.

Acknowledgement

The authors thank Mark de Berg and Olivier Deviller for their observations regarding the complexity of rounded arrangements.

References

[1] P. K. Agarwal and M. Sharir. Applications of a new space-partitioning technique. *Discrete Comput. Geom.*, 9:11–38, 1993.

[2] I. J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 211–219, 1995.

[3] J. Canny, B. R. Donald, and E. K. Ressler. A rational rotation method for robust geometric algorithms. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 251–260, 1992.

[4] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[5] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray shooting and hidden surface removal. *Algortihmica*, 12:30–53, 1994.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Heidelberg, Germany, 1997.

[7] S. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. *Discrete Comput. Geom.*, 22(4):593–618, 1999.

[8] M. Goodrich, L. J. Guibas, J. Hershberger, and P. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 284–293, 1997.

[9] D. H. Greene. Integer line segment intersection. Unpublished Manuscript.

[10] D. H. Greene and F. F. Yao. Finite-resolution computational geometry. In *Proc. 27th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 143–152, 1986.

[11] L. Guibas and D. Marimont. Rounding arrangements dynamically. *Internat. J. Comput. Geom. Appl.*, 8:157–176, 1998.

[12] J. Hobby. Practical segment intersection with finite precision output. *Comput. Geom. Theory Appl.*, 13:199–214, 1999.

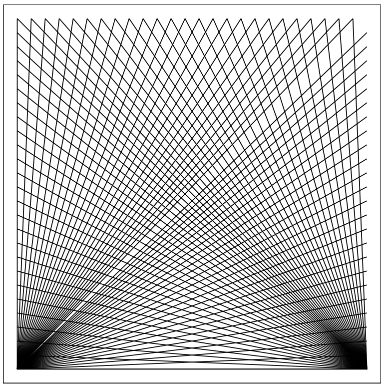
[13] V. J. Milenkovic. Rounding face lattices in d dimensions. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 40–45, 1990.

[14] V. J. Milenkovic. Shortest path geometric rounding. *Algorithmica*, 27(1):57–86, 2000.

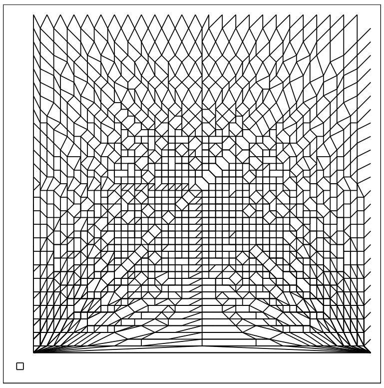
[15] S. Raab. Controlled perturbation for arrangements of polyhedral surfaces with application to swept volumes. M.Sc. thesis, Dept. Comput. Sci., Bar Ilan University, Ramat Gan, Israel, 1999. http://www.math.tau.ac.il/~raab/master_thesis.ps.

[16] S. Schirra. Robustness and precision issues in geometric computation. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 597–632. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 1999.

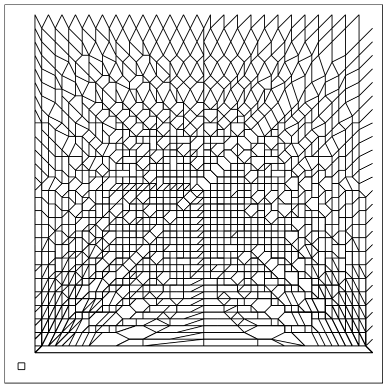
[17] C. K. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 35, pages 653–668. CRC Press LLC, Boca Raton, FL, 1997.



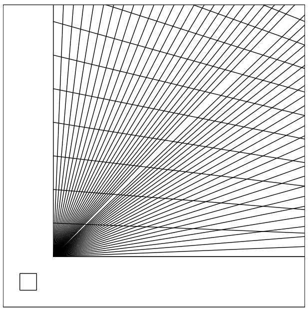
Input



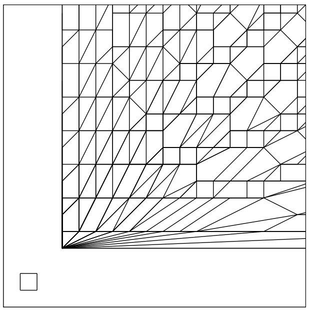
SR output



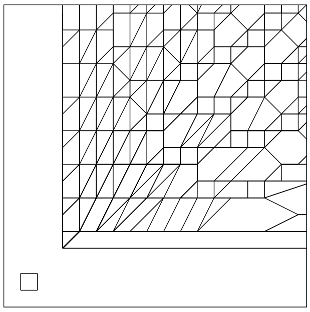
ISR output



Input zoom in



SR output zoom in



ISR output zoom in

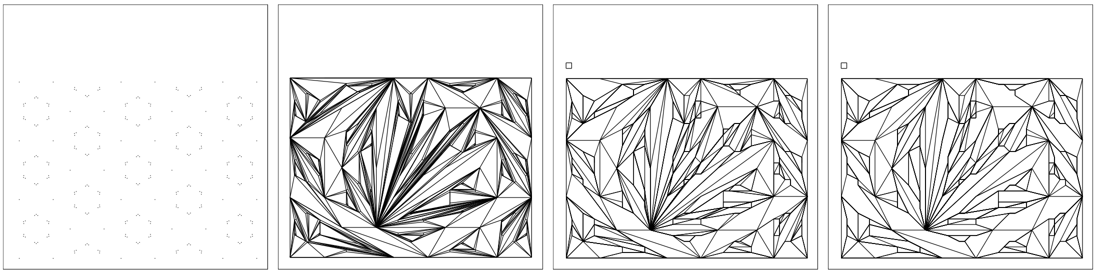
nkd	nfhp	tt
1	613477	100% = 213.2 s
2	513551	87.2%
3	474997	83.6%
4	478749	84%
5	479507	84.3%
6	463025	83.4%
7	456882	83%
8	456269	84%
9	455334	84.8%
10	456196	86.3%

Different number of kd-trees

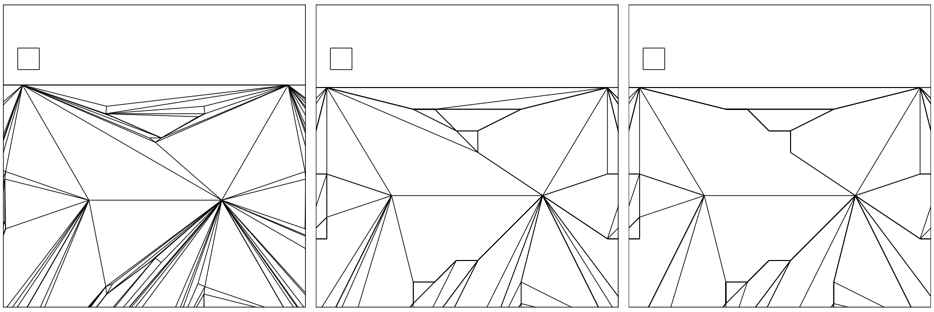
		isr						sr					
ps	nhp	md	ad	mnv	anv	mdvs	ncvs	md	ad	mnv	anv	mdvs	ncvs
0.125	8488	1.01	0.19	120	90.96	0.08	0	0.09	0.09	106	87.95	0.04	17
0.25	8261	1.5	0.41	124	94.15	0.15	0	0.17	0.17	112	89.16	0.06	58
0.5	7711	1.68	0.67	135	97.9	0.28	0	0.35	0.35	126	91.66	0.08	135
1	6003	1.58	0.99	154	101.85	0.55	0	0.71	0.71	153	95.99	0.07	328
2	2538	1.51	1.41	101	72.9	1.26	0	1.41	1.41	101	72.87	0.88	3
3	1143	2.12	1.84	67	49.1	2.12	0	2.12	1.84	67	49.09	1.34	1
4	673	2.82	2.7	51	37.56	2.82	0	2.82	2.7	51	37.56	2.82	0
5	439	3.53	3.32	41	30.31	3.53	0	3.53	3.32	41	30.3	2.23	1
10	120	7.07	6.6	21	15.58	7.07	0	7.07	6.6	21	15.58	7.07	0

ISR and SR comparison

Figure 8: Congestion data



Input points　　Input triangulation　　SR output　　ISR output



Input zoom in　　SR output zoom in　　ISR output zoom in

nkd	nfhp	tt
1	4872	100% = 15.4 s
2	4789	103.2%
3	4852	102.6%
4	4597	101.9%
5	4487	102.6%
6	4349	103.2%
7	4349	102.6%
8	4399	102.6%
9	4419	103.2%
10	4358	102.6%

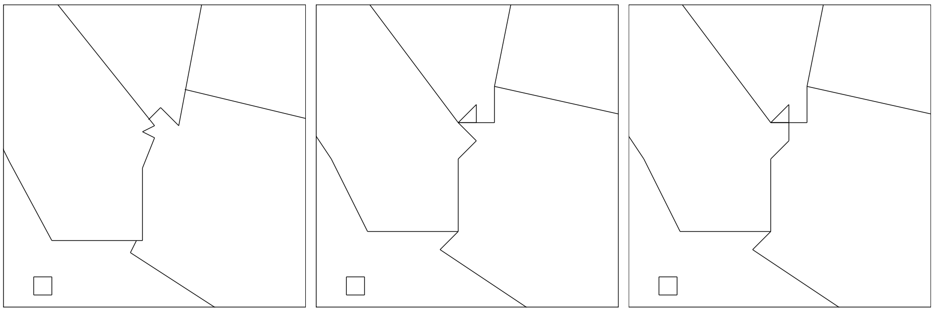
Different number of kd-trees

		isr						sr					
ps	nhp	md	ad	mnv	anv	mdvs	ncvs	md	ad	mnv	anv	mdvs	ncvs
2	306	2.231	0.825	6	2.219	1.223	0	1.341	0.812	6	2.198	0.318	9
5	300	9.804	2.691	7	2.625	3.14	0	3.494	2.442	6	2.48	0.741	50
10	249	17.194	5.18	9	2.761	5.368	0	7.028	4.847	7	2.637	1.414	45
15	195	22.088	6.985	10	2.75	9.486	0	10.559	6.512	10	2.622	2.631	67
20	162	32.207	7.614	9	2.621	11.767	0	13.914	7.19	8	2.532	4.85	45

ISR and SR comparison

Figure 9: Triangulation data

Input SR output ISR output



Input zoom in SR output zoom in ISR output zoom in

inkd	nkd	nfhp	tt
1	1	293	100% = 9.11 s
2	2	306	102%
3	3	302	103.1%
4	4	284	103.8%
5	5	293	105%
6	6	275	106%
7	7	260	106.8%
8	6	269	106.1%
9	8	272	107.9%
10	8	253	107.9%

Different number of kd-trees

		isr						sr					
ps	nhp	md	ad	mnv	anv	mdvs	ncvs	md	ad	mnv	anv	mdvs	ncvs
0.125	486	0.097	0.088	4	2.098	0.111	0	0.088	0.088	4	2.096	0.045	1
0.25	485	0.353	0.177	5	2.113	0.196	0	0.176	0.176	5	2.107	0.039	2
0.5	480	0.392	0.353	4	2.104	0.377	0	0.353	0.353	4	2.100	0.039	2
1	475	1.414	0.715	5	2.137	0.569	0	0.707	0.707	5	2.115	0.196	3
2	432	2.236	1.063	5	2.137	1.264	0	1.414	1.043	5	2.102	0.392	9
3	379	3.807	1.353	5	2.037	2.121	0	2.121	1.336	5	2.020	1.341	2
4	338	3.333	1.764	6	1.991	2.828	0	2.828	1.758	5	1.983	1.264	2
5	299	4.735	2.124	5	1.897	3.535	0	3.535	2.110	4	1.884	1.581	3
10	177	10.606	3.732	5	1.615	7.071	0	7.071	3.696	5	1.602	7.071	0

ISR and SR comparison

Figure 10: Geographic data