APPLIED aspects of COMPUTATIONAL GEOMETRY

Geometric Rounding: Snap Rounding
Arrangements of Segments

Dan Halperin
School of Computer Science
Tel Aviv University

Overview

- why round?
- snap rounding basics: definitions, properties, first algorithm
- better performance algorithms
- the combinatorics of snap-rounded arrgs
- better rounding-quality variants

the presentation of the last three topics interspersed

d1

Overview, the progression of things

- why round?
- snap rounding basics: def's, properties, first algorithm
- output sensitive algo I: the erasure algorithm
- the combinatorics of snap-rounded arrgs
- iterated snap roudning
- output sensitive algo II: the bundles algorithm
- the best of both worlds: the erasure-bundles algorithm

Slide 3

d1 danha, 22/05/2009

d2

Geometric rounding

- transforming an arbitrary precision object into a fixed precision representation
- Why round?

  precision too high to be useful; no finite exact numerical representation (irrational numbers)
- is it OK to round?

  depends on the application; most often consistent rounding is acceptable and desirable
- naive rounding has problems: new intersections, largely displaced intersections, topological inversion, and more
- consistent rounding is hard

Slide 4

d2 danha, 22/05/2009

d3

Example: vertical decomposition of arrgs of triangles

- The coordinates (x,y,z) of every triangle corner are each represented with a 16-bit over 16-bit rational

Slide 5

d3 danha, 22/05/2009

Complexity of numbers, input coordinates

```
Triangle 1:
     (-9661 / 499, 898 / 2689, -92949 / 3802),
     (-15034 / 1583, -8174 / 1759, -57116 / 3851),
     (13605 / 1261, -90590 / 3669, -11791 / 518)

Triangle 2:
     (-77665 / 4036, -130679 / 3347, -31167 / 1630),
     (-5851 / 297, 36471 / 893, -53137 / 2704),
     (132613 / 3310, 3 / 8, -21926 / 1111)

Triangle 3:
     (-37497 / 1939, -131078 / 3301, 591 / 3680),
     (-74461 / 3822, -28120 / 3397, 7607 / 346),
     (21622 / 1037, -12461 / 1441, 17957 / 827)

Triangle 4:
     (-10760 / 521, -58546 / 3057, 27619 / 1322),
     (-65262 / 3181, 74693 / 3622, 17898 / 863),
     (48898 / 2419, 1602 / 1627, 26390 / 1273)

Triangle 5:
     (-73482 / 3845, 88794 / 2203, 2720 / 3661),
     (-20591 / 1049, 9257 / 983, 57830 / 2693),
     (28590 / 1363, 38699 / 3957, 62390 / 2957)
```

Complexity of numbers, computed coordinates

A normalized coordinate of the worst feature of the partial decomposition — 237 digits long:

```
PD feature = 4979983882610488719277551621904699 4702
46182802505912364621748587334692109923893960959 0257
26989674024022169299702332971 / 5027790709859107937
56310374453264400561991943404298432389624397772 4409
28440717068821348688514967315807043013459806716
```

A normalized coordinate of the worst feature of the full decomposition — 559 digits long:

```
FD feature = 232793152439246761557989586883829045 85
98820358559036174083951968125496814516274709807 2652
14185860750272304623936720977656925977667887164 0355
47670312162391255854958478912398297412995827870 4985
39074448357766210408523170834023252512236899001 3542
79996132937206816849552931288112929 81 / 22458231406
21609487820297612679005432469881643247844751180 2089
66536364125006650143376953847480774294727058110 9819
67467591634125473414866344409019925427614200985 0182
41944472606066134207792617904534411070470548862 3957
68080930621026919963783708875743035453027734313 5738
809521441456
```

Rounding vs. simplification

- the two problems are closely related but different
- sample simplification problem: given a simple polygonal chain P and a positive real parameter ε find the fewest-links polygonal chain “equivalent” to P with distance at most ε from P
- practical and efficient solution: the Douglas-Peuker algorithm
- we will focus on rounding rather than simplification; we are concerned with convenient (fixed size) number representation and otherwise wish to preserve as many properties of the original object as possible
- often a byproduct of rounding is simplification

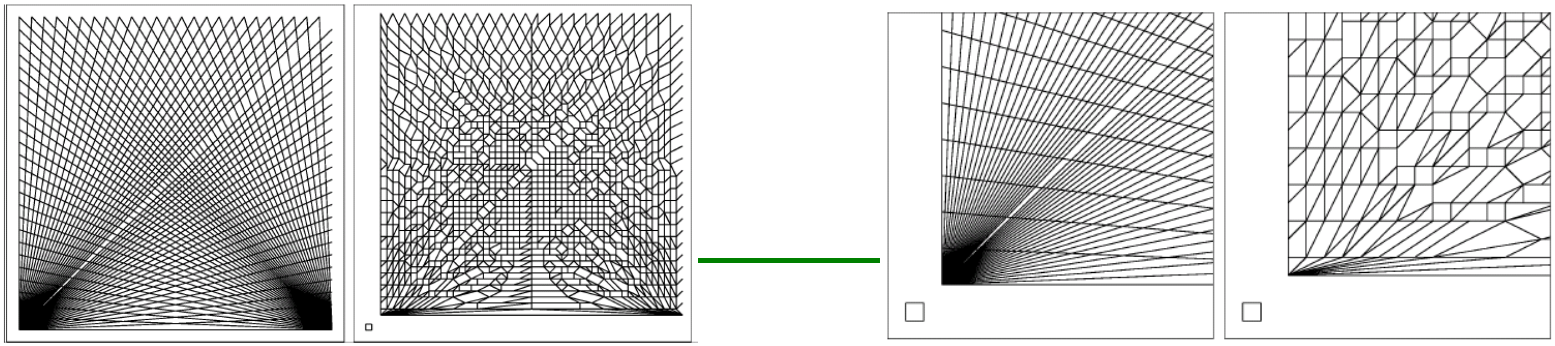
The role of exact computing in rounding

- we will use exact geometric computing in all the algorithms that we present below
- this is a disadvantage; is it necessary?
- Kurt Mehlhorn [*Mini-course on Geometric Rounding*, 2002]: “I doubt however, that there is a general strategy for geometric rounding which avoids the construction of the `exact object' as a first step”

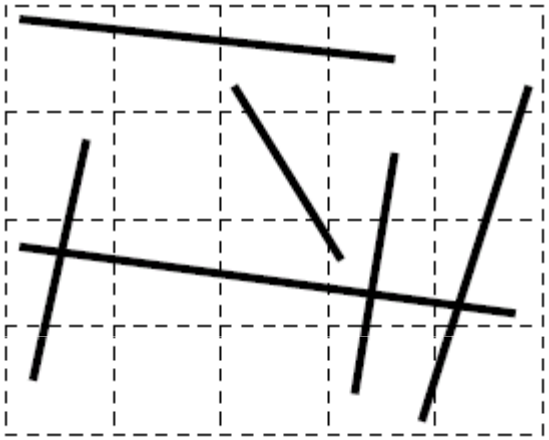
if the above conjecture is true then “what rounding is for constructions” is inferior to “what filtering is for predicates” (but rounding has other benefits of course)

Snap rounding (SR) arrgs of segments, I/O

- input: n segments and a grid of pixels such that the center points of the pixels have integer coordinates (the centers of pixels form the integer grid)
- reminder, the vertices of the arrangement: either segment endpoints (2n) or intersection of segments (I, at most $\theta(n^2)$); in this context a.k.a.: critical points
- we call the input segments ursegments
- output: an arrangement of segments where all the vertices are centers of grid pixels; details follow

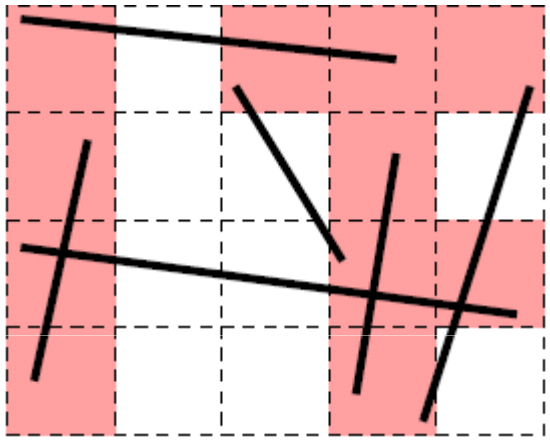


Snap rounding arrgs of segments, definition



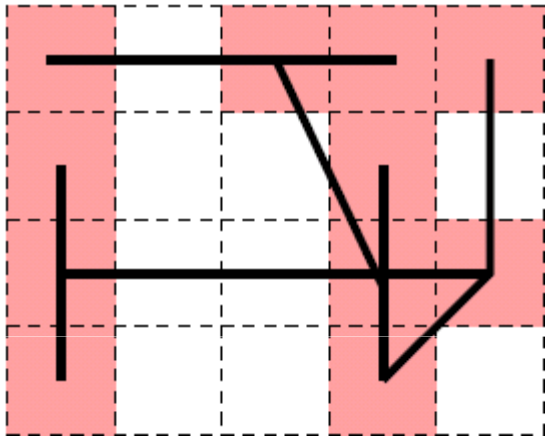
- a pixel containing an arrg vertex is a hot pixel

Snap rounding arrgs of segments, definition



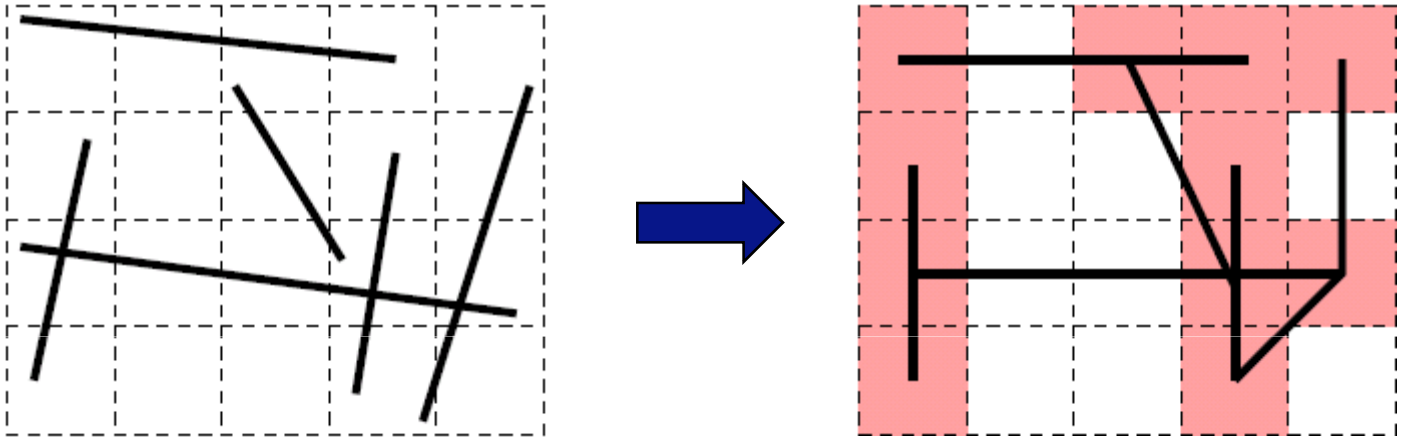
- a pixel containing an arrg vertex is a hot pixel
- for each ursegment s construct its approximating polygonal chain s* by connecting the centers of the hot pixels that s crosses in the order of crossing

Snap rounding arrgs of segments, definition



- a pixel containing an arrg vertex is a hot pixel
- for each ursegment s construct its approximating polygonal chain s* by connecting the centers of the hot pixels that s crosses in the order of crossing

Snap rounding arrgs of segments, definition



- a pixel containing an arrg vertex is a hot pixel
- for each ursegment s construct its approximating polygonal chain s* by connecting the centers of the hot pixels that s crosses in the order of crossing
- the chain is made of links or subsegments

Remark: what's in a hot pixel

- a hot pixel contains:
  - the interior of the pixel
  - the interior of the pixel's bottom edge
  - the interior of the pixel's left edge
  - the bottom-left corner of the pixel
- thus every point inside the grid limits belongs to exactly one hot pixel
- real coordinate r is rounded to $\lfloor r+1/2 \rfloor$, namely if u=round(r), $r \in [u-1/2,u+1/2)$
- the grid may be translated and scaled; the rounding computation is similar

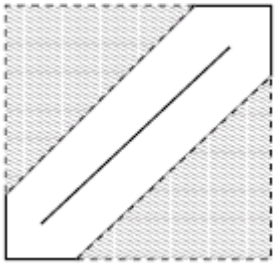
Properties of snap rounding

- fixed precision representation

need to show: no new vertices are created (below, together with topological similarity)

- geometric proximity
- topological similarity

Properties, geometric proximity



claim: the rounding chain s* is in the Minkowski sum of s with a pixel centered at the origin p(0)

proof:

- s and the pixel are convex and so is their Minkowski sum
- sufficient to show that each vertex of s* (which is the center of a hot pixel) is in this Minkowski sum; recall: s intersects p(c) iff c is inside s $\oplus$ -p(0) (= p(0))

Properties, topological similarity

- transforming s into s* viewed as a continuous deformation process; features may collapse but a curve does not cross over a vertex
- two-stage deformation process: each segment is broken by the hot pixels into external fragments and internal fragments
- the endpoints of external fragments lie on hot pixel boundaries at nodes and throughout the process a chain is defined as the polysegment through these nodes
- stage 1: collapsing the hot pixels horizontally onto their medial vertical axis
- stage 2: collapsing the vertical axis onto its middle point
- corollary, fixed precision representation: no new vertices are created

A simple algorithm for snap rounding

(1) Bentley-Ottman sweep to identify hot pixels

(2) reconstructing the chain s* by the hot pixels through which s passes

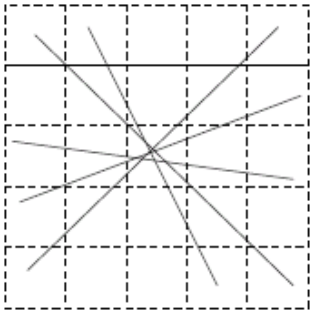
- intersection detection by sweepline
- Y structure = the dynamic sweepline
- X structure = the dynamic event queue
- events: left endpoint, right endpoint, intersection point
- output sensitivity

overall running time $O((n+I)\log n + L)$

L – overall complexity of the rounding chain

recall: n - # of input segs, I - # of intersections in the original arrg

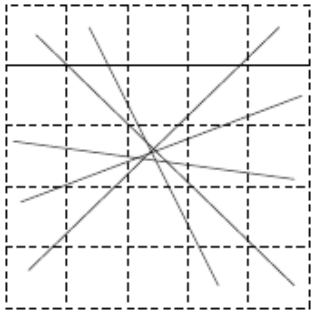
Output sensitive algorithm, Take I: The hot-pixel erasure algorithm



- avoid computing all intersections when unnecessary
- running time $O(n \log n + \sum_{h \in H} |S_h| \log n)$, where H is the set of hot pixels, and S_h is the set of segments crossing the pixel h

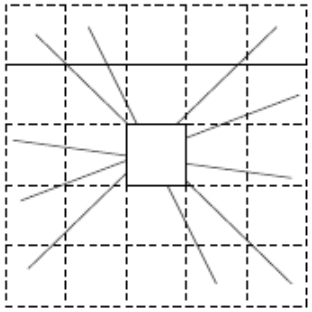
- worst-case running time: will be discussed later, after understanding the combinatorial complexity of snap rounding

The erasure algorithm, bird's eye view



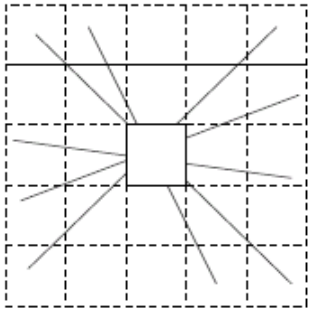
- erase the arrangement inside a hot pixel, the moment the sweepline reaches the leftmost critical point inside that pixel

The erasure algorithm, bird's eye view



- erase the arrangement inside a hot pixel, the moment the sweepline reaches the leftmost critical point inside that pixel

The erasure algorithm, bird's eye view



1) erase the arrangement inside a hot pixel, the moment the sweepline reaches the leftmost critical point inside that pixel
2) stretch the nodes (endpoints of external fragments) to the corresponding hot pixels centers - trivial

The erasure algorithm, details

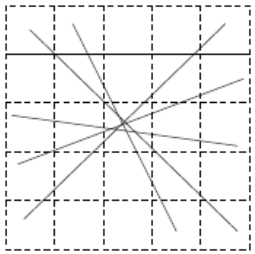
- sweepline events:
  - original segment endpoints
  - segment intersection
  - segment/pixel-boundary intersection
  - original segment re-insertion
  - right end of hot pixel boundary

The erasure algorithm, details, cont'd

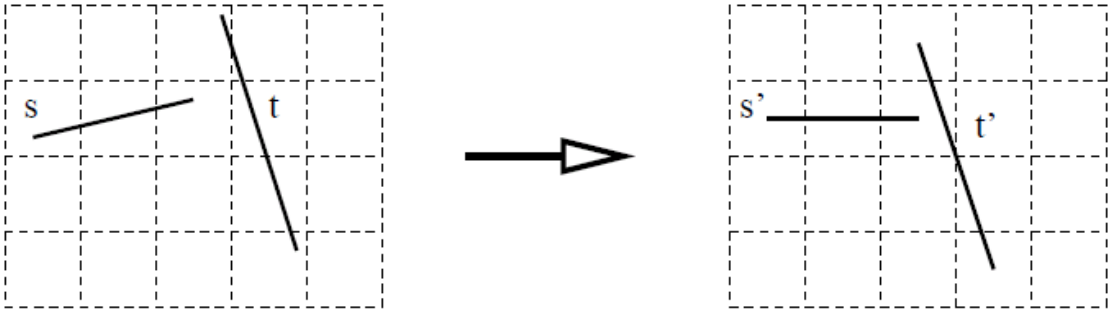
- pixel detected as hot at its leftmost critical point (discovered at x=xpos during the sweep)
- the top and bottom boundaries of the hot pixel to the right of the detection point are added to the sweep line
- the portions of ursegments to the right of xpos and within the hot pixel are erased; if prevail outside the pixel, reinserted at the pixel boundary
- for each hot pixel we keep four lists of segments abutting on the boundaries of the rectangle contained in the hot pixel and defined by intersecting the hot pixel with the right half plane delimited by x=xpos (needed for step (2), stretching)
- the sweep inside the hot pixel does not stop at xpos; for example, if a left endpoint of an urseg is detected inside the pixel, the urseg is added to the y-structure

The erasure algorithm, analysis

- similar to standard sweep-line algorithm
- work on a hot pixel proportional to the number (!) S_h of ursegments that cross it, rather than to the complexity of the arrg inside it, with O(log n) work per crossing ursegments
- running time: $O(n \log n + \sum_{h \in H} |S_h| \log n)$
- Q: how much time does it work on this?



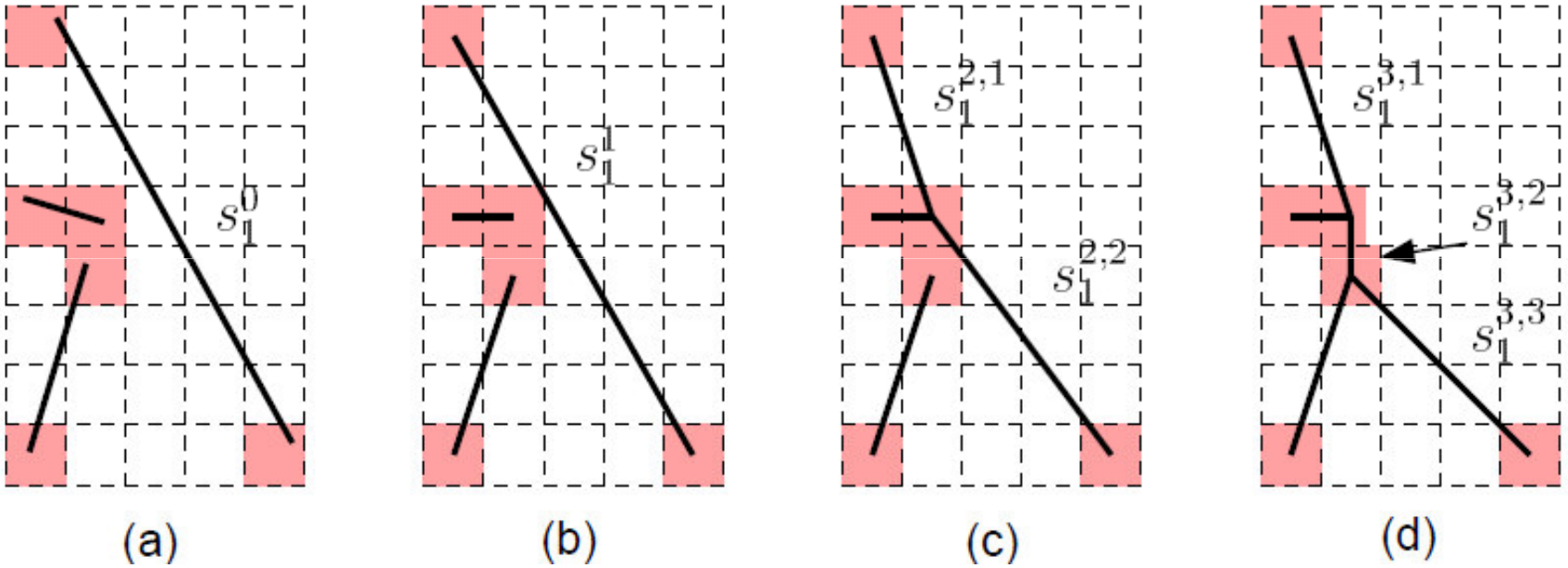
A problem with SR's rounding quality



- if we allow b bits per integer, our pixel grid has 2^b x 2^b square pixels
- the distance between a vertex and a non-incident edge

$$1/\sqrt{(2^b-1)^2+1} \approx 2^{-b}$$

A solution: Iterated Snap Rounding (ISR)



ISR, algorithm

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

ISR, algorithm

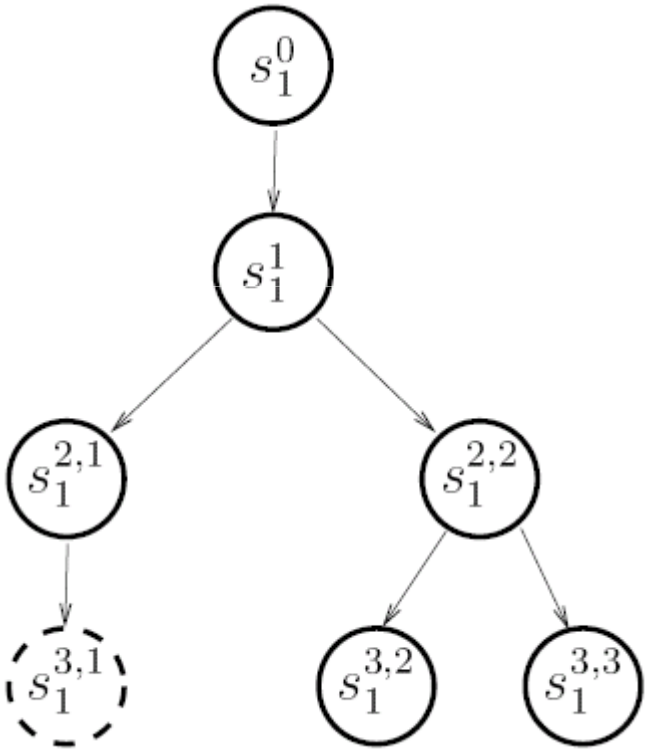
REROUTE(s)

// s is the input segment with endpoints p and q

1. query D to find H_s, the set of hot pixels intersected by s
2. **if** H_s contains a single hot pixel // s is entirely inside a pixel
3. **then** add the center of the hot pixel containing s to OUTPUT_CHAIN
4. **else**
5. 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
6. **if** ($r = 2$ **and** p, q are centers of pixels)
7. **then** add the link $\overline{m_1 m_2}$ to OUTPUT_CHAIN
8. **else**
9. **for** $i = 1$ to $r - 1$
10. REROUTE($\overline{m_i m_{i+1}}$)

ISR, alternative view

- Reroute for one ursegment is described by a tree



nodes denoted by full-line circles contain segments with which we query the structure D

the dashed circle denotes a segment which is an exact copy of the segment of its parent

ISR, properties

- finite process

ISR = a finite number of rounds of SR

no new pixels created

a polysegment s* is (weakly) x-monotone and (weakly) y-monotone

- preserves the topology in the same sense that SR does
- a vertex is at least half a unit away from any non-incident edge
- the rounding chain is in the Minkowski sum of the ursegment and a square of side size k (= depth of the recurrence of Reroute) centered at the origin

ISR, complexity

- we compute the hot pixels in O(n log n + I) time
- using multi-level partition trees to answer segment/pixel queries, split between vertical pixel boundaries and horizontal boundaries
- each tree, when allowed M unit of storage, $N \leqslant M \leqslant N^2$

takes $O(M^{1+\varepsilon})$ preprocessing time, and answer queries in $O(N^{1+\varepsilon}/\sqrt{M} + g)$ time, where g is the number of hot pixels found

- using standard tricks we can balance between preprocessing and query time without knowning the number of queries in advance
- the number of queries at most 2L, where L is the total number of links in all the chains together

ISR, complexity summary

- time $O(n \log n + I + L^{2/3} N^{2/3+\varepsilon} + L)$ for any $\varepsilon > 0$
- space $O(n + N + L^{2/3} N^{2/3+\varepsilon})$

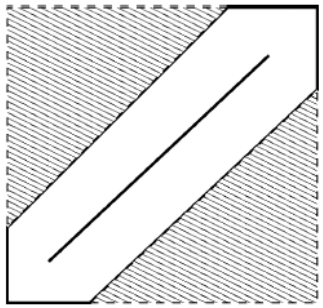
n - # of segments

I - # of intersections in the original arrg

N - # of hot pixels

L - overall complexity of the rounding chains

ISR, implementation

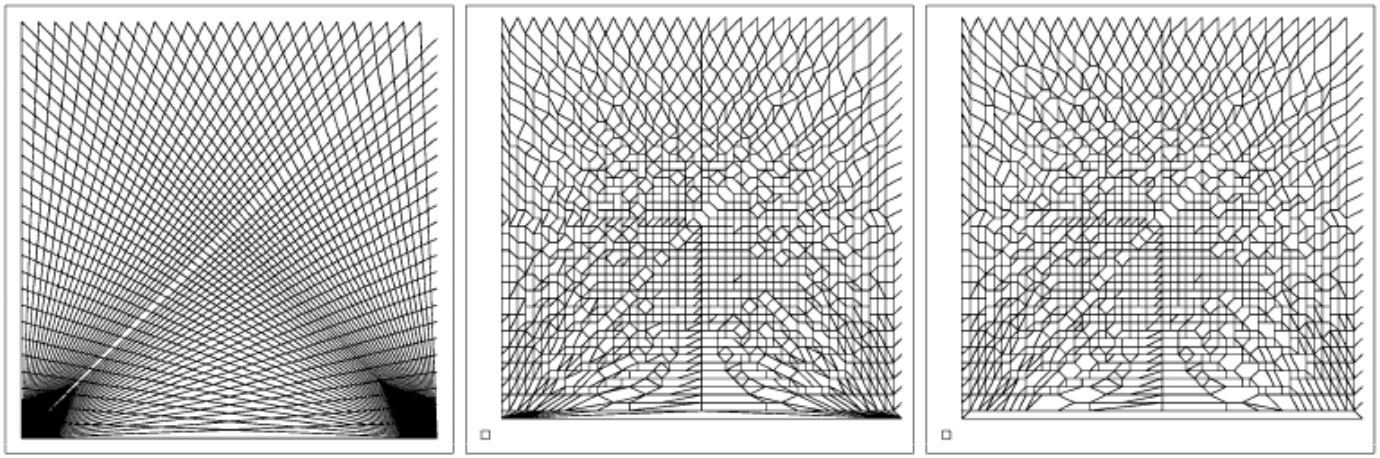


- substituting multi-level partition trees by c-oriented kd-trees
- selective rotations (creating a tree only for many potential queries)
- using exact rotations

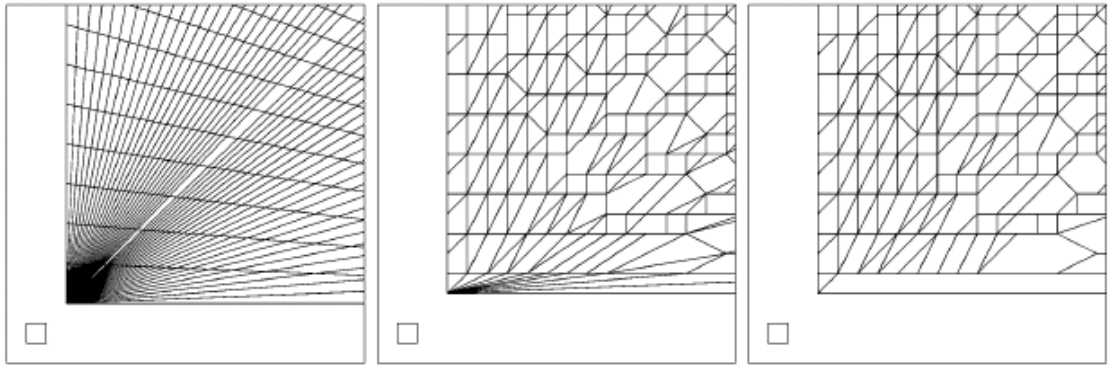
ISR, CGAL

- ISR is currently CGAL's SR software
- the user can curb the number k of iterations (height of the tree)
- setting k=1 results in a standard snap-rounded arrangement

SR vs. ISR, example 1

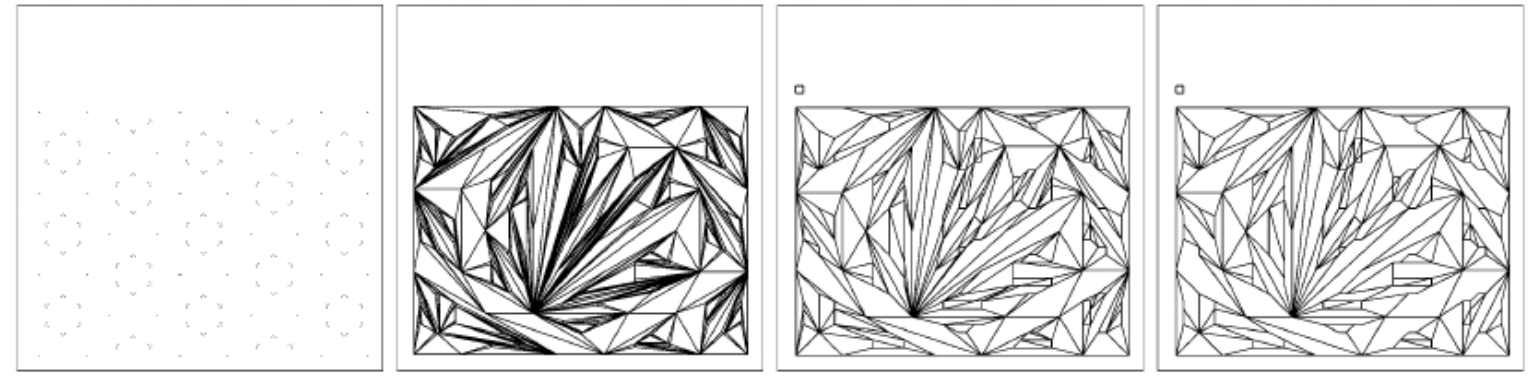


Input SR output ISR output

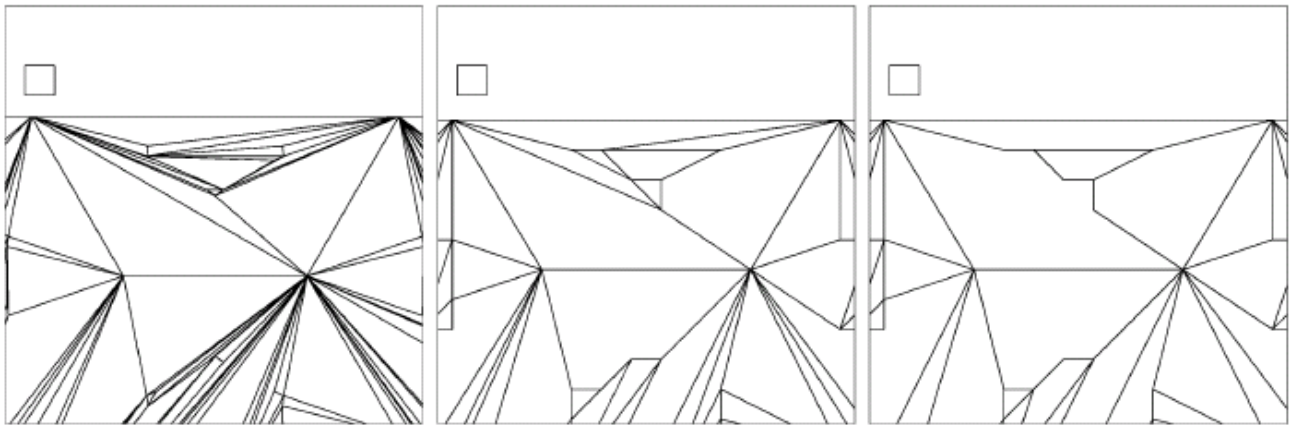


Input zoom in SR output zoom in ISR output zoom in

SR vs. ISR, example 2

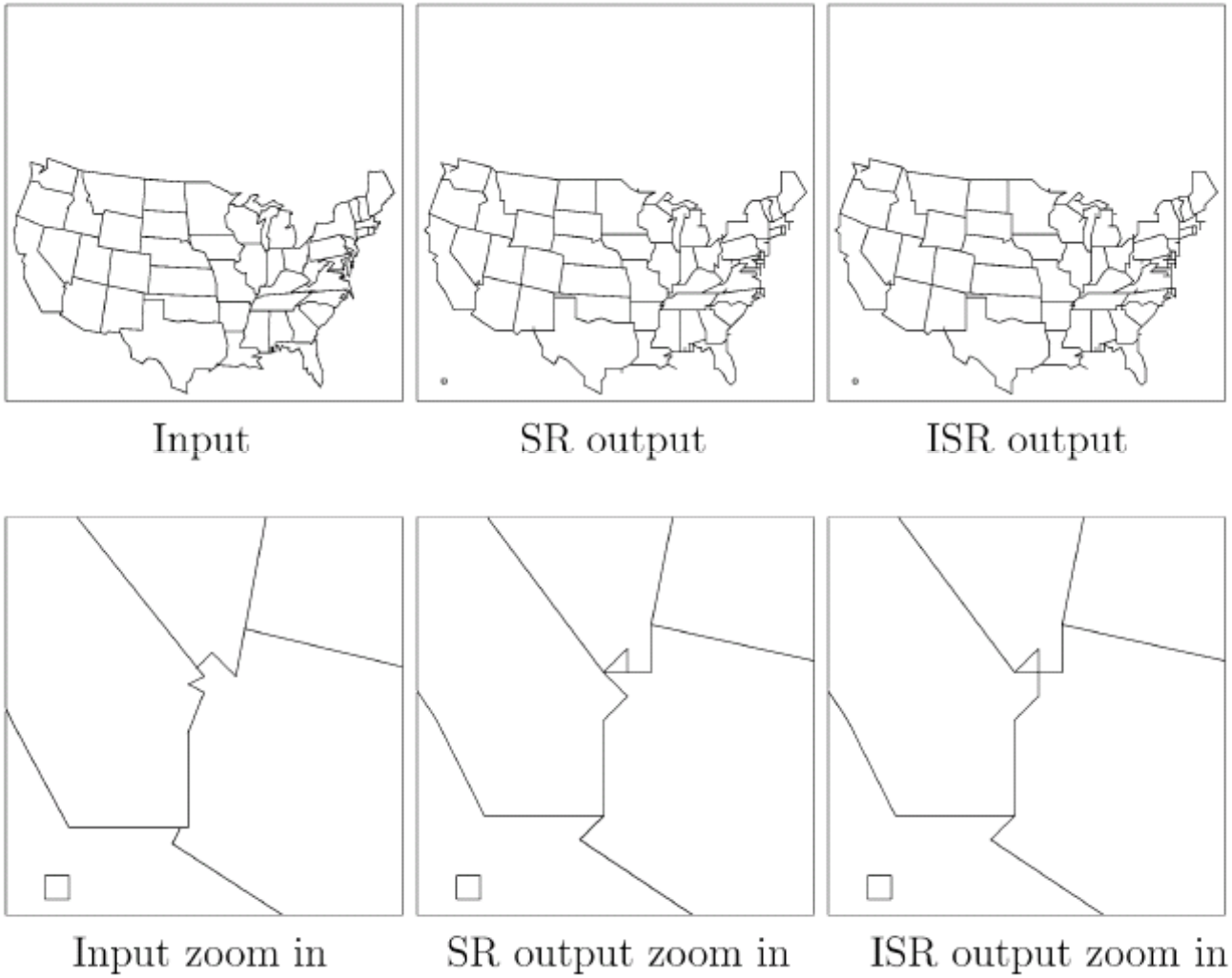


Input points | Input triangulation | SR output | ISR output

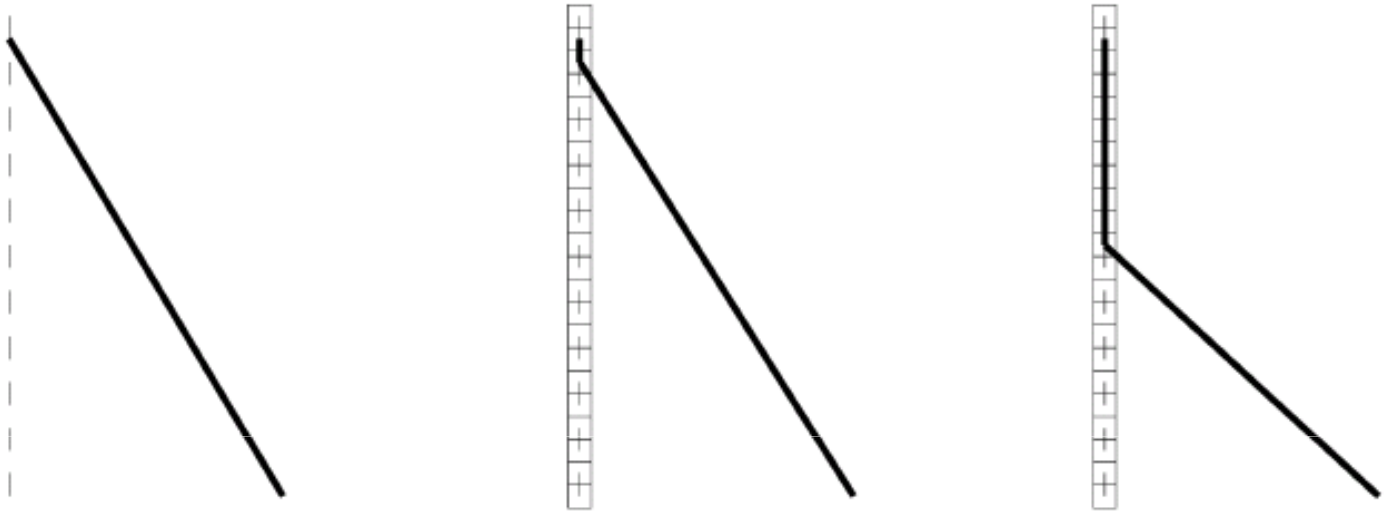


Input zoom in | SR output zoom in | ISR output zoom in

SR vs. ISR, example 3

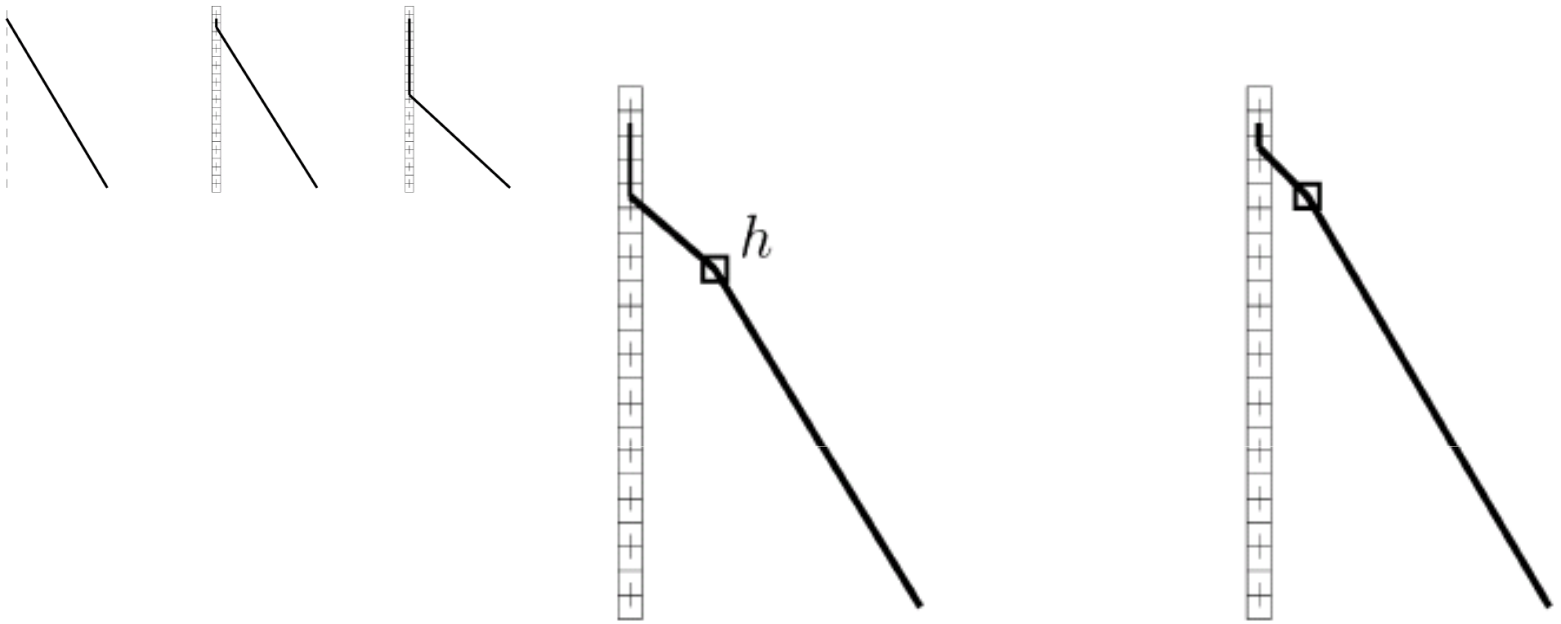


A problem with ISR



- input segment, SR, ISR
- huge drift
- can be as large as $\theta(n^2)$ pixels, for n input segs

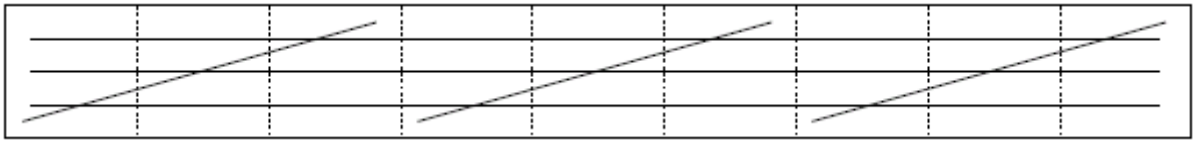
A solution: ISR with Bounded Drift

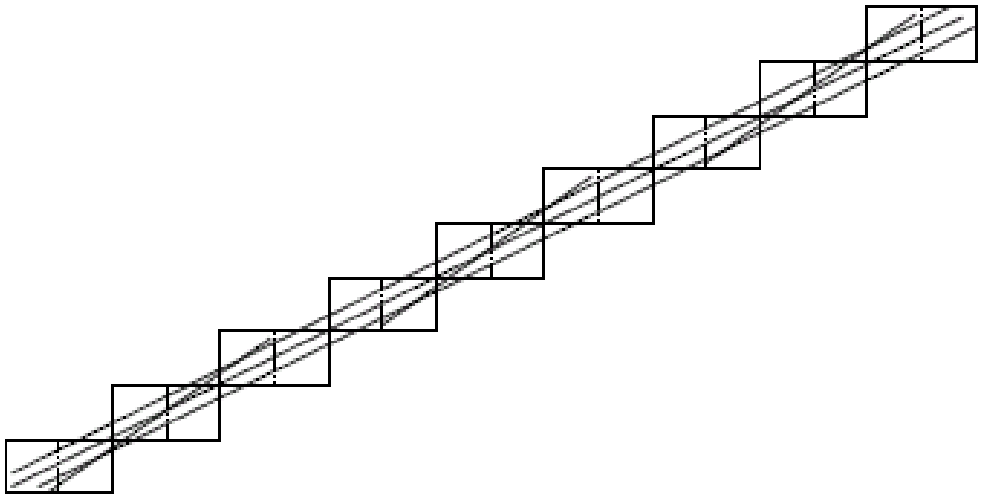


- ISRBD results with two different allowable drift parameter
- for details see bib list

The complexity of SR and ISR

- the complexity of a single chain s* can be $\theta(n^2)$
- the overall complexity of all the chains can be $\theta(n^3)$

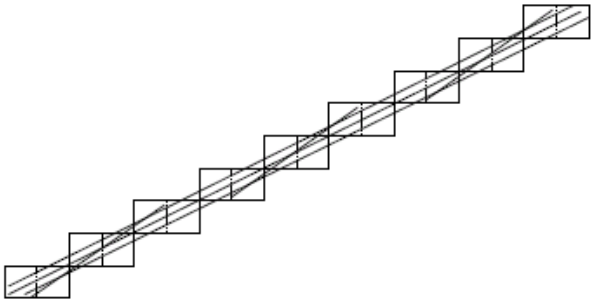




The complexity of SR and ISR, cont'd

- the maximum complexity of an SR'ed (or ISR'ed) arrangement is $O(n + I)$ which is at most $O(n^2)$

Worst-case performance of the erasure algorithm



- Q: back to the standard snap rounding:

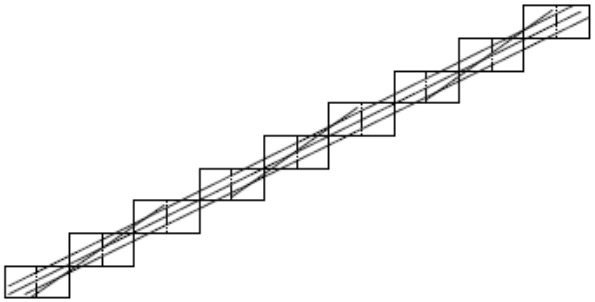
  what is the worst case complexity of the erasure algorithm on the above input?

- A: $O(n \log n + \Sigma_{h \in H} |S_h| \log n)$

which can be $\Theta(n^3 \log n)$

- can we do better?

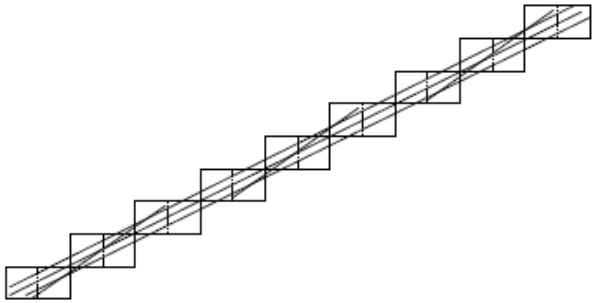
The collapse of many similar segments



- all the algorithms we have seen so far work $\Omega(n^3)$ time on the example above, although the (in fact any) rounded arrangement has complexity $O(n^2)$
- many segments collapse to the same final chain
- if we only care about the final rounded arrg then we can handle similar ursegments collectively; we avoid handling ursegments individually wherever possible

Intersection-sensitive snap rounding: The bundles algorithm

- input: as before
- output: the rounded arrangement represented as a graph
  - the nodes represent the centers of hot pixels
  - an arc connects two nodes that correspond to hot pixels h and g iff an ursegment passes through h and g and crosses no other hot pixel in-between
- the bundles algorithm runs in time O((n+I) log n)
- Q: how much time it takes on this?



The bundles algorithm, overview

1) identifies the set V of nodes of the output graph G=(V,E), namely the hot pixels (standard, nothing new)
2) determines E, namely decides which pairs of nodes of G should be connected by an arc

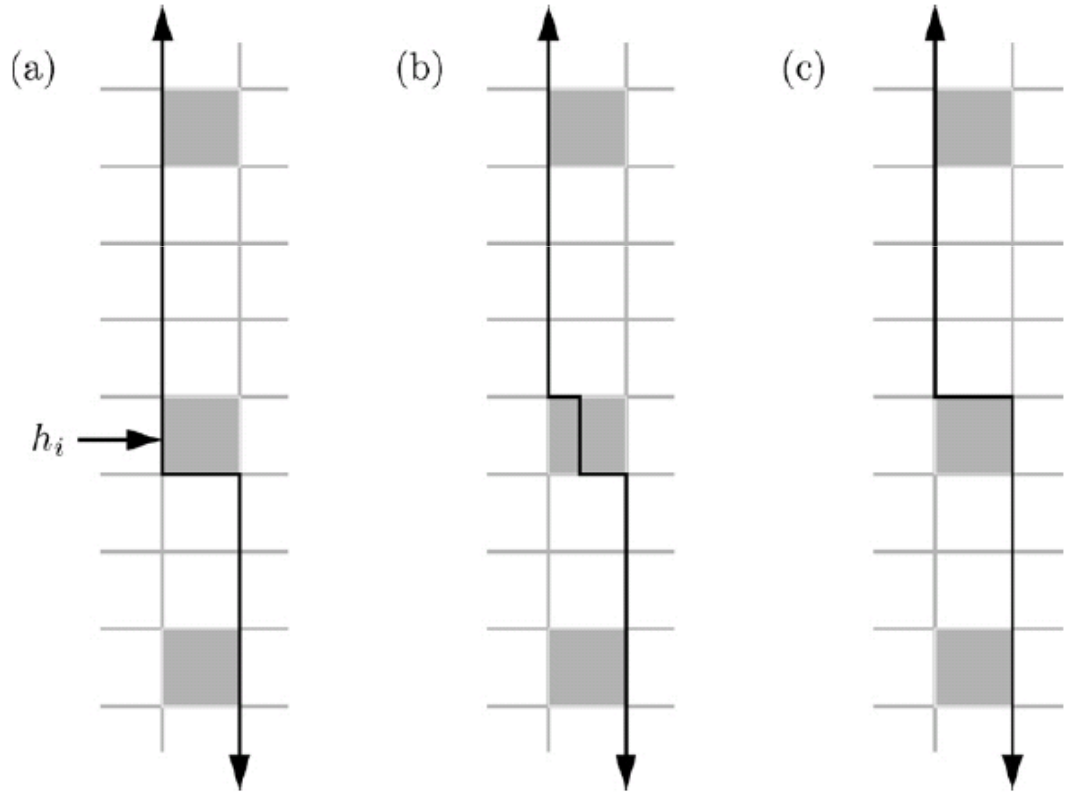
- the second stage is carried out in two sweep-line passes; we split the ursegs into S^+ and S^-, ursegs with positive and negative slopes respectively, and each is treated separately
- we assume for simplicity that there are no axis-aligned ursegs – can be easily relaxed; still we may have axis-aligned edges in the rounded arrg

Determining the arcs of G for S^+

- when the sweep-line is at x=xpos, a bundle is a set of ursegments intersecting the sweep-line, all originating from the same hot pixel to the left of xpos, and whose "fan" does not intersect any hot pixel
- for a bundle $b=(S_b,h_b)$
  - S_b: the ursegments in the bundle
  - h_b : the hot pixel of origin
  - u(b): its uppermost urseg
  - l(b): its lowest urseg
  - a search tree T_b that stores all the ursegs of b
- a tree T storing all the u(b)'s and l(b)'s in their order along the sweep-line

The sweep poly line

- consists of up to five axis-aligned segments
- sweeping over a hot pixel:



Actions upon reaching a hot pixel h

1) find all bundles reaching h
   - by querying the status line with the interval [se(h):nw(h)] of south-east, north-west corners
2) split the bundles and create new bundles
   - one continuing above, one below, and one starting at h (two of the above may be empty)
3) update the status structures, T and T_b's
   - while sweeping inside h

- total running time O((n+I) log n)

The best of both worlds: The erasure-bundles algorithm

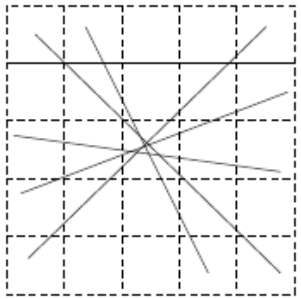
- input: as before
- output: the rounded arrangement represented as a graph
  - the vertices represent the centers of hot pixels
  - an edge connects to vertices that correspond to hot pixels h and g iff an ursegment passes through h and g and crosses no other hot pixel in-between
  - extra data structure from which the original ursegments which collapsed to an edge can be retrieved

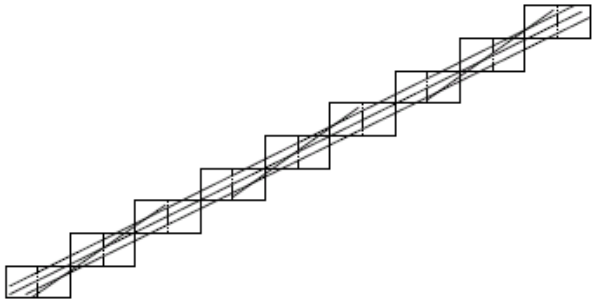
The erasure-bundles algorithm, running time

- the algorithm runs in time

  $O(n \log n + \sum_{h \in H} is(h) \log n)$,

  where $is(h)$ is the number of ursegments that have an endpoint or an intersection inside h
- Q: what is the running time in the following?





The erasure-bundles algorithm, details

- the algorithm, as its name implies, marries the erasure algorithm with the bundles algorithm, with the addition of maintaining a persistent data structure, which compactly encodes which ursegments collapse into each edge of the rounded arrg
- homework: complete the details

  see bib list

Snap rounding: pros and cons

pros

- convenient numerical output
- fairly efficient
- preserves certain topological properties

cons

- very limited range of effective applicability
- requires special machinery for preprocessing (is it necessary?)

major challenge

- devise consistent and efficient rounding schemes for planar arrangements of curves and for arrangements of surfaces in higher dimensions

References

- precursor of SR, contains examples why naïve rounding is bad
  [Greene-Yao ’86]
  Finite-resolution computational geometry, FOCS, 143-152
- the basic SR approach
  [Hobby ‘99]
  Practical segment intersection with finite precision output
  CGTA 13:199–214
- the erasure algorithm
  [Guibas-Goodrich-Hershberger-Tanenbaum ‘97]
  Snap rounding line segments efficiently in two and three dimensions
  SoCG, 284-293
- ISR
  [H-Packer ‘02]
  Iterated snap rounding, CGTA 23:209-225

References, cont'd

- properties of SR
  [Guibas-Marimont '98]
  Rounding arrangements dynamically, IJCGA 8:157-178
- the bundles algorithm
  [de Berg-H-Overmars '07]
  An intersection-sensitive algorithm for snap rounding
  CGTA 36:159-165
- the erasure-bundles algorithm
  [Hershberger '08]
  Improved output-sensitive snap rounding
  DCG, 39:298-318
- ISRBD
  [Packer '08]
  Iterated snap rounding with bounded drift, CGTA 40:231-251

References, more work on rounding

- [Fortune '99]

  Vertex-rounding a three-dimensional polyhedral subdivision
  DCG 22:593-618
- [Devillers-Guiges '06]

  Inner and outer rounding of Boolean operations on lattice polygonal regions, CGTA 33:3-17
- [Milenkovic '00]

  Shortest path geometric rounding, Algorithmica 27:57-86

  DCG, 39:298-318
- [Eigenwillig-Kettner_Wolper '07]

  Snap rounding of Bezier curves, SoCG 158-167

THE END