Geometric Rounding

Snap rounding arrangements of segments

Dan Halperin

danha@tau.ac.il

Tel Aviv University

Rounding geometric objects

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

```
T1:
x= 28027/25243, y= 43613/18457, z= 14423/37273
x= 20353/2617, y= 26497/32299, z= 3673/63667
x= 55897/42403, y= 499/27767, z= 31253/10243
T2:
x= 53593/24763, y= 62501/63317, z= 11827/5693
x= 57143/65423, y= 40483/59447, z= 27739/62327
x= 57283/22027, y= 41231/45817, z= 9433/48673
T3:
x= 5693/48527, y= 11597/7757, z= 58367/44017
x= 2377/59471, y= 23831/3163, z= 57287/25343
x= 16657/46507, y= 57283/14783, z= 9437/6911

Intersection:
x (normalized rational)=
297428938184216477745466885306417207432716614825860310134480766082004
2660360261669891047833223434838758837012 / 831908664650212786986421 52
774750081010 83902598620752970358356141334716771364757960126316537540
08326934339751
```

Rounding vs. simplification

the two problems are closely related

typical 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

The role of exact computing in rounding

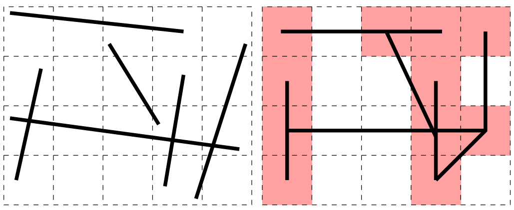
we will use exact geometric computing in all the algorithms that we present below

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 arrg's of segments, definition



- a pixel containing an arrg vertex is a hot pixel
- for each original segment s pass its approximating polygonal chain s^* thru the hot pixels that s crosses

the chain is made of links or subsegments

Snap rounding, history

- [Greene+Yao '86] rounding polygonal subdivisions: topology preserving, many kinks/links
- [Greene], [Hobby '99] SR arrgs of segments
- [Guibas+Marimont '98] dynamic SR
- [GGHT '97] quasi output sensitive SR I
- [H-Packer '02] iterated snap rounding
- [de Berg-H-Overmars '04] quasi output sensitive SR II

Snap rounding, geometric proximity

the rounding chain is in the Minkowski sum of the original segment and a pixel centered at the origin

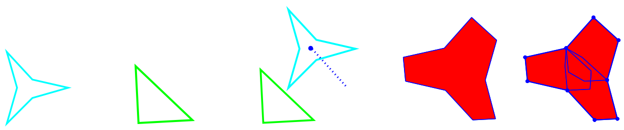
Minkowski sums (see the next slide for an illustration)

- A and $B(v)$ intersect iff $v \in A \oplus -B(0)$ where $B(v)$ is a copy of B whose reference point is located at $v \in \mathbb{R}^2$
- the Minkowski sum of convex sets is convex

Minkowski sums and collision detection

given two planar sets A and B, their Minkowski sum $A \oplus B$ is the set $\{a + b \mid a \in A, b \in B\}$

the robot B collides with the obstacle A whenever the reference point of B is inside the sum $A \oplus (-B)$



Snap rounding, topology preservation

[Guibas-Marimont '98]

features may collapse but a curve does not cross over a vertex

a 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

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

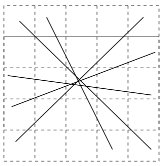
overall running time $O((n + I)\log n + L)$

n - # of segments
I - # of intersections in the original arrg
L - overall complexity of the rounding chains

Quasi output-sensitive algorithm I

[GGHT:Goodrich-Guibas-Hershberger-Tanenbaum '97]

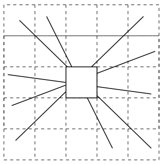
avoid computing all intersections when unnecessary



running time: $O(n \log n + \Sigma_{h\in H}|S_h| \log n)$, where H is the set of hot pixels, and S_h is the set of segments passing thru the pixel h

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

GGHT, bird's eye view



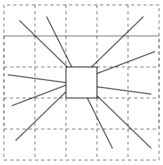
(1) erase the arrangement inside hot pixels

(2) stretch the nodes (endpoints of external fragments) to the corresponding hot pixel centers: trivial

(special handling of segments that lie fully inside a (hot) pixel

how to efficiently detect hot pixels and erase their interior

GGHT, algorithmic details: definitions



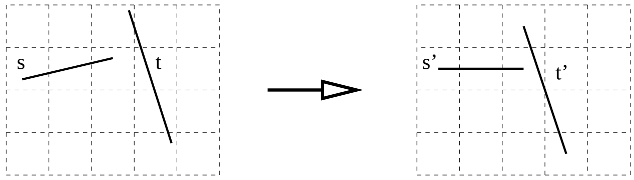
- `xpos`: the x coordinate where a critical event was spotted
- `pix`: the newly discovered hot pixel
- `pix.toplist`: x-ordered list of original segments that intersect the top of `pix`
- `pix.botlist`: similar for the bottom of `pix`
- `Hcur`: y-ordered list of hot pixels at `xpos`

GGHT, algorithmic details: events

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

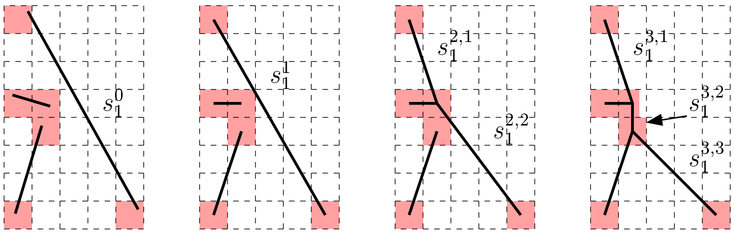
running time: $O(n \log n + \Sigma_{h \in H} |S_h| \log n)$

The distance between a vertex and a non-incident edge



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

Iterated snap rounding (ISR)



(a) (b) (c) (d)

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 $\mathcal{S}^*$

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

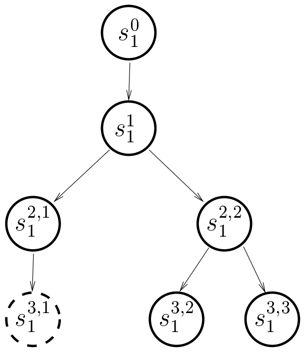
else

for $i = 1$ to $r - 1$

REROUTE($\overline{m_i m_{i+1}}$)

ISR, an alternative view

REROUTE for one segment 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 node containing an exact copy of the segment of its parent

ISR: Properties

- finite process

ISR = a finite number of rounds of SR

no new hot pixels created

- 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 original segment and a square of side size k (= depth of the recurrence of REROUTE) centered at the origin

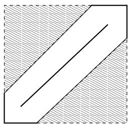
ISR: Complexity

- using multi-level partition trees to answer segment/pixel queries
- 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

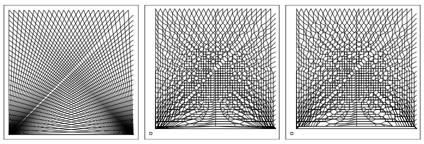
- substituting multi-level partition trees by c-oriented kd-trees



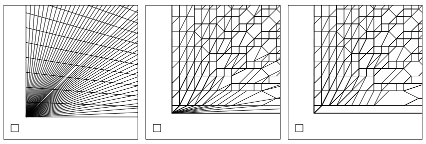
- selective rotations (creating a tree only for many potential queries)
- using exact rotations
- CGAL (sweep, kd-trees), LEDA (rational)

available as CGAL extension package

SR vs. ISR: example 1

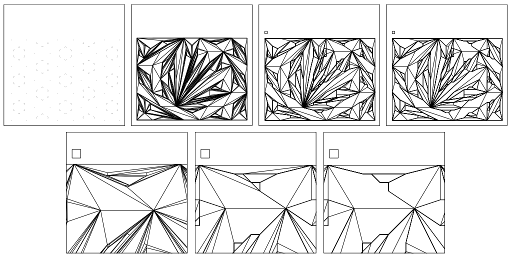


Input SR output ISR output

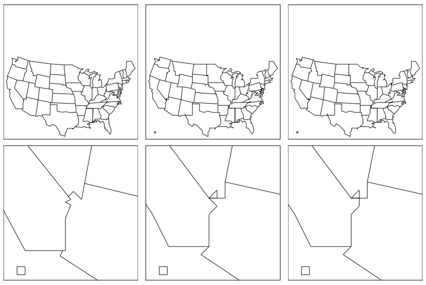


Input zoom in SR output zoom in ISR output zoom in

SR vs. ISR: example 2



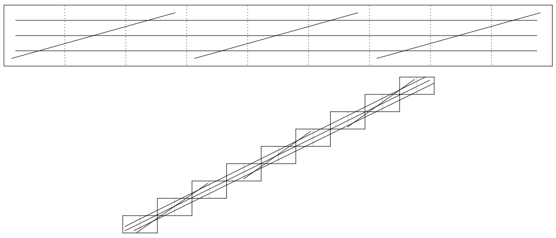
SR vs. ISR: example 3



The complexity of SR and ISR

the complexity of a single chain s^* can be $\Theta(n^2)$

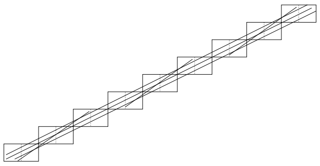
the overall complexity of all the chains can be $\Theta(n^3)$



The maximum complexity of an SR'ed (or ISR'ed) arrangements is $O(n+I)$ where I is the number of intersections in the original arrangement, which is at most $O(n^2)$

Quasi output-sensitive algorithm II

back to the standard snap rounding
what is the worst case complexity of GGHT?



$O(n \log n + \Sigma_{h \in H} |S_h| \log n)$, which can be $\Theta(n^3 \log n)$

[de Berg-H-Overmars '04]: an algorithm with worst-case running time $O(n^2 \log n)$

dBHO: key idea

all the algorithms we have seen so far work at least time proportional to the overall complexity of the chains, which is $\Theta(n^3)$ in the worst case

aiming for subcubic running time, we cannot afford to work on each chain separately

instead we group original segments into bundles: a bundle is a set of original segments all coming out of the same hot pixel; a bundle persists as long as it does not 'cross' another hot pixel

Problem: the ultimate output sensitive algo

devise an algorithm which at once (i) avoids computing unnecessary intersections (as GGHT does) and (ii) is sensitive to the complexity of the rounded arrg rather than to the overall complexity of the chains (as dBHO is)

Alternative rounding: shortest path rounding

- [Milenkovic '00] rounding with shortest paths

Rounding in 3D

- [GGHT '97] segments in 3D
- [Milenkovic '90]
- [Fortune '99] 3D snap rounding

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

the material presented in class is based on the following papers:

- Goodrich, Guibas, Hershberger, and Tanenbaum '97, Snap rounding line segments efficiently in two and three dimensions, *Proc. 13th ACM Symposium on Computational Geometry*, pp. 284–293
- Guibas and Marimont '98, Rounding arrangements dynamically, *International J. Computational Geometry and Application*, vol. 8, pp. 157–176
- Halperin and Packer '02, Iterated snap rounding, *Computational Geometry: Theory & Applications*, 23(2), pp. 209-222
- de Berg, Halperin and Overmars '04, An intersection-sensitive algorithm for snap rounding, manuscript: `www.cs.tau.ac.il/~danha/papers/issr.pdf`

THE END