GEOINFO 2015
XVI Brazilian Symposium on GeoInformatics
November 29th to December 2nd, Campos do Jordão, SP, Brazil

USING RATIONAL NUMBERS AND PARALLEL COMPUTING TO EFFICIENTLY AVOID ROUND-OFF ERRORS ON MAP SIMPLIFICATION

Maurício G. Gruppi[1]
Salles V. G. de Magalhães[1,2]
Marcus V. A. Andrade[1]
W. Randolph Franklin[2]
Wenli Li[2]

[1]Departamento de Informática - Universidade Federal de Viçosa
[2]Rensselaer Polytechnic Institute - USA

Contents

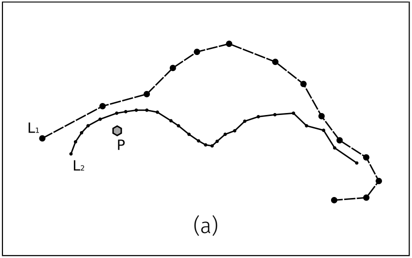
INTRODUCTION

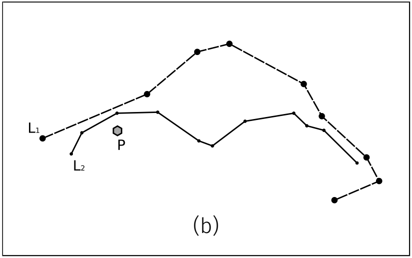
Map Simplification

What is Map Simplification?

- It's the process of reducing the amount of detail of a map.
- Such as reducing the number of vertices of a **polygonal chain** when altering scale.
- However, there are key features that must be preserved.

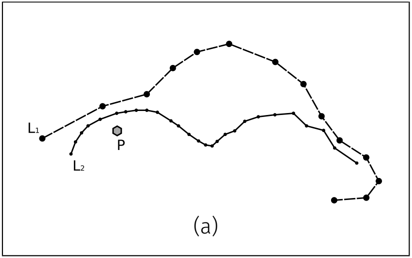
Map Simplification

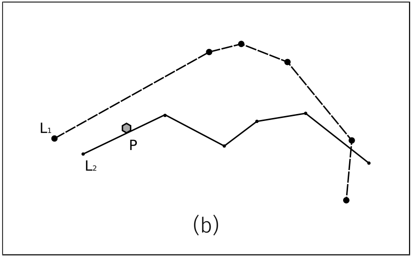




(a) shows an example of input set. A **topologically consistent** simplification of (a) is shown in (b).

Map Simplification





(b) shows a **topologically inconsistent** simplification of (a).

RELATED WORKS

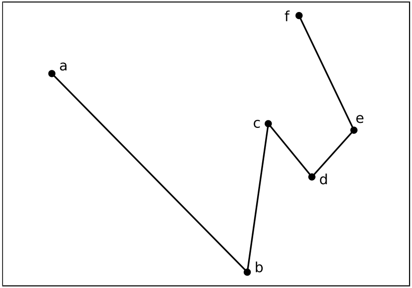
Ramer-Douglas-Peucker's Algorithm (RDP)
[Douglas and Peucker, 1973][Ramer, 1972]

- Simplification by *selection*.
- May produce inconsistency.
  - [Saalfeld, 1999]
  - [Li et al., 2013]

Visvalingam-Whyatt's Algorithm (VW) [Visvalingam and Whyatt, 1993]

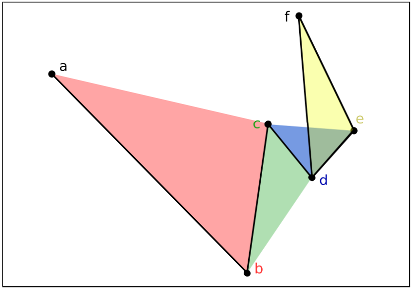
- Simplification by *elimination*.
- Ranks points by *effective area*.
- Removes points whose *effective area* is smaller than a given threshold.

VW's Algorithm



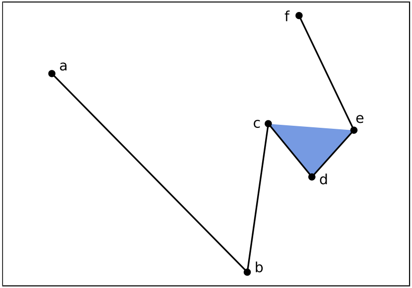
VW's Algorithm

- Calculates every point's *effective area* in L.
- **Definition:** The *effective area* of a polyline vertex v_i is the area of the triangle formed by v_{i-1}, v_i and v_{i+1}.



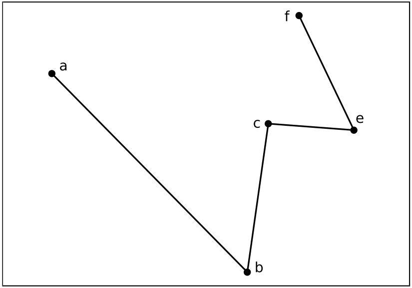
VW's Algorithm

- Calculates every point's *effective area* in L.
- Find the point p with smallest effective area.



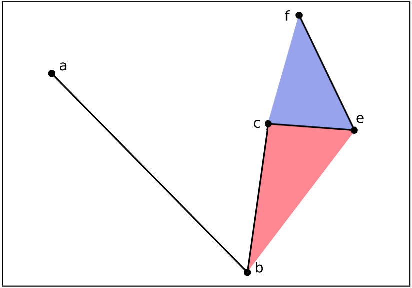
VW's Algorithm

- Calculates every point's *effective area* in L.
- Find the point p with smallest effective area.
- Remove p from L.



VW's Algorithm

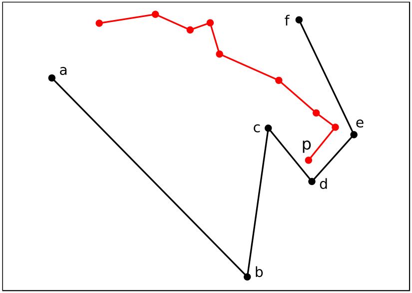
- Calculates every point's *effective area* in L.
- Find the point p with smallest effective area.
- Remove p from L.
- Calculate the new effective area for p's neighbors.



VW's algorithm can produce topologically inconsistent results.

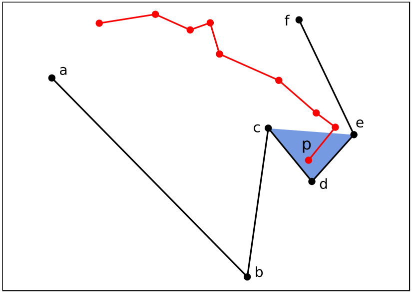
VW's Algorithm

- Input: 2 polygonal chains (**black** and red).



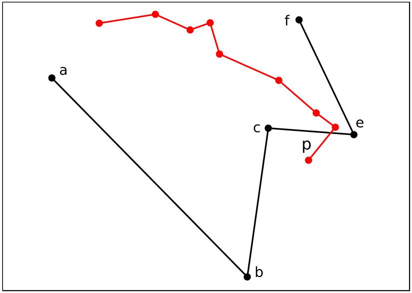
VW's Algorithm

- Input: 2 polygonal chains (**black** and **red**).
- Remove d from L.



VW's Algorithm

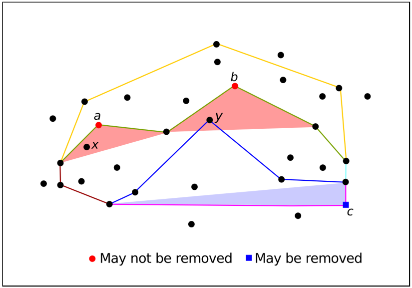
- Input: 2 polygonal chains (**black** and **red**).
- Remove d from L.
- An intersection has been created between both lines.



TopoVW

TopoVW [Gruppi et al., 2015]

- Ranks points by effective area.
- Checks for points inside each p's triangle.
- Removes p if there is none.
- Stops when a certain number of points have been removed.

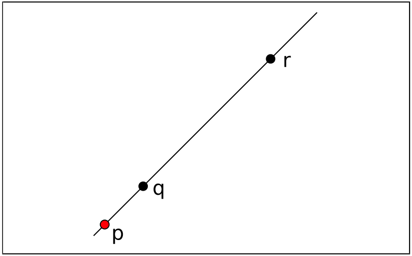


Round-off Errors In Floating Point Arithmetic

- Algorithms previously mentioned were designed for floating-point arithmetic.
- Arbitrary precision numbers represented as fixed precision numbers.
- May incur round-off errors.
- Therefore producing **wrong** results.

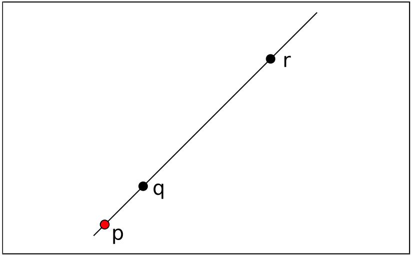
Round-off Errors In Floating Point Arithmetic

- Round-off errors affect planar orientation predicate [Kettner et al., 2008].
- The problem of finding whether three points *p*, *q*, *r*:
  - are collinear.
  - make a left-turn.
  - make a right-turn.



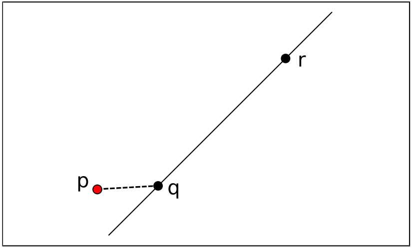
Round-off Errors In Floating Point Arithmetic

- Round-off errors affect planar orientation predicate [Kettner et al., 2008].
- The problem of finding whether three points *p*, *q*, *r*:
  - are collinear.
  - make a left-turn.
  - make a right-turn.

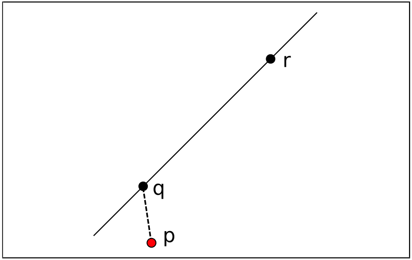


Round-off Errors In Floating Point Arithmetic

- Round-off errors affect planar orientation predicate [Kettner et al., 2008].
- The problem of finding whether three points *p*, *q*, *r*:
  - are collinear.
  - make a left-turn.
  - make a right-turn.



- Round-off errors affect planar orientation predicate [Kettner et al., 2008].
- The problem of finding whether three points *p*, *q*, *r*:
  - are collinear.
  - make a left-turn.
  - make a right-turn.



Round-off Errors In Floating Point Arithmetic

$$orientation = sign\left(\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}\right)$$

$$orientation = sign\left(\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}\right)$$

Sign:

- +: left turn.
- -: right turn.
- 0: collinear.

Round-off Errors In Floating Point Arithmetic

$$orientation = sign\left(\begin{vmatrix} 1 & p_x & p_y \\ 1 & q_x & q_y \\ 1 & r_x & r_y \end{vmatrix}\right)$$

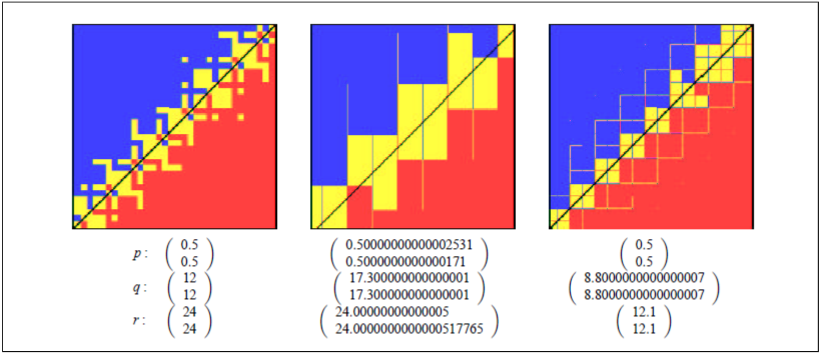
Sign:

- +: left turn.
- -: right turn.
- 0: collinear.

Possible problems:

- *rounding to zero.*
- *perturbed zero.*
- *sign-inversion.*

Round-off Errors In Floating Point Arithmetic



Result of the planar orientation problem using floating-point arithmetic. Source: [Kettner et al., 2008].

ROUND-OFF ERRORS ON MAP SIMPLIFICATION

We have tested floating-point round-off errors on map simplification:

- We needed to determine whether p is inside triangle T formed by (r, s, t).
- This was done by using *barycentric coordinates* of p in T.

Let a, b and c be scalars such that:

- $p_x = ar_x + bs_x + cr_x$
- $p_y = ar_y + bs_y + cr_y$
- a + b + c = 1

p lies inside T if and only if $0 \leq a, b, c \leq 1$

- A function $is\_inside(r, s, t, p)$ was implemented in C++ using floating-point numbers.

Round-off Errors on Map Simplification

- A function *is_inside*(r, s, t, p) was implemented in C++ using floating-point numbers.
- ***false inside***: outer point said inside.

Round-off Errors on Map Simplification

- A function *is_inside*(r, s, t, p) was implemented in C++ using floating-point numbers.
- ***false inside***: outer point said inside.
- May prevent simplification.

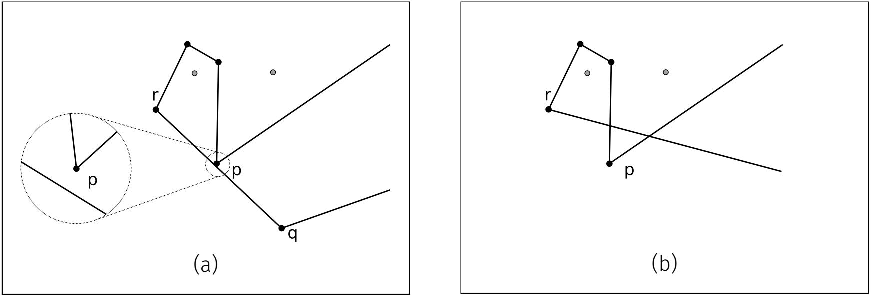
Round-off Errors on Map Simplification

- A function $is\_inside(r, s, t, p)$ was implemented in C++ using floating-point numbers.
- ***false inside***: outer point said inside.
- ***false outside***: inner point said outside.

Round-off Errors on Map Simplification

- A function $is\_inside(r, s, t, p)$ was implemented in C++ using floating-point numbers.
- ***false inside***: outer point said inside.
- ***false outside***: inner point said outside.
- May create improper intersections and self-intersections.

Round-off errors on map simplification



p was a **false outside**. Thus the removal of *q* was possible, creating self-intersections.

Solution to Round-off Errors

- ϵ-tweaking
- Snap-rounding
- Exact Arithmetic

ϵ-TWEAKING

ϵ-*tweaking* uses a tolerance value when comparing two numbers:

$$x = y, \text{ if } |x - y| \leq \epsilon.$$

- Automatically activates *rounding to zero*.
- Finding ϵ is difficult. Especially for big datasets.

Snap-rounding

Snap-rounding splits the map into pixels (cells). Rounds every endpoint to the center of its bounding pixel.

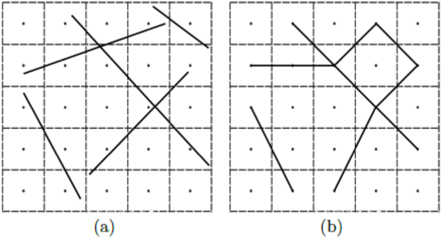


Figure: (a) before snap-rounding. (b) after snap-rounding. Intersections were introduced.

Exact Arithmetic With Rational Numbers

Exact Arithmetic with Rational Numbers:

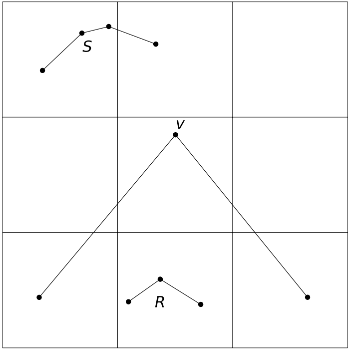
- Non-integer variables are represented as arbitrary precision rational numbers.
- Slower than floating-point arithmetic but round-off errors free.
- Overhead can be reduced using parallel computation.

Our Method

- EPLSimp uses exact arithmetic for simplifying polylines.
- A uniform-grid structure is used for determining which points needed to be tested for each triangle.
- Parallel computing used for performance.
- Lines are then subdivided into sets that can be simplified in parallel.

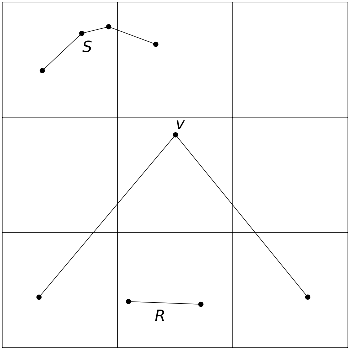
Our Method

- Construct a uniform-grid in parallel.



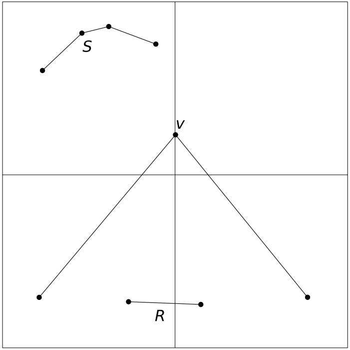
Our Method

- Construct a uniform-grid in parallel.
- Simplify line R.



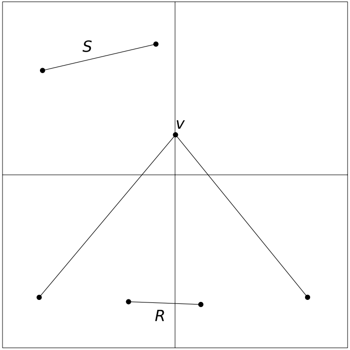
Our Method

- Construct a uniform-grid in parallel.
- Simplify line *R*.
- Decrease grid resolution.
- More lines inside single cells.
- Allows parallel simplification.



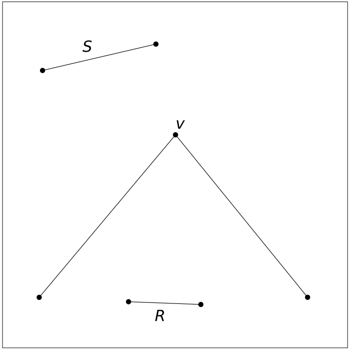
Our Method

- Simplify line S.



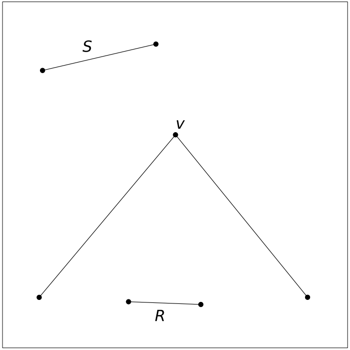
Our Method

- Simplify line S.
- Decrease grid resolution.



Our Method

- Simplify line *S*.
- Decrease grid resolution.
- Simplify the remaining lines (if any).



EXPERIMENTAL RESULTS

- *EPLSimp* was implemented in C++ using the GMPXX library [Granlund and the GMP development team, 2014].
- Artificial datasets were created to evaluate the occurrence of round-off errors.
- *EPLSimp* did not produce any topological inconsistencies.

Experimental Results - Performance

Table: Times (in ms) for the main steps of the map simplification algorithms. Rows *Max* represent the time for removing the maximum amount of points from the map while rows *Half* represent the time to remove half of the points.

	Dataset	1		2		3	
	Method	TopoVW	*EPLSimp*	TopoVW	*EPLSimp*	TopoVW	*EPLSimp*
Max.	Initialize	4	22	28	190	1828	5353
	Simplify	39	60	626	445	46069	57095
	Total	43	82	654	635	47897	62448
Half	Initialize	4	22	28	186	1847	5447
	Simplify	25	41	357	331	23021	48090
	Total	29	63	384	517	24868	53537

Experimental Results - Performance

Table: Times (in ms) for initializing and simplifying maps from the 3 datasets considering different number of threads. The simplification was configured to remove the maximum amount of points from the maps.

		Initialization			Simplification		
	Dataset	1	2	3	1	2	3
Threads	1	71	655	26833	176	1574	250237
	2	91	568	15483	152	1150	131310
	4	54	422	9853	99	689	82641
	8	34	240	6552	61	483	62089
	16	22	190	5353	60	445	57095

Conclusions

- We were able to avoid round-off errors using exact arithmetic with rational numbers.
- Parallel computing helped alleviating the overhead, approaching floating-point's processing time.
- Future works include:
    - Adapting *EPLSimp* for simplifying vector drawings and 3D objects.
    - Use exact arithmetic for other GIS algorithms.

Thank You

CAPES

CNPq

Conselho Nacional de Desenvolvimento Científico e Tecnológico

FAPEMIG

NSF

Rensselaer

mauricio.gruppi@ufv.br
salles@ufv.br
marcus@ufv.br
mail@wrfranklin.org