

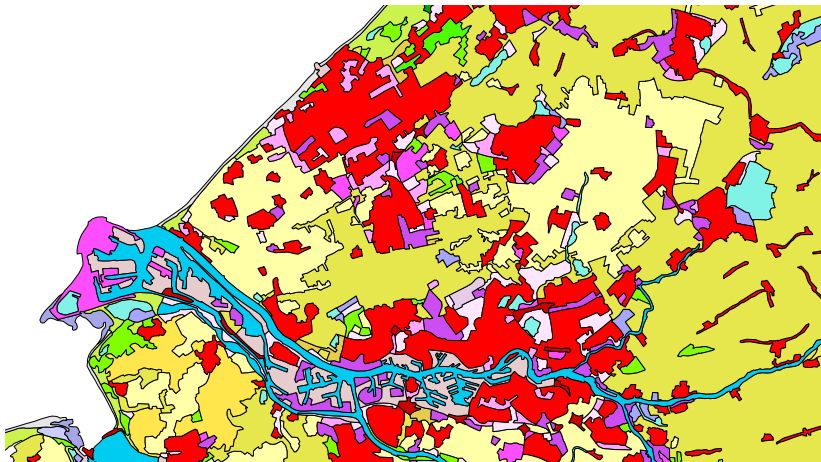
Automatically repairing polygons and planar partitions with *prepair* and *pprepair*

Ken Arroyo Ohori Hugo Ledoux Martijn Meijers

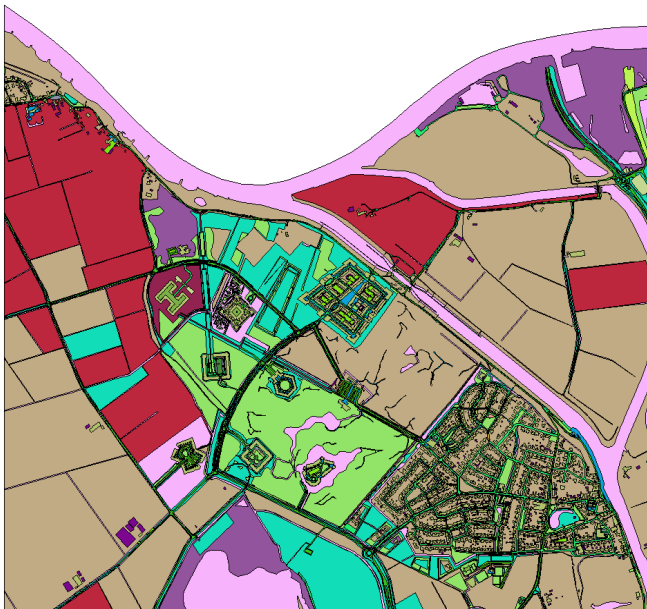


OSGIS 2012
September 5, 2012

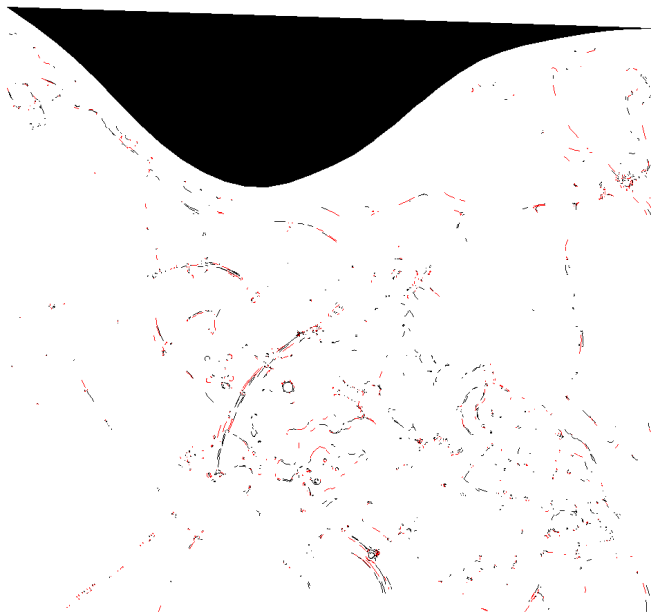
Planar partition = no gaps, no overlaps



Real data = problems



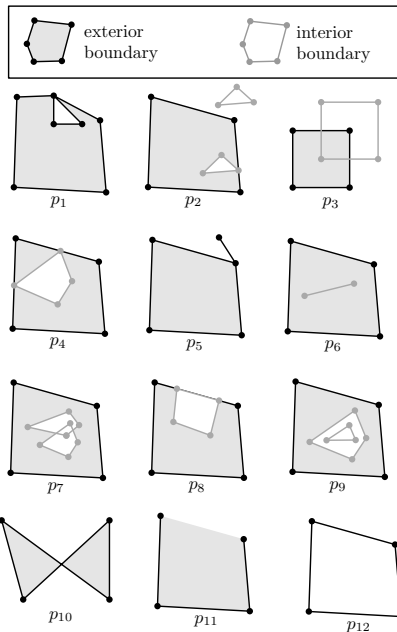
Real data = problems



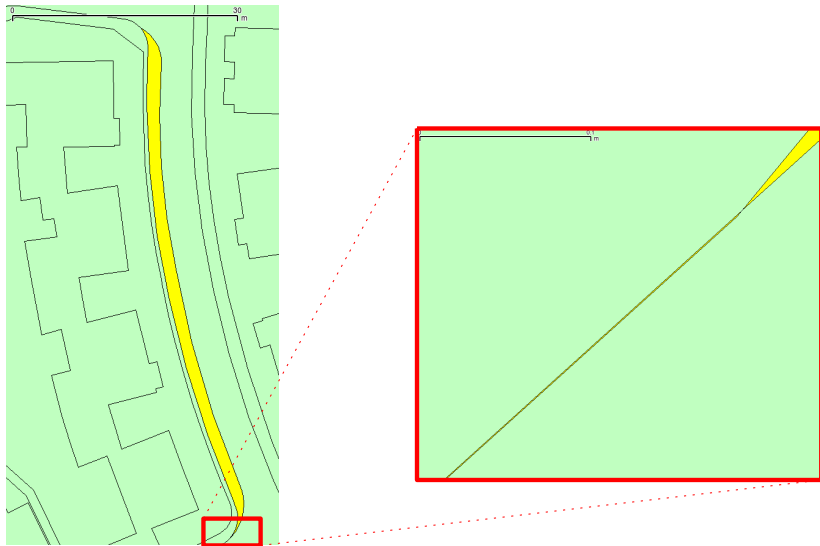
What about polygons?

OGC Simple Features +
ISO19107:

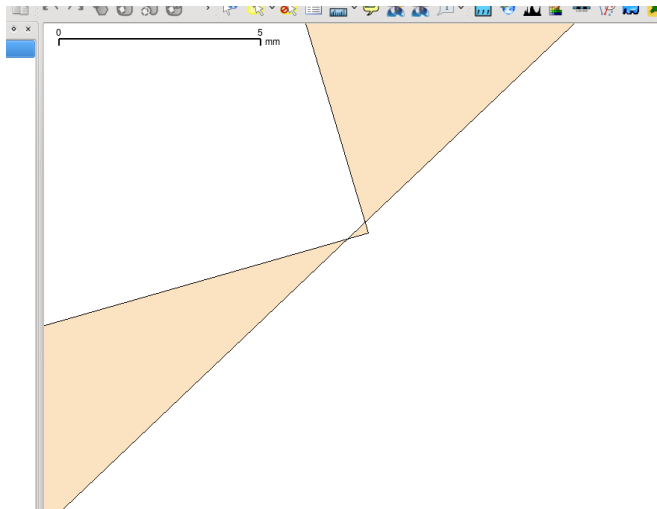
- 1 no self-intersection
- 2 closed boundaries
- 3 rings can touch but not overlap
- 4 no duplicate points
- 5 no dangling edges
- 6 connected interior



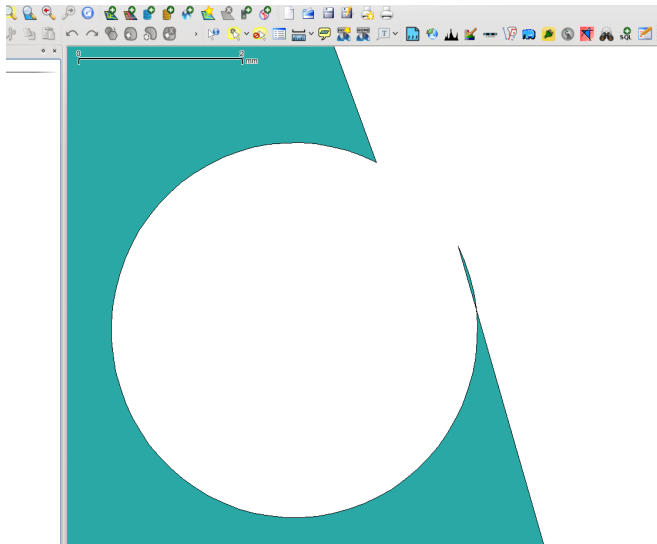
Real data = problems



Real data = problems

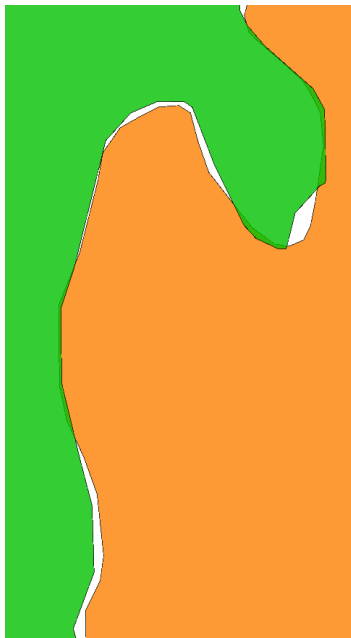


Real data = problems



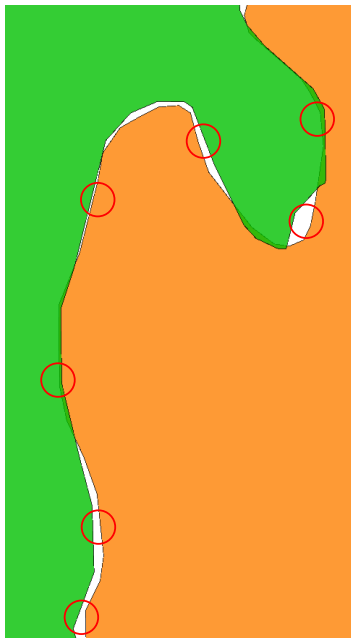
Standards/definitions tell us what is valid, but...
what to do with invalid data?

- Planar partitions: snapping / topology rules / manual work
- Polygons: “buffer-by-0” / PostGIS 2.0’s `ST_MakeValid()` / “visual repair”

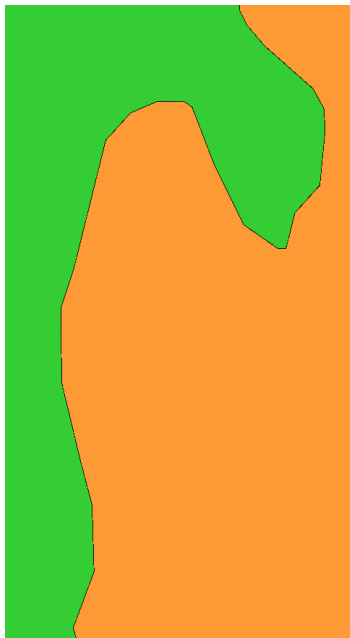


- Tolerance (*threshold*) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems

Snapping

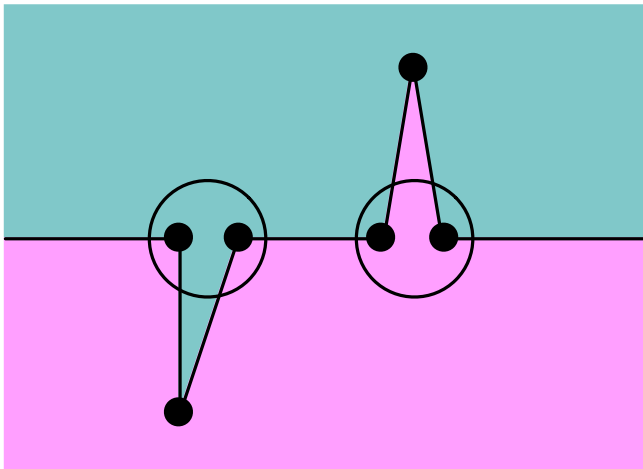


- Tolerance (*threshold*) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems



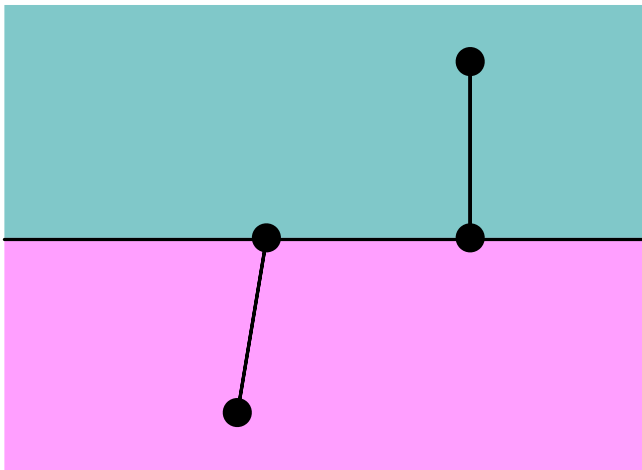
- Tolerance (*threshold*) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems

Snapping



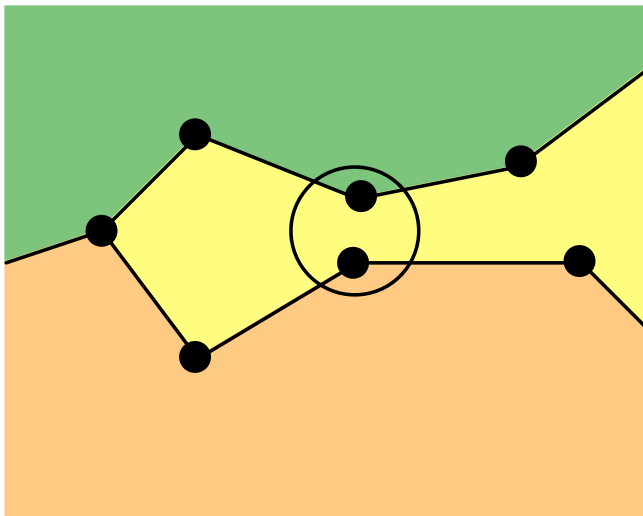
Spikes and punctures can create invalid polygons

Snapping



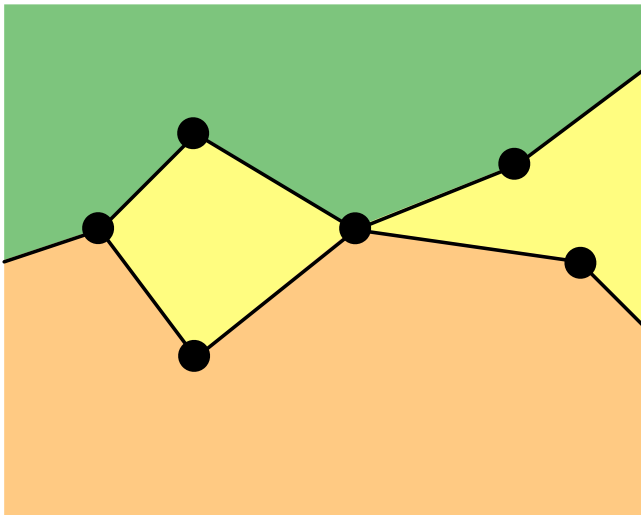
Spikes and punctures can create invalid polygons

Snapping



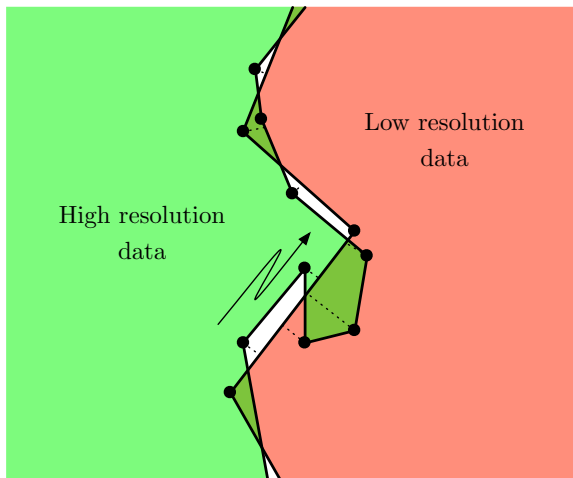
Splitting of polygons into several polygons

Snapping



Splitting of polygons into several polygons

Snapping



Topologically invalid result

Topology rules

Untitled - ArcMap - ArcInfo

File Edit View Bookmarks Insert Selection Tools Window Help

Spatial Analyst Layer: Editor Task: Create New Feature

Layer: corine_Topology Topology: corine_Topology

Layers

- corine_Topology
 - Area Errors
 - Line Errors
 - Point Errors
- e41n27

Error Inspector

Show: <Errors from all rules> 117 errors

Search Now ☒ Errors ☐ Exceptions ☒ Visible Extent on

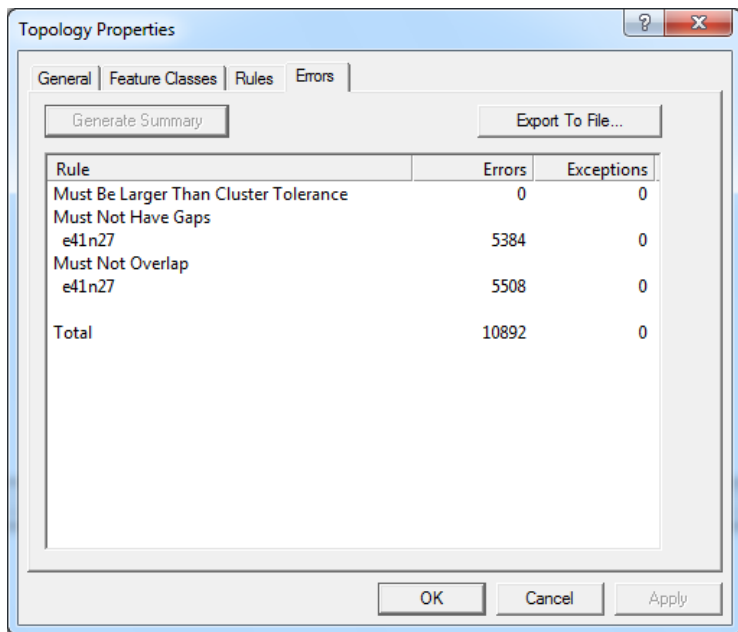
Rule Type	Class 1	Class 2	Shape
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon
Must Not Have Gaps	e41n27		Polygon

Cartography Tools
Conversion Tools
Data Interoperability Tools
Data Management Tools
Geocoding Tools
Geostatistical Analyst Tools
Linear Referencing Tools
Mobile Tools
Multidimension Tools
Network Analyst Tools
Samples
Schematics Tools
Server Tools
Spatial Analyst Tools
Spatial Statistics Tools
Tracking Analyst Tools

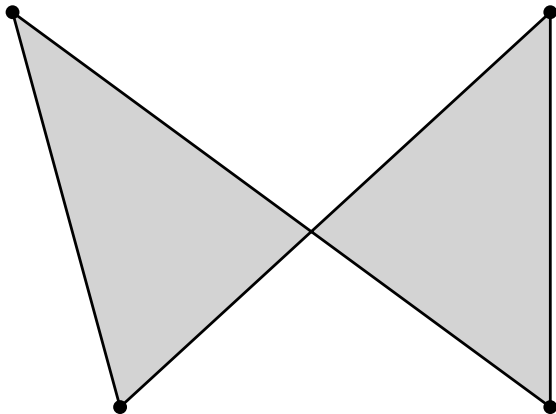
Favorites Index Search Results

Selects the edit task

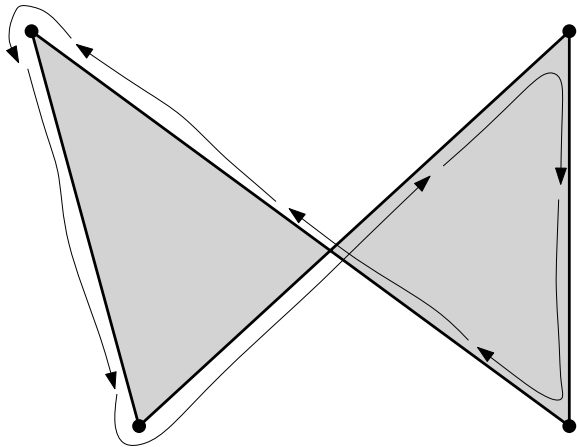
4131266.871 2759918.441 Meters

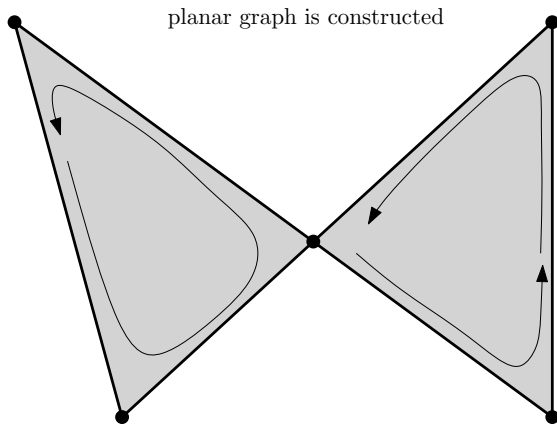


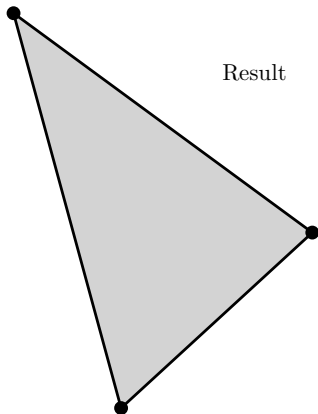
“Buffer-by-0”



“Buffer-by-0”

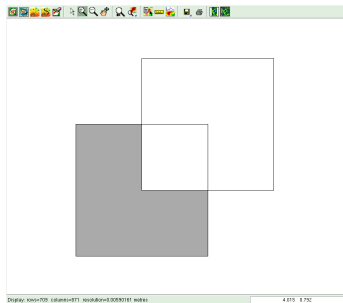
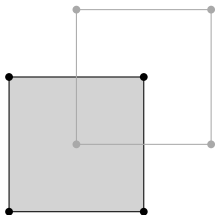




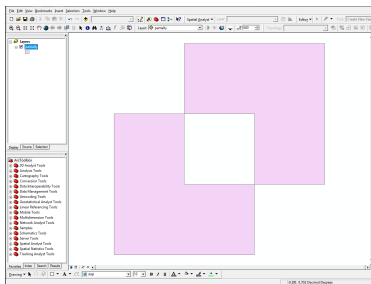


- high-level automatic repair function
- diff functions called depending on geometric and topological configurations of rings
- based on construction of planar graph (GEOS is used)
- not documented (“read the code”) = predicting behaviour is difficult
- very slow for big polygons

“Visual repair”



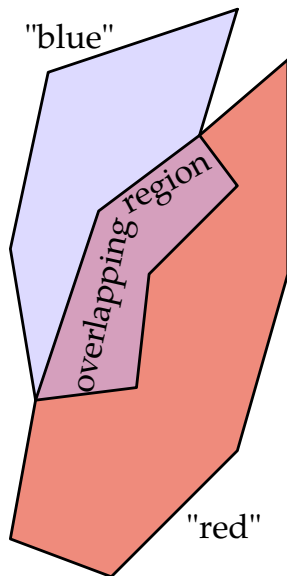
GRASS



ArcGIS

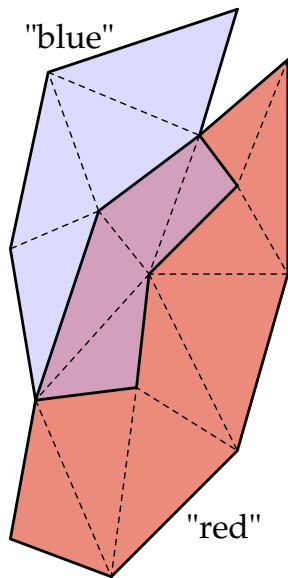
Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons



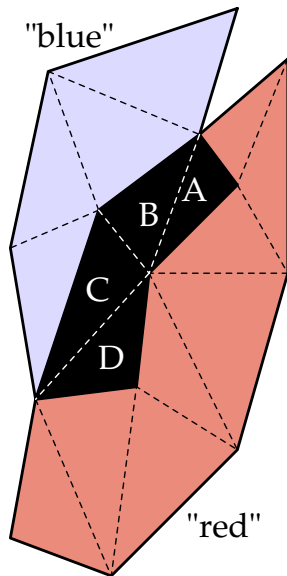
Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons



Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons



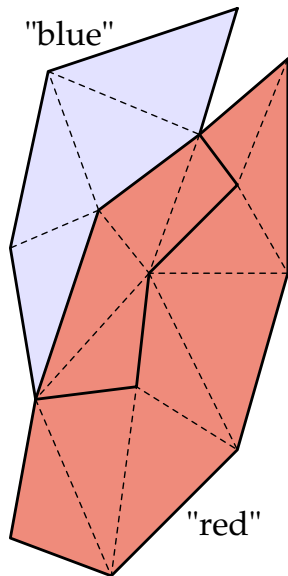
Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons

Triangle	Repair
A	"red"
B	"red"
C	"red"
D	"red"

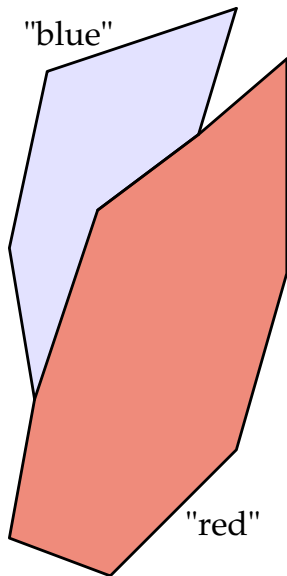
Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons



Our solution = constrained triangulation (CT)

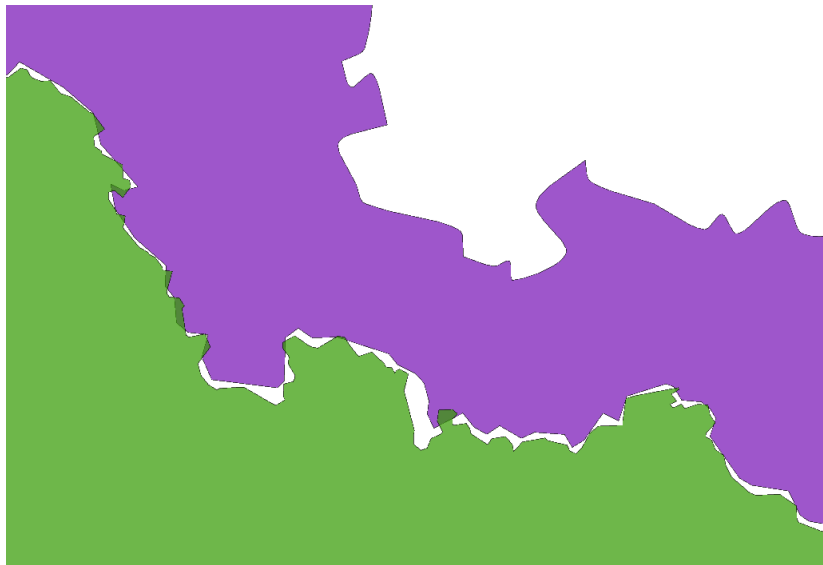
- 1 Construct CT of input polygons
- 2 Flag each triangle with its polygon / interior & exterior
- 3 Make each triangle have *exactly* one tag
- 4 Reconstruct polygons



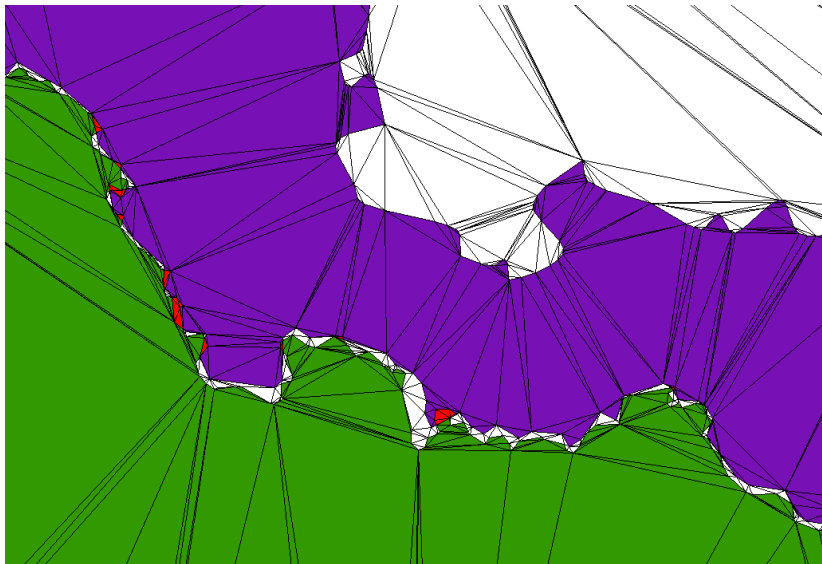
Local control: 6 different operators

Repair operation	Type	Criteria
Triangle by number of neighbours	Focal	The label present in the largest number of adjacent faces, overlaps included.
Triangle by absolute majority	Focal	Label present in two or more valid adjacent faces
Triangle by longest boundary	Focal	Label present along the longest portion of the boundary of the adjacent faces
Regions by longest boundary	Focal of zonal	Label present along the longest portion of the boundary of the adjacent faces
Regions by random neighbour	Focal of zonal	Random label from the adjacent faces
Triangle by priority list	Varies	Label that has the highest priority according to a predefined priority list

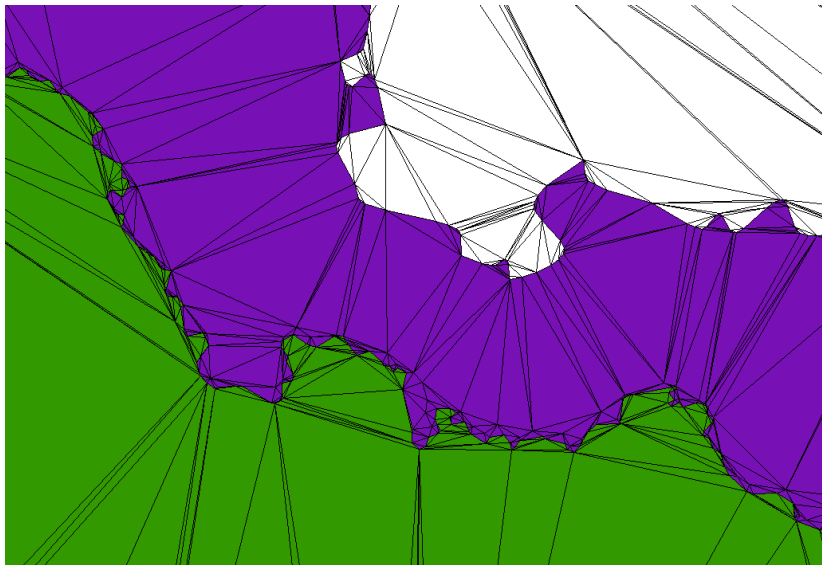
Local control: one concrete example



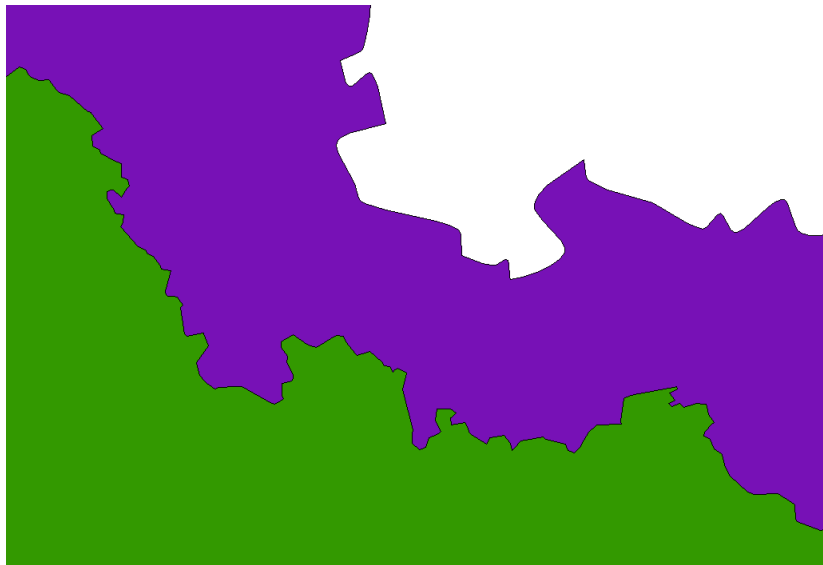
Local control: one concrete example



Local control: one concrete example



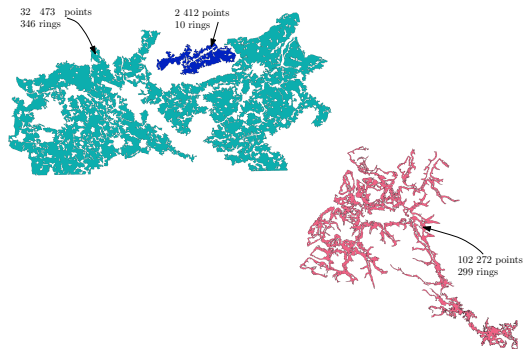
Local control: one concrete example



Local control: one concrete example

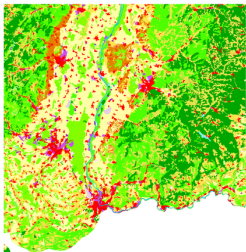


Experiments with big polygons: CORINE2006

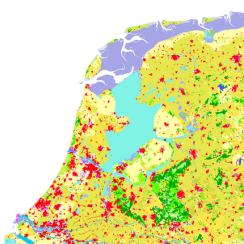


	points	rings	<i>prepair</i>	ST_MakeValid()
EU-47552	2 412	10	0.5s	0.8s
EU-47997	32 473	346	11.4s	314.0s
EU-180927	102 272	299	52.2s	740.2s

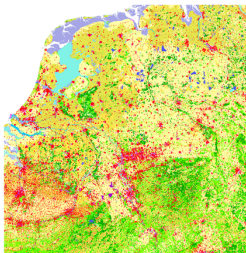
Experiments with large real-world datasets



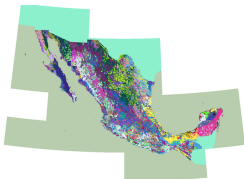
(a) E41N27



(b) 4tiles



(c) 16tiles



(d) Mexico

Experiments with large real-world datasets

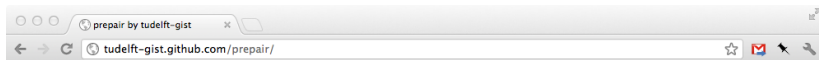
	# polygons	# pts	# pts largest polygon	avg # pts per polygon
E41N27	14 969	496 303	26 740	34
4tiles	4 984	365 702	16 961	75
16tiles	63 868	6 622 133	95 112	104
Mexico	26 866	4 181 354	117 736	156

Comparison with other GIS packages

	pprepair		ArcGIS		FME		GRASS	
	memory	time	memory	time	memory	time	memory	time
E41N27	145 MB	19s	145 MB	1m3s	158 MB	31s	59 MB	3m09s
4tiles	116 MB	17s	113 MB	37s	105 MB	31s	49 MB	53s
16tiles	1.45 GB	4m47s	crashes	–	636 MB	15m48s	crashes	–
Mexico	1.01 GB	3m31s	216 MB	>1d	264 MB	2m45s	408 MB	11m38s

The code is robust and freely available

- <http://tudelft-gist.github.com/pprepare>
- Uses OGR and CGAL
- BSD license → soon GPLv3



pprepare

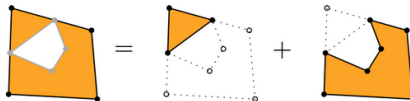
Automatic repair of single polygons

[View the Project on GitHub](#)
tudelft-gist/pprepare

Download
ZIP File

Download
TAR Ball

Fork On
GitHub



What is pprepair?

pprepare permits you to easily repair "broken" GIS polygons, and that according to the international standards ISO 19107. In brief, given a polygon stored in *WKT*, it *automatically* repairs it and gives you back a valid WKT. Automated repair methods can be considered as interpreting ambiguous or ill-defined polygons and giving a coherent and clearly defined output.

It performs more or less the same as the new PostGIS 2.0's function `ST_MakeValid()`, but is several order of magnitude faster, scales better to massive polygons, and predicting its behaviour is simple (so one can guess how her polygons will be repaired).

pprepare is based on a constrained triangulation, and [CGAL](#) and [OGR](#) are used.

Thanks for your attention

Ken Arroyo Ohori

`g.a.k.arroyoohori@tudelft.nl`

Hugo Ledoux

`h.ledoux@tudelft.nl`

Martijn Meijers

`b.m.meijers@tudelft.nl`

