

# Edge-matching polygons with a constrained triangulation

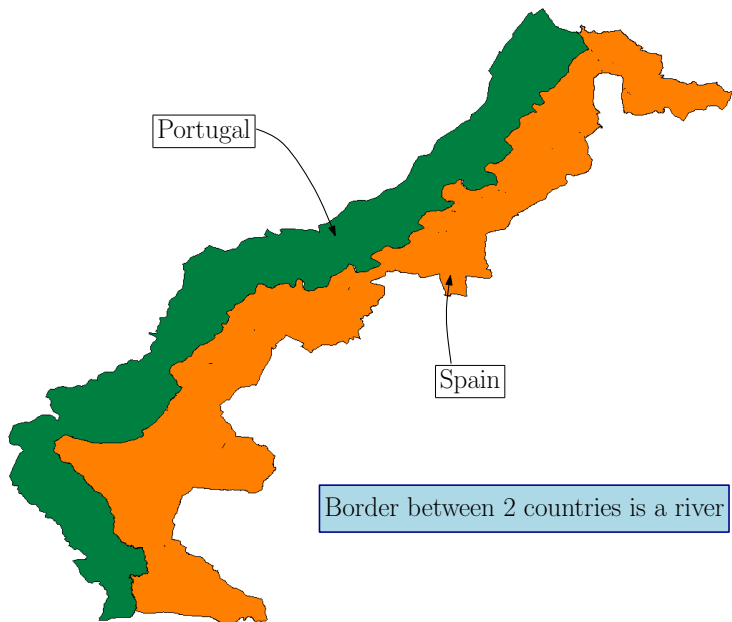
Hugo Ledoux    Ken Arroyo Ohori



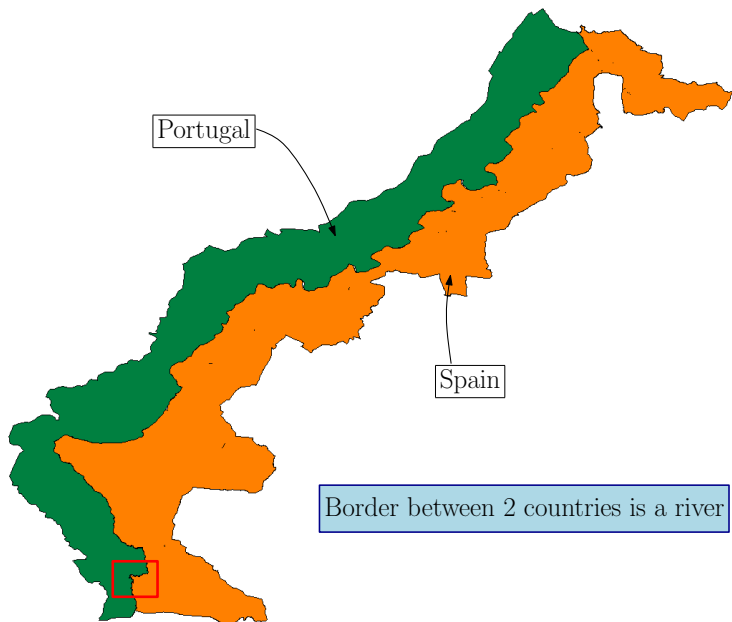
2011/01/26

GIS Ostrava 2011, Czech Republic

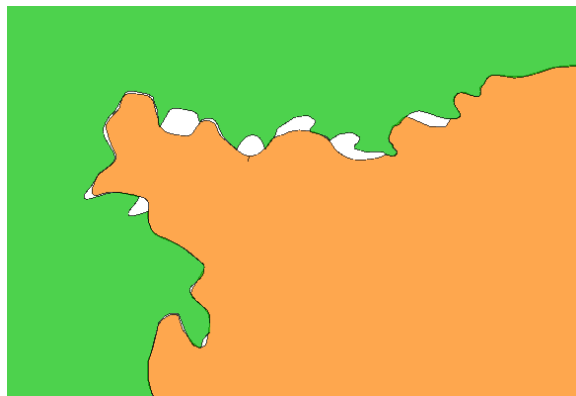
# Edge matching: ensuring the border is coherent



# Edge matching: ensuring the border is coherent

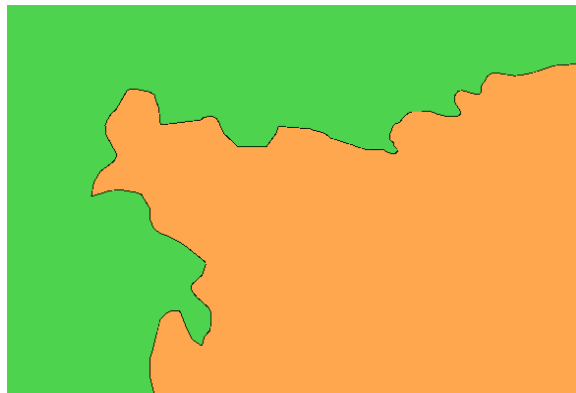


## Often the polygons do not “match”



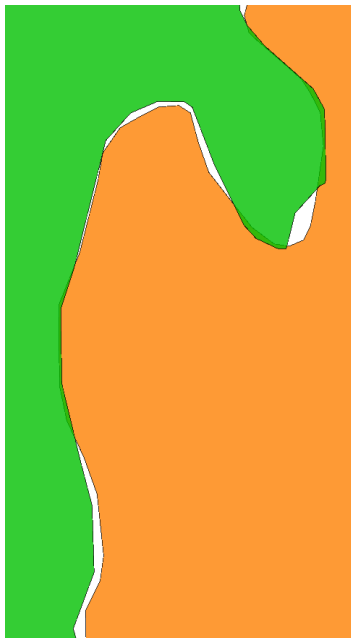
- 1 Diff instruments to collect data
- 2 Diff scales
- 3 Diff coordinate reference systems

## Often the polygons do not “match”



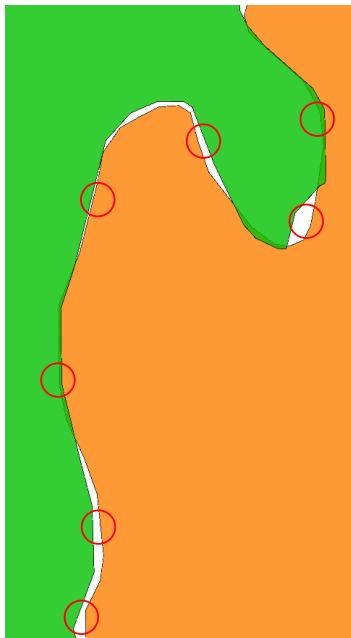
- 1 Diff instruments to collect data
- 2 Diff scales
- 3 Diff coordinate reference systems

## How edge matching is usually done: snapping



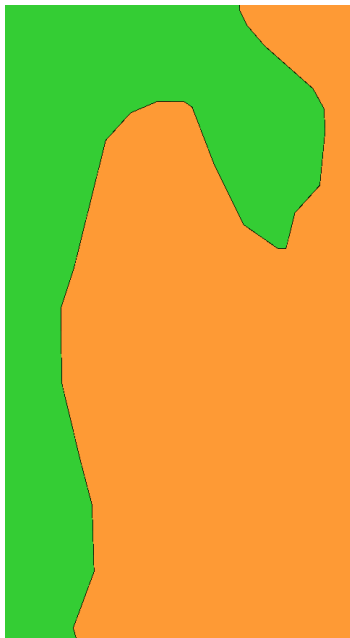
- Tolerance (*threshold*) is used for *snapping* vertices
- Tolerance based on scale of datasets
- Works fine for *simple* problems

## How edge matching is usually done: snapping



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

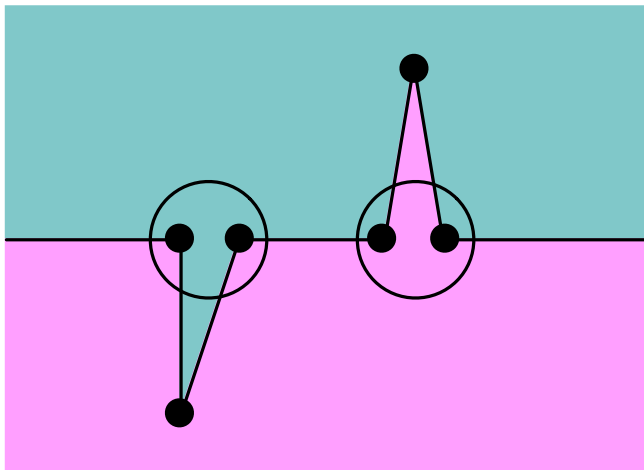
## How edge matching is usually done: snapping



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

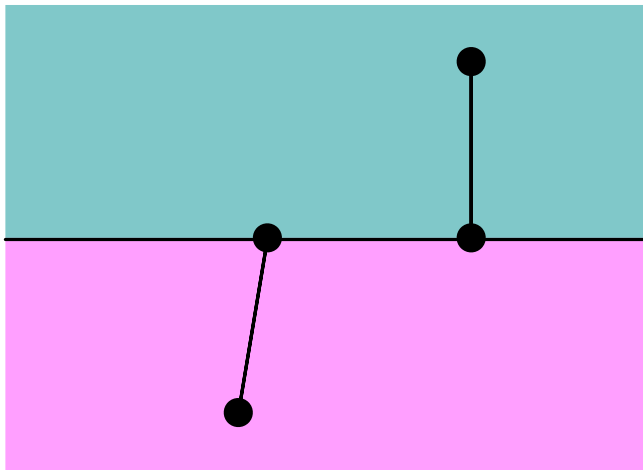


# Snapping is error-prone and “dangerous”



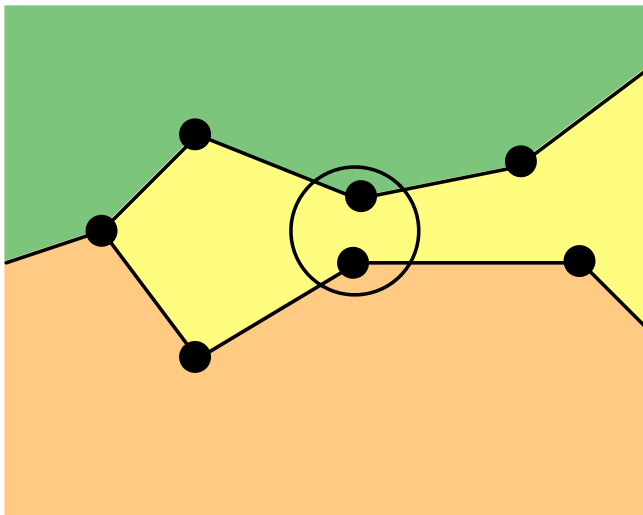
Spikes and punctures can create invalid polygons

# Snapping is error-prone and “dangerous”



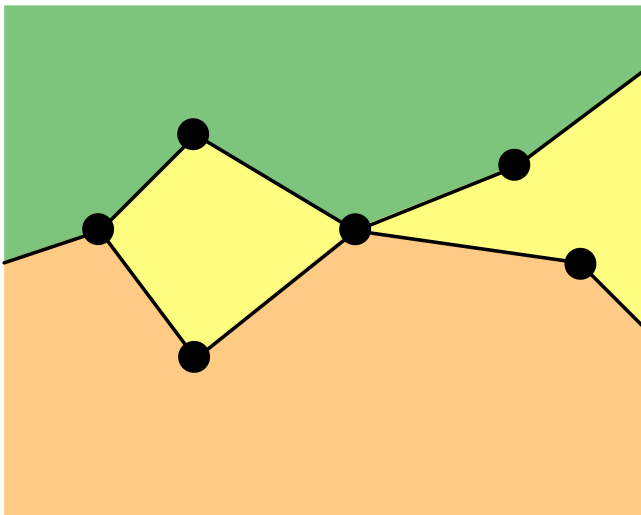
Spikes and punctures can create invalid polygons

# Snapping is error-prone and “dangerous”



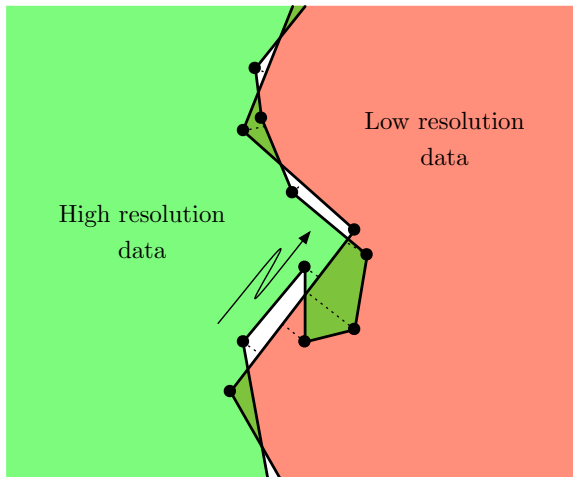
Splitting of polygons into several polygons

# Snapping is error-prone and “dangerous”



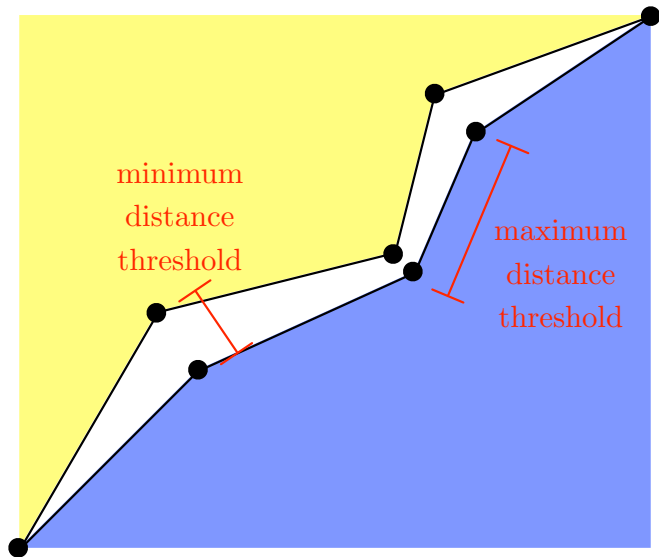
Splitting of polygons into several polygons

# Snapping is error-prone and “dangerous”



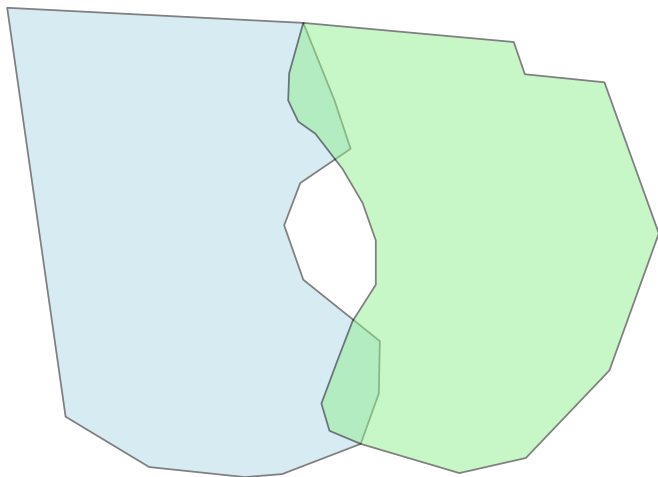
Topological invalid result

# The optimal threshold to snap might not exist



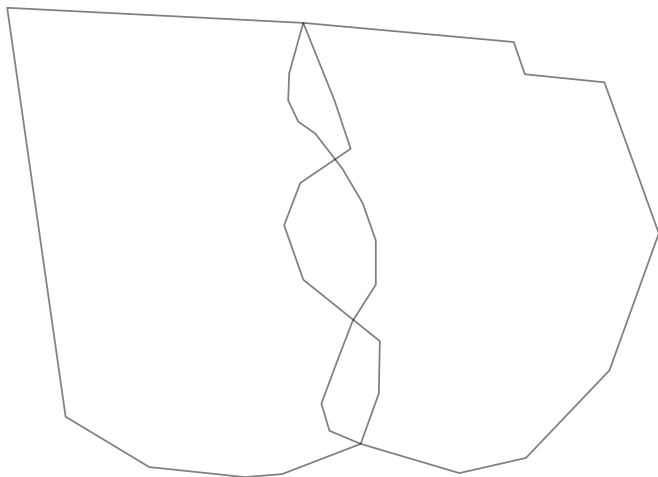
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



# Our solution = constrained triangulation (CT)

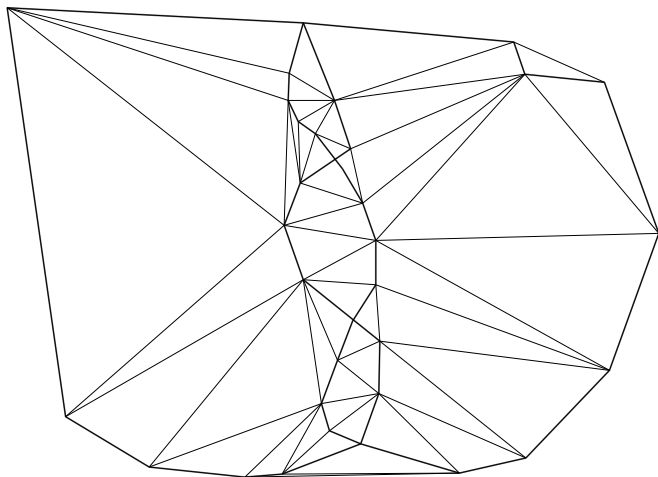
- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels





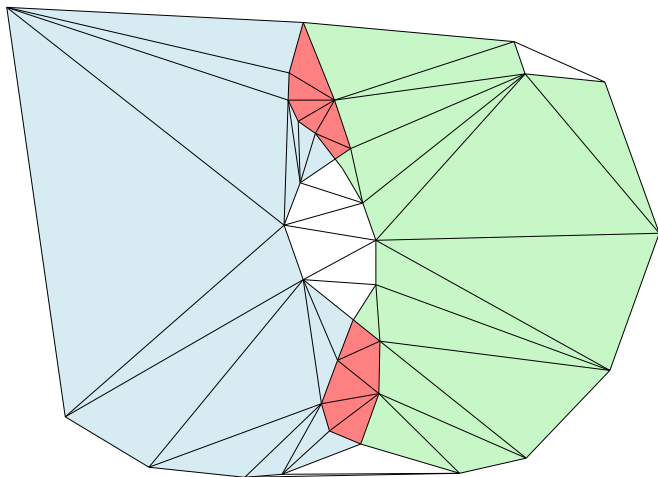
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



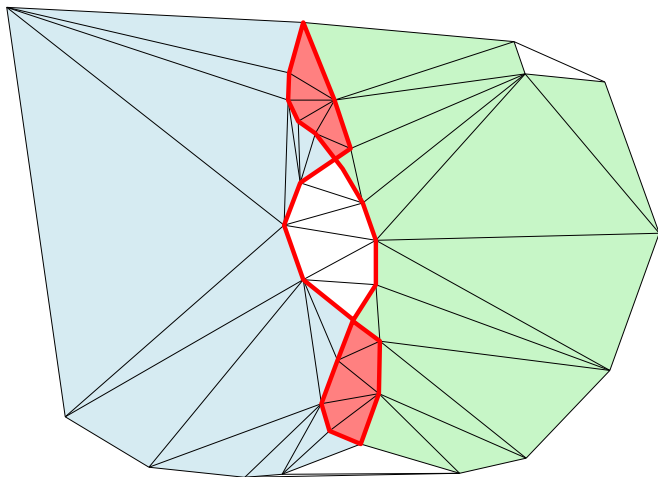
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



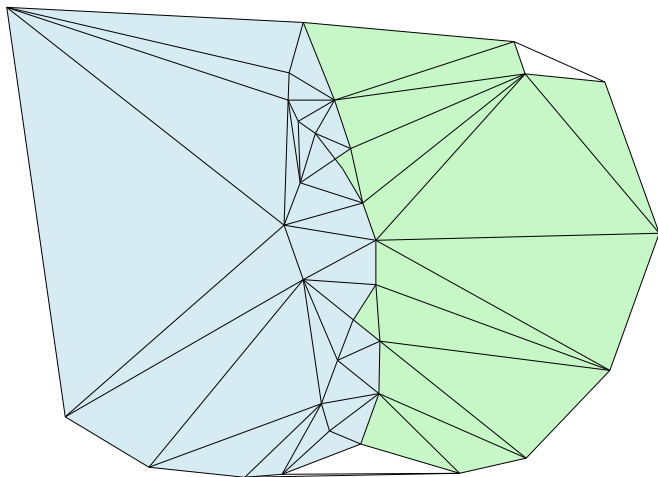
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



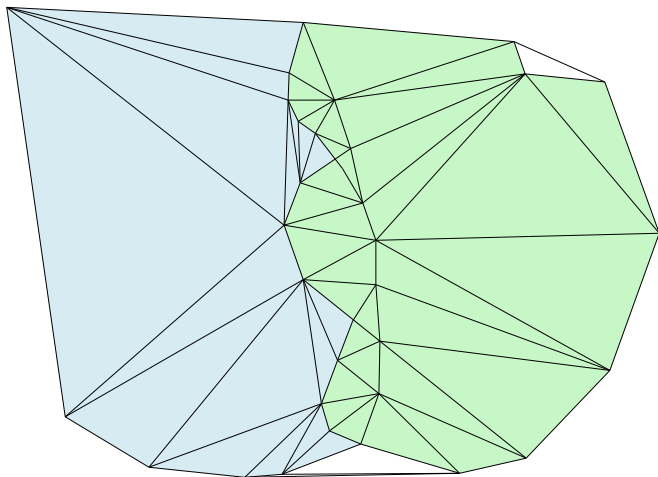
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



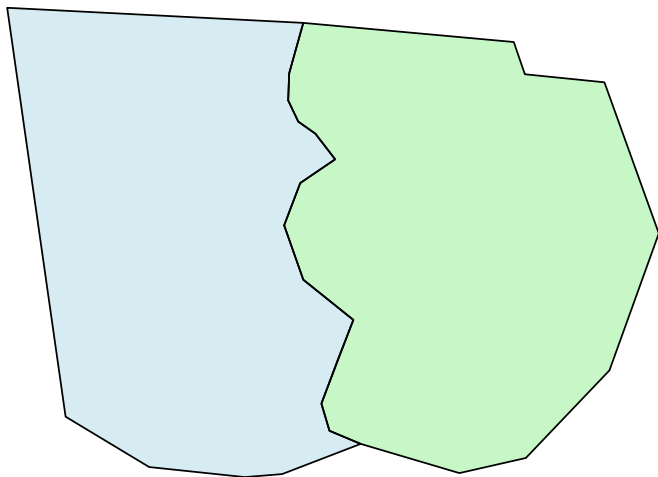
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



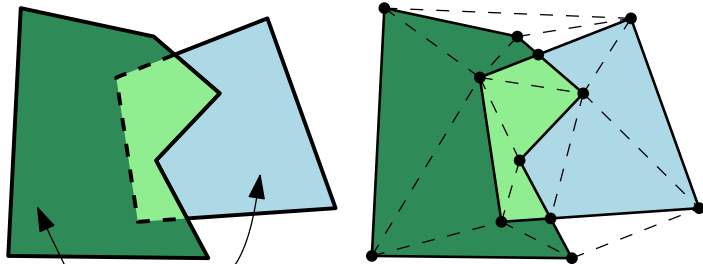
# Our solution = constrained triangulation (CT)

- 1 Construct CT of input polygons
- 2 Flag each triangle with label of its polygon
- 3 Problems = triangles with no label or  $> 1$  labels
- 4 Repair gaps/overlaps *locally* by changing labels



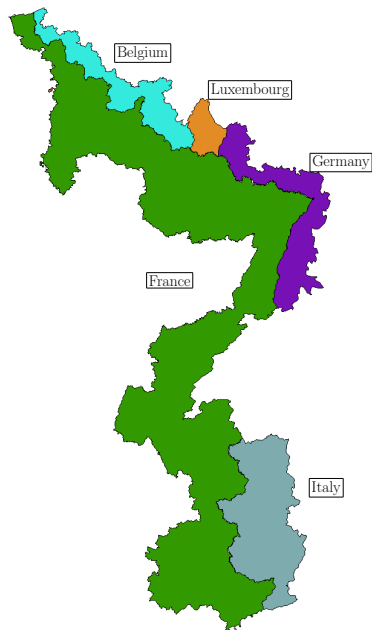
# Implementation

- 1 Fast implementation in C++, with CGAL and OGR
- 2 Open-source code, you can test it
- 3 Numerical and geometric robustness. Points do not move during processing.



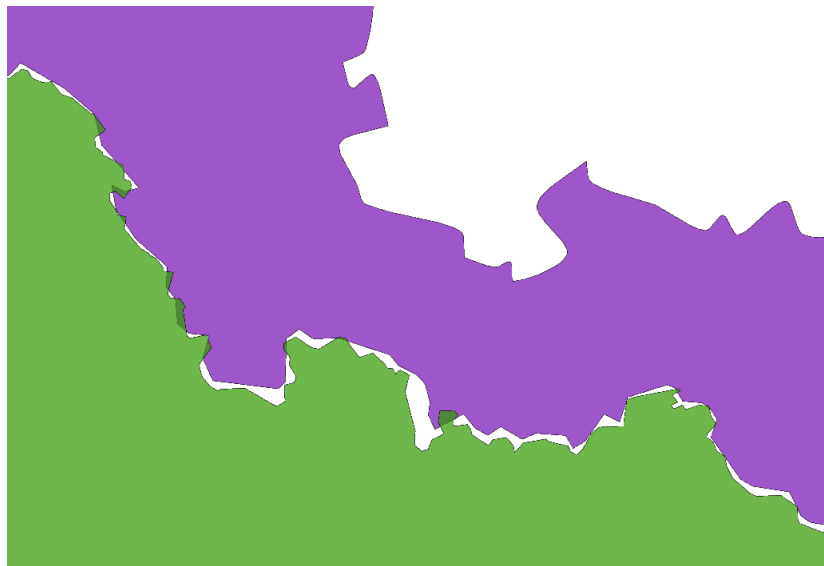
2 input polygons that overlap

# Experiments with real-world datasets

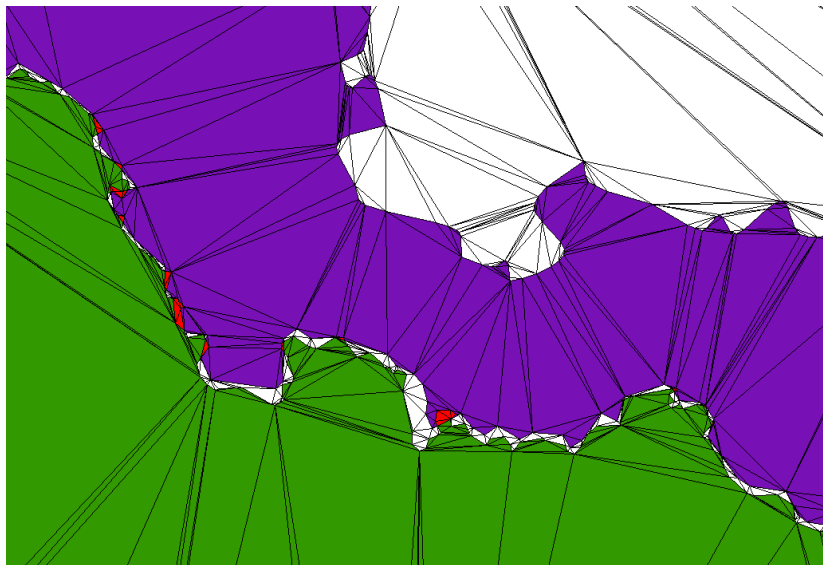




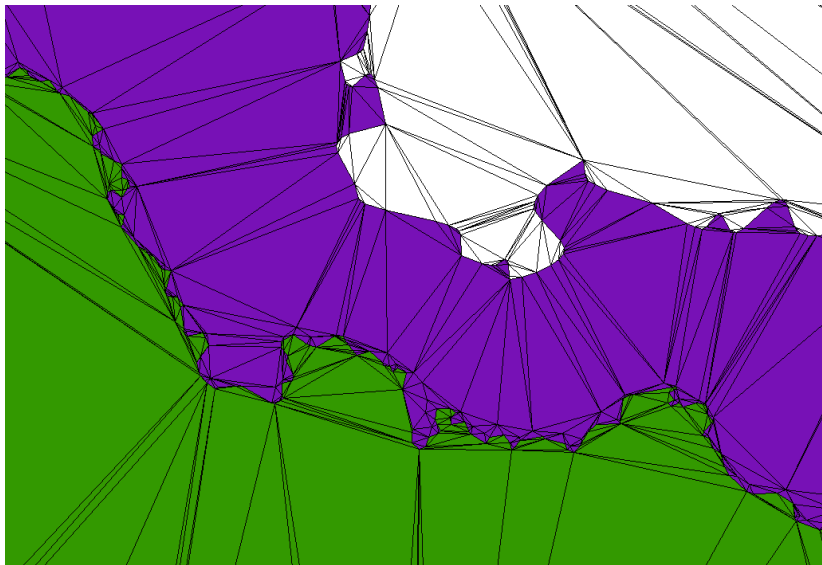
# Experiments with real-world datasets



# Experiments with real-world datasets



# Experiments with real-world datasets



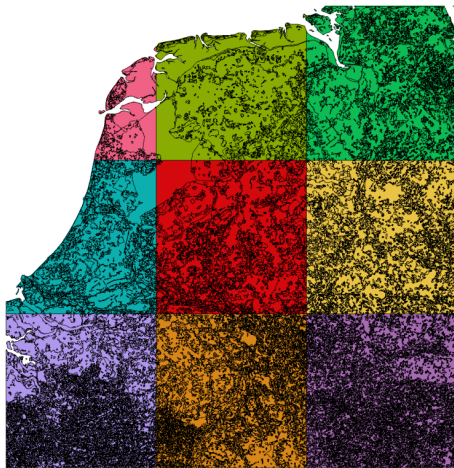
# Experiments with real-world datasets



# Experiments with real-world datasets



# Experiments with CORINE dataset



Can process around 40 tiles in  $< 1h$   
(around 120 000 polygons with 4GB main memory)

# Web Processing Service (WPS)

The screenshot shows a web browser window with the address bar containing the URL `scspruijt.dyndns.org/zoo/index.php?page=edgematch`. The page title is "Edgematching Shapefiles". Below the title, there are three tabs: "Upload", "Edgematch", and "Map", with "Edgematch" being the active tab. The main content area is divided into two columns: "Files to match" and "Available files".

**Files to match**

- b2.shp
- france.shp

**Available files**

- output52.shp
- output61.shp
- hugoreresults.shp
- output66.shp
- out\_random.shp
- test2.shp
- output62.shp
- output53.shp
- dep.shp
- out\_longest.shp
- output65.shp
- 100KME40N32.shp
- out\_priority.shp

# Advantages of a triangle-based approach

- Fast and robust (based on CGAL)
- No user-defined tolerance needs to be defined
- Local control of the the edge-matching process
- Results are guaranteed to be valid (geometrically and topologically)



Thanks for your attention

**Hugo Ledoux**

`h.ledoux@tudelft.nl`

**Ken Arroyo Ohori**

`kenohori@gmail.com`

