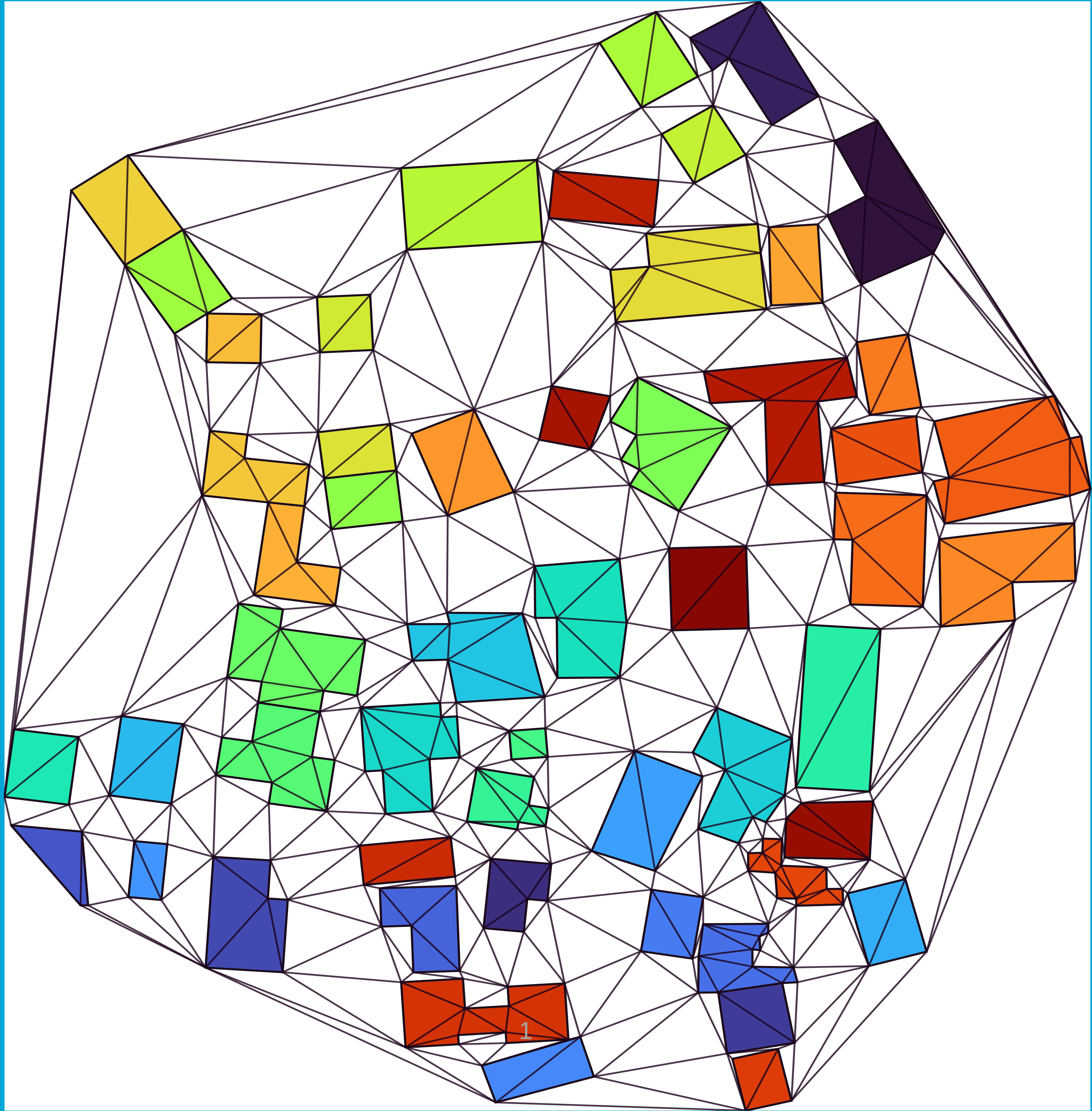
MSc thesis in Geomatics

Snap rounding polygons with a triangulation

Supervisors:
Hugo Ledoux & Ken Arroyo Ohori
Delegate:
Giorgio Agugiaro
Co-reader:
Martijn Meijers

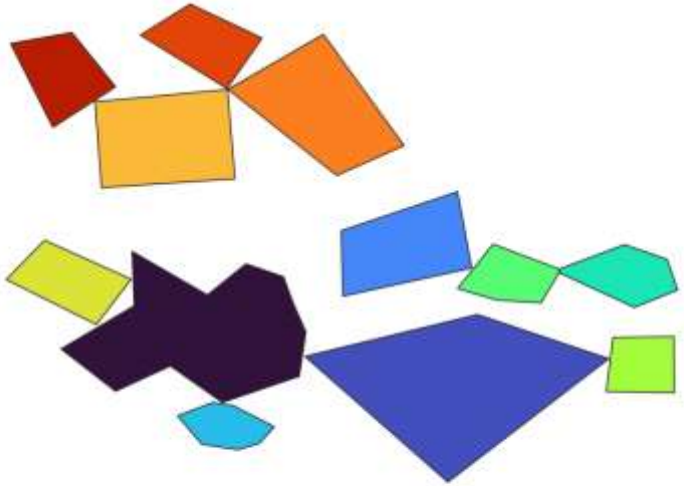
Fengyan Zhang
student #5462150

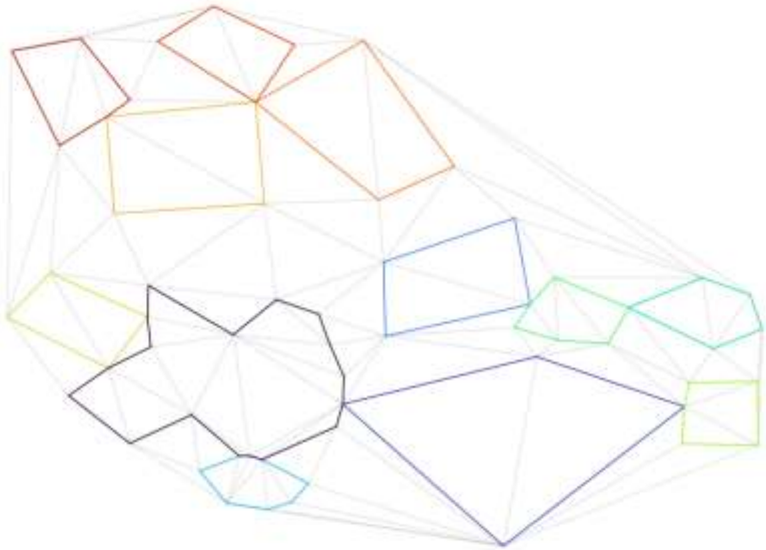
TU Delft



Content

- Introduction
- Methodology
- Implementation
- Results
- Conclusion & Future work





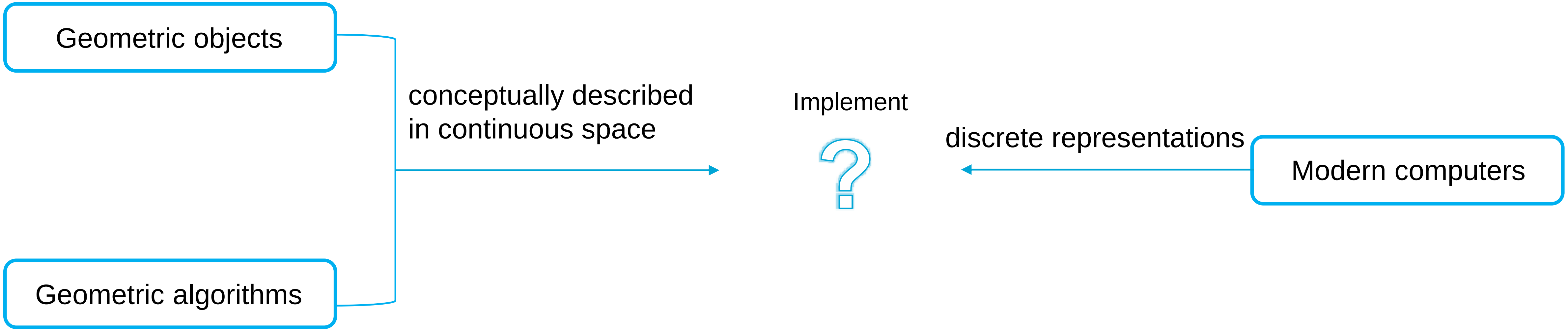
TU Delft

1. Introduction

TU Delft

Part 1.1 Research Motivation

Why do we need snap rounding?



Modern computers handles information that is represented in **discrete** and **finite** values. These values are typically expressed using binary digits, commonly known as bits, which can be either 0 or 1.

Part 1.1 Research Motivation

Implement geometric algorithms

exact arithmetic

- using data structures and algorithms that can handle numbers with **arbitrary precision**
- dynamically allocate memory as needed to represent numbers with the required precision.
- limited only by the **available memory** resources of the computer
- can be computationally more expensive

finite-precision arithmetic

- typically utilize floating-point arithmetic with **finite precision**
- limited by the number of bits (to store the significand and the exponent) available for representation.
- introduces the **round-off errors** and can lead to inaccuracies in computations.
- more efficient compared to exact arithmetic

Part 1.1 Research Motivation

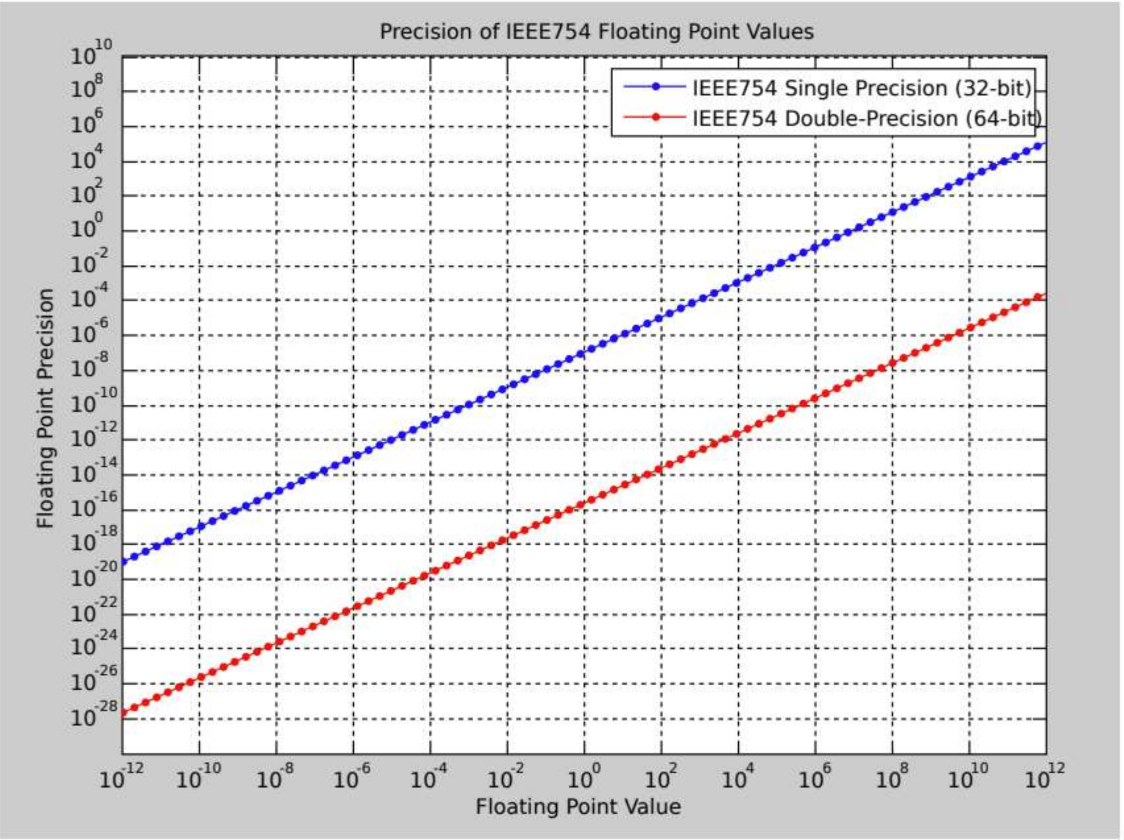


Figure 2.1: Precision of binary32 and binary64 in the range 10^{-12} to 10^{12}. Source: IEEE 754. (2023, April 11). In Wikipedia. https://en.wikipedia.org/wiki/IEEE_754. Author: By Vectorization: Alhadis - Own work based on: IEEE754.png by Ghennessey, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=87066073

$$\pm d.dd\ldots d \times \beta^e$$

significand, base, exponent

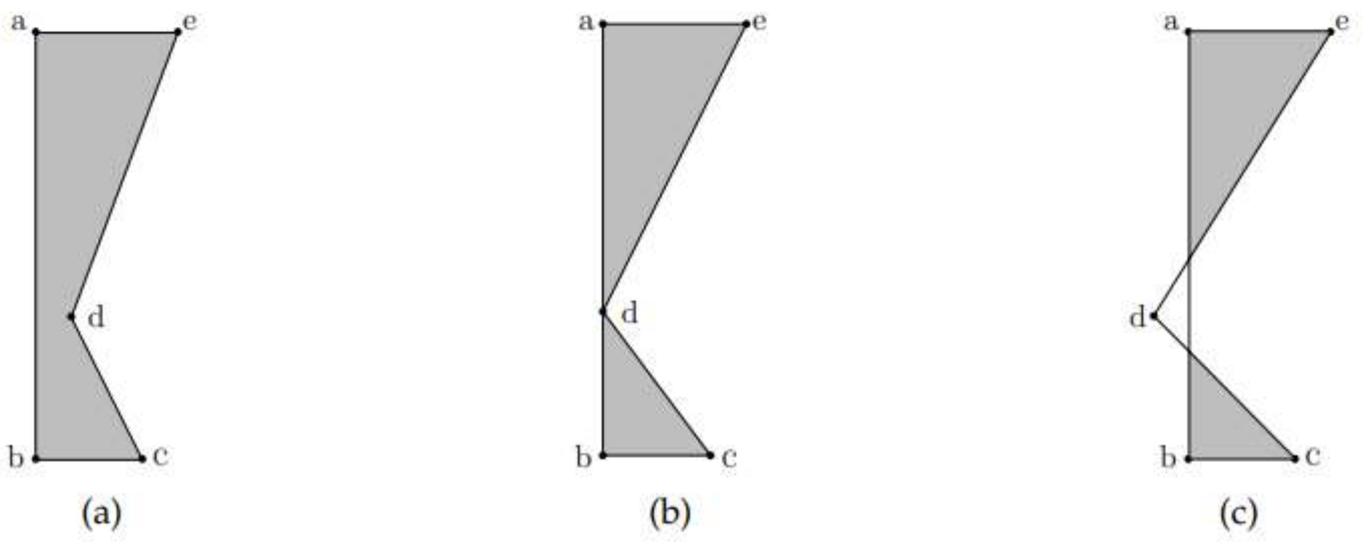
It includes p digits (p also represents the precision, the more digits after the decimal point are, the higher the precision is)

(8.57390526216, 4.469619220309) -> higher precision

(85739.0554312, 446961.2992009) -> lower precision

Part 1.1 Research Motivation

Precision issues of floating-point arithmetic (round off errors)



Coordinates shifting caused by round off errors could possibly make a valid polygon invalid.

(8.57390526216, 4.469619220309)

CRS_a

(85739.0554312, 446961.2992009)

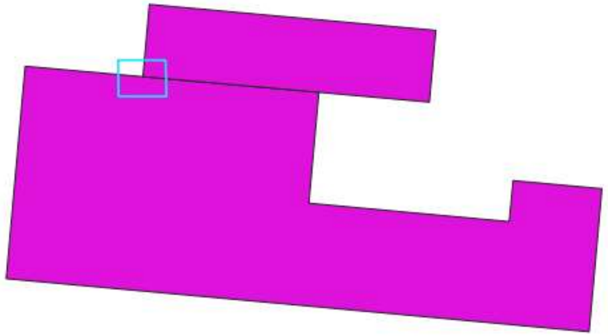
CRS_b

TU Delft

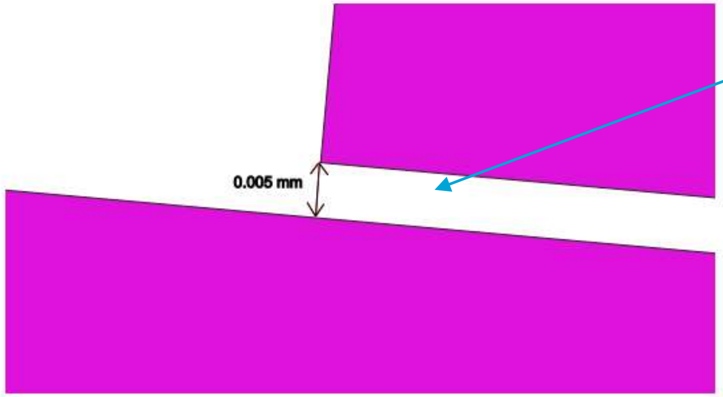
Part 1.1 Research Motivation

error-prone polygon arrangements in the datasets

Data acquisition, storage, exchange, manipulation

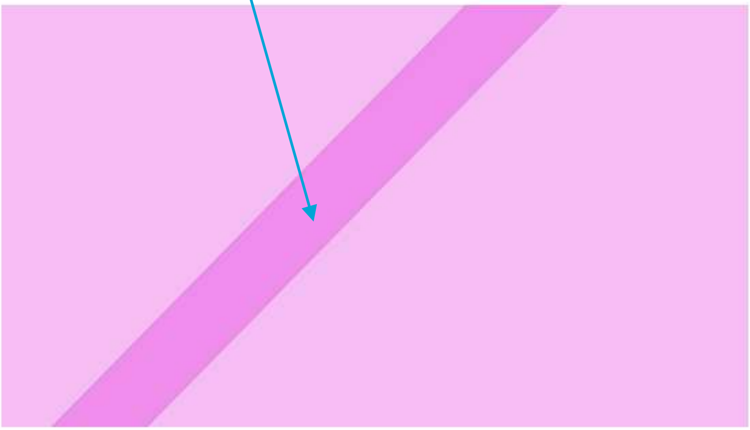


(a)



Would possibly cause problems of geometric algorithms, e.g. adjacency query.

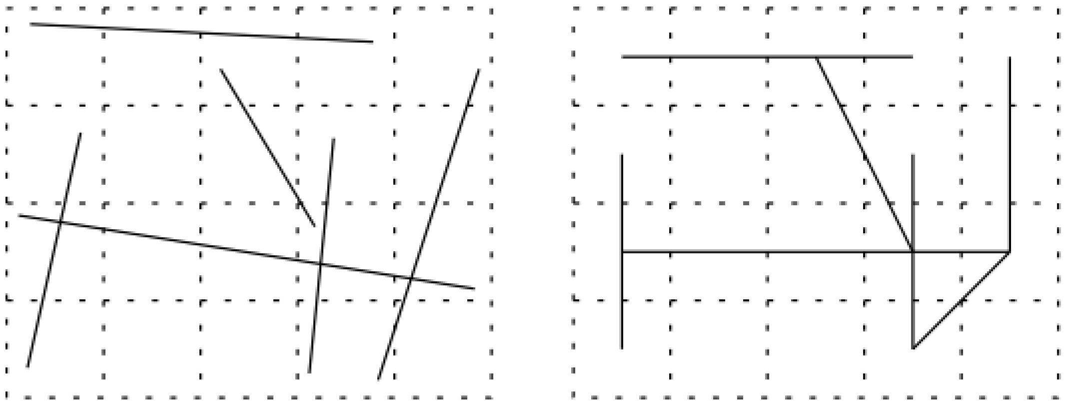
(a) two overlapping polygons with a scale of 1 : 200



(b) two overlapping polygons with a scale of 50 : 1

Part 1.1 Research Motivation

Snap rounding (SR): convert an arrangement of objects (e.g. line segments) from an arbitrary-precision representation to a fixed-precision representation (finite precision estimation)



Before (left) and after (right) snap rounding (SR) from Halperin and Packer (2002)*

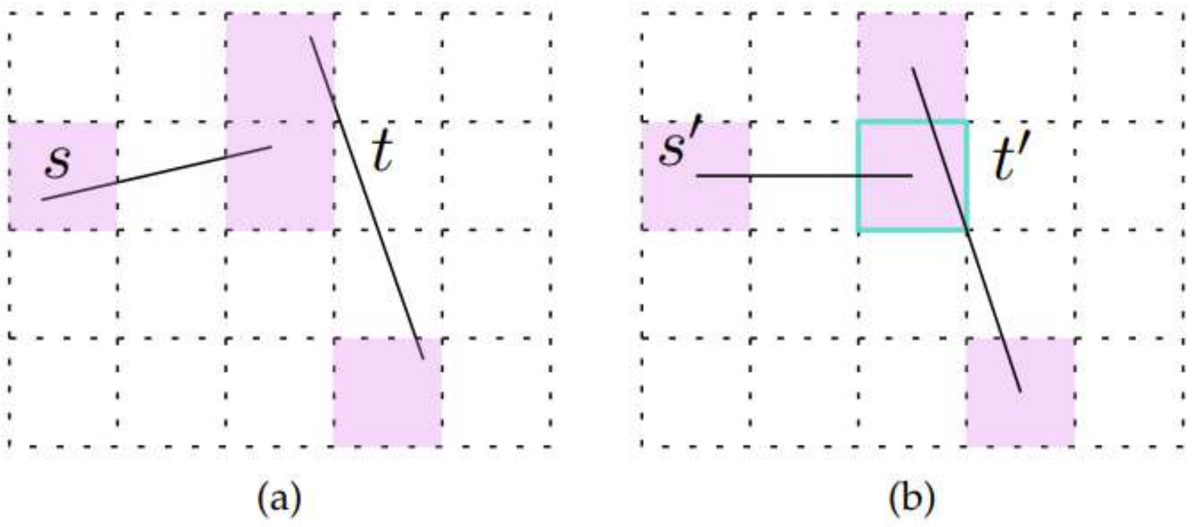
- the ending points are snapped to the center of the grid cells
- the length of cell is the resolution of the grid (tolerance)
- the resulting geometric objects are well-separated
- improves geometric robustness

TU Delft

* Halperin, D. and Packer, E. (2002). Iterated snap rounding. Computational Geometry, 23(2):209–225.

Part 1.1 Research Motivation

Limitations



A vertex becomes very close to a non-incident edge after (b) snap rounding. The figure is derived from Halperin and Packer (2002)*. The cells (pixels) highlighted in pink are *hot pixels* (containing vertices)

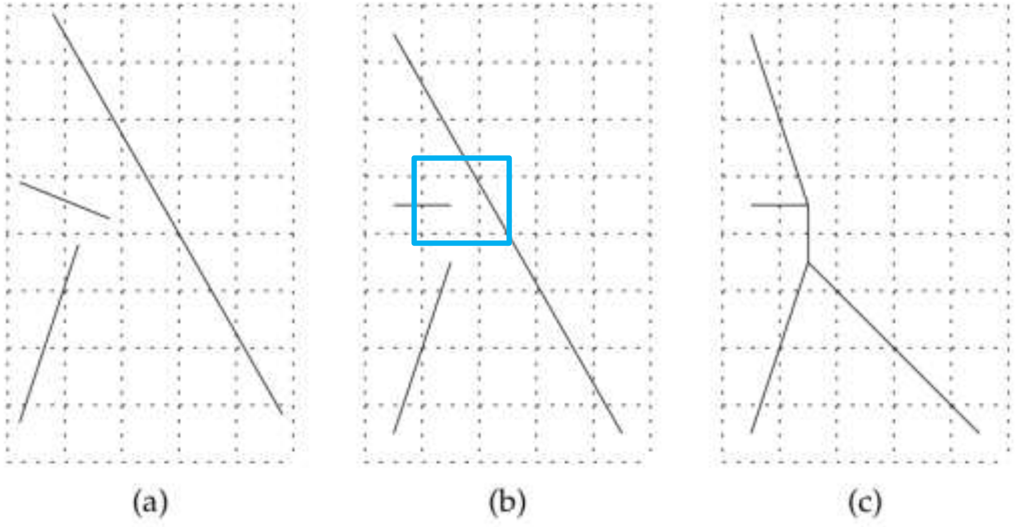
After SR, vertices can still be very close (smaller than the given tolerance) to non-incident edges.

TU Delft

* Halperin, D. and Packer, E. (2002). Iterated snap rounding. Computational Geometry, 23(2):209–225.

Part 1.1 Research Motivation

Variation of SR algorithms



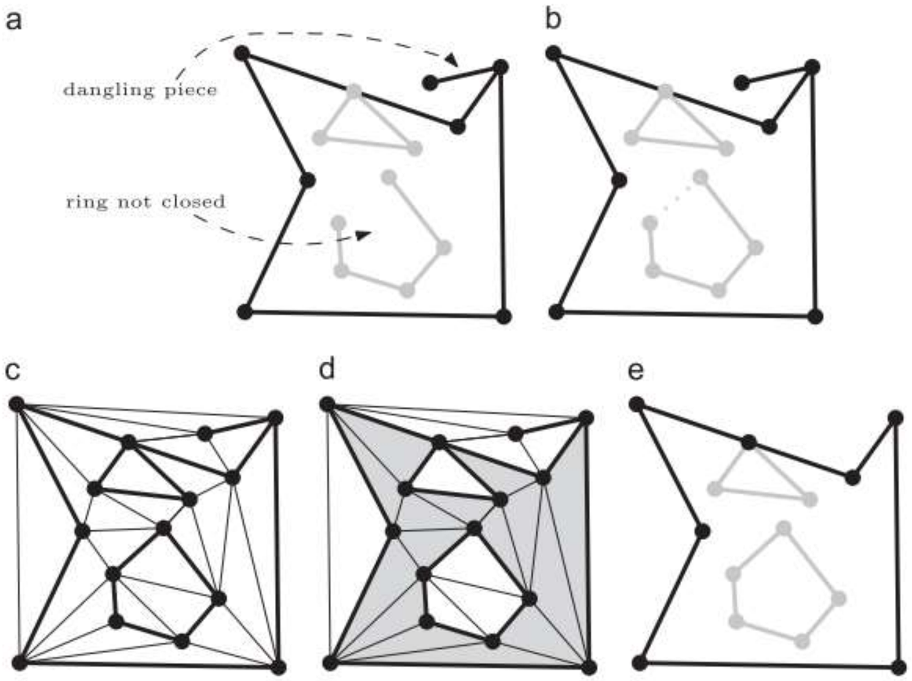
Iterated Snap Rounding (ISR). (a) An arrangement of segments before. (b) After Snap Rounding. (c) After Iterated Snap Rounding (Halperin and Packer (2002)[1]).

- **Iterated Snap Rounding (ISR)**
  the rounded counterpart of the input can be shifted or distorted away from its original position due to the iterative process
- **Iterated Snap Rounding with Bounded Drift (ISRBD)[2]**
  ISRBD ensures that the deviation between the input line segment arrangement and its rounded counterpart is less than a predefined value.
- **Stable Snap Rounding[3]**
  idempotent, improves the robustness and stability of ISRBD and ISR, however, it does not guarantee to eliminate near-degenerate cases while ISRBD does.
- **Snap Rounding with Restore (SRR)[4]**
  for the situation where a vertex is too near to a non-incident edge after rounding, the non-incident edge is moved in the opposite direction (this means it is being moved back towards its original location) instead of performing a SR operation.

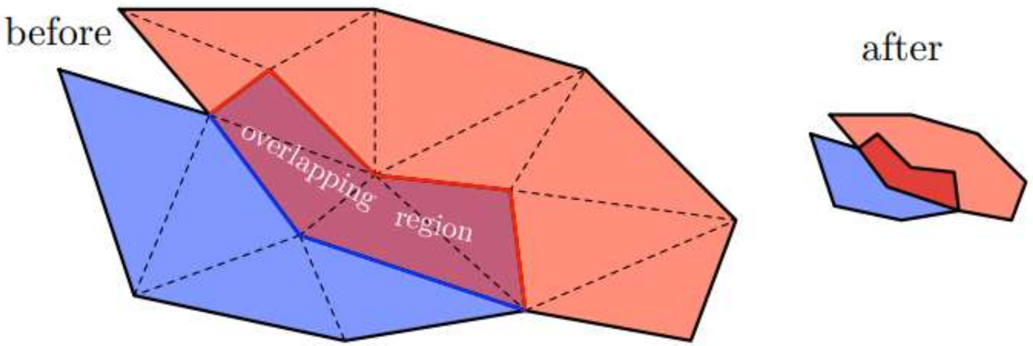
[1] Halperin, D. and Packer, E. (2002). Iterated snap rounding. Computational Geometry, 23(2):209–225.
[2] Packer, E. (2006). Iterated snap rounding with bounded drift. In Proceedings of the twentysecond annual symposium on Computational geometry, pages 367–37.
[3] Hershberger, J. (2011). Stable snap rounding. In Proceedings of the twenty-seventh annual symposium on Computational geometry, pages 197–206.
[4] Belussi, A., Migliorini, S., Negri, M., and Pelagatti, G. (2016). Snap rounding with restore: An algorithm for producing robust geometric datasets. ACM Transactions on Spatial Algorithms and Systems (TSAS), 2(1):1–36.

Part 1.1 Research Motivation

Fix geometries using a constrained triangulation (CT)



Repair polygons based on a constrained triangulation from Ledoux et al., 2012[1]



Repair a planar partition based on a constrained triangulation, from，Ohori et al, 2012[2]

[1] Ledoux, H., Arroyo Ohori, K., and Meijers, M. (2012). Automatically repairing invalid polygons with a constrained triangulation. In Proceedings of the AGILE 2012 International Conference, April 2012, Avignon, France, pp. 13-18. Agile.
[2] Ohori, K. A., Ledoux, H., and Meijers, M. (2012). Validation and automatic repair of planar partitions using a constrained triangulation. Photogrammetrie-FernerkundungGeoinformation, 5(10):613–630.

Part 1.2 Research Objectives

Snap rounding + Constrained triangulation

↓

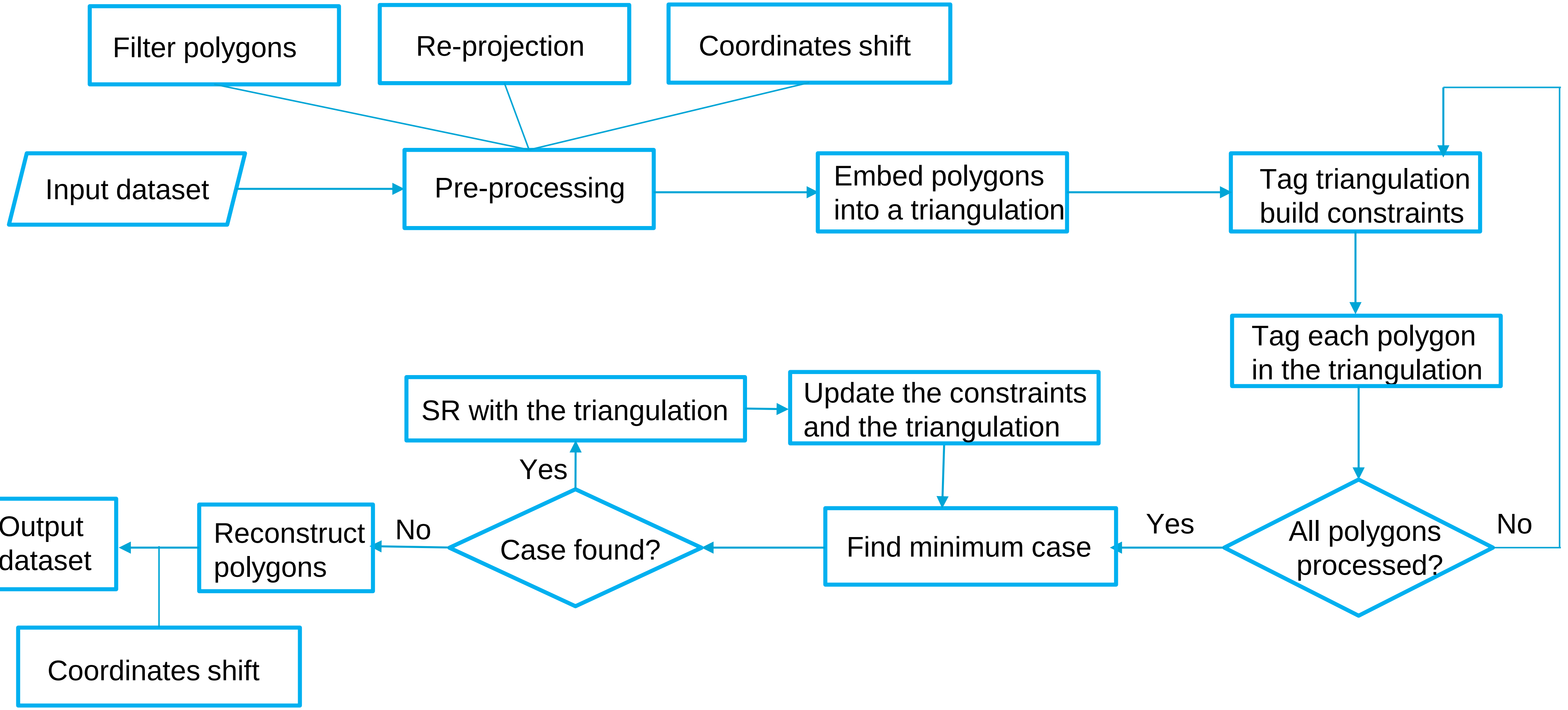
2-dimensional polygons ?

How can a constrained triangulation (CT) be used as a supporting data structure to robustly perform snap rounding of polygons in 2- dimensional space?

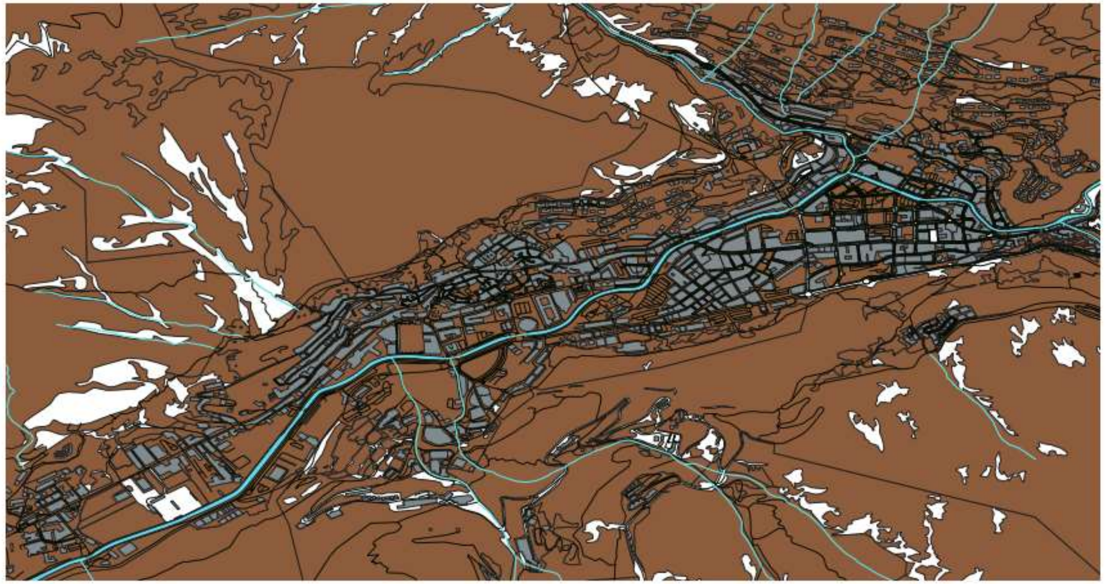
- How to integrate a CT with the input polygons (possibly with interior holes)?
- How to link the triangulation with the original polygons? If the triangulation is modified, how to update the changes to the polygons?
- Polygons usually have attributes attached (e.g. polygon id, area of a polygon), how to preserve them in a proper way during the snap rounding process?
- Given a certain threshold, there may exist several snap rounding cases (e.g. multiple polygonal vertices and boundaries), what is the most reasonable order when snap rounding is being performed?
- How to measure and evaluate the distortions of the polygons before and after snap rounding?

2. Methodology

Part 2.1 Overview



Part 2.1 Pre-processing



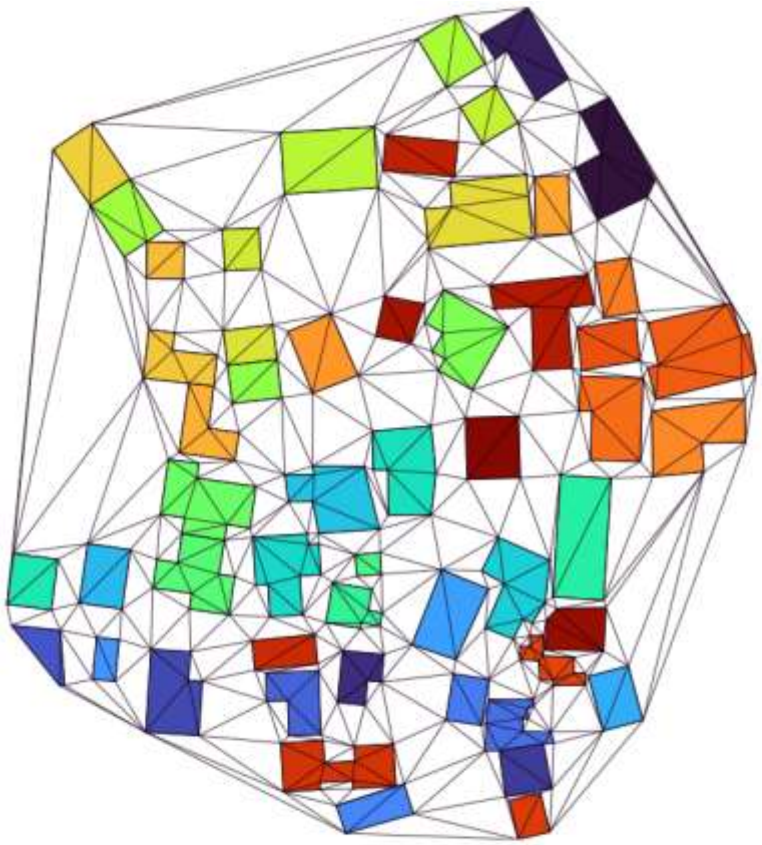
An excerpt of Andorra dataset. Buildings are depicted in grey, landuse is shown in brown, water body and waterways are in blue, the black straight lines represent the roads and railways (data source: GeoFabrik*).

Input polygons
with re-projected XY coordinates

TU Delft

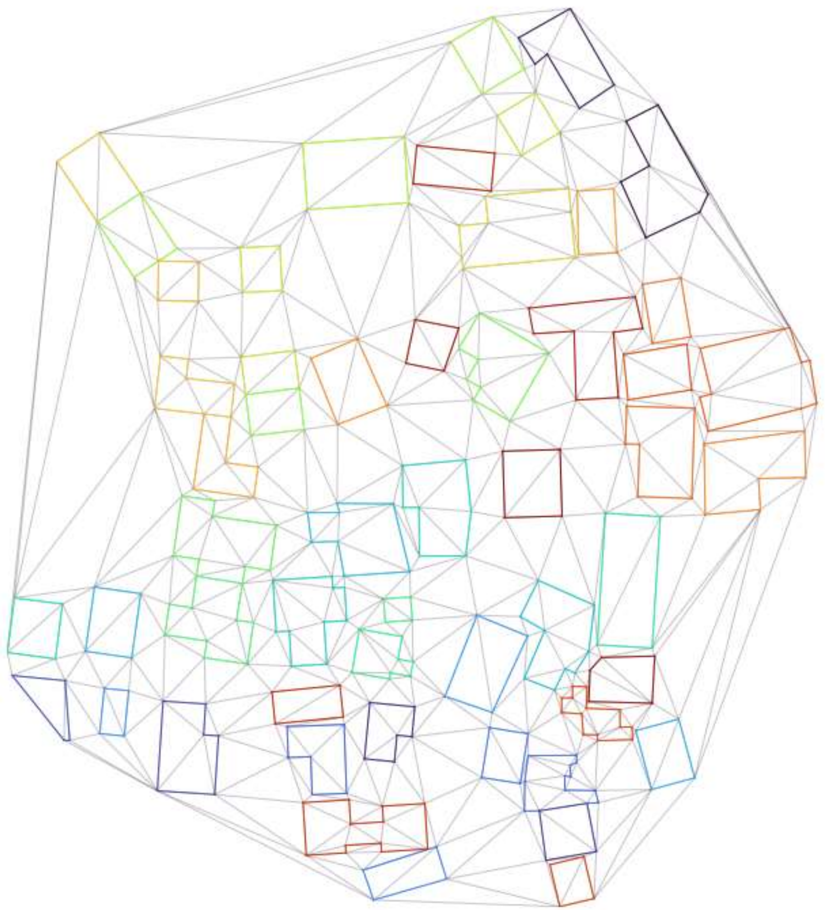
* Geofabrik: Download server for openstreetmap data. Web Based Download Application: http://download.geofabrik.de/. Last checked on April 2023.

Part 2.2 Tag the Triangulation



Embedded polygons into a triangulation (CT)

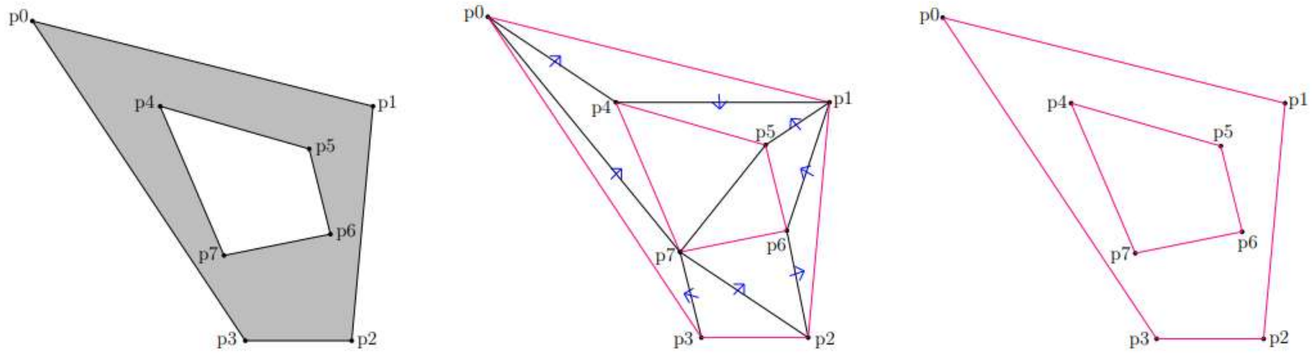
Boundaries are used as constrained edges



Constructed constraints (colorful line segments) will be stored in a separate container.

Part 2.2 Tag the Triangulation

valid input

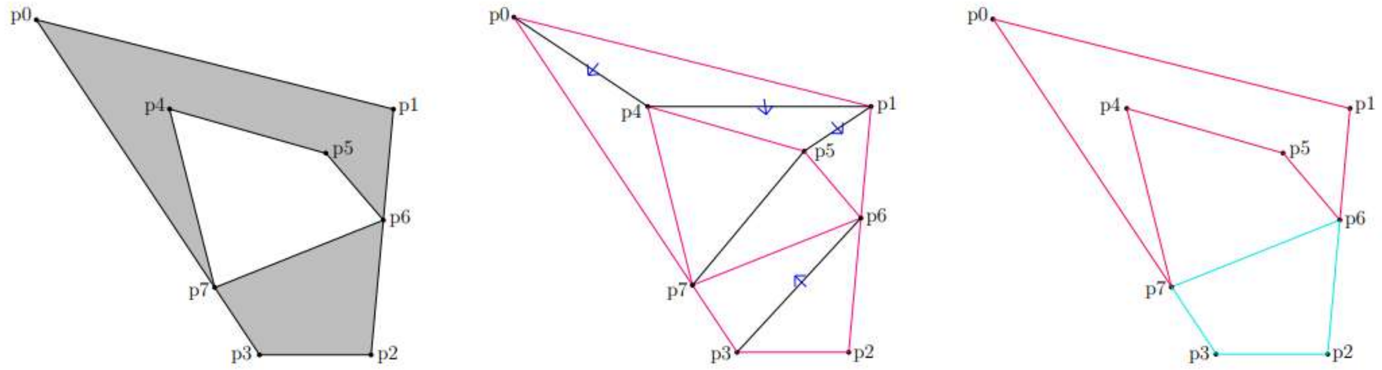


(a) input polygon (b) tagging process (c) tagging result

Breadth-first search (BFS)

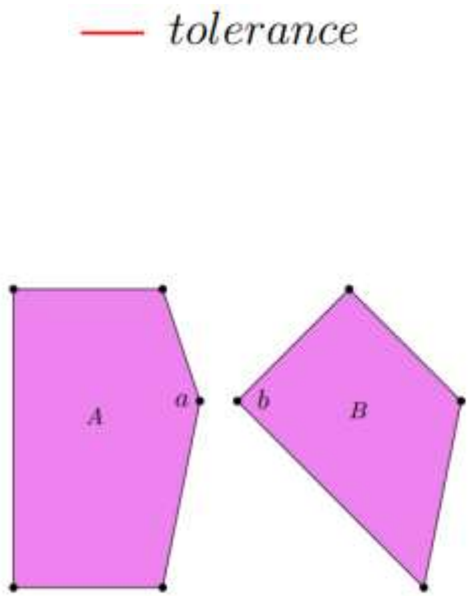
Capable of handling overlap area

faulty input

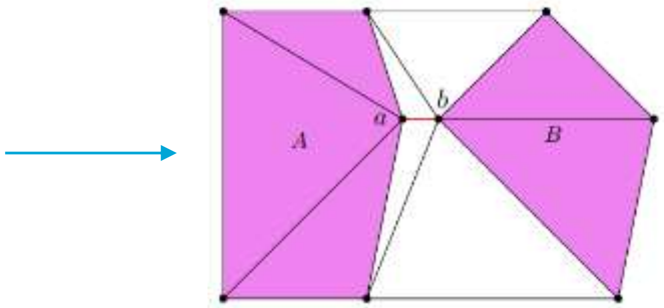


(a) input polygon (b) tagging process (c) tagging result

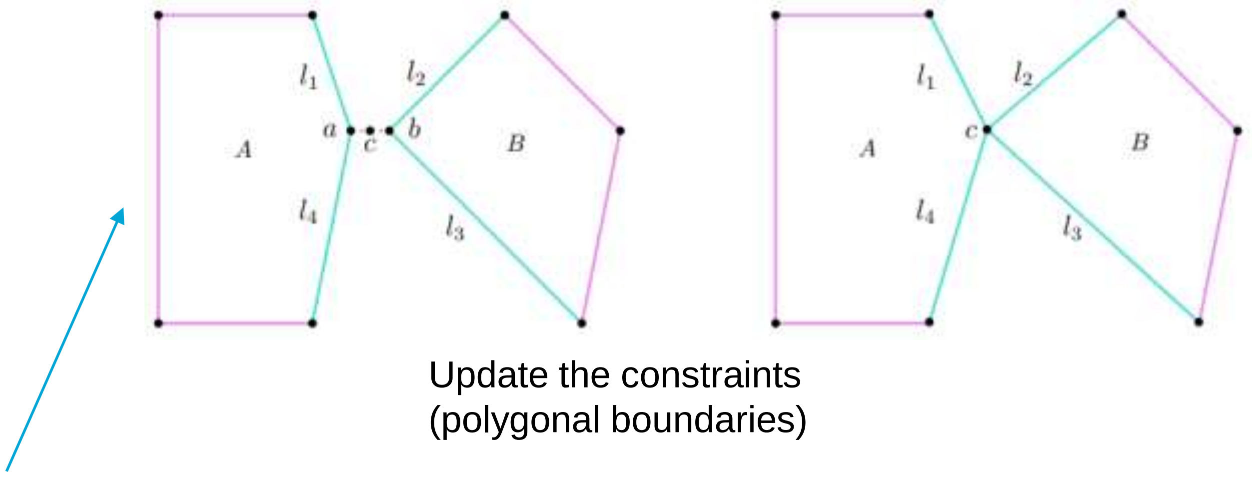
Part 2.3 SR – Close Polygonal Vertices



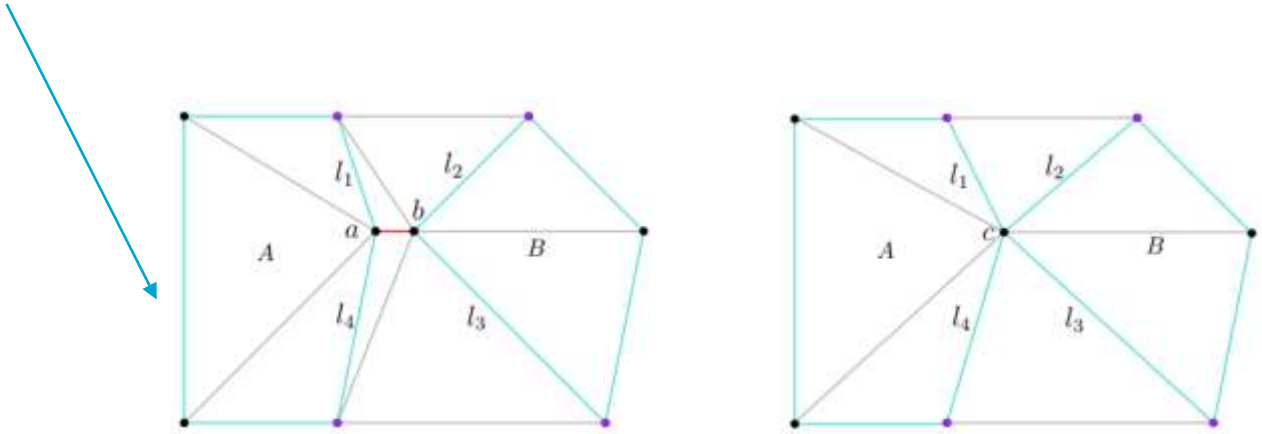
Input polygons



Embed into a CT, identify the close polygonal vertices

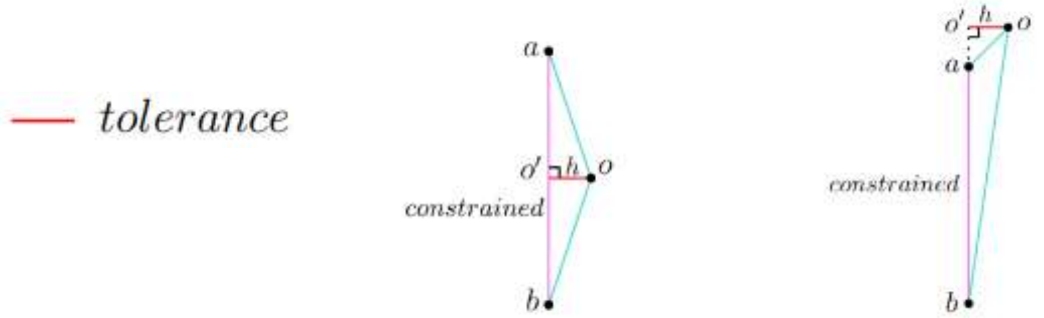


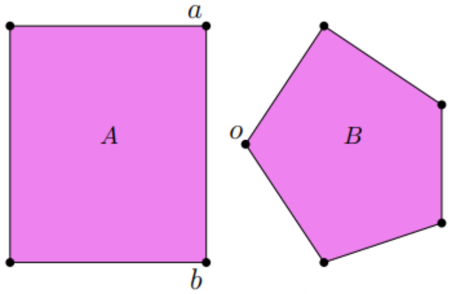
Update the constraints (polygonal boundaries)



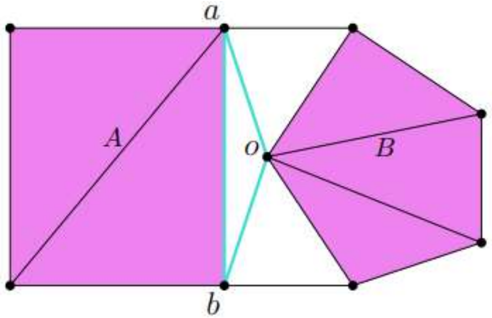
Update the CT

Part 2.3 SR – Close Polygonal Vertex and Boundary

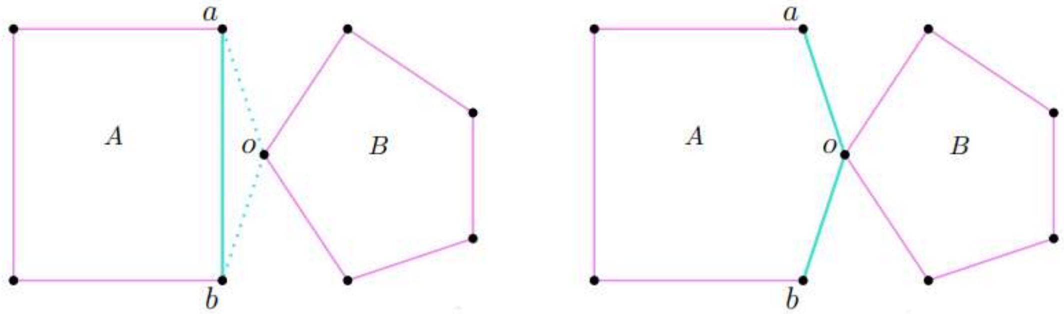




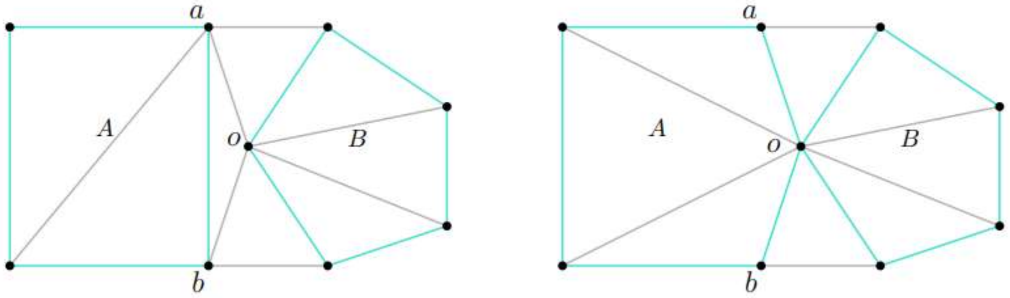
Input polygons



Embed into a CT, identify the close polygonal vertex and boundary



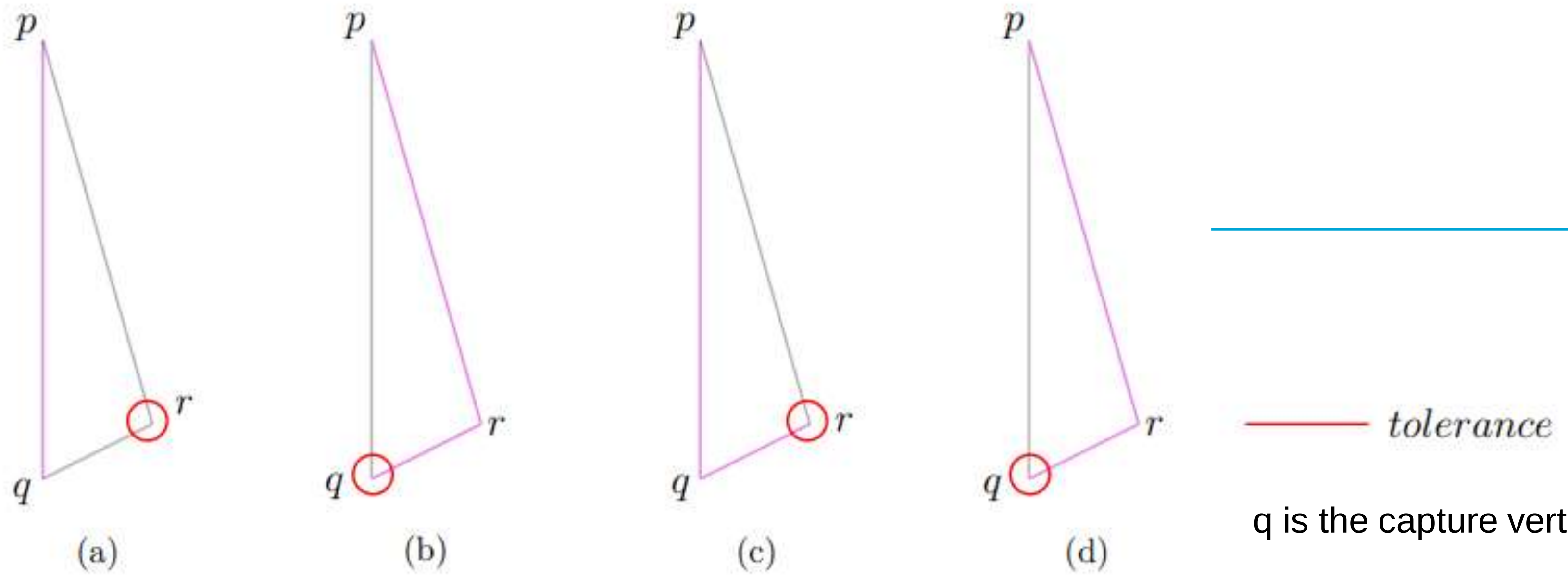
Update the constraints (polygonal boundaries)



Update the CT

Part 2.3 SR – Close Polygonal Vertex and Boundary

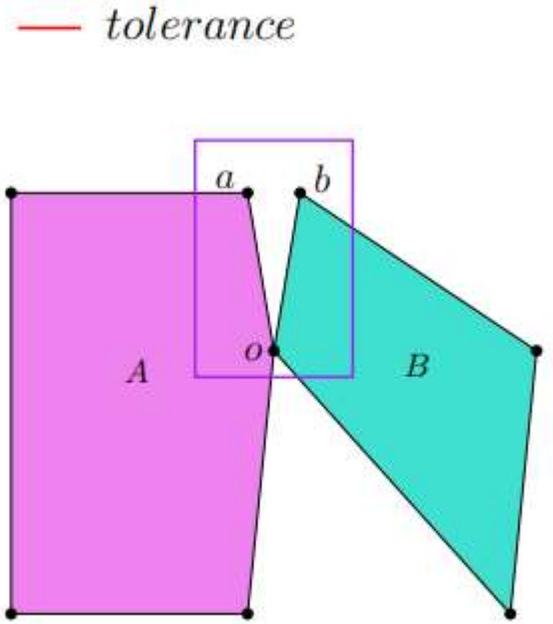
avoid infinite loops



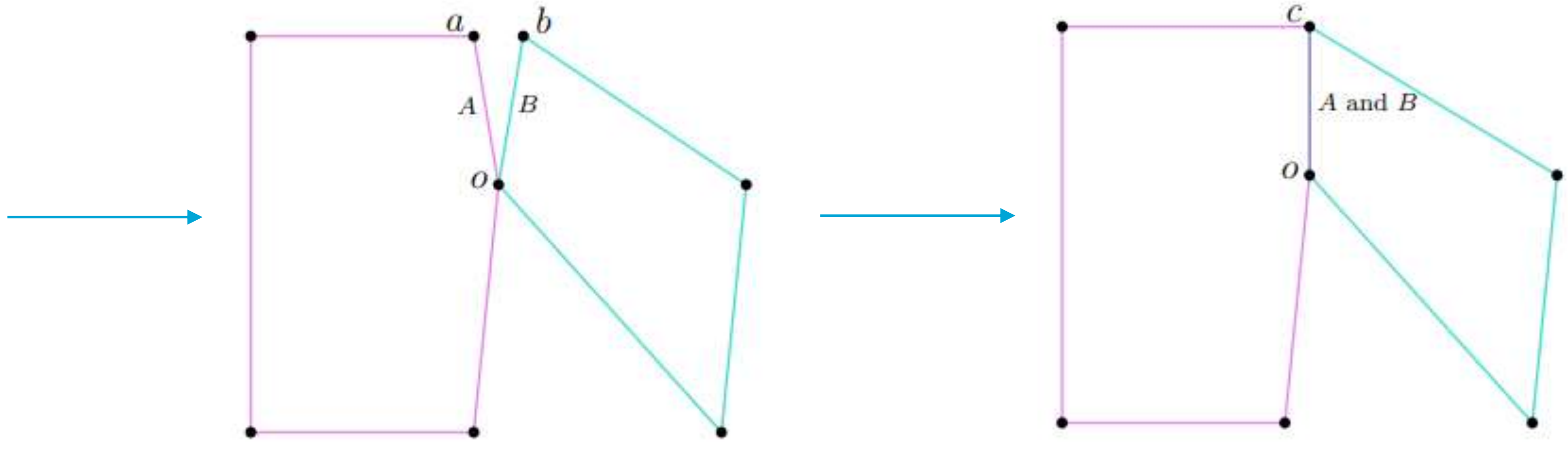
In this case vertex q and r should be snapped

q is the capture vertex

Part 2.4 Resolve Redundancies



When introducing new constraints, there will be two constraints with different tags.

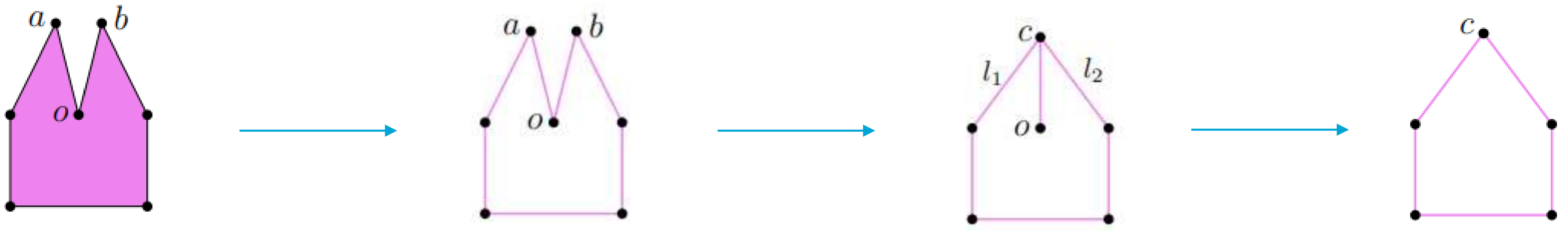


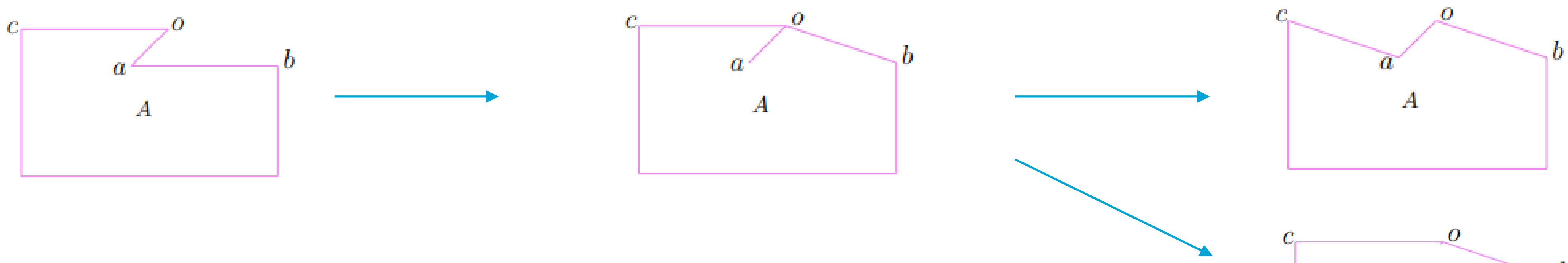
Merge the id information if there is already a constraint.

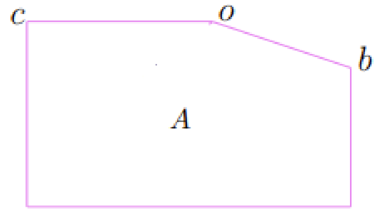
Part 2.5 Remove Dangling Elements (Optional)

tolerance

Can be handled in the polygon reconstruction but would possibly cause cascading effects, e.g. the dangling element requires SR again.

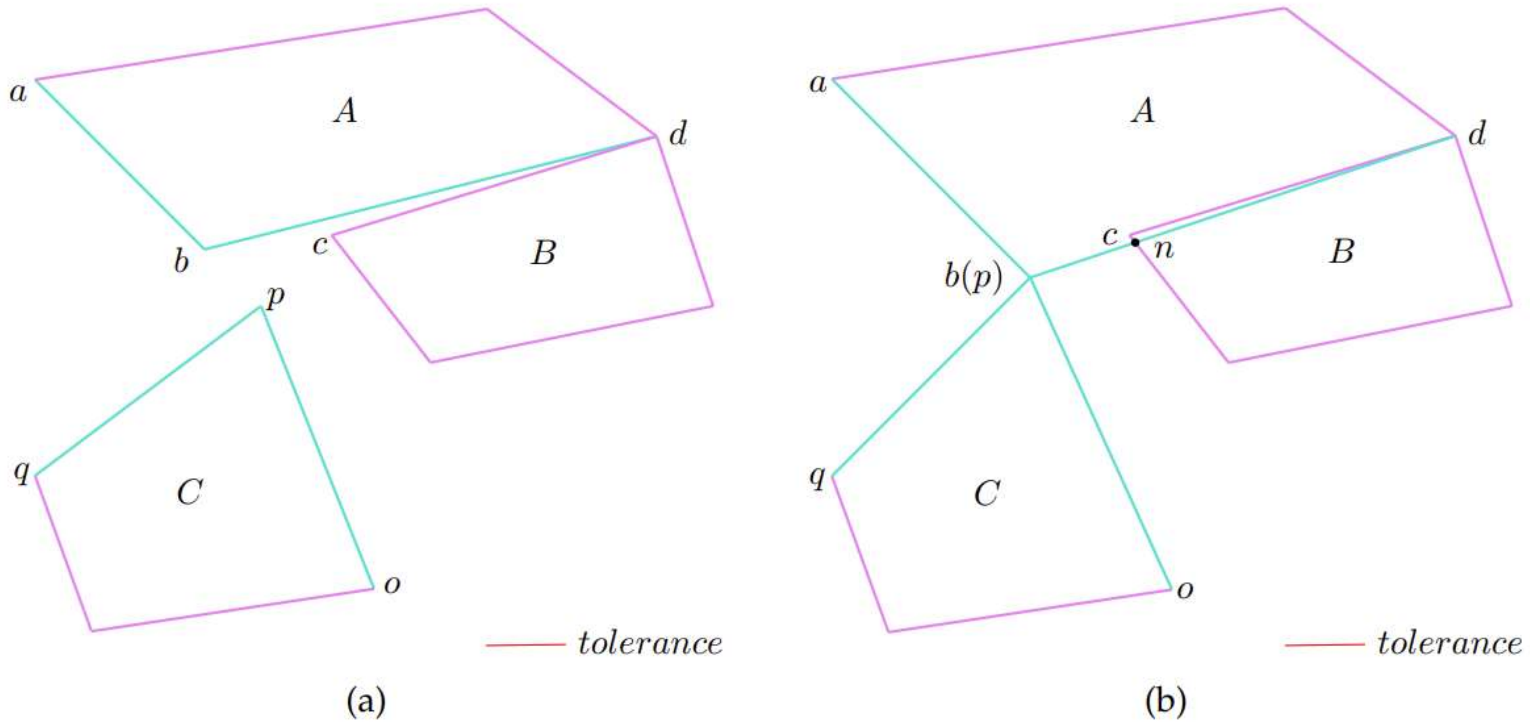






Improve the quality of rounded result but time-consuming

Part 2.6 Process From the Minimum Case

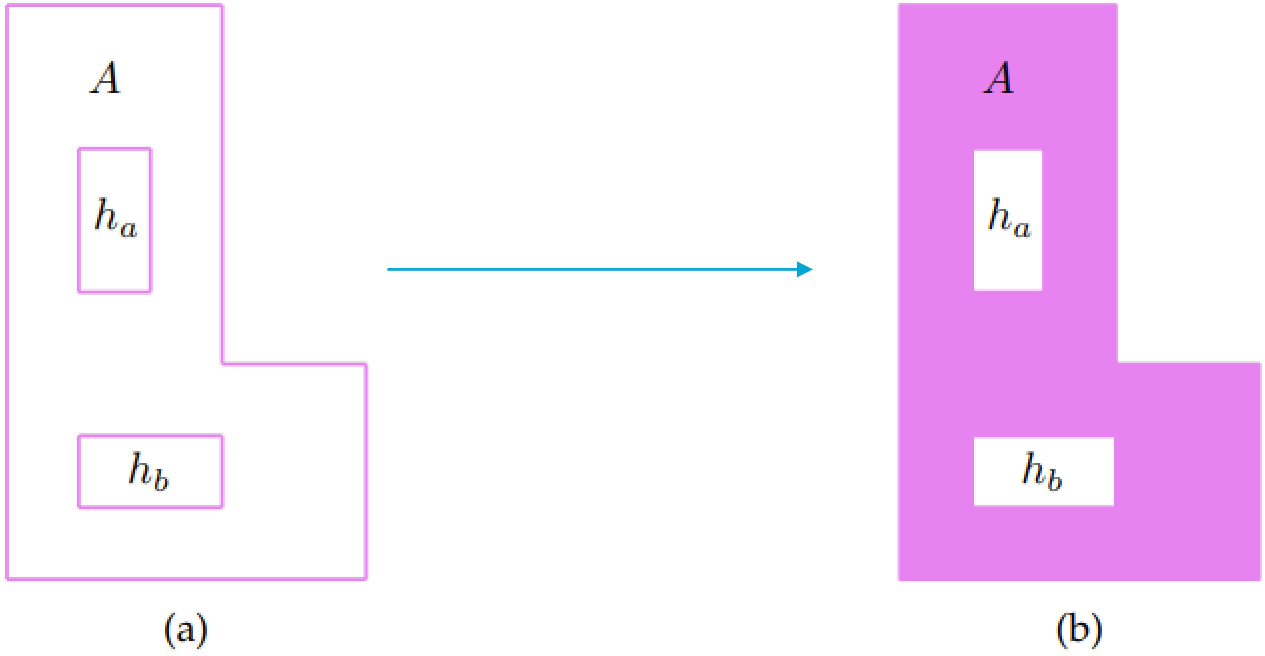


Cascading effects – a SR operation (bp) creates more SR cases (vertex n) proceeding c and bd first will avoid this problem

Apply SR on a set of polygons. (a) Input polygons. Boundaries that will be modified are highlighted in blue. (b) Polygons after SR. Boundary bd intersects polygon B at vertex n after first rounding operation.

TUDelft

Part 2.7 Reconstruction of Polygons



Construct a graph

Identify potential closed rings

Validate geometric and topological correctness of the ring

constructs polygon geometries using the formed rings

An example of using polygonizer to polygonize a set of line segments. (a) A set of line segments. (b) The resulting polygon of Polygonizer containing two interior holes.

3. Implementation

TU Delft

Part 3.1 Prototype

Open-source

Implemented with C++ 17

Third-party libraries:
GDAL – for reading / writing files
CGAL – for using triangulation packages and other auxiliary data structures
GEOS – for reconstruction of the polygons

Available at:
https://github.com/zfengyan/snapoly

Originally developed and tested on Windows 10 platform

Cross-platform version is also provided via CMake

```
D:\snapoly\bin\x64\Release>snapoly 0.01
default snap rounding tolerance is: 0.01
the tolerance is set to: 0.01
the tolerance is set to: 0.01
reading polygons ...
        Path: D:\snapoly\data\DenHaag\DenHaagCentral.gpkg
        Type: GPKG
        Num of Layers: 1
        number of polygons: 19878
        number of fields: 1
extent:
min X: 77379.6  max X: 80555
min Y: 452876   max Y: 455692
done
Time: 0.00545958min
inserting polygons to triangulation ...
done
Time: 0.0243881min
adding tags to triangulation ...
tagging polygons: 1000
tagging polygons: 2000
tagging polygons: 3000
tagging polygons: 4000
tagging polygons: 5000
tagging polygons: 6000
tagging polygons: 7000
tagging polygons: 8000
tagging polygons: 9000
tagging polygons: 10000
```

TUDelft

4. Results

Part 4.1 Datasets

	Abbreviation	Number of polygons	Type
Den Haag Central	DHC	19878	Dense urban area
Faroe Islands	Faroes	30926	Island area
Delft	Delft	38376	Normal
Freiburg	Freiburg	53723	Normal
Rotterdam Central	RCD	127795	Dense urban area

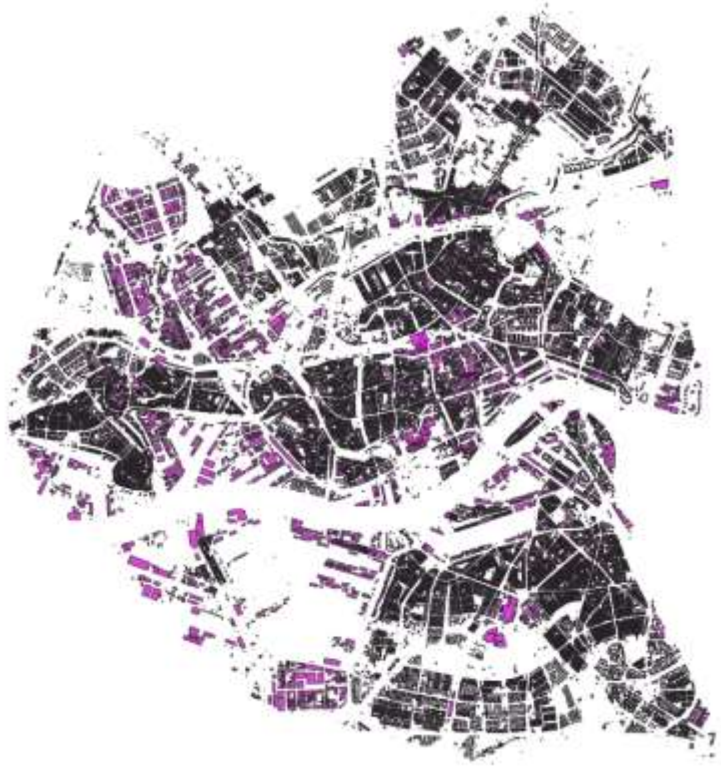
Data Source: GeoFarbik*

TU Delft

* Geofabrik: Download server for openstreetmap data. Web Based Download Application: http://download.geofabrik.de/. Last checked on April 2023

Part 4.1 Datasets

Den Haag Central (DHC)
Num of polygons: 19878
Type: Dense urban area



Rotterdam Central (RCD)
Num of polygons: 127795
Type: Dense urban area

Data Source: GeoFarbik*

* Geofabrik: Download server for openstreetmap data. Web Based Download Application: http://download.geofabrik.de/. Last checked on April 2023.

TU Delft

Part 4.1 Datasets

Delft
Num of polygons: 38376
Type: Normal

Freiburg
Num of polygons: 53723
Type: Normal

TU Delft

Data Source: GeoFarbik*

* Geofabrik: Download server for openstreetmap data. Web Based Download Application: http://download.geofabrik.de/. Last checked on April 2023

Part 4.1 Datasets

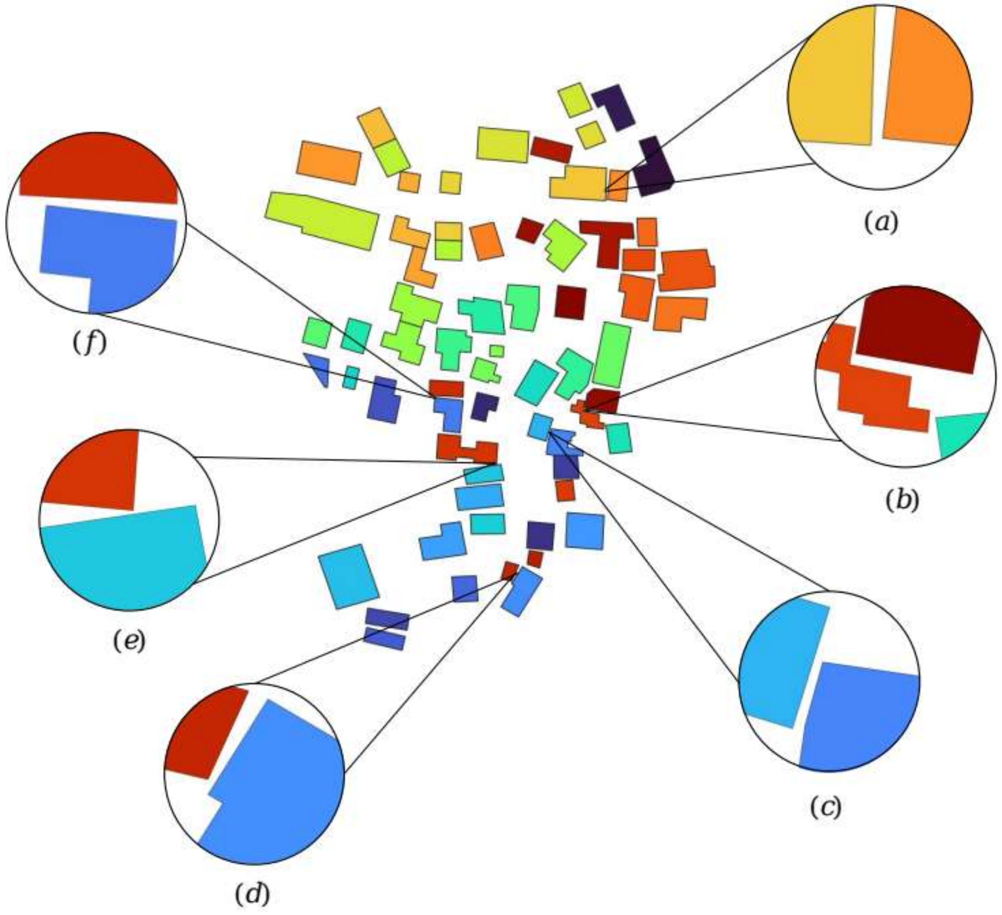
Faroe Islands (Faroes)
Num of polygons: 30926
Type: Island

Data Source: GeoFarbik*

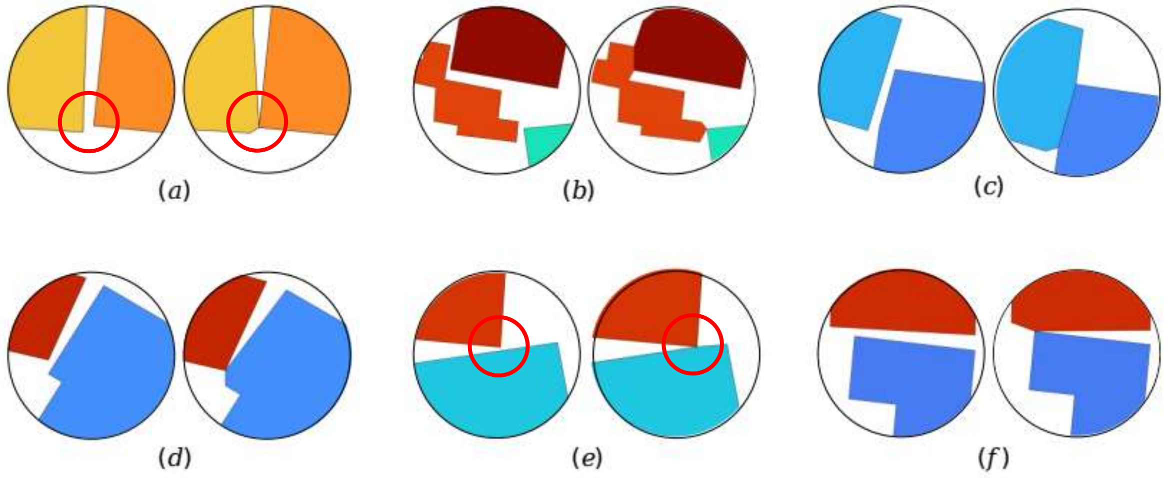
TU Delft

* Geofabrik: Download server for openstreetmap data. Web Based Download Application: http://download.geofabrik.de/. Last checked on April 2023

Part 4.2 Exemplary results



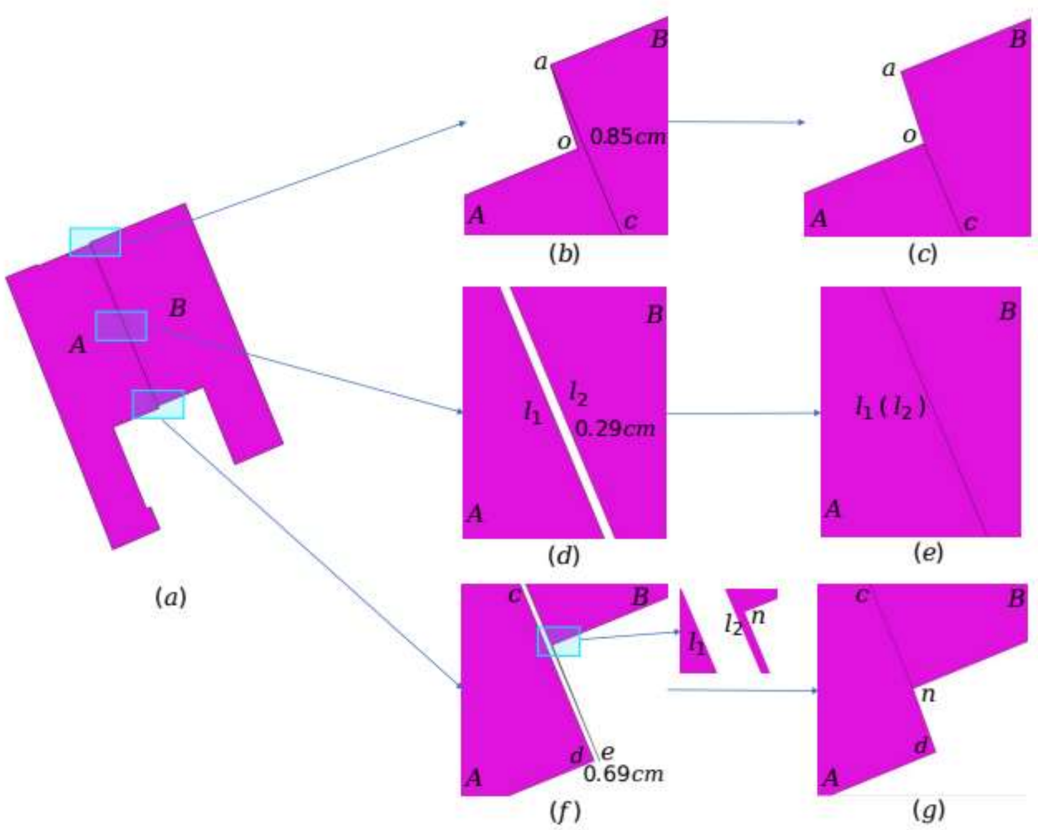
An excerpt of Faroe Islands dataset.

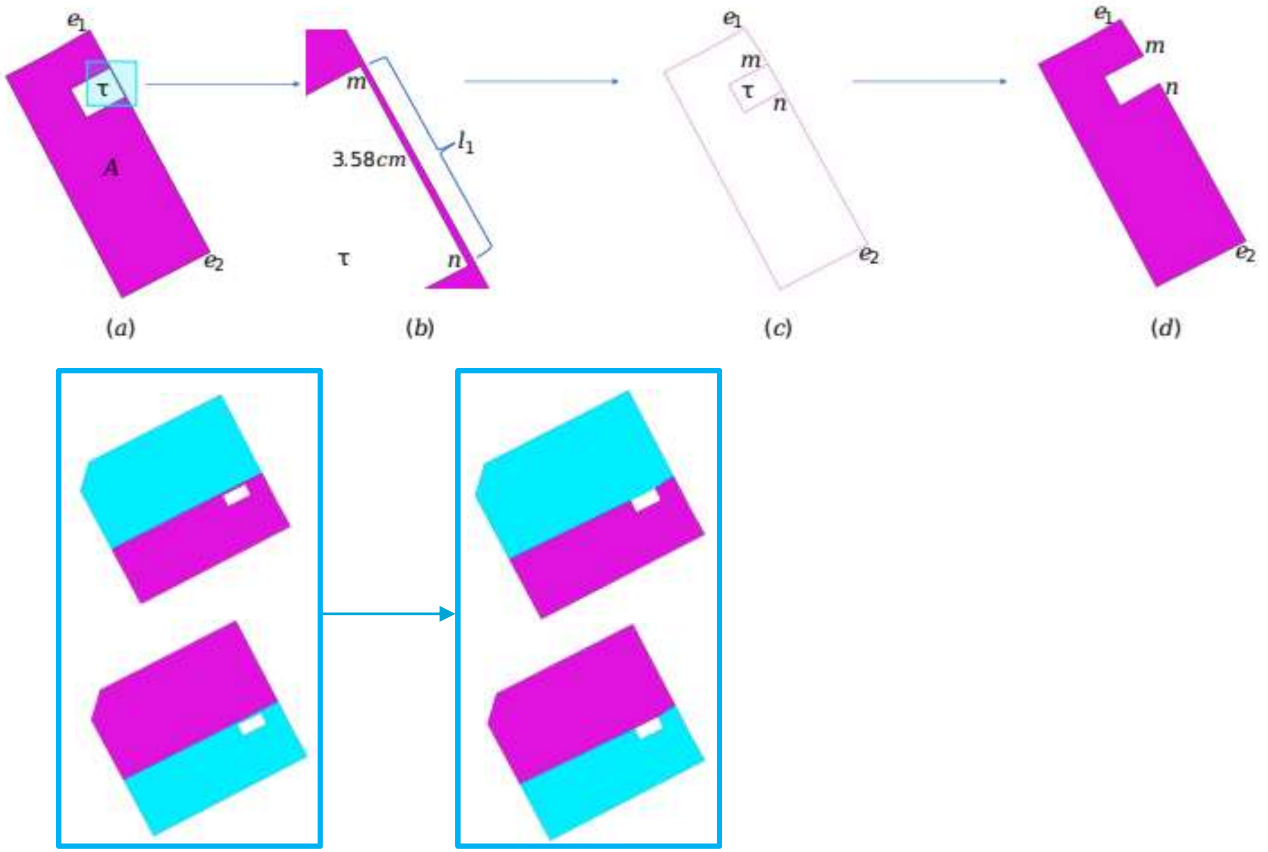


SR cases and corresponding counterparts.

perpendicularity

Part 4.2 Exemplary results





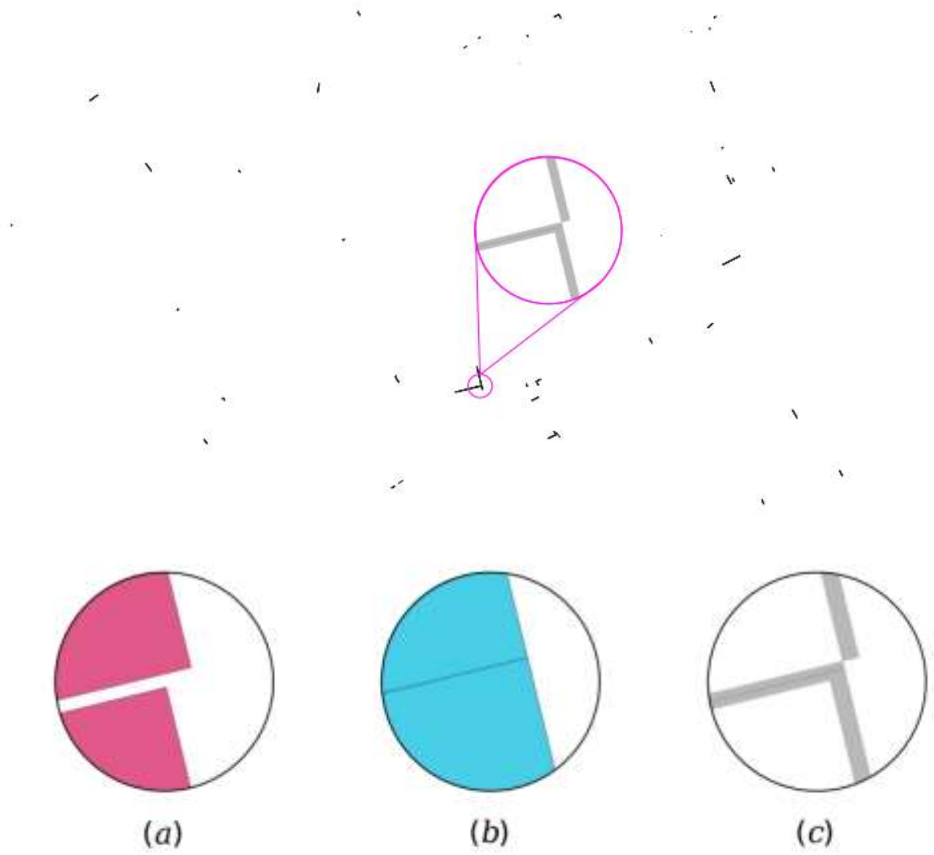
Exemplary polygons from Den Haag Central dataset.

TU Delft

Part 4.3 Observe & measure distortions

Input

Rounded result



Symmetrical difference

An excerpt of Delft dataset.

Visualized in QGIS

Part 4.3 Observe & measure distortions

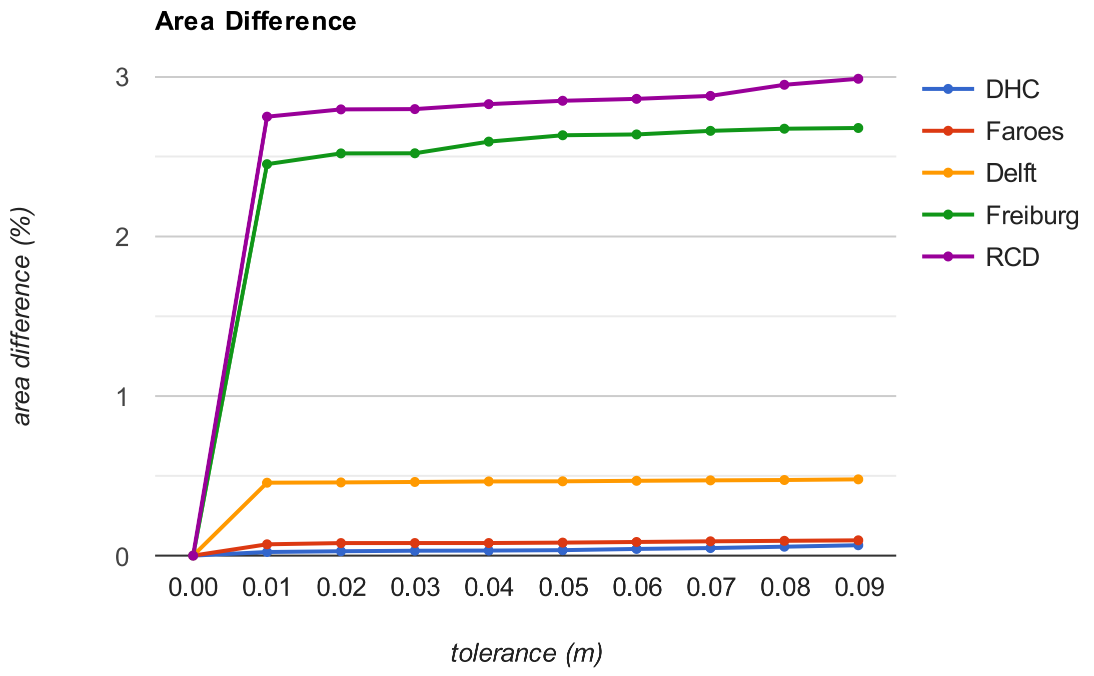
Quantitative measurement: area difference

$$area\_diff = \frac{\sum_{i=1}^{n} |\mathcal{A}(\mathcal{P}i) - \mathcal{A}(\mathcal{P}i')|}{\sum_{i=1}^{n} \mathcal{A}(\mathcal{P}i)} \times 100\%.$$

$\mathcal{A}(\mathcal{P}i)$ The area of the original polygon

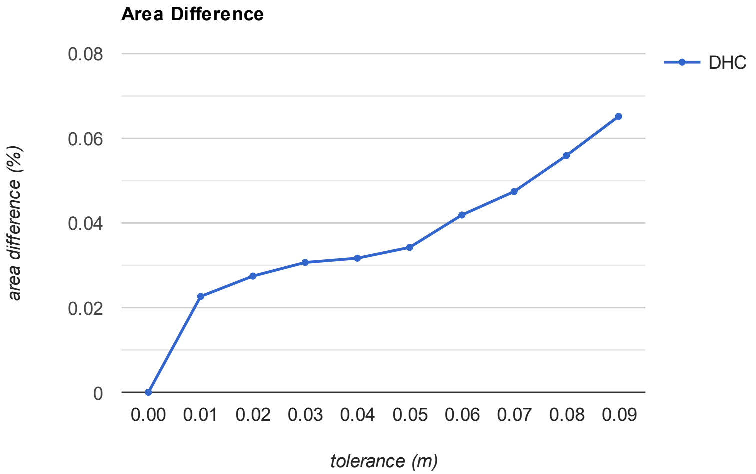
$\mathcal{A}(\mathcal{P}i')$ The area of the rounded polygon

n The total number of the polygons

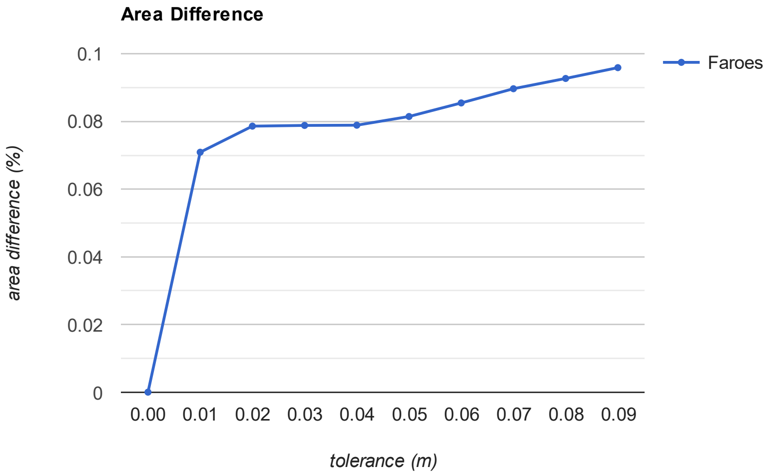


Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

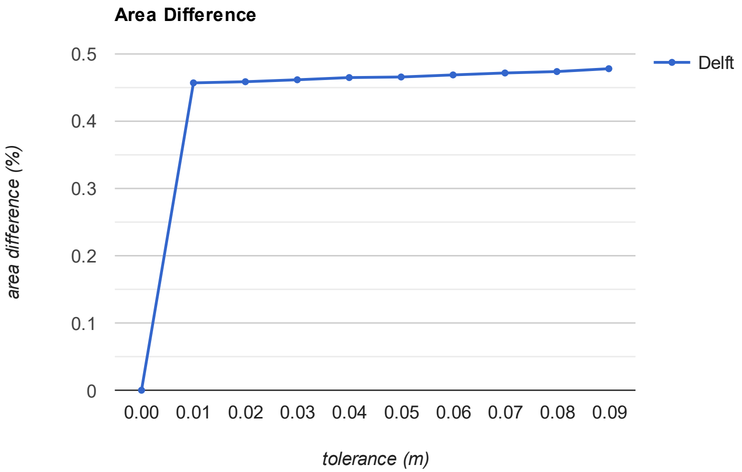
Part 4.3 Observe & measure distortions



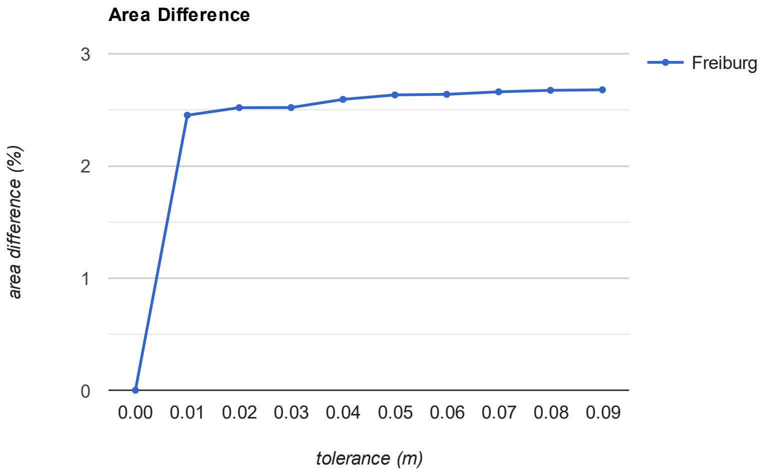
(a) DHC (19878 polygons)



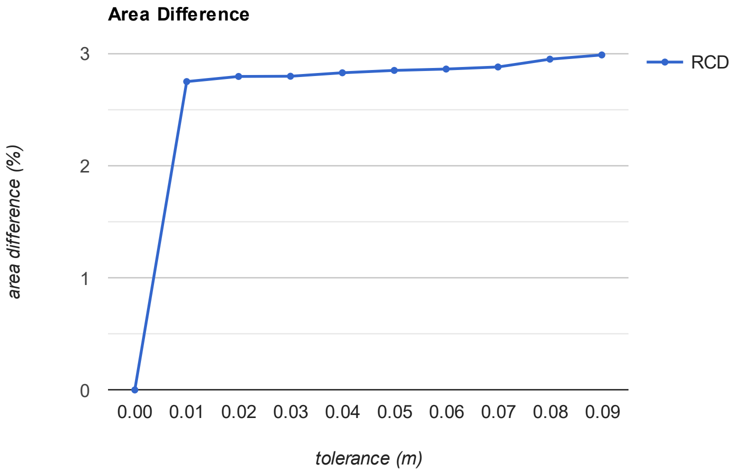
(b) Faroes (30926 polygons)



(c) Delft (38376 polygons)



(d) Freiburg (53723 polygons)



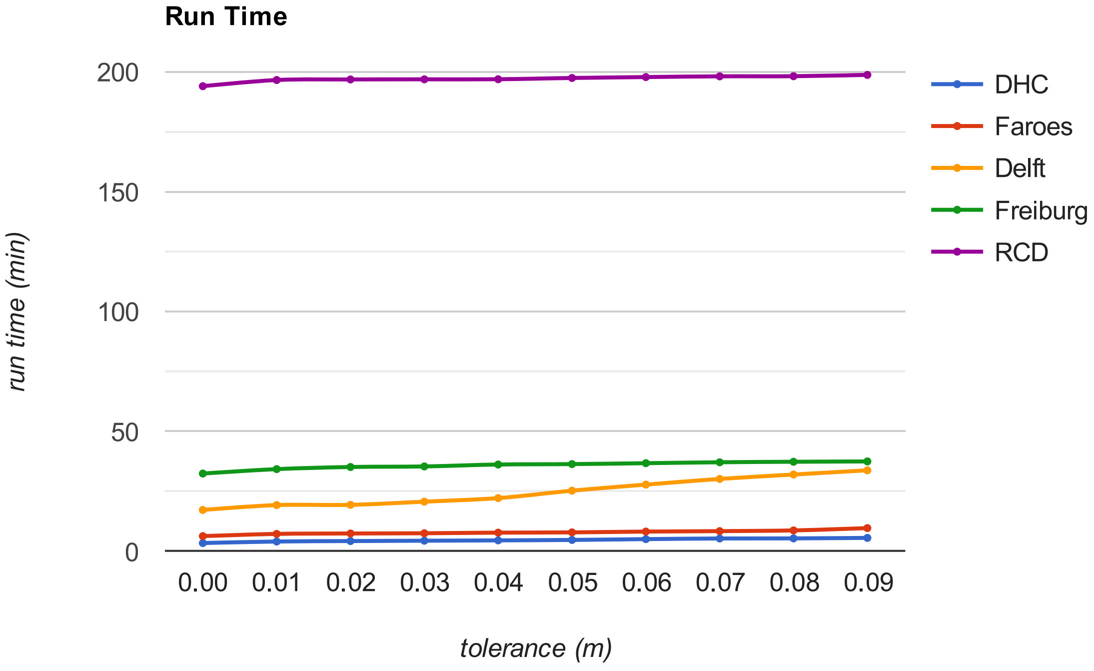
(e) RCD (127795 polygons)

TU Delft

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

Part 4.4 Benchmarking

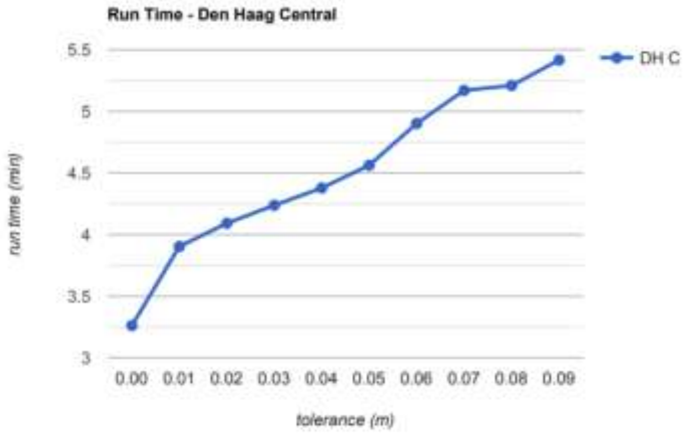
Dataset	Number of polygons	Tolerance(m)	Total run time(min)
DHC	19878	0.00	3.267
DHC	19878	0.01	3.908
DHC	19878	0.02	4.096
DHC	19878	0.03	4.242
DHC	19878	0.04	4.382
DHC	19878	0.05	4.566
DHC	19878	0.06	4.905
DHC	19878	0.07	5.173
DHC	19878	0.08	5.212
DHC	19878	0.09	5.419
Faroes	30926	0.00	6.159
Faroes	30926	0.01	7.088
Faroes	30926	0.02	7.263
Faroes	30926	0.03	7.354
Faroes	30926	0.04	7.611
Faroes	30926	0.05	7.725
Faroes	30926	0.06	8.056
Faroes	30926	0.07	8.245
Faroes	30926	0.08	8.543
Faroes	30926	0.09	9.514
Delft	38376	0.00	17.090
Delft	38376	0.01	19.130
Delft	38376	0.02	19.223
Delft	38376	0.03	20.558
Delft	38376	0.04	22.050
Delft	38376	0.05	25.145
Delft	38376	0.06	27.686
Delft	38376	0.07	30.051
Delft	38376	0.08	31.593
Delft	38376	0.09	33.650
Freiburg	53723	0.00	32.305
Freiburg	53723	0.01	34.153
Freiburg	53723	0.02	35.036
Freiburg	53723	0.03	35.283
Freiburg	53723	0.04	36.072
Freiburg	53723	0.05	36.253
Freiburg	53723	0.06	36.631
Freiburg	53723	0.07	37.018
Freiburg	53723	0.08	37.232
Freiburg	53723	0.09	37.368
RCD	127795	0.00	194.148
RCD	127795	0.01	196.729
RCD	127795	0.02	196.970
RCD	127795	0.03	197.022
RCD	127795	0.04	197.056
RCD	127795	0.05	197.583
RCD	127795	0.06	197.960
RCD	127795	0.07	198.275
RCD	127795	0.08	198.323
RCD	127795	0.09	198.875



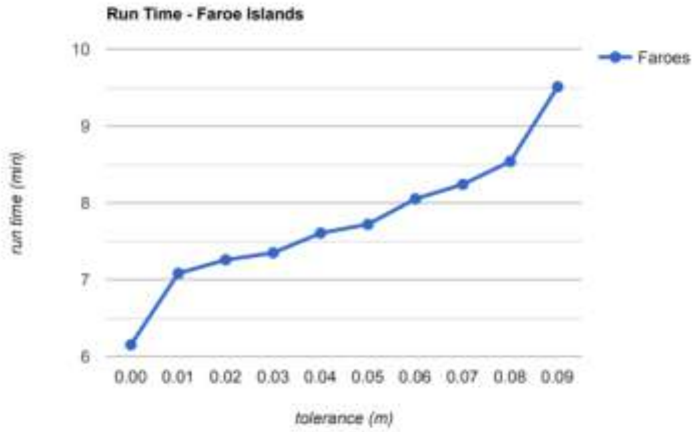
Overall performance for the selected datasets.

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

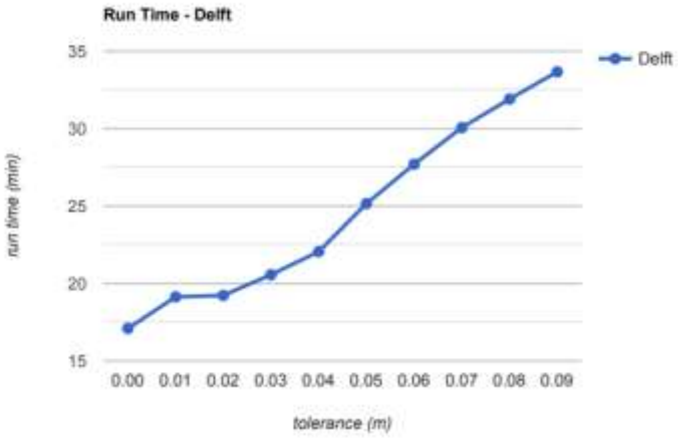
Part 4.4 Benchmarking



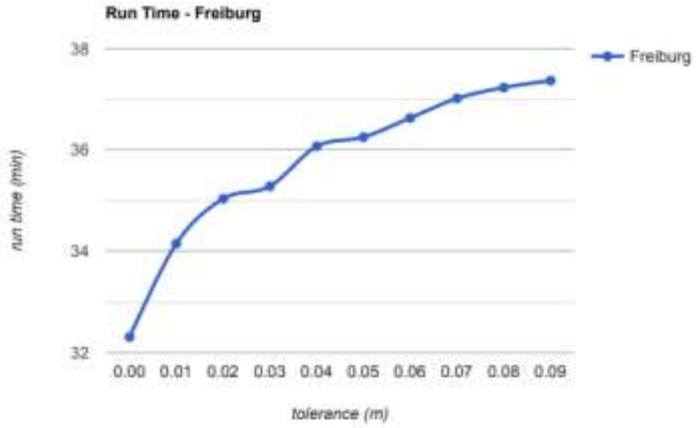
(a) DHC (19878 polygons)



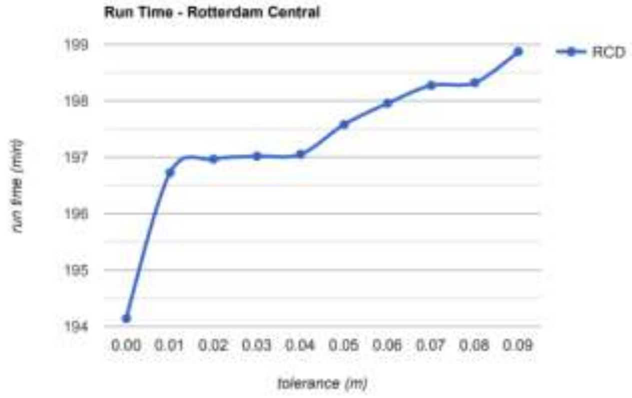
(b) Faroes (30926 polygons)



(c) Delft (38376 polygons)



(d) Freiburg (53723 polygons)



(e) RCD (127795 polygons)

TU Delft

Overall performance for the selected datasets.

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

Part 4.4 Benchmarking

Num of polygons (DHC)	Num of vertices	Num of segments	Num of faces	Runtime (min)
10	383	1125	743	0.00168242
100	1715	5118	3404	0.00380131
500	6545	19603	13059	0.00956036
1000	8491	25441	16951	0.0147298
5000	29981	89911	59931	0.176387
10000	57252	171718	114467	0.666789
19878 (total)	115589	346735	231147	3.25784
Num of polygons (Faroes)	**Num of vertices**	**Num of segments**	**Num of faces**	**Runtime (min)**
10	79	224	146	0.00257392
100	744	2214	1471	0.00498292
500	3677	11012	7336	0.018107
1000	7425	22252	14828	0.0376588
5000	30913	92721	61809	0.268987
10000	58106	174301	116196	0.724415
20000	115388	346146	230759	2.45576
30926 (total)	174557	523642	349086	7.24895
Num of polygons (Delft)	**Num of vertices**	**Num of segments**	**Num of faces**	**Runtime (min)**
10	148	432	285	0.00254325
100	2872	8600	5729	0.0066124
500	11852	35525	23674	0.0382818
1000	18810	56396	37587	0.0857326
5000	33377	100092	66716	0.342467
10000	69117	207312	138196	1.13968
20000	122748	368210	245463	3.91653
30000	173797	521356	347560	10.9141
38376 (total)	210917	632723	421807	18.0034

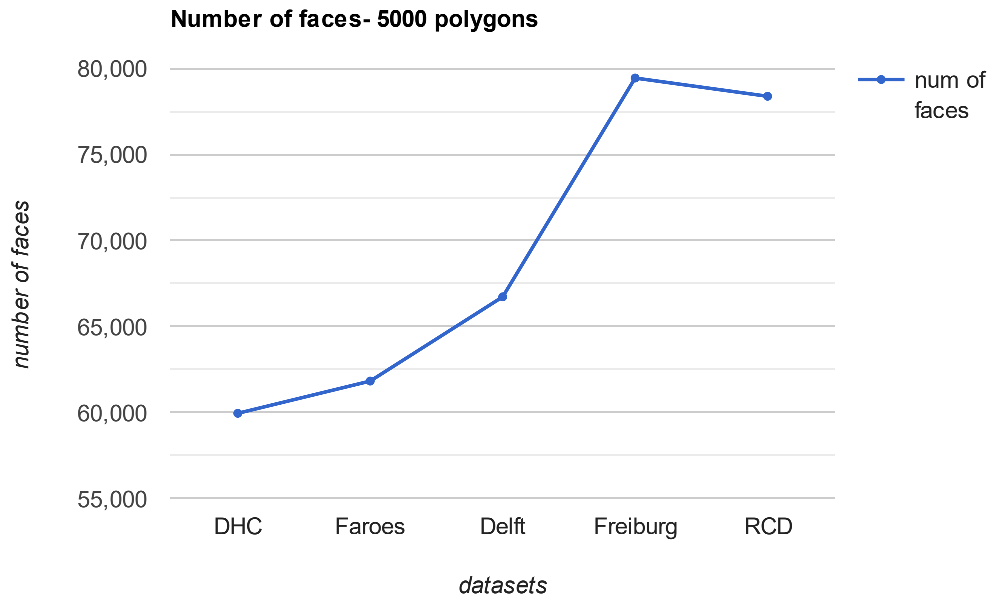
Num of polygons (Freiburg)	Num of vertices	Num of segments	Num of faces	Runtime (min)
10	391	1154	764	0.00361835
100	1854	5539	3686	0.00564597
500	6008	18009	12002	0.0220264
1000	10131	30376	20246	0.0453322
5000	39743	119203	79461	0.354143
10000	78735	236176	157442	1.18357
20000	148623	445838	297216	4.89471
30000	201209	603598	402390	11.7602
40000	243381	730114	486734	19.7922
53723 (total)	300126	900347	600222	35.5441
Num of polygons (RCD)	**Num of vertices**	**Num of segments**	**Num of faces**	**Runtime (min)**
10	337	987	651	0.00259865
100	1653	4936	3284	0.00538976
500	10019	30031	20013	0.0324458
1000	16146	48412	32267	0.0726439
5000	39212	117610	78399	0.423932
10000	75164	225464	150301	1.38921
50000	246332	738964	492633	25.2106
100000	482126	1446343	964218	123.575
127795 (total)	616293	1848840	1232548	204.754

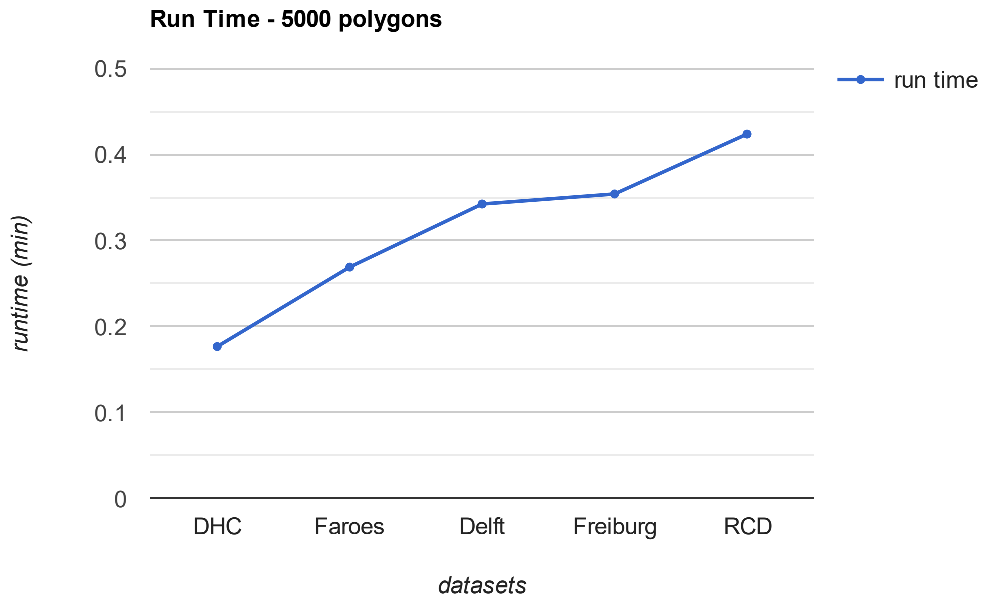
tolerance = 0.01m

TUDelft

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

Part 4.4 Benchmarking

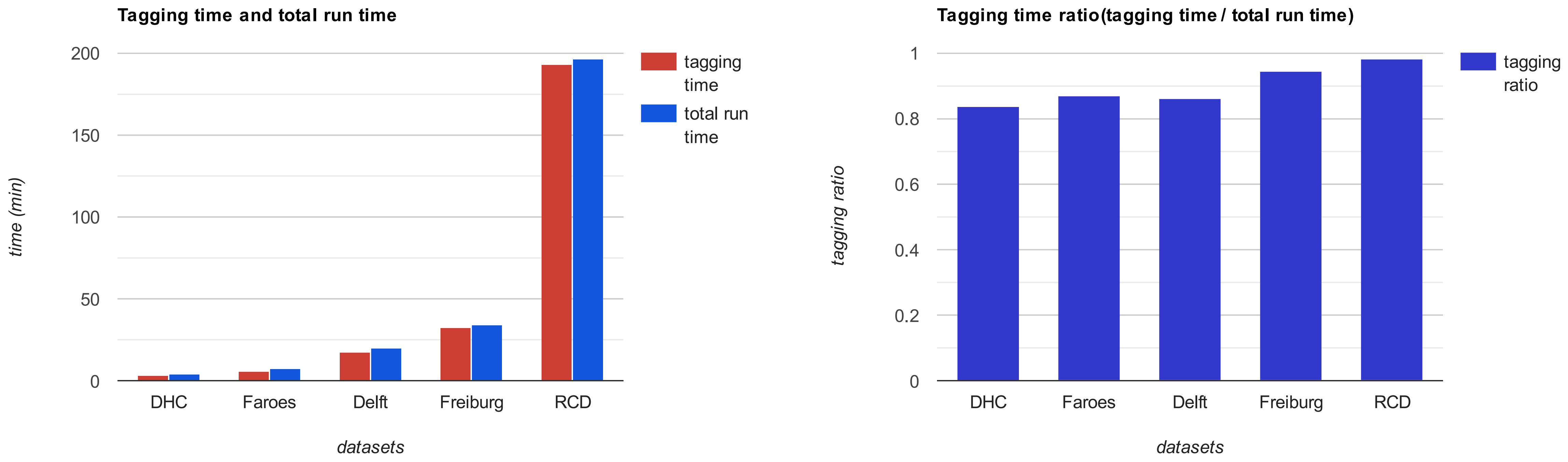




The number of polygons

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

Part 4.4 Benchmarking



Tagging time and total run time. (a) Tagging time and total runtime regarding the selected datasets (tolerance = 0.01m). (b) Tagging time ratio (tagging time / total run time) regarding the selected datasets (tolerance = 0.01m)

TU Delft

Testing platform: Windows-x64 Intel(R) Core(TM) i7-10875H CPU @ 2.30GHz RAM: 16.0 GB Windows version: Windows10 21H2

5. Conclusion & Future work

Part 4.5 Conclusion

Snap rounding + Constrained triangulation

↓

2-dimensional polygons √

- How to integrate a CT with the input polygons (possibly with interior holes)
  using the boundaries of the polygons (including exteriors and interiors) as constraints.
- How to link the triangulation with the original polygons? If the triangulation is modified, how to update the changes to the polygons?
  Use a separate container to store the boundaries of polygons (and also the related information), update the boundaries and the triangulation dynamically.
- Polygons usually have attributes attached (e.g. polygon id, area of a polygon), how to preserve them in a proper way during the snap rounding process?
  The boundaries (constraints) are constructed with tag attached, indicating which polygon they should belong to, hence the attributes of the polygon can be preserved.
- Given a certain threshold, there may exist several snap rounding cases (e.g. multiple polygonal vertices and polygonal vertex and boundaries), what is the most reasonable order when snap rounding is being performed?
  Process the snap rounding operation from the minimum case (has minimum distance with a certain tolerance).
- How to measure and evaluate the distortions of the polygons before and after snap rounding?
  Symmetrical difference + area difference.

TUDelft

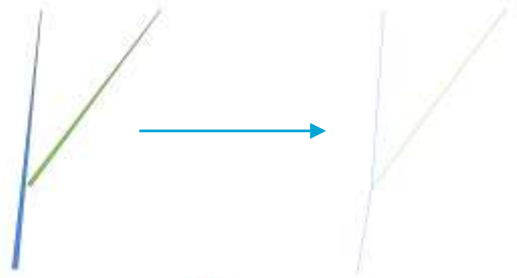
Part 4.5 Conclusion

- The small gaps between vertices or between vertex and boundaries are removed.
- Capable of handling not only valid geometries but also invalid ones, such as overlapping area.
- Originally topological and geometric characteristics are preserved as much as possible
- Work with datasets of different size, e.g. from 100 polygons to 100, 000 polygon

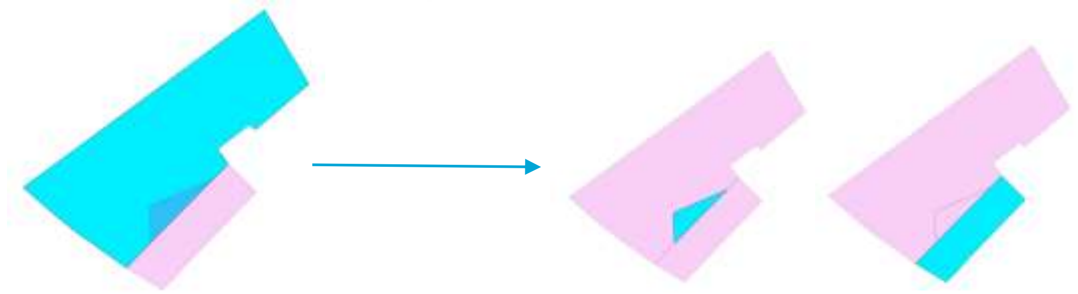
- The distortions are natural to the SR and can not be completely avoided, further modifications can be implemented to guarantee certain properties of the polygons, e.g. perpendicularity.
- Not fully robust regarding large tolerance values as it will cause geometry degeneracies (e.g. polygons collapsing to line segments or points) and possibly cause undefined behavior of the prototype
- The tagging stage accounted for over 80% of the total run time, can be optimized but will possibly not have the potential of handling more complex scenarios such as overlap area.

Part 4.6 Future work

- Support for rounding polygons and line segments.



- More flexible processing of overlapping areas.



- Improve efficiency by utilizing std::unordered_set instead of std::list to store the constraints. naturally std::unordered_set only supports for built-in data types. Storing customized objects (e.g. constraints) would require constructing customized hash functions.

TUDelft

Part 4.6 Future work

- **Improve efficiency** by using additional data structures such as std::priority_queue to store the snapping cases in accordance with the distance (need to be proven due to the cascading effects)
- **Improve robustness for large tolerance values**.
  SR usually works for appropriate tolerance values. However, large tolerance value can be given and it may cause undefined behaviours of the prototype as the mechanism of SR is designed for handling closely positioned vertices and boundaries.
- **More automated process**
  Re-projection process can be integrated into the prototype if the CRS can be known in advance.
  Integrating auto-correction techniques can further enhance the prototype's automation capabilities, such as integrating *prepair** to repair individual polygons before the application of SR.
  Automatic selection of an appropriate tolerance value (conduct a preliminary analysis of the input dataset, such as examining adjacency relationships and computing distances between elements).
- **Compare with ISR of polygons**
  Some cells in the grid of traditional SR do not store any elements hence the grid is not fully utilized and there will be waste -> compare the memory usage
- **Support for more data formats**

*Ledoux, H., Arroyo Ohori, K., and Meijers, M. (2012). Automatically repairing invalid polygons with a constrained triangulation. In Proceedings of the AGILE 2012 International Conference, April 2012, Avignon, France, pp. 13-18. Agile.

Thanks for listening.