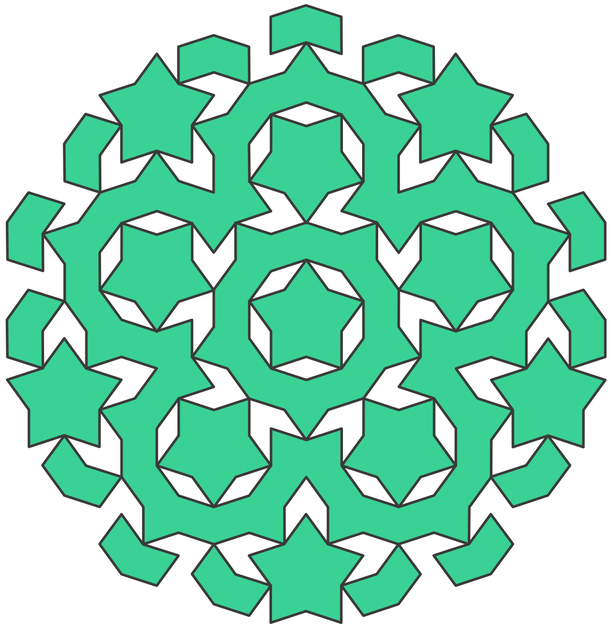


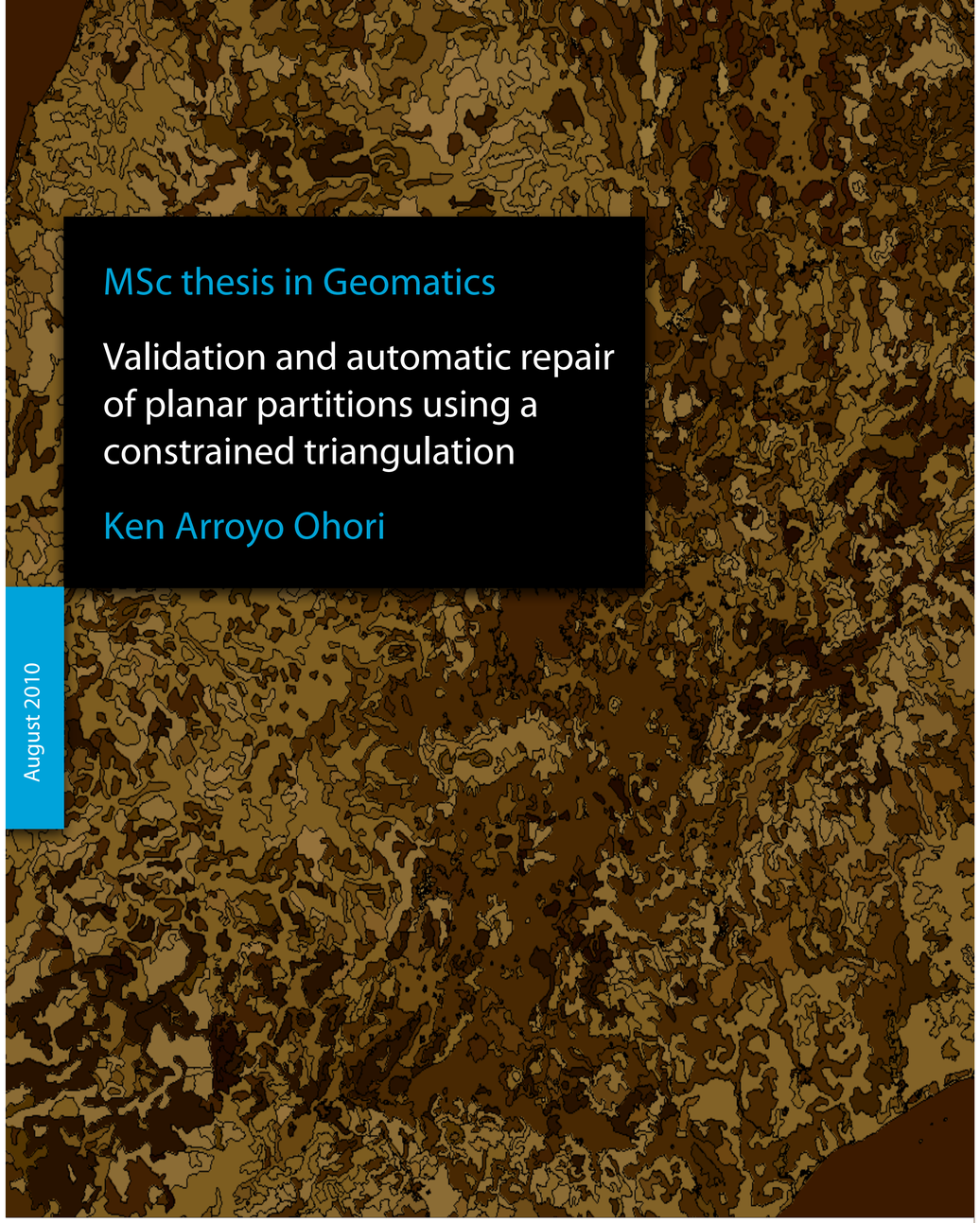
CGAL (Multi)polygon repair

3D Tea

Ken Arroyo Ohori 19.09.2024

Context and timeline

- My MSc thesis (2010) included a method to repair polygons and an implementation (***prepair***)
- Slowly added improvements since then, including some parts that require a (fragile) deeper integration with CGAL
- Chance to work on it during the Google Summer of Code 2023
- Merged into CGAL 6.0 (out in beta now)



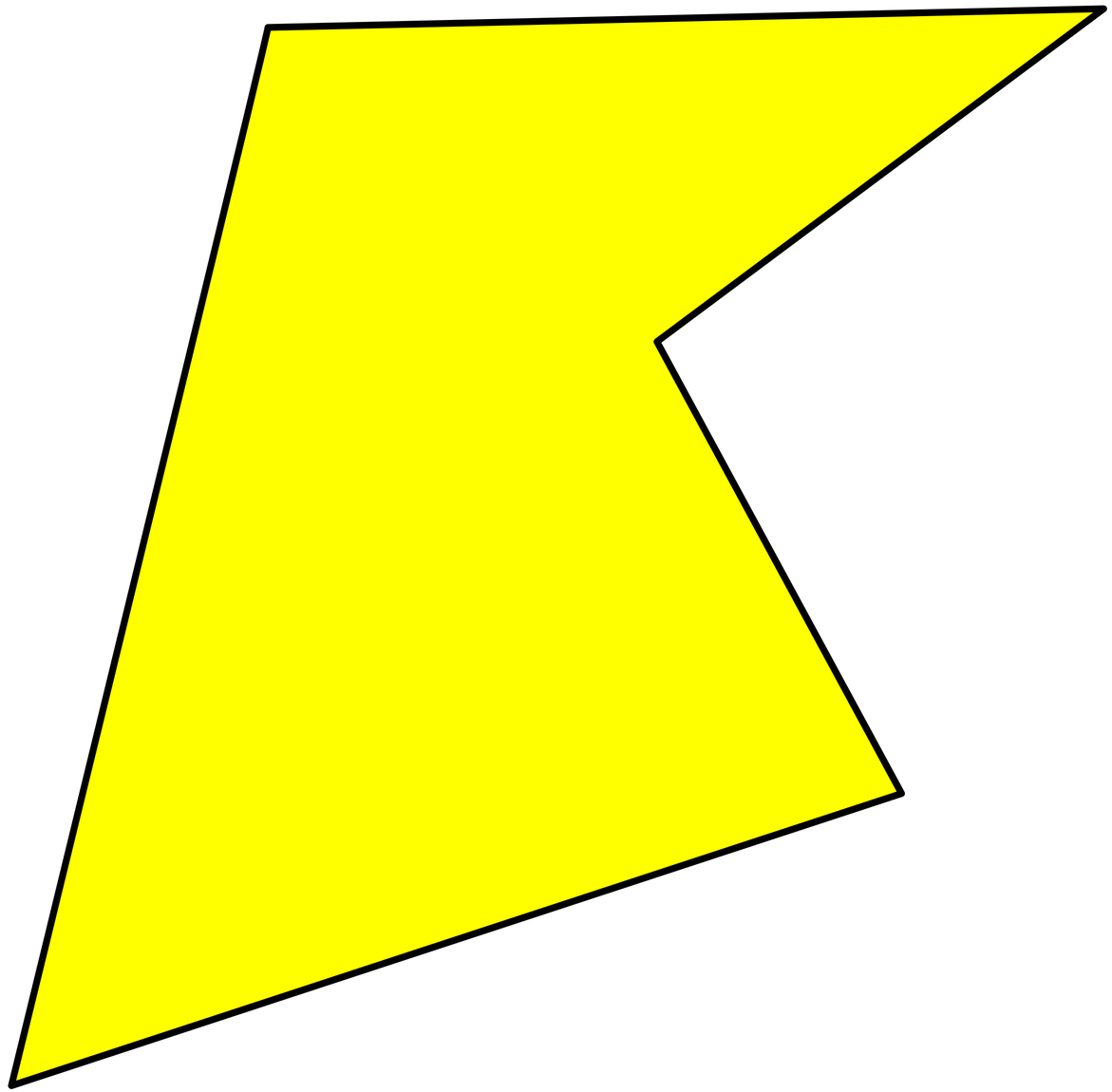
Why?

- Common: invalid polygons are prevalent in large datasets
- Problematic: invalidity causes errors, undefined behaviour and invalid output in further processing
- Complex: checking validity of (multi)polygons with holes isn't straightforward

Validity

Polygon

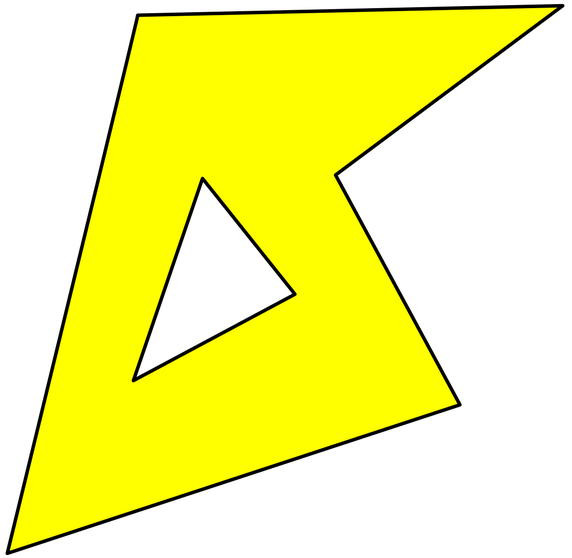
- Point set in $\mathbb{R}^2$ bounded by a cycle of linear edges (outer boundary).
- The outer boundary should be simple: the interiors of its edges are pairwise disjoint and all of its vertices have a degree of two.
- Topologically equivalent to a disk.

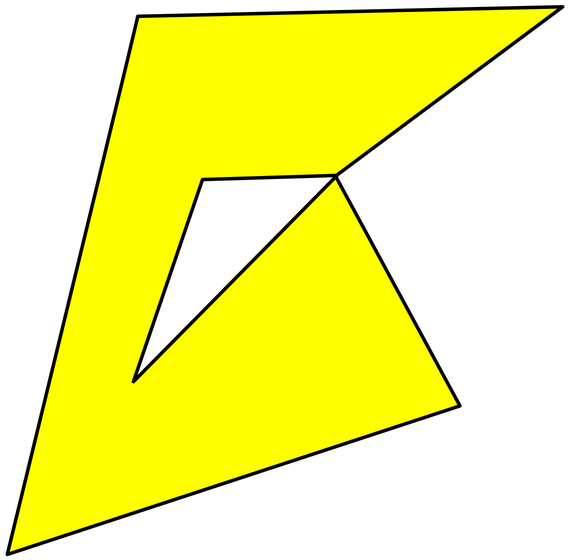


Validity

Polygon with holes

- Point set in $\mathbb{R}^2$ bounded by one outer boundary and zero or more inner boundaries, where each inner boundary represents a hole in the polygon.
- Considered independently, each boundary should be simple.
- The different boundaries of a polygon are allowed to intersect tangentially at common vertices (with no common edges), forming vertices with degrees of a multiple of two at the tangential points.
- The interior of a polygon with holes should form a connected point set.
- Note: A valid polygon can also be represented as a valid polygon with holes (with zero holes).

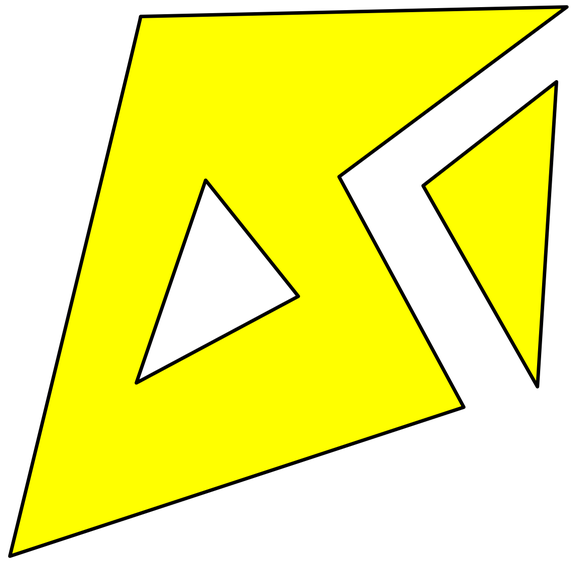


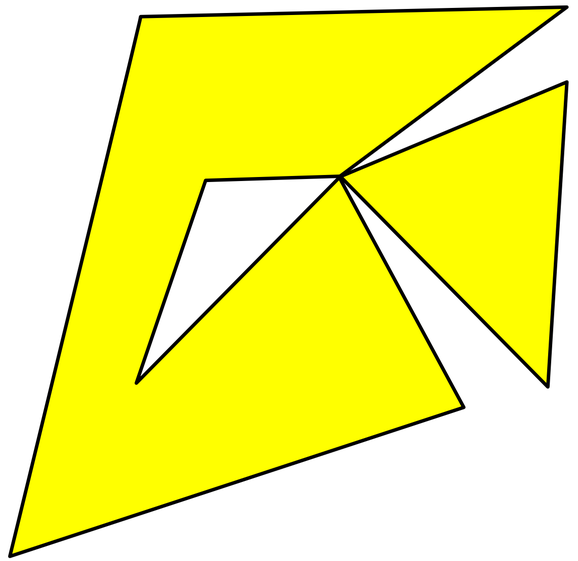


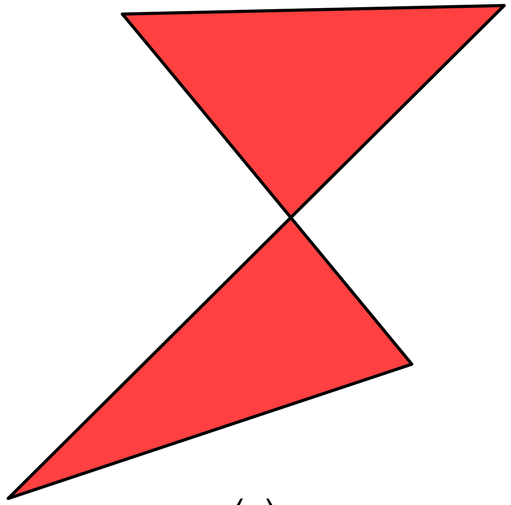
Validity

Multipolygon with holes

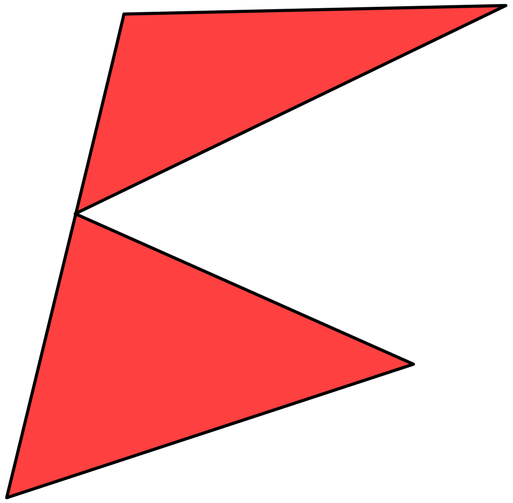
- Point set in $\mathbb{R}^2$ represented by a set of zero or more valid polygons with holes.
- The interiors of the polygons with holes should be pairwise disjoint, but they are allowed to intersect tangentially at their common vertices.
- Note: A valid polygon with holes can also be represented as a valid multipolygon with holes (with only one polygon).



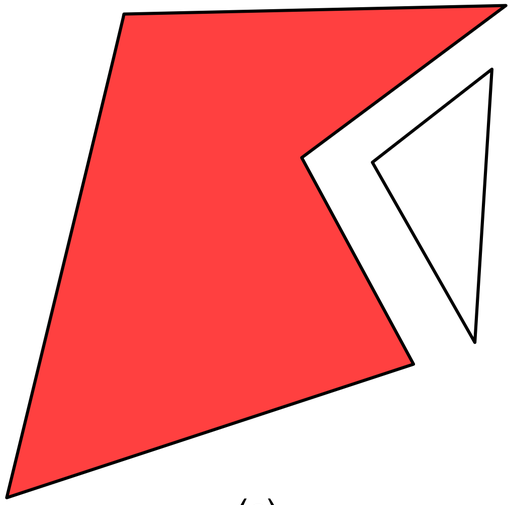




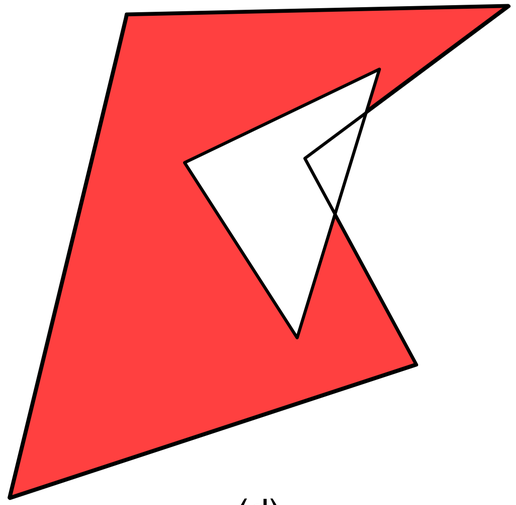
(a)



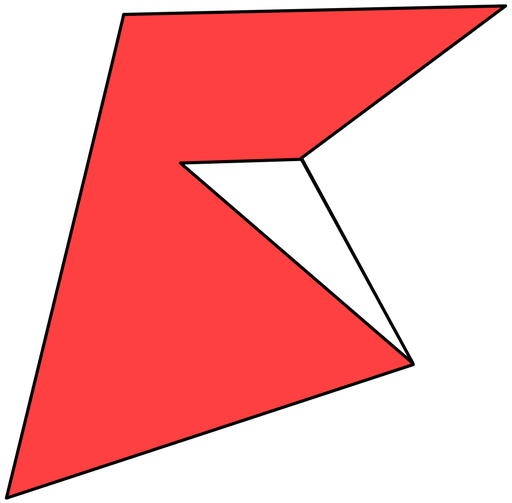
(b)



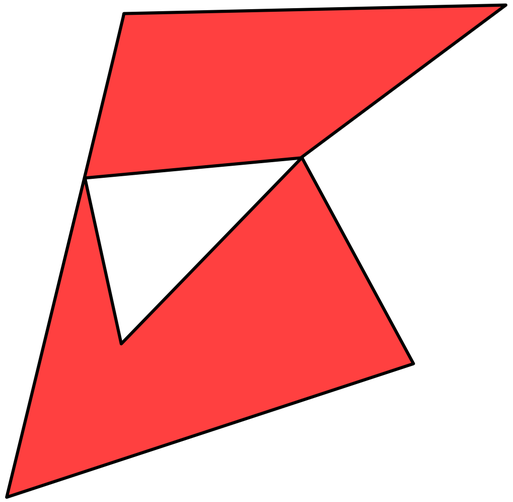
(c)



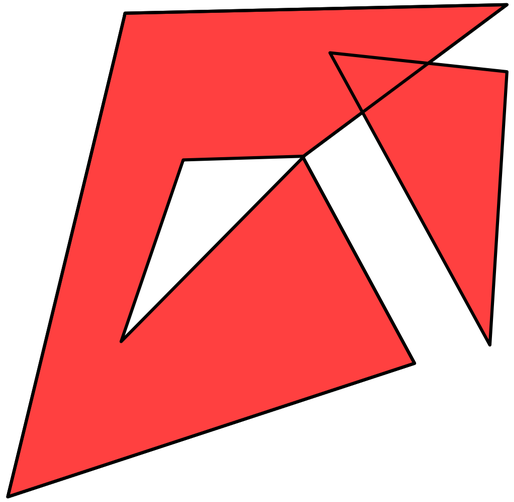
(d)



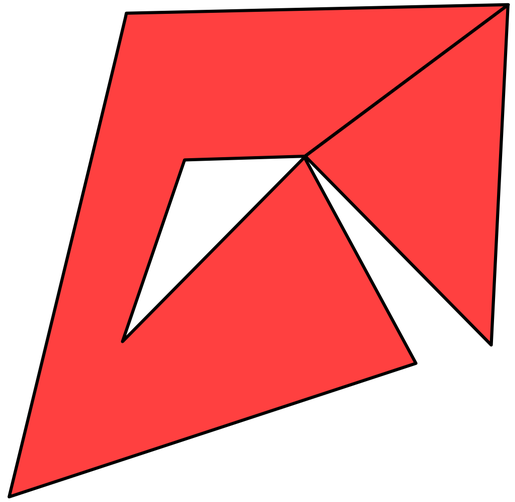
(e)



(f)



(g)



(h)

Possibly invalid input (polygon, polygon with holes or multipolygon with holes)

Valid output (multipolygon with holes)

Valid output

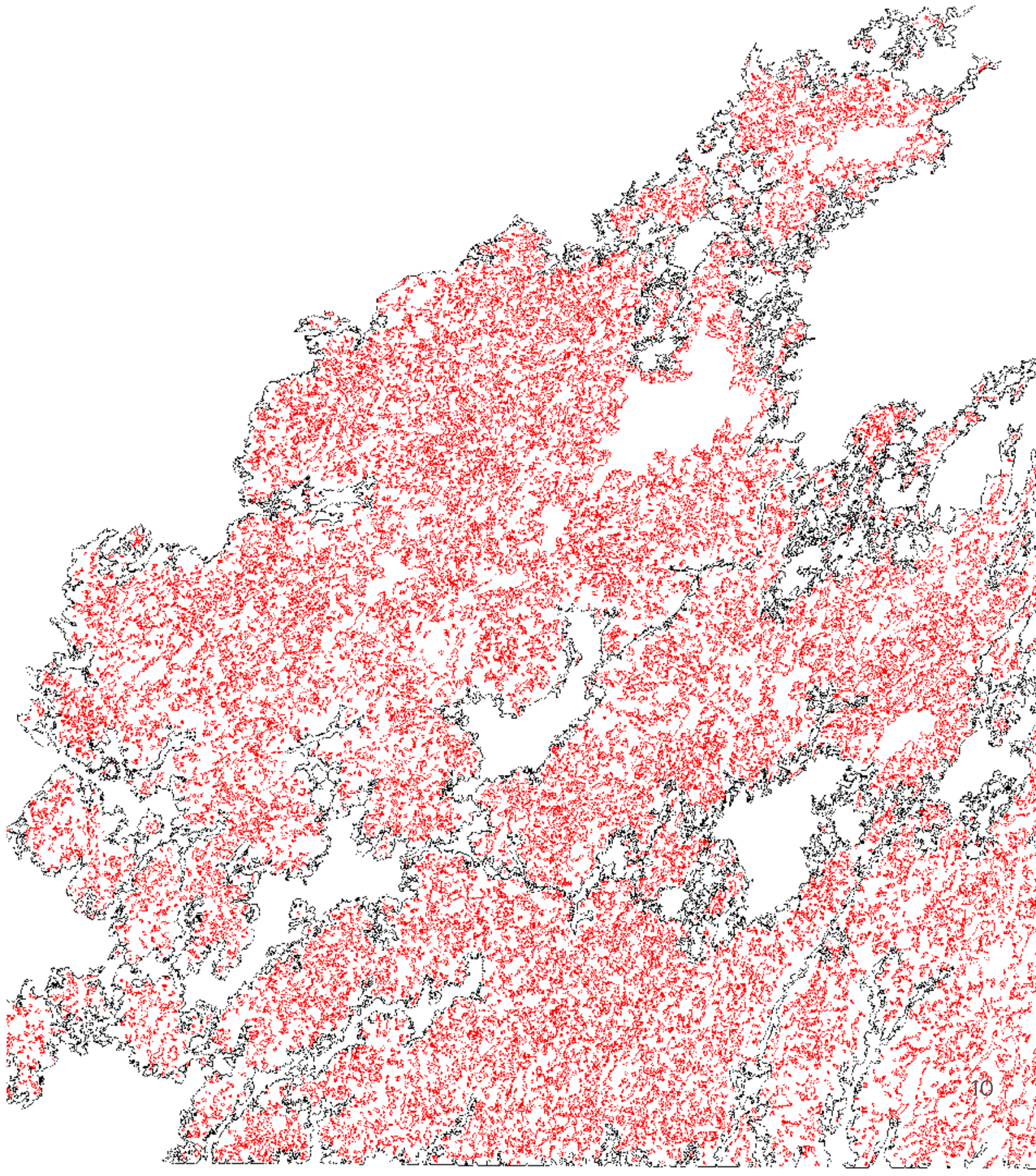
Ensuring that it is unique and deterministic

- Adjacent collinear edges touching at vertices of degree two are merged
- The sequence of vertices representing a boundary starts from its lexicographically smallest vertex
- Outer boundaries are oriented counter-clockwise and inner boundaries are oriented clockwise
- The inner boundaries of a polygon with holes are stored in lexicographic order
- The polygons with holes of a multipolygon with holes are also stored in lexicographic order

Algorithm

Overview

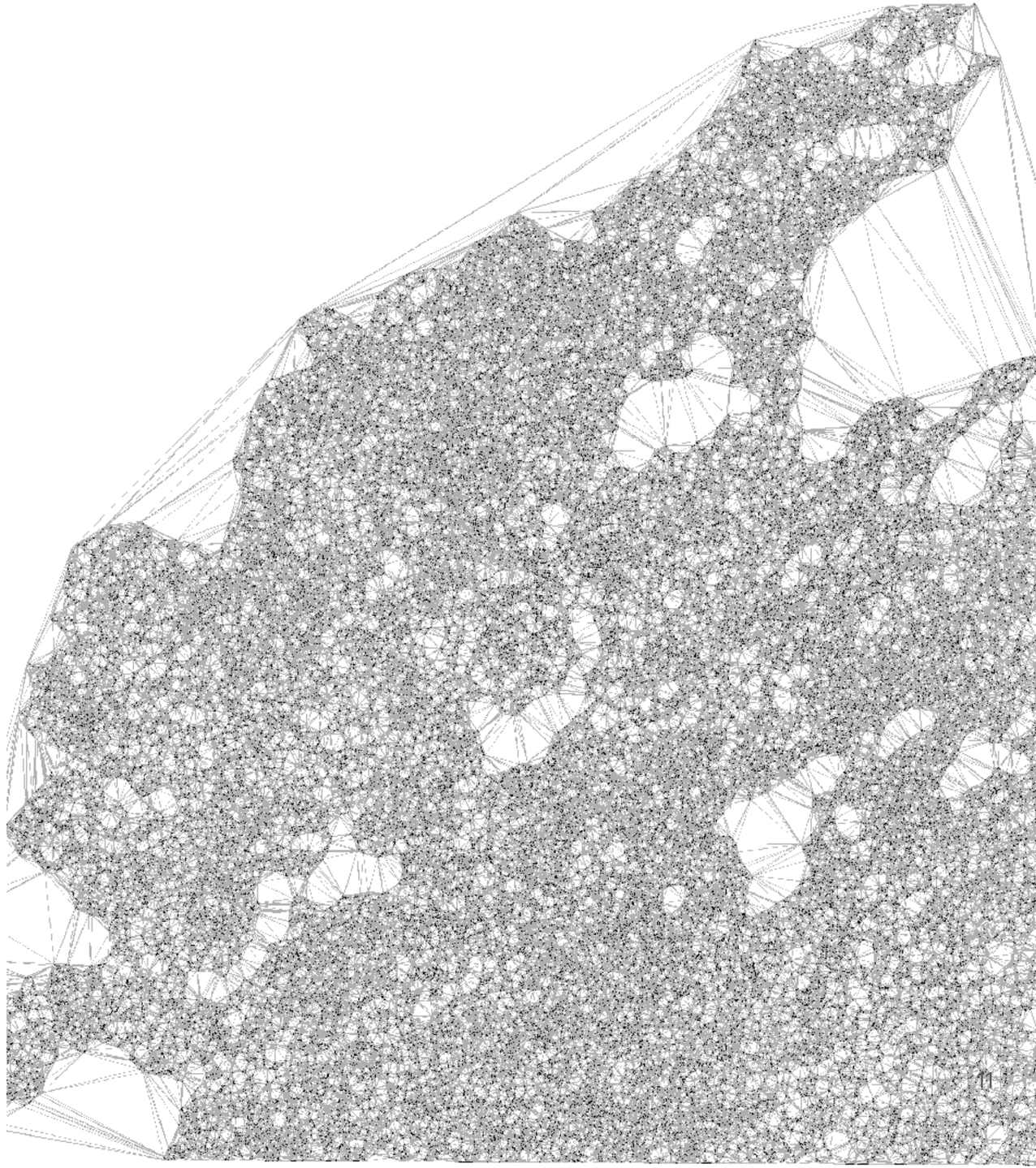
- 1. Arrangement
- 2. Labelling of the faces
- 3. Reconstruction of a multipolygon



Algorithm

1. Arrangement

- The input line segments are added to the arrangement as edges.
    - Internally, this is done using a constrained triangulation where they are added as constraints.
- With the even-odd rule, only the edges that are present an odd number of times in the input will be edges in the final arrangement.
    - When these edges are only partially overlapping, only the parts that overlap an odd number of times will be edges in the final arrangement.
- This procedure is done in two steps:
    - 1. preprocessing to eliminate identical edges that are present an even number of times (optimisation), and
    - 2. adding edges incrementally while applying an even-odd counting mechanism, which erases existing (parts of) edges when new overlapping ones are added.

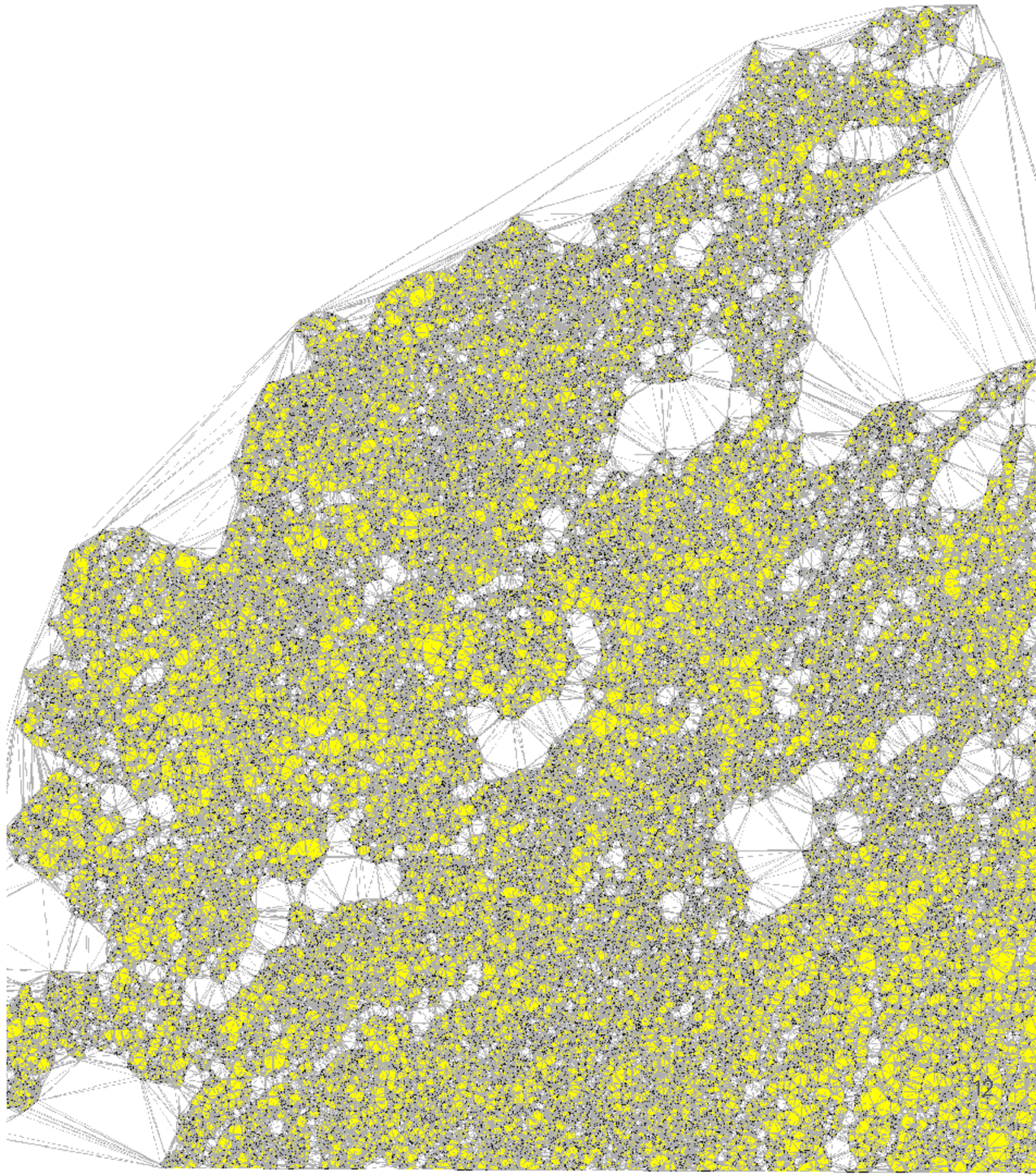


Algorithm

2. Labelling

- First, the polygon exterior is labeled. For this, all of the faces that can be accessed from the exterior without passing through an edge are labeled as exterior faces.

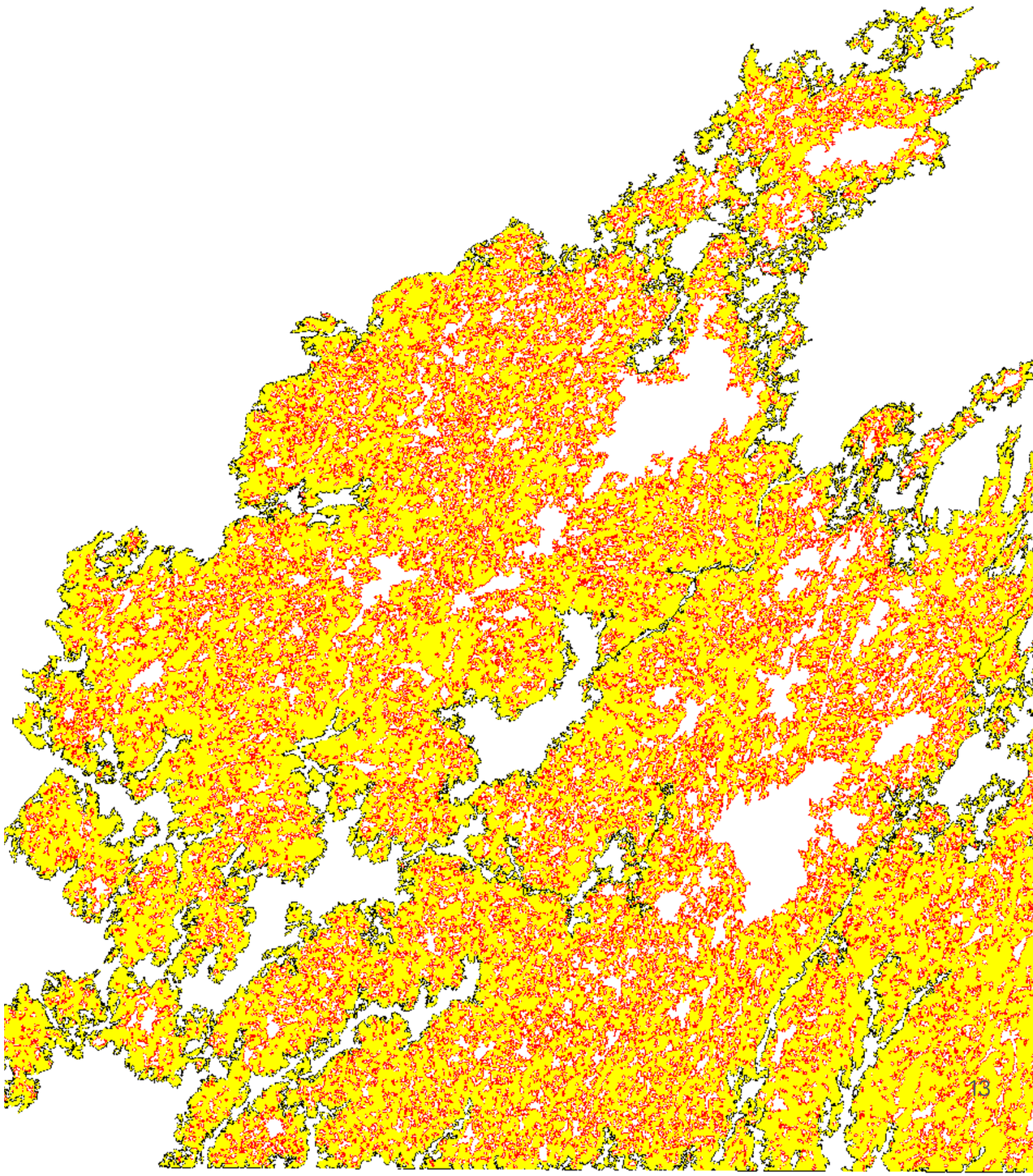
- Then, all other faces are labeled. For the even-odd rule, the label applied alternates between polygon interior and hole every time that a constrained edge is passed.

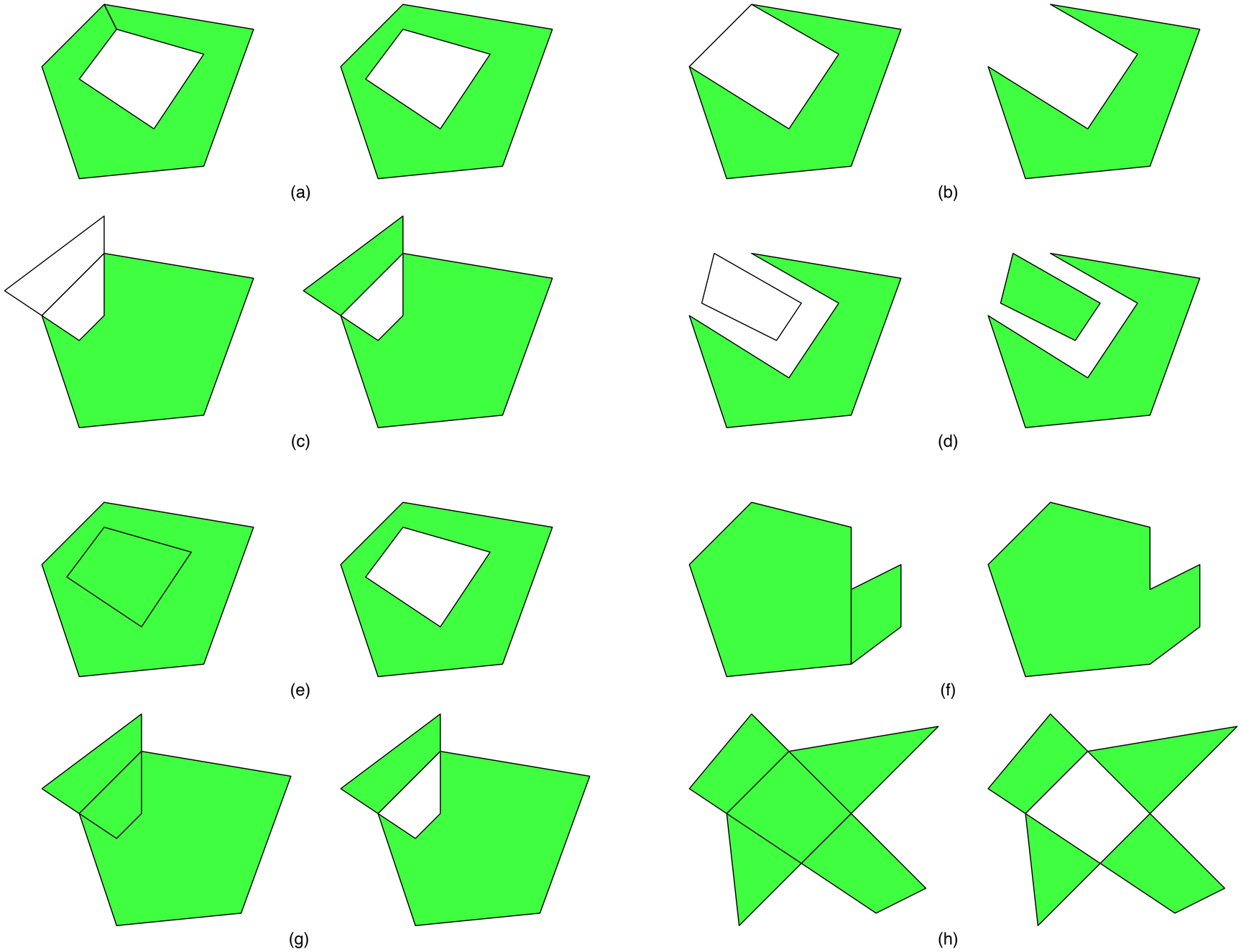


Algorithm

3. Reconstruction

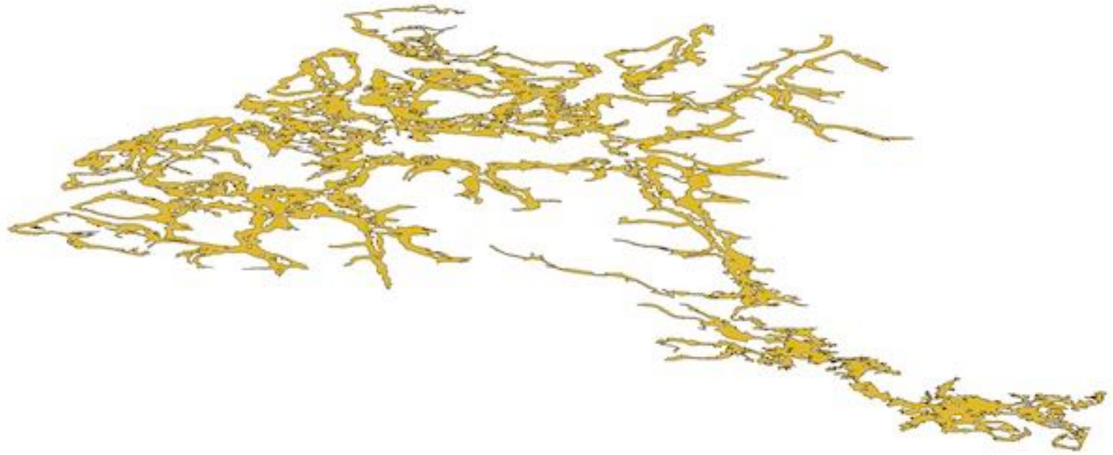
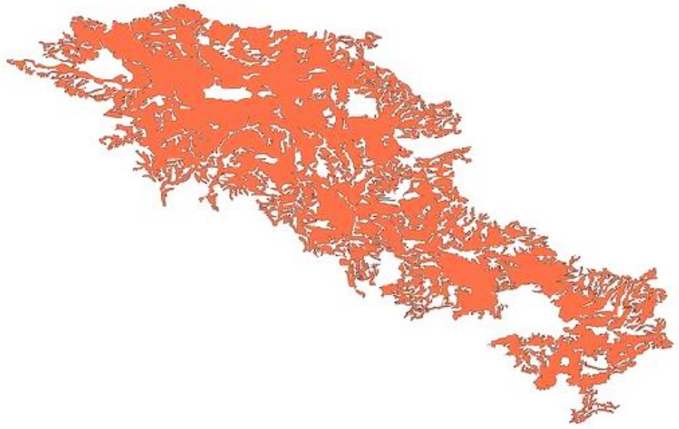
- The algorithm reconstructs the multipolygon boundary by boundary, obtaining counter-clockwise cycles for outer boundaries and clockwise cycles for inner boundaries.
- Once all boundaries have been reconstructed, the boundaries are assembled into multipolygons
  - using the face labels to knowwhich polygon with holes inner/outer boundaries belong to, and
  - using the orientation to distinguish between the outer and inner boundaries of each polygon with holes.




(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)

Performance

In practice, millions of vertices in a few seconds

Polygon	Vertices	Holes	Time
	101973	298	0.652 sec
	43925	125	0.190 sec

…but potentially quadratic intersections in pathological cases

```cpp
#include <iostream>
#include <fstream>

#include <CGAL/Exact_predicates_inexact_constructions_kernel.h>
#include <CGAL/Polygon_repair/repair.h>
#include <CGAL/IO/WKT.h>

using Kernel = CGAL::Exact_predicates_inexact_constructions_kernel;
using Point_2 = Kernel::Point_2;
using Polygon_2 = CGAL::Polygon_2<Kernel>;
using Polygon_with_holes_2 = CGAL::Polygon_with_holes_2<Kernel>;
using Multipolygon_with_holes_2 = CGAL::Multipolygon_with_holes_2<Kernel>;

int main() {
  std::ifstream in("data/bridge-edge.wkt");
  Polygon_with_holes_2 pin;
  CGAL::IO::read_polygon_WKT(in, pin);

  Multipolygon_with_holes_2 mp = CGAL::Polygon_repair::repair(pin);
  if (mp.number_of_polygons_with_holes() > 1) {
    CGAL::IO::write_multi_polygon_WKT(std::cout, mp);
  } else {
    CGAL::IO::write_polygon_WKT(std::cout, mp.polygons_with_holes()[0]);
  }

  return 0;
}
```

Other useful bits

- Multipolygons in CGAL (I/O, CGAL basic viewer, etc.)
- Function to check if a (multi)polygon *is_valid()*, although undocumented for now
- Odd-even counting of triangulation constraints
- Architecture to easily implement other repair rules in the future
- Everything integrated into CGAL testing architecture (different platforms, kernels, etc.)

Questions?