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

Ken Arroyo Ohori   Hugo Ledoux   Martijn Meijers

July 2, 2012

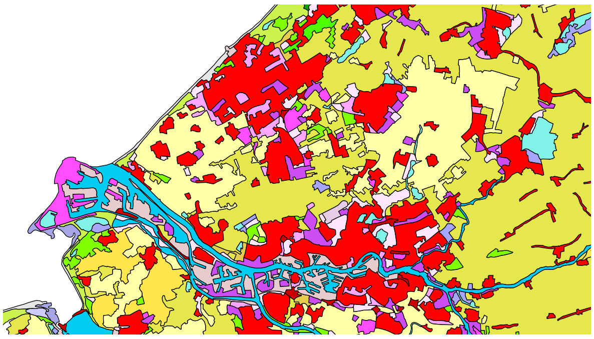


Figure 1: Part of the Corine Land Cover dataset for the region around Delft, The Netherlands.

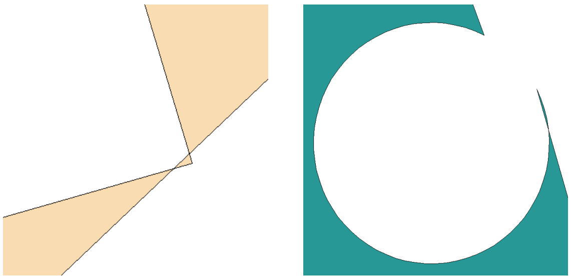


Figure 2: Two invalid (self-intersecting) polygons in a municipality dataset.

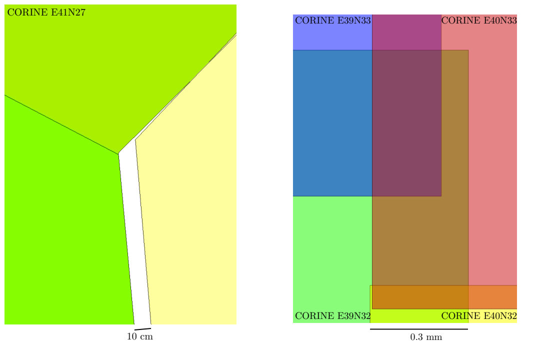
1 Introduction

Planar partitions are frequently used in GIS to model concepts such as land cover, the cadastre, or the administrative boundaries of a given country. As shown in Figure 1, a planar partition is a subdivision of a region into non-overlapping polygons. In practice, planar partitions are often represented, and stored in a computer, as a set of individual polygons to which one or more attributes are attached, and the topological relationships between polygons are not explicitly stored (shared boundaries are thus represented and stored twice). The preferred method of practitioners is representing polygons according to the *Simple Features* specification [OGC, 2006], for instance as an ESRI *Shapefile* [ESRI, 1998] or in a database, such as PostGIS[1].

Polygons are frequently built from imperfect data (e. g. line segments with overshoots and undershoots), or digitized semi-manually in a variety of CAD software, such as the example in Figure 2. Thus, invalid polygons continue to be prevalent in practice despite the existence of a variety of validation tools (see Van Oosterom et al. [2004]), causing errors such as: duplicate vertices, and unclosed or self-intersecting polygons. Similar issues occur with planar partitions. If they are stored as a set of individual polygons, then in practice errors, mistakes and inconsistencies will often be introduced when they are built, updated or exchanged. Examples of common errors are: overlapping polygons, gaps between polygons, and polygons not connected to others. Figure 3 shows examples of a gap and an overlap in the Corine dataset. Notice that such problems are often not visible at the scale that the data is usually viewed, exacerbating the problem [Laurini and Milleret-Raffort, 1994]. These errors can be, among others, due to human error, the use of floating-point arithmetic, or limited precision [Schirra, 1997]. They can have catastrophic consequences for practitioners since most software and algorithms using planar partitions as input assume that this input is valid. At best erroneous

[*] Authors' draft of an abstract to be published in the Proceedings of the 4th Open Source GIS UK Conference in Nottingham, United Kingdom

[1] `http://postgis.refractions.net/`



(a) A gap between two polygons.

(b) An overlapping region between four tiles of the dataset.

Figure 3: Examples of errors in the Corine dataset.

results are returned, at worst it causes a software failure, often without any warning to the user.

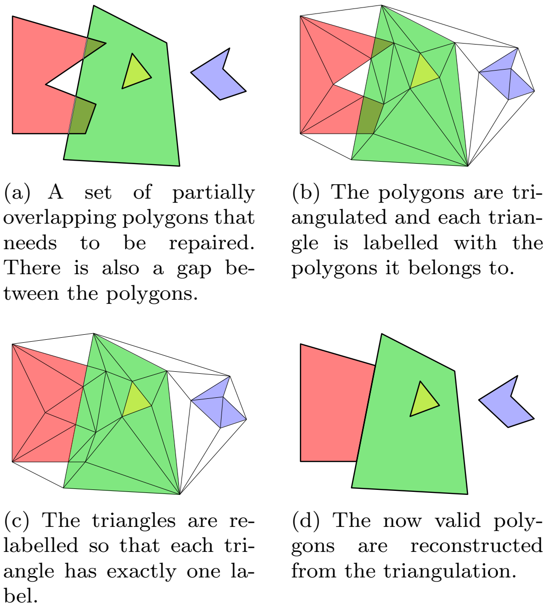
Solving this problem involves repairing errors that occur both at the polygon and planar partition levels. Doing so automatically is highly desirable, since real-world datasets can easily contain thousands of polygons and millions of vertices, sometimes resulting in hundreds of errors. Automatic polygon repair has been tackled in the past with a variety of *ad hoc* tricks (see Ramsey [2010] for an excellent overview). Meanwhile, automatic planar partition repair is usually based on snapping nearby vertices (up to a user-defined threshold). However, these approaches are limited (they cannot recover from any situation), and do not guarantee that the validity of their output. As for commercial tools (e. g. ArcGIS), they offer only semi-automatic methods.

We have developed a method—and implemented it—to automatically repair polygons and planar partitions stored according to the Simple Features specification. It uses a constrained triangulation (CT) of the polygons as a support, which is by definition a planar partition. This structure allows us to give a consistent interpretation to invalid polygons. Repair is performed by simple operations on the CT: (re)labelling the triangles, and standard graph traversal algorithms (such as depth-first search). Since vertices are never moved, we can guarantee that a given repair operation will preserve the topological consistency of both individual polygons and the entire planar partition. We give in Section 2 an overview of the method.

We have created a robust implementation of our method, and have made it open source and publicly available at `https://github.com/tudelft-gist`. Our software takes as input polygons stored according to the Simple Features specification, automatically repairs them if they contain errors, and returns a new set of valid polygons that is guaranteed to be a valid planar partition. We also report in Section 3 our experiments with several real-world datasets (some of them rather large), and we compare our method and its implementation to alternatives, both for validation and for repair.

2 Our implementations: prepair and pprepair

Our approach to automatically repair polygons and planar partitions uses a constrained triangulation (CT) as a supporting structure, and as Figure 4 indicates, it has four steps:



(a) A set of partially overlapping polygons that needs to be repaired. There is also a gap between the polygons.

(b) The polygons are triangulated and each triangle is labelled with the polygons it belongs to.

(c) The triangles are relabelled so that each triangle has exactly one label.

(d) The now valid polygons are reconstructed from the triangulation.

Figure 4: The steps to repair a planar partition.

(a) The CT of the line segments forming the boundaries of the polygon(s) is constructed. For a (valid) planar partition, each segment will be present twice (except those forming the outer boundary of the set of input polygons), and these duplicates should be ignored. When segments are found to intersect, they are split with a new point created at the intersection. Both of these operations are available in triangulation libraries, such as CGAL [CGAL, 2011], Triangle [Shewchuk, 1997] and GTS [GTS, 2006].

(b) Each triangle in the CT is labelled appropriately. For the polygon case, two labels representing its interior and exterior are used, labelling by starting from a triangle known to be in the exterior of the polygon, e. g. using the "big triangle" concept [Facello, 1995; Liu and Snoeyink, 2005], and switching labels when passing through a constrained edge. For the planar partition case, each triangle is labelled with the the polygon inside which it is located, so that gap triangles have zero labels, and overlap triangles have two or more.

(c) To repair a planar partition, triangles are relabelled to ensure that each triangle has exactly one label. This operation is based on local criteria, such as the length of its boundary with adjacent features, which can be done triangle by triangle, or by first selecting regions of contiguous triangles with the same set of labels.

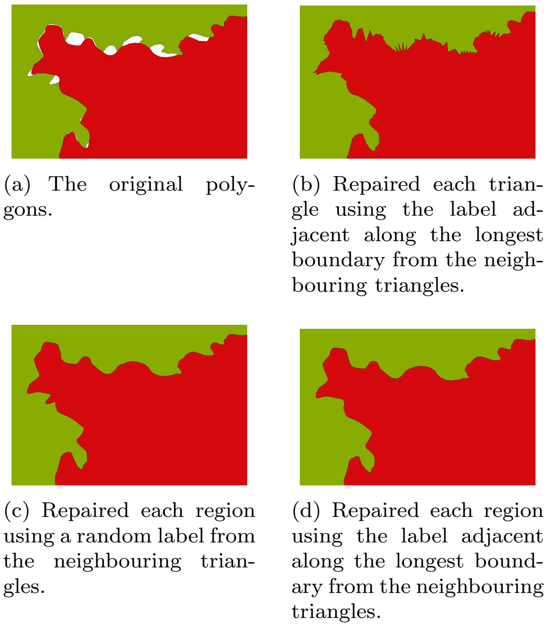
(d) The polygon(s) are extracted from the triangulation, one by one and according to their labels. To do this, we use a depth-first clockwise search, and a stack-based algorithm that generates separate closed rings for the outer boundary and each of the inner boundaries.

Based on this process, we have implemented two software tools that are able to automatically repair polygons and planar partitions. These are called **prepair** (for polygons), and **pprepair** (for planar partitions), which are open source and freely available under a BSD license (although we use GPL libraries).

Our implementation is written in C++ and uses the OGR Simple Features Library[2] for input and output, and CGAL's[3] 2D Triangulations package to triangulate the polygons. It can use either floating point or exact (lazy evaluated) representations of the points in the triangulation. The different classes in our software and how they use OGR and CGAL classes are shown in Figure 5.

More details are available in Ledoux et al. [2012] and in Arroyo Ohori et al. [2012].

We have also implemented different repair operations, which are based on local properties of a triangle, a region of triangles with the same labels, or using a list of feature classes with different priorities. Figure 6 presents some of the available repair operations.


(a) The original polygons.

(b) Repaired each triangle using the label adjacent along the longest boundary from the neighbouring triangles.

(c) Repaired each region using a random label from the neighbouring triangles.

(d) Repaired each region using the label adjacent along the longest boundary from the neighbouring triangles.

Figure 6: Different repair operations used in the two polygons for the Arribes del Duero Natural Park in Spain (red) and the International Douro Natural Park in Portugal (green). All of them can be considered best by a certain criterion, like preserving the area ratio between the two polygons (b), smoothness of the boundary (d), or a balance between the two (c).

[2] http://www.gdal.org/ogr/

[3] Computational Geometry Algorithms Library: http://www.cgal.org/

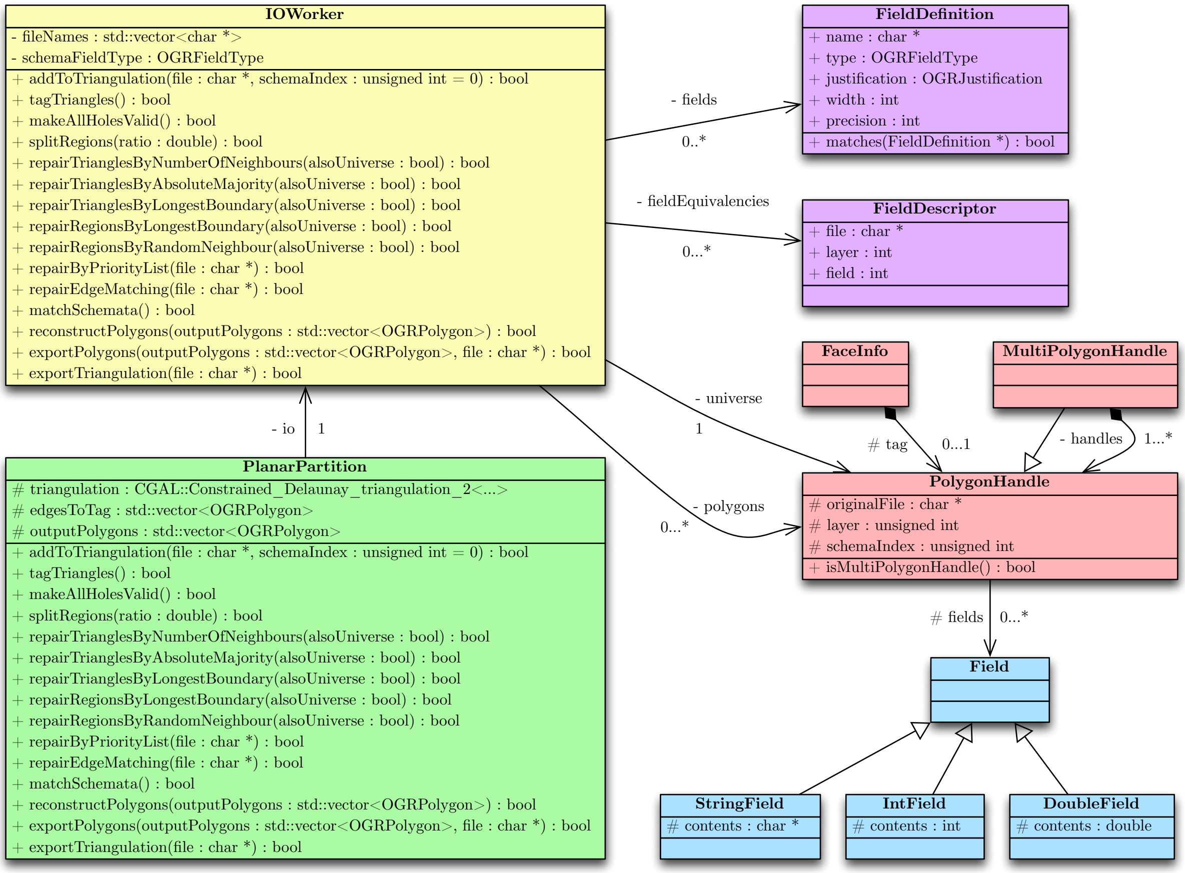
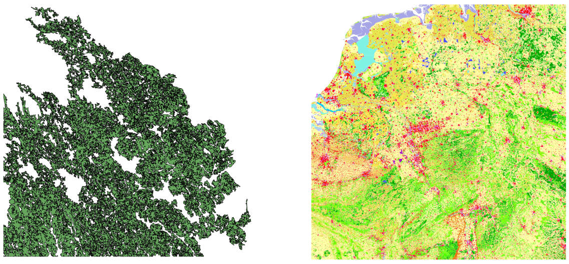

IOWorker
- fileNames : std::vector<char *>
- schemaFieldType : OGRFieldType
+ addToTriangulation(file : char *, schemaIndex : unsigned int = 0) : bool
+ tagTriangles() : bool
+ makeAllHolesValid() : bool
+ splitRegions(ratio : double) : bool
+ repairTrianglesByNumberOfNeighbours(alsoUniverse : bool) : bool
+ repairTrianglesByAbsoluteMajority(alsoUniverse : bool) : bool
+ repairTrianglesByLongestBoundary(alsoUniverse : bool) : bool
+ repairRegionsByLongestBoundary(alsoUniverse : bool) : bool
+ repairRegionsByRandomNeighbour(alsoUniverse : bool) : bool
+ repairByPriorityList(file : char *) : bool
+ repairEdgeMatching(file : char *) : bool
+ matchSchemata() : bool
+ reconstructPolygons(outputPolygons : std::vector<OGRPolygon>) : bool
+ exportPolygons(outputPolygons : std::vector<OGRPolygon>, file : char *) : bool
+ exportTriangulation(file : char *) : bool
- fields
0..*
FieldDefinition
+ name : char *
+ type : OGRFieldType
+ justification : OGRJustification
+ width : int
+ precision : int
+ matches(FieldDefinition *) : bool
- fieldEquivalencies
0...*
FieldDescriptor
+ file : char *
+ layer : int
+ field : int
- universe
1
FaceInfo
MultiPolygonHandle
# tag
0...1
- handles
1...*
- io
1
PlanarPartition
# triangulation : CGAL::Constrained_Delaunay_triangulation_2<...>
# edgesToTag : std::vector<OGRPolygon>
# outputPolygons : std::vector<OGRPolygon>
+ addToTriangulation(file : char *, schemaIndex : unsigned int = 0) : bool
+ tagTriangles() : bool
+ makeAllHolesValid() : bool
+ splitRegions(ratio : double) : bool
+ repairTrianglesByNumberOfNeighbours(alsoUniverse : bool) : bool
+ repairTrianglesByAbsoluteMajority(alsoUniverse : bool) : bool
+ repairTrianglesByLongestBoundary(alsoUniverse : bool) : bool
+ repairRegionsByLongestBoundary(alsoUniverse : bool) : bool
+ repairRegionsByRandomNeighbour(alsoUniverse : bool) : bool
+ repairByPriorityList(file : char *) : bool
+ repairEdgeMatching(file : char *) : bool
+ matchSchemata() : bool
+ reconstructPolygons(outputPolygons : std::vector<OGRPolygon>) : bool
+ exportPolygons(outputPolygons : std::vector<OGRPolygon>, file : char *) : bool
+ exportTriangulation(file : char *) : bool
- polygons
0...*
PolygonHandle
# originalFile : char *
# layer : unsigned int
# schemaIndex : unsigned int
+ isMultiPolygonHandle() : bool
# fields
0...*
Field
StringField
# contents : char *
IntField
# contents : int
DoubleField
# contents : double

Figure 5: A simplified UML diagram of pprepair.

3 Experiments

We have tested our software with several datasets, such as Corine[4], Mexican and Canadian land cover data, and edge-matching datasets.

Our software has been able to successfully repair every dataset that we have tested, including some very large ones. Figure 7 shows two such examples, with the time and memory it took to repair them. These tests were run on a Intel Core 2 Duo 2.66 GHz MacBook Pro under Mac OS X 10.7.3.



(a) The largest polygon of the Corine dataset, consisting of 1 189 903 vertices and having 7672 holes. It was repaired in 1m4s using 940 MB of memory.

(b) 25 tiles of the Corine dataset, consisting of 105 712 polygons and 5 122 108 vertices. It was repaired in 6m10s using 2.34 GB of memory.

Figure 7: Examples of large datasets successfully repaired by prepair and pprepair.

References

K. Arroyo Ohori, H. Ledoux, and M. Meijers. Validation and automatic repair of planar partitions using a constrained triangulation. *Journal of Photogrammetry, Remote Sensing and Geoinformation Processing*, 2012. Accepted for publication.

CGAL. *CGAL 3.8 User and Reference Manual*. CGAL Editorial Board, 2011.

ESRI. Shapefile technical description. White paper, ESRI, July 1998.

M. A. Facello. Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. *Computer Aided Geometric Design*, 12(4):349–370, 1995.

GTS. *GTS Library Reference Manual*, 2006. URL `http://gts.sourceforge.net/reference/book1.html`.

R. Laurini and F. Milleret-Raffort. Topological reorganization of inconsistent geographical databases: A step towards their certification. *Computers & Graphics*, 18(6):803–813, December 1994.

H. Ledoux, K. Arroyo Ohori, and M. Meijers. Automatically repairing invalid polygons with a constrained triangulation. In *Proceedings of the 15th AGILE International Conference on Geographic Information Science*, 2012.

Y. Liu and J. Snoeyink. The “far away point” for Delaunay diagram computation in $\mathbb{E}^d$. In *Proceedings 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 236–243, Seoul, Korea, 2005.

OGC. *OpenGIS Implementation Specification for Geographic Information - Simple Feature Access - Part 1: Common Architecture*, 1.2.0 edition, October 2006.

P. Ramsey. PostGIS: Tips for power users. Presentation at the FOSS4G 2010 Conference, Barcelona, Spain, 2010. `http://s3.opengeo.org/postgis-power.pdf`.

S. Schirra. *Precision and Robustness in Geometric Computations*, volume Algorithmic Foundations of Geographic Information Systems of *Lecture Notes in Computer Science*, chapter 9, pages 255–287. Springer Berlin / Heidelberg, 1997.

J. R. Shewchuk. *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA, 1997.

P. van Oosterom, W. Quak, and T. Tijssen. About invalid, valid and clean polygons. In P. F. Fisher, editor, *Developments in Spatial Data Handling—11th International Symposium on Spatial Data Handling*, pages 1–16. Springer, 2004.

[4] Corine is a European land cover dataset. It is freely available at `http://www.eea.europa.eu/data-and-maps/data/corine-land-cover-2000-clc2000-seamless-vector-database`.