Validation of planar partitions using constrained triangulations*

Hugo Ledoux and Martijn Meijers
Delft University of Technology (OTB—section GIS Technology)
Jaffalaan 9, 2628BX Delft, the Netherlands
{h.ledoux-b.m.meijers}@tudelft.nl

March 12, 2010

Planar partitions—full tessellations of the plane into non-overlapping polygons—are frequently used in GIS to model concepts such as land cover, cadastral parcels or administrative boundaries. Since in practice planar partitions are often stored as a set of individual objects (polygons) to which attributes are attached (e.g. stored with a *shapefile*), and since different errors/mistakes can be introduced during their construction, manipulation or exchange, several inconsistencies will often arise in practice. The inconsistencies are for instance overlapping polygons, gaps and unconnected polygons. We present in this paper a novel algorithm to validate such planar partitions. It uses a constrained triangulation as a support for the validation, and permits us to avoid different problems that arise with existing solutions based on the construction of a planar graph. We describe in the paper the details of our algorithm, our implementation, how inconsistencies can be detected, and the experiments we have made with real-world data (the CORINE2000 dataset).

1 INTRODUCTION

Planar partitions are frequently used in GIS to model concepts such as land cover, the cadastral parcels or the administrative boundaries of a given area. As shown in Figure 1, a planar partition is a full tessellation of the plane into non-overlapping polygons. The spatial extent is partitioned

*Preliminary version of a paper that will be published in the proceedings of SDH 2010, held in Hong Kong.

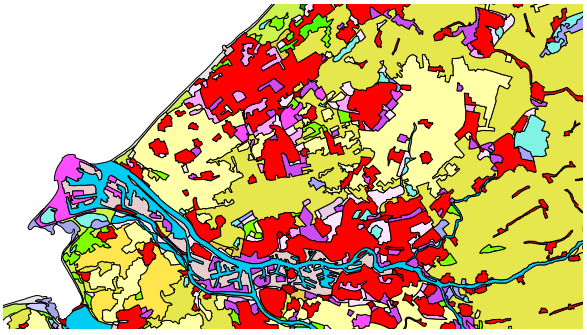


Figure 1: Part of the CORINE2000 dataset for a part of the Netherlands.

into polygons, and every location must be covered by one and only one polygon (gaps are thus not allowed). In GIS-related disciplines planar partitions, like other geographical phenomena, 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 stored. The preferred method is with the *Simple Features* paradigm, which is an international standard (OGC, 2006); the *de facto* standard ESRI's shapefile and most databases (e.g. PostGIS) are based on this standard. We discuss in Section 2 details of the Simple Features paradigm that complicate the representation of planar partitions.

If a planar partition is stored as a set of individual polygons, then in practice errors, mistakes and inconsistencies will often be introduced when the planar partition is constructed, updated or exchanged. The inconsistencies most likely to occur are: (i) overlapping polygons (e.g. slivers); (ii) gaps between polygons; (iii) polygons not connected to the others.

In this paper we present a novel algorithm to *validate* such a planar partition, i.e. given a set of polygons stored with the Simple Features paradigm, our algorithm verifies if this set forms a planar partition, or not. As explained in Section 2 different solutions currently exist, these are based on the construction of the planar graph of the polygons and on the use of geometrical and topological validation rules. The solution we propose—using a constrained triangulation as a supporting structure for validation—is described in Section 3 and has in our opinion several advantages over existing methods. We report in Section 4 on our implementation of the algorithm (it uses the stable and fast triangulator of CGAL[1]) and on the experiments we have made with the CORINE Land Cover 2000 dataset. Finally, we discuss the advantages of our method in Section 5.

2 RELATED WORK

Validation of planarity of area partitions has its roots in the definition of what is a valid surface representation for real world features (i.e. a *polygon*). In this section we will first review the Simple Features specification (SFS) that describes what is a valid polygon and secondly how a set of polygons can be validated, so that it forms a planar partition.

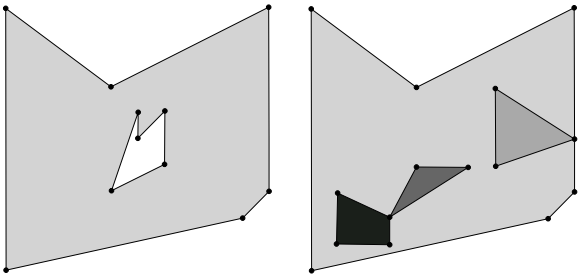
2.1 Simple Features

The Simple Features specification is a recognised and used international standard for the storage and access of geographical objects in vector format such as points, lines and polygons. SFS defines a polygon by stating that: "A Polygon is a planar Surface defined by 1 exterior boundary and 0 or more interior boundaries. Each interior boundary defines a hole in the Polygon." (OGC, 2006). In the specification 6 assertions are given that together define a valid polygon. Essential for a valid polygon is that the boundaries of the polygon must define one connected area (each point inside the polygon can be reached through the interior of the polygon from any other point inside the polygon). Additionally, a polygon can contain holes. We say that the exterior boundary of the polygon is the *outer ring*, and a hole is an *inner ring*. As shown in Figure 2, these holes can be filled by one or more polygons, which we call islands. Island polygons can recursively contain holes which are filled by islands. Observe also that holes are allowed to interact with each others and the outer boundary under certain circumstances, e.g. they are allowed to touch at one point (as in Figure 2b), as long as the interior of the polygon stays one connected area.

The polygons can be represented either in text (well-known text–WKT) or binary (well-known binary–WKB) formats. Each polygon is stored independently from other polygons; even those adjacent (it is not possible to store topological relationships between the polygons).

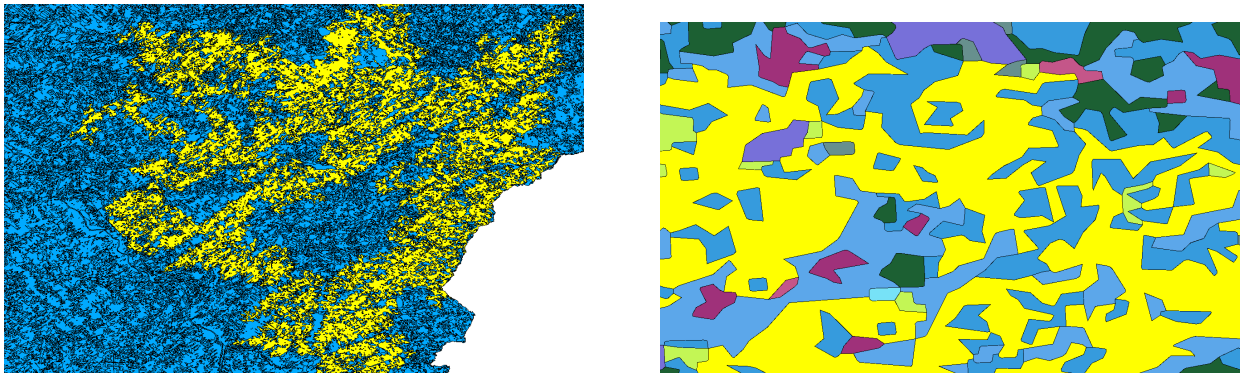
Integrity checking of an individual polygon entails checking whether the polygon fulfils the above definition. A naive way of validity checking could be based on checking each segment of each linear

[1]The Computational Geometry Algorithms Library: `http://www.cgal.org`



(a) (b)

Figure 2: Two examples of polygons and their WKT. **(a)** One polygon with one hole. **(b)** Another polygon with three holes, and each of them is filled with an island. Observe that two holes/islands touch each other at one point, and that one hole touch the outer boundary of the polygon at one location.



(a) (b)

Figure 3: **(a)** One polygon (in yellow) from the Canadian Land Cover Map. Its outer boundary has around 35000 points and it has around 3200 holes. **(b)** Zoom in on this polygon, observe that holes are filled by islands, and that these touch other rings at several points.

ring with all other segments of the polygon for intersections, which is apparently quite costly with respect to computation.

In this work, we have adopted the Simple Feature definition for what we consider is a valid polygon. In the paper by Oosterom *et al.* it was shown in an experiment with different systems and a set of 37 'killer polygons' that in practice the use of this definition is not self-evident and that different products have different interpretations of what is a valid polygon. The authors concluded that "the consistent use of a polygon definition is not yet a reality" (van Oosterom et al., 2002). The validation of one polygon according to the SFS specification has since then found its way into software implementations and is easily possible with different libraries, e.g. GEOS [2] and JTS [3] being two of those (open source) libraries.

2.2 Planar partitions

Planar partitions, such as the CORINE2000 dataset, are freely available in shapefile format where each polygon has one value attached (its code for the land cover). Polygons in such datasets are usually fairly complex (see for instance Figure 3) and the number of polygons is generally very large. The specifications of the dataset states that all polygons form a planar partition, but in practice this

[2]Geometry Engine Open Source: `trac.osgeo.org/geos`
[3]Java Topology Suite: `www.vividsolutions.com/jts/jtshome.htm`

is not the case (see Section 4).

Having a definition for what is a valid polygon alone is not enough for certain applications: for those applications it is necessary to define what a valid *set* of polygons is. Therefore, the *disjoint* spatial relation has to be introduced. Two polygons are said to be disjoint, when their interior does not overlap (i.e. does not have any spatial relation). A brute force approach to enforce planarity of a partition with independent, loose-lying polygons is rather cumbersome: it is necessary to check whether each polygon its interior is disjoint with all other polygons, which means a lot of computation (in the order of $O(n^2)$ where n is the number of features to be checked). Furthermore detection of holes between polygons in the partition is only possible by obtaining the union of all individual polygons, which again is computational intensive. As final remark: this also assumes that each individual polygon has already been checked.

In a series of papers on the topic of formal and correct spatial object modelling (Plümer and Gröger, 1996; Gröger and Plümer, 1997; Plümer and Gröger, 1997), a set of mathematical axioms is given for checking the 'validity' of a map, i.e. a collection of polygons. The axioms that Gröger and Plümer form are a correct and complete set of rules to enforce correctness for all polygons in a map together with their adjacency relationships. Checking of the axioms has static and dynamic aspects. Static integrity checking "concerns the question whether a given database as a whole is consistent" (Gröger and Plümer, 1997). An example of the dynamic aspects—how to keep a (in this case cadastral) dataset consistent under updates or transactions—is found in (Matijević et al., 2008). Our approach only examines the state of a geographic dataset as a whole (thus enforcing the integrity rules for a given set of polygons). For the remaining part of this paper, we will focus on static integrity checking.

Furthermore, it is important to note that Plümer and Gröger (1997) base their axioms on concepts from graph theory, but they also highlight the fact that a graph-based approach alone is not enough: the graph has to be augmented with geometrical knowledge (each vertex has geometry attached, i.e. the coordinates of points have to be stored). Validation is thus underpinned by both geometrical and topological concepts and systems thus have to deal with those two concepts at the same time.

For validating all polygons in a dataset in a single operation, it is necessary to perform a conversion to a graph-based description, which is consecutively checked for consistency (following a set of rules similar to the axioms described by Gröger and Plümer). For this conversion different approaches are available (Shamos and Hoey, 1976; van Roessel, 1991). Implementation of this conversion to a graph-based representation is sometimes difficult, especially if the polygon contains holes. The graph of the boundary is then unconnected and extra machinery is necessary to still represent the knowledge on holes in the graph structure. The fact that holes are also allowed to touch complicates the task of validation even further: holes are supposed to form an unconnected planar graph, but if they touch the graph is connected.

3 Validation with the constrained triangulation

Our approach to validation of planar partitions uses a constrained triangulation (CT) as a supporting structure because, as explained in Section 3.1, CTs are by definition planar partitions. The workflow of our approach is as follows:

1. the CT of the input segments forming the polygons is constructed;
2. each triangle in the CT is flagged with the ID of the polygon inside which it is located;
3. problems are detected by identifying triangles having no IDs, and by verifying the connectivity between triangles.

The flagging and the verification of the connectivity of the input polygons is performed by using graph-based algorithms on the dual graph of the CT.

We describe in this section the concepts needed and we give a detailed description of the different steps. It should be noticed that we assume that each input polygon to our approach is individually valid (as explained in Section 2 this is an easy task and tools are readily available).

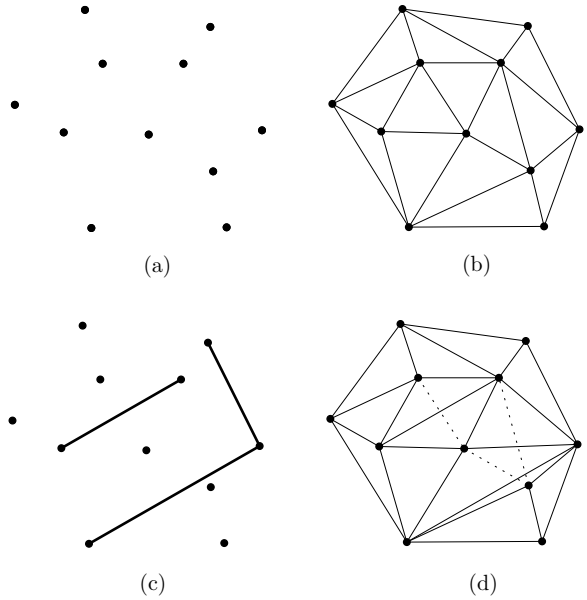


Figure 4: **(a)** A set S of points in the plane. **(b)** A triangulation of S; the union of all the triangles forms conv(S). **(c)** The set S with 3 constrained segments. **(d)** The constrained triangulation of the set of points and segments. The dashed lines are the edges of the triangulation of S that are removed since they are not conform to the input segments.

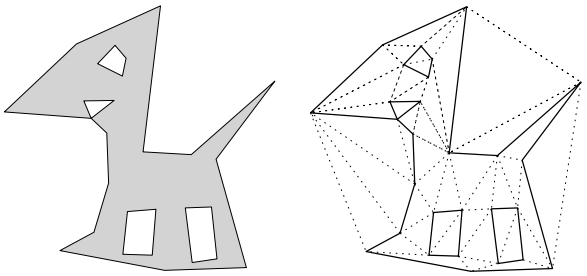


Figure 5: **(a)** A polygon with 4 holes. **(b)** The constrained triangulation of the segments of this polygon.

3.1 Constrained triangulation

A triangulation decomposes an area into triangles that are non-overlapping. As shown in Figure 4a–b, given a set S of points in the plane, a triangulation of S will decompose its convex hull, denoted conv(S). It is also possible to decompose the convex hull of a set T where points and straight-line segments are present, with a constrained triangulation (CT). In CT(T) every input segment of T appears as an edge in the triangulation (see Figure 4c–d).

If T contains segments forming a loop (which defines a polygon), it permits us to triangulate the interior of this loop (i.e. a triangulation of the polygon). It is known that any polygon (also with holes) can be triangulated without adding extra vertices (de Berg et al., 2000; Shewchuk, 1997). Figure 5 shows an example.

In our approach, the triangulation is performed by constructing a CT of all the segments representing the boundaries (outer + inner) of each polygon. If the set of input polygons forms a planar partition, then each segment will be inserted twice (except those forming the outer boundary of the set of input polygons). This is usually not a problem for triangulation libraries because they ignore points and segments at the same location (as is the case with the solution we use, see Section 4).

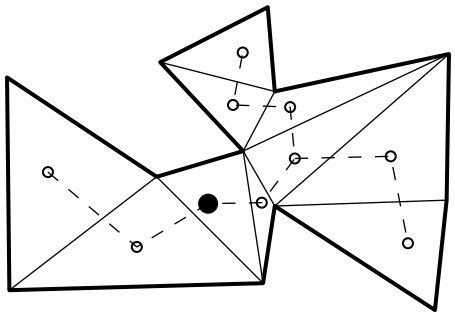


Figure 6: One polygon (thick lines) with its triangulation (normal black lines). The dual graph of the triangulation is drawn with dashed lines, and the filled black point is the centroid of the polygon from where the walk starts.

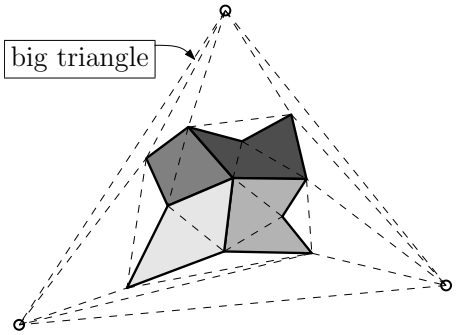
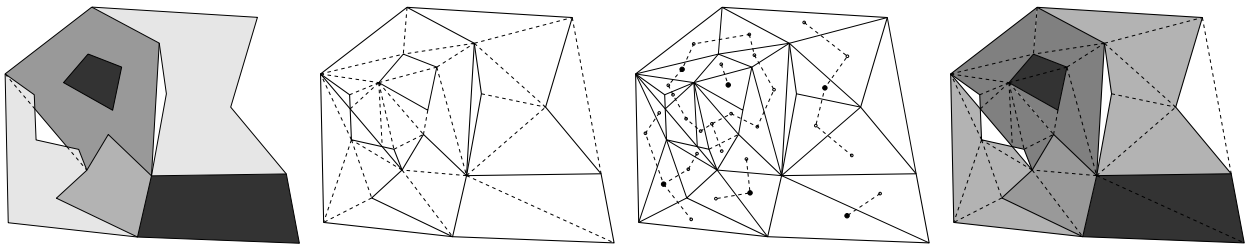


Figure 7: The 4 input polygons are triangulated and are inside the big triangle. A walk from one location outside the 4 polygons would appropriately flag as "universe" the 4 triangles inside the convex hull of the 4 polygons.

3.2 Flagging triangles

Flagging triangles means assigning the ID of each polygon to the triangles inside that polygon (the triangles that decompose the polygon). To assign this ID, we first compute one point inside each polygon. This point is what we subsequently call the "centroid" — observe here that this cannot be always the geometric centroid of the polygon as this could be outside the polygon. Our algorithm finds a location inside the polygon and makes sure that this location is not inside one of the holes of the polygon. Then for each centroid c we identify the triangle that contains c, and we start a "walk" on the dual graph of the triangulation, as shown in Figure 6. The walk is a depth-first search (DFS) on the dual graph, and observe that constrained edges in the triangulation will act as blockers for the walk. Observe also that islands are not a problem (see Figure 8c).

Big triangle. To appropriately flag all the triangles of the CT (those inside the convex hull of the input points/segments but not inside an input polygon) we exploit one particularity of libraries to compute triangulation: the so-called "big triangle", which is also being called the "far-away point" (Liu and Snoeyink, 2005). Many implementations indeed assume that the set S of points is entirely contained in a big triangle τ_{big} several times larger than the range of S. Figure 7 illustrates the idea. With this technique the construction of the CT is always initialised by first constructing τ_{big}, and then the points/segments are inserted. Doing this has many advantages, and is being used by several implementations (Facello, 1995; Mücke, 1998; Boissonnat et al., 2002). To assign an ID "universe" to the triangles, we simply start at one triangle incident to one vertex of τ_{big} and perform the same walk as for the other polygons.



(a) (b) (c) (d)

Figure 8: **(a)** Six polygons form the input planar partition. **(b)** The constrained triangulation of the boundaries of the input polygons. **(c)** The dual graph of the triangles is drawn with dashed lines; the dark points are the points from which the walk in each polygon starts. **(d)** The result contains triangles that not flagged (white triangles). The white triangle on the right is not a problem since it is a "universe" triangle.

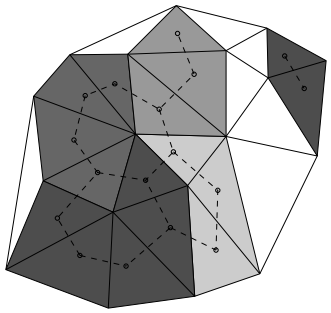


Figure 9: Five polygons, with one unconnected to the other ones. The dual graph for the flagged triangles is shown in with dashed lines.

3.3 Identifying problems

If the set of input polygons forms a planar partition then all the triangles will be flagged with one and only one ID. Notice that because of the big triangle, triangles outside the spatial extent of the planar partitions will be flagged as "universe". Notice also that if a polygon contains a hole, then for the planar partition to be valid this hole must be filled completely by another polygon (an island).

If there are gaps and/or overlaps in the input planar partition then some triangles will not be flagged. We can detect these easily by verifying the IDs. Figure 8 illustrates one input planar partition that contains 6 polygons; notice that one has an island and that some polygons overlap and that there are also gaps. The walk starting from each centroid is shown in Figure 8c, and the resulting flagging of triangles is shown in 8d (the grey shadings represent the IDs). When 2 or more polygons overlap then depending on the location of the centroids some triangles will not be flagged (because the constrained edges block the walks).

Another problem that could arise is when the union of the input polygons forms more than one polygon. Figure 9 shows one example with 5 input polygons: 4 of them form a valid planar partition but one is not connected to the others (thus the 5 polygons do not form a planar partition). We solve that problem by starting a walk from any centroid, but that walk is not stopped by the constrained, only by the triangles flagged as "universe". The connectivity problem simply boils down to ensuring that all the triangles flagged with an ID other than "universe" can be reached.

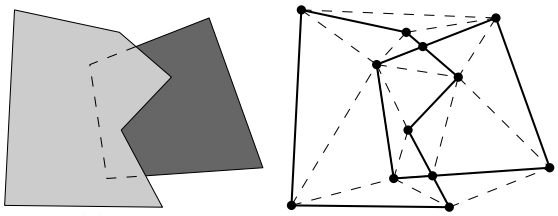


Figure 10: **(a)** Two overlapping polygons. **(b)** CGAL's constrained triangulation of the polygons.

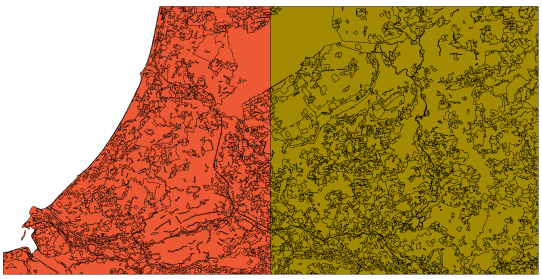


Figure 11: CORINE2000's tiles E39N32 and E40N32.

4 IMPLEMENTATION AND EXPERIMENTS

We implemented the algorithm described in this paper with the Python language[4]. Our implementation reads as input either a *shapefile* or a set of WKTs, and tells the user what problems are present in the input polygons (if any).

For the constrained triangulation, we rely entirely on the implementation of CGAL (we use the Python bindings of CGAL[5]). Each segment of the input polygons is inserted incrementally in the CT. When 2 segments are identical, the second one is simply ignored. Since the input if formed of individual polygons, it is faster (and simpler) to rely on the spatial indexing scheme of CGAL to detect the duplicate edges than to pre-process them with an auxiliary data structure. It should be noticed that we use the default tolerance in CGAL to determine if 2 points are at the same location.

We also rely on CGAL for ensuring that a valid triangulation is formed when 2 or more polygons overlap. As shown in Figure 10, if 2 polygons overlap their segments will intersect (which would not be a valid planar graph). However, CGAL has built-in operations to calculate the intersection of 2 segments and to create new sub-segments.

We have tested our implementation with different parts of the CORINE2000 dataset. This is a dataset modelling the land cover for the whole of Europe, and it is freely available[6]. The dataset is divided into tiles and each tile can be downloaded as a shapefile. Although the specifications of CORINE2000 state that the polygons form a planar partition and that validation rules are used, we found several errors.

One example is when creating one planar partition from two adjacent tiles, as shown in Figure 11. The process of tiling the whole dataset has obviously introduced errors because several sliver polygons were detected during our experiments. We have also found one case where a polygon had been obviously "shifted" manually by a user (see Figure 12).

[4] `http://www.python.org`
[5] `http://cgal-python.gforge.inria.fr`
[6] More information can be found on `http://www.eea.europa.eu/themes/landuse/clc-download`

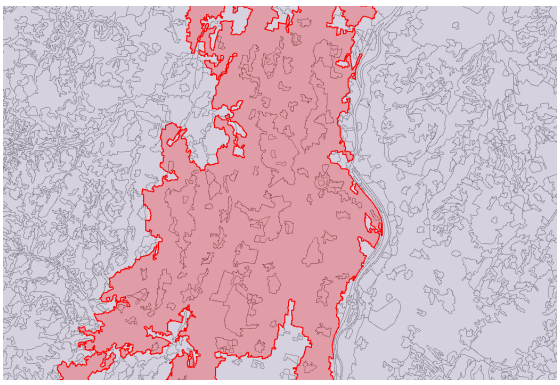


Figure 12: A polygon manually shifted (from CORINE2000 tile E41N27) – it is overlapping with neighbours on one side and gaps are present on the opposite side.

5 DISCUSSION AND CONCLUSIONS

The problem of validating a planar partition stored with Simple Features is theoretically a simple one: construct the planar graph of the input, and define a set of geometric and topological validation rules. Unfortunately, the implementation of a planar graph construction algorithm and of the validation rules is far from being trivial (especially when the input polygons contain holes) and can often not scale to big datasets containing millions of polygons.

We have presented in this paper a new algorithm and we have successfully implemented it. Our approach solves most of the current problems and has in our opinion several advantages:

1. The algorithm is simple and can be implemented easily over a CT library such as CGAL. The only things needed are: (i) to be able to add attributes to triangles (for the IDs); (ii) having access to the data structure. All the validation rules simply boil down to flagging triangles and graph-based searches.

2. The holes/islands inside polygons are easily handled by the CT. No additional data structure or special mechanisms are necessary, as is the case with planar graph approaches.

3. The implementation can be built over well-known and optimised CT implementations, which are fast and can handle millions of objects. It is known that triangulations of several millions points can be managed in main memory (Amenta et al., 2003; Blandford et al., 2005).

4. If problems are present in the input, we believe the CT could be used to *automatically repair* the planar partition. That would simply involve (re)flagging the IDs of problematic triangles (based on some user-defined rules) and then "following" the boundaries between IDs to reconstruct polygons and give them back to the user in Simple Features format. We see great potential for such an application.

5. Apart from static integrity checking, our approach could be used for keeping a dataset consistent under a set of edits (dynamic checking). The CT can then be used to locally check the validity of an update.

For future work, we plan on implementing the algorithm in C++ to be able to scale to massive datasets, and we also plan on working on the automatic repairing and incremental updates with the help of the CT. Finally, the ideas presented in this paper are all valid in higher dimensions and we plan on implementing them for constrained tetrahedralization (Si, 2004), which would permit us to validate 3D city models for instance.

6 ACKNOWLEDGEMENTS

We would like to thank Gustavo Adolfo Ken Arroyo Ohori for proof-reading this document and suggesting useful improvements.

References

Nina Amenta, Sunghee Choi, and Günter Rote. Incremental constructions con BRIO. In *Proceedings 19th Annual Symposium on Computational Geometry*, pages 211–219, San Diego, USA, 2003. ACM Press.

Daniel K. Blandford, Guy E. Blelloch, David E. Cardoze, and Clemens Kadow. Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24, 2005.

Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19, 2002.

Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry: Algorithms and applications*. Springer-Verlag, Berlin, second edition, 2000.

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

Gerhard Gröger and Lutz Plümer. Provably correct and complete transaction rules for GIS. In *Proceedings 5th ACM international workshop on Advances in geographic information systems*, pages 40–43, New York, NY, USA, 1997.

Yuanxin Liu and Jack 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.

Hrvoje Matijević, Zvonko Biljecki, Stipica Pavičić, and Miodrag Roić. Transaction processing on planar partition for cadastral application. In *Proceedings FIG Working Week 2008—Integrating Generations*, Stockholm, Sweden, 2008.

Ernst P. Mücke. A robust implementation for three-dimensional Delaunay triangulations. *International Journal of Computational Geometry and Applications*, 8(2):255–276, 1998.

OGC. OpenGIS implementation specification for geographic information—simple feature access. Open Geospatial Consortium inc., 2006. Document 06-103r3.

Lutz Plümer and Gerhard Gröger. Nested maps—a formal, provably correct object model for spatial aggregates. In *Proceedings 4th ACM International Symposium on Advances in GIS*, pages 76–83, New York, NY, USA, 1996. ACM.

Lutz Plümer and Gerhard Gröger. Achieving integrity in geographic information systems—maps and nested maps. *GeoInformatica*, 1(4):345–367, 1997.

Michael Ian Shamos and Dan Hoey. Geometric intersection problems. In *FOCS*, pages 208–215. IEEE, 1976.

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

Hang Si. Tetgen: A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator. User's manual v1.3 9, WIAS, Berlin, Germany, 2004.

Peter van Oosterom, Jantien Stoter, Wilko Quak, and Siyka Zlatanova. The balance between geometry and topology. In Dianne Richardson and Peter van Oosterom, editors, *Advances in Spatial Data Handling—10th International Symposium on Spatial Data Handling*, pages 209–224. Springer, 2002.

Jan W. van Roessel. A new approach to plane-sweep overlay: Topological structuring and line-segment classification. *Cartography and Geographic Information Science*, 18:49–67, 1991.