From Minkowski Sum to Concave Hull: Two Case Studies of Open Source Development at Ordnance Survey

Sheng Zhou[*1] and Jonathan Simmons[†1]

[1]Data Science and Analytics Team
Data Office, Ordnance Survey
Great Britain

April 15, 2019

Summary

We present our open source implementations of two important computational geometry concepts: 2D Minkowski sum/difference and concave hull. Our Minkowski difference implementation provides a functionality that is not widely available yet in open source community. Compared to other implementations, our concave hull implementation offers rare features such as multi-part hulls and hull with holes. We presented some experimental results from application of these two libraries and discussed some generic issues related to open source development.

KEYWORDS: Geospatial analytics and modelling, Open Source, Minkowski Sum and Difference, Concave Hull, Cluster Analysis

1. Introduction

Open source geospatial software has flourished over the past decade. At Ordnance Survey open source geospatial software are commonly used alongside proprietary software, often as the preferred option. Ordnance Survey has benefited greatly and is making contributions to the open source movement in various ways.

In this paper we present our open source implementations for Minkowski Sum/Difference and Concave Hull computation. We also briefly discuss some generic issues related to open source development.

2. An Experimental Implementation of Minkowski Sum/Difference

2.1. A Brief Introduction to Minkowski Sum and Difference

As an important computational geometry concept, Minkowski sum (or addition) of two sets A and B in Euclidean space is constructed by performing vector addition between each point in A and B (**Figure 1**):

$$A \oplus B = \{ a + b \mid a \in A , \quad b \in B \} \tag{1}$$

Similarly, Minkowski difference (or subtraction) is:

$$A \ominus B = \{ c \mid (c + B) \subseteq A\} \ = (A^c \oplus (-B))^c \tag{2}$$

$A \ominus B$ is interpreted as: the complementary set of Minkowski sum of the complementary set of A and the reflection of B through the origin. Here A^c is the complementary set of A.

[*] Sheng.Zhou@os.uk
[†] Jonathan.Simmons@os.uk

$A \oplus (-B)$ is sometimes also called as Minkowski difference (should not be confused with $A \ominus B$).

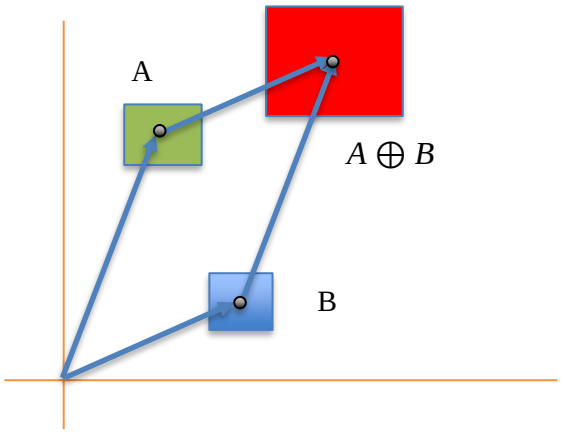

Figure 1 Minkowski Sum.

2.2. Applications of Minkowski Sum/Difference

Minkowski sum is commonly used for collision detection between two polygonal objects (A and B in **Figure 2**). $A \oplus (-B)$ defines the region a collision occurs if the base point of object B enters it.

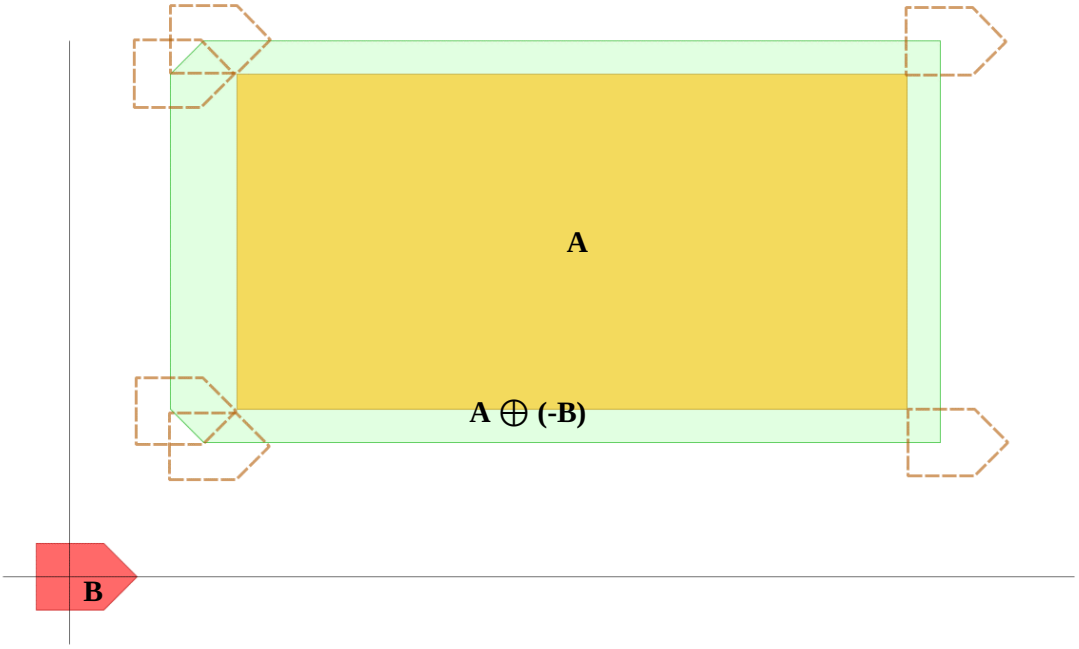

Figure 2 Collision detection using Minkowski Sum.

Minkowski difference can be used to decide whether a polygon B (e.g. outline of an octagon helicopter landing pad) may be fitted into another polygon A (e.g. a field next to the scene of an emergency incident). If $A \ominus B \neq \varnothing$, then B can be fitted into A (**Figure 3**).

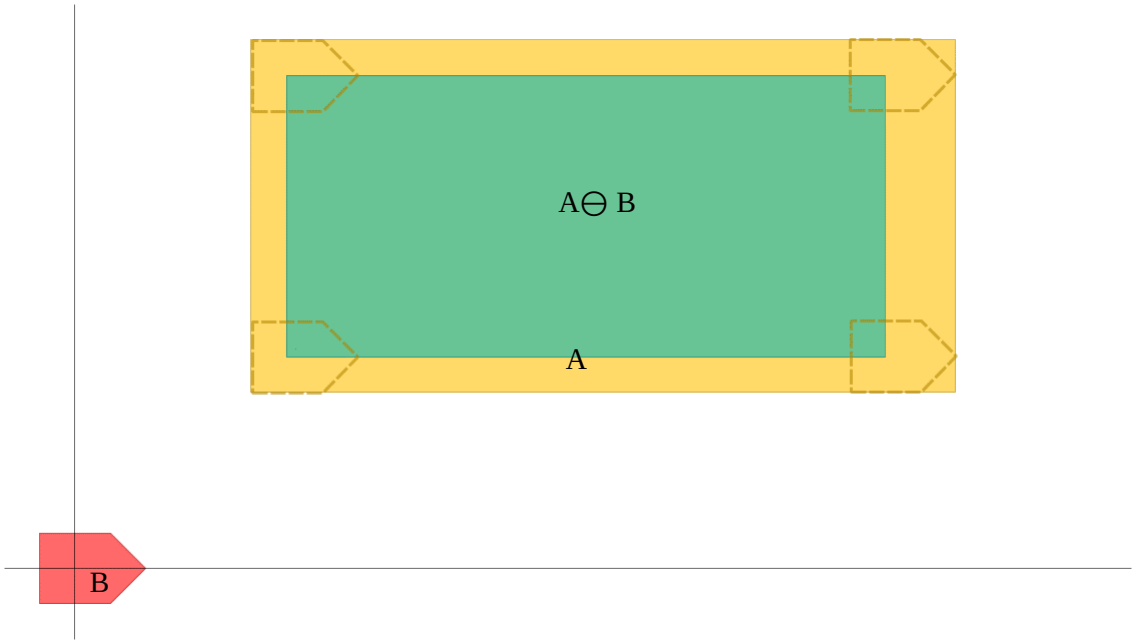

Figure 3 Polygon inclusion test with Minkowski difference

2.3. A Constructive Approach to Minkowski Sum/Difference Computation

Due to the failure to find any useful code for Minkowski difference computation, we developed an experimental Minkowski sum/difference library based on the open source JTS library to meet some customer requirements.

Starting from the definition of Minkowski Sum/Difference, we followed a less efficient but much simpler approach to construct Minkowski sum/difference from "vector addition" of two segments.

The "segment vector addition" of two segments $\mathbf{seg}_{src}((x_1, y_1), (x_2, y_2))$ and $\mathbf{seg}_{ref}((x_3, y_3), (x_4, y_4))$ is:

$$SegVecAdd(\mathbf{seg}_{src}, \mathbf{seg}_{ref}) = Polygon((x_3+x_1, y_3+y_1), (x_4+x_1, y_4+y_1), (x_4+x_2, y_4+y_2), (x_3+x_2, y_3+y_2)) \quad (3)$$

Given a "source" polygon $\mathbf{plg}_s$ and a "reference" polygon $\mathbf{plg}_r$ (without hole and with exterior boundary linestring $\mathbf{l}_s$ and $\mathbf{l}_r$ respectively), we compute the segment vector addition between each line segment in $\mathbf{l}_s$ and that in $\mathbf{l}_r$, and then we union all segment addition results to create a polygon $\mathbf{plg}_u$. The Minkowski sum $\mathbf{plg}_s \oplus \mathbf{plg}_r$ is the polygon constructed from the exterior ring of $\mathbf{plg}_u$ and any holes of $\mathbf{plg}_u$ that is outside $\mathbf{plg}_s$. The Minkowski difference $\mathbf{plg}_s \ominus \mathbf{plg}_r$ is the polygon constructed from holes of $\mathbf{plg}_u$ that is inside $\mathbf{plg}_s$ (which could be empty). The case of $\mathbf{plg}_s$ with holes is more complicated but can still be solved in similar constructive manner.

This algorithm runs at $O(m*n)$ time for convex source and reference polygons with m and n vertices respectively, which in theory is slower than the conventional $O(m+n)$ algorithm (de Berg et al 2009). However, in practice n is normally constant and very small. Therefore the performance is not as bad as suggested by its time complexity. While one or both polygons are concave, its complexity is the same as the conventional algorithm ($O(m*n)$ if one is concave and $O(m^2n^2)$ if both are concave).

This library depends on JTS only and source code is available at:

https://github.com/OrdnanceSurvey/OS_Minkowski_Sum_Diff

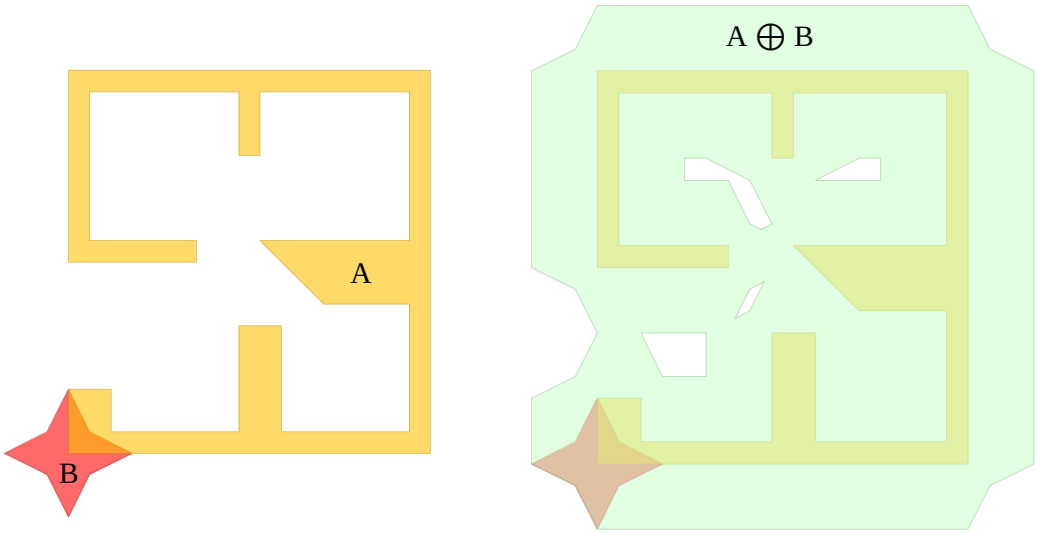

Figure 4 Sample result of Minkowski Sum (CGAL sample data, our library)

Figure 5 illustrates some experiment results for finding potential emergency helicopter landing zones near residential areas, using Minkowski difference. Terrain and elevated obstructions are also considered. An octagon landing pad (which is effectively orientation-less) is used here. For other applications (e.g. airstrips) where orientation is a factor, the process has to be re-run for each orientation (**Figure** 6).

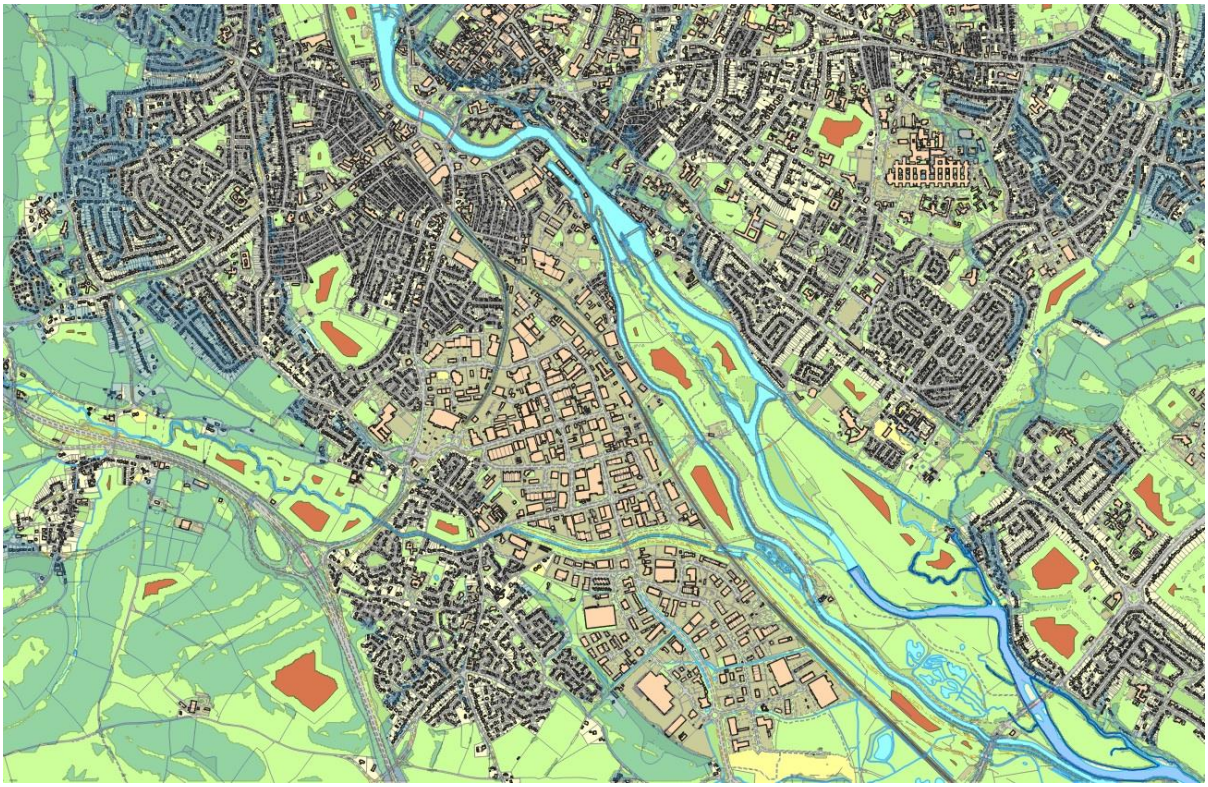

Figure 5 Potential landing zones (areas in RED) for emergency service helicopters

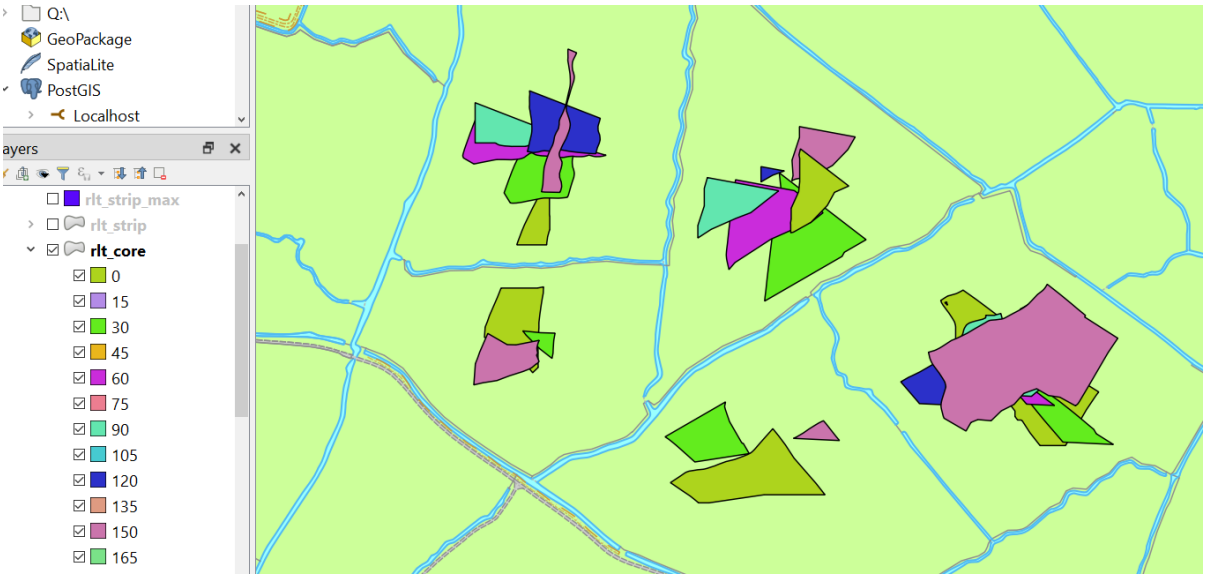

Figure 6 Fitting 200x20m airstrip at multiple directions (terrain and obstacles are considered)

3. An improved Concave Hull implementation

3.1. Concave Hull

Concave hull may be loosely defined as result of removing area according to certain criteria from the convex hull polygon of a dataset. Generally speaking, concave hull provides a more compact representation of the outline of a spatial dataset.

Conventional concave hull is generated by first constructing the triangulation of the dataset and then starting from triangle edges on dataset convex hull boundary and "dig" inwards to remove triangles that satisfy given criterion, such as maximum edge length by Duckham M et al (2008), alpha-shape (maximum radius of circumscribed circle) by Edelsbrunner H et al (1983), or a more complicated criterion by Park and Oh (2012).

In **Figure 7** the convex hull of 5 points is *Polygon*(1, 2, 3, 4) where point 5 is an internal point. The dataset is triangulated. If the internal triangle *Triangle*(4, 1 5) satisfies certain criterion (e.g. the length **L** of boundary edge 4-1 or the radius **R** of circumscribed circle of the triangle is over a given threshold), the triangle may be removed from the initial polygon defined by the convex hull and we have the concave hull of *Polygon*(1, 2, 3, 4, 5). Obviously, there are two options for the "digging" process: depth-first or width-first, which will result in different concave hulls.

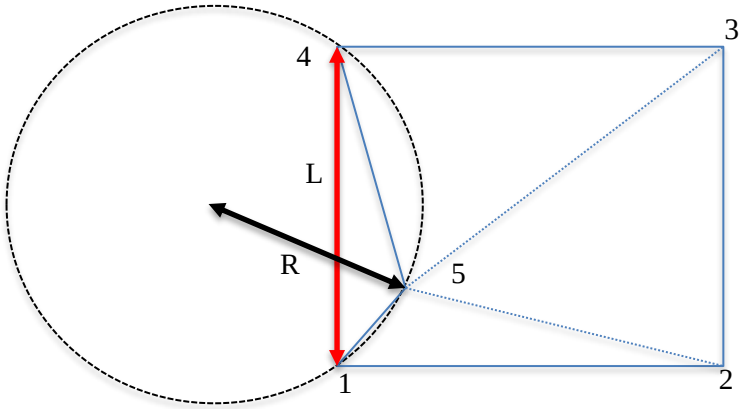

Figure 7 Concave hull construction by "digging"

Concave hull implementations are available in most GIS or spatial database software. However, most of them support single criterion only, and without support for multi-parts (note that this is different from computing concave hull for each geometry in a collection) or internal holes.

3.2. A generic concave hull implementation with multi-part and hole support

To address these issues, we designed and implemented a new concave hull library based on JTS, which is a generic process using a Java interface to decide whether a triangle should be removed. Consequently, different customised criteria may be used. We used a split-able double linked circular list to support multiple-part hull generation. Hull with holes is also implemented.

Java source code of this library is at:

https://github.com/OrdnanceSurvey/OS_ConcaveHull

3.3. Examples and applications

The left column of **Figure 8** are 1000 random points on single or double rings in a square of 1000m side. The central and right columns area concave hull polygons generated without and with hole support, using 50m maximum edge length. Note that this may not be the best example for concave hull, it illustrates the hole-finding functionality of our code very well.

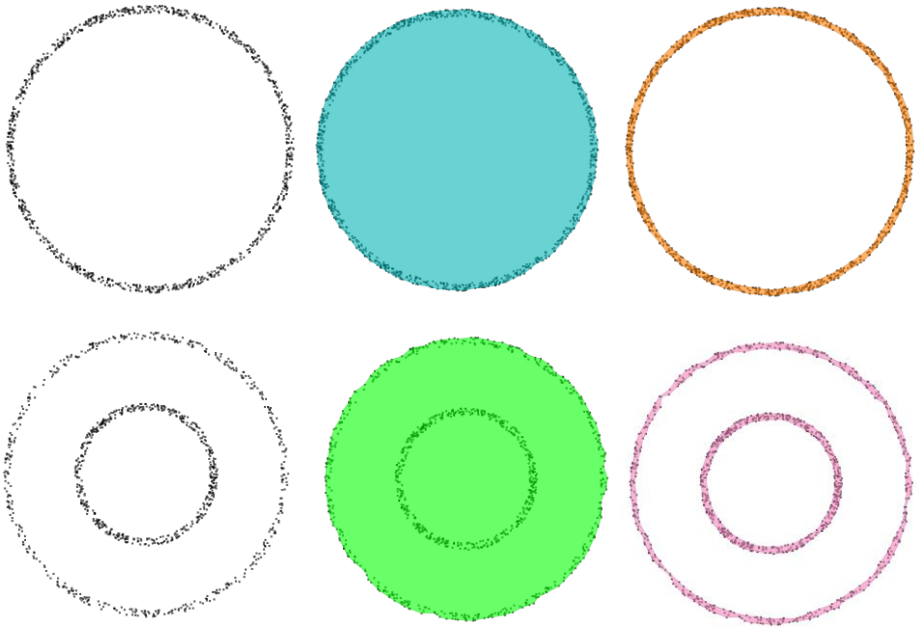

Figure 8 Random points on single/double rings, their concave hulls without/with holes

An observation that may be made is concave hull generation could be applied as a post-process to results of clustering analysis to create a compact outline for each cluster, or concave hull may be used to perform spatial clustering and cluster boundary generation in a single process (**Figures 9 and 10**).

Figure 9 Mean highwater line of GB, concave hulls without and with holes

Figure 10 are some detailed views of the generated hulls. Highwater lines are shown as backdrop. Note that multiple islands are connected to make a single part of the multi-part hull.

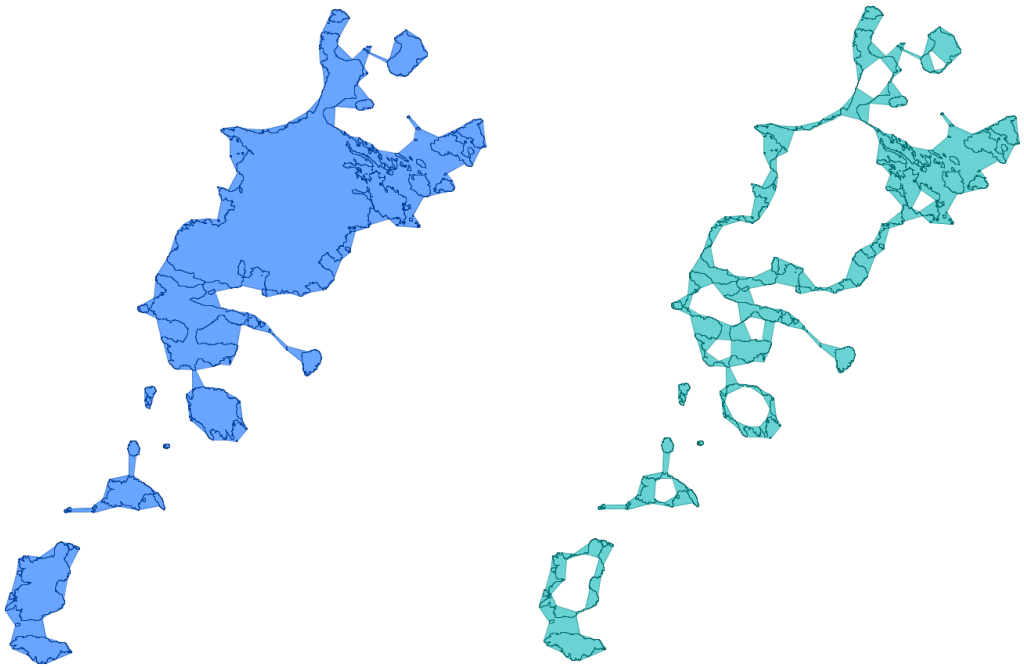

Figure 10 local details of concave hulls without/with holes

Figure 11 Isle of Wight coast represented by concave hull with holes

4. Discussion

Our experience confirmed the widely recognised benefits of open source development and application: flexibility and extensibility, which enables efficient rapid prototyping, especially when certain required functionality is inadequate in or absent from off-the-shelf software packages. On the other hand, we also experienced some of the difficulties common in open source development: inadequate documentation, quality variation within one library/software, unreliable long-term support and stability (an example is the change of package name during JTS 1.14 upgrade to 1.15, which caused a huge dependency chaos among JTS users).

In general, adaption of open source may generate higher demands for developer and development time. Nevertheless, we believe that when a good balance is stroked between proprietary and open source software, the benefit will overweight the costs. We will continue our open source development and our future contributions may be found at:

https://github.com/OrdnanceSurvey

References

De Berg M, van Kreveld M, Overmars O and Schwarzkopf O (2008). *Computational Geometry: Algorithms and Applications, 3rd Ed*. Springer-Verlag, Berlin Heidelberg.

Duckham M, Kulik L, Worboys MF and Galton A (2008). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41(10), 3224-3236.

Edelsbrunner H, Kirkpatrick DG and Seidel R (1983). On the shape of a set of points in the plane", *IEEE Transactions on Information Theory*, 29 (4), 551–559.

Guibas L, Ramshaw L and Stolfi J (1983). A kinetic framework for computational geometry. *24th Annual Symposium on Foundations of Computer Science*, 100-111.

JTS (2019). *https://projects.eclipse.org/projects/locationtech.jts*

Park JS and Oh SJ (2012) A new concave hull algorithm and concaveness measure for n-dimensional datasets. *Journal of Information Science and Engineering* 28, 587-600.

Biographies

Sheng Zhou is a Senior Data Scientist at Ordnance Survey Data Office. He obtained his PhD in GIS/Urban Planning from Cardiff University. Prior to joining OS Research, he participated in research projects on multiscale spatial database and multi-agent system for active maps at Glamorgan and Cardiff universities. His research interests include machine learning algorithms and applications on spatial data; computational geometry; spatial databases; spatial analysis; map generalisation.

Jonathan Simmons is the Head of Data Science and Analytics at Ordnance Survey Data Office.