AGILE: GIScience Series, 5, 35, 2024. https://doi.org/10.5194/agile-giss-5-35-2024
Proceedings of the 27th AGILE Conference on Geographic Information Science, 4–7 Sept. 2024.
Editors: Alison Heppenstall, Mingshu Wang, Urska Demsar, Rob Lemmens, and Jing Yao.
This contribution underwent peer review based on a full paper submission.

The Challenges of Line Buffers: Issues and Methods

Šimon Leitgeb [1]

[1]Department of Geography, Masaryk University, Brno, Czechia

Correspondence: Šimon Leitgeb (leitgeb.simon@gmail.com)

Abstract. This paper discusses the challenges of flat-cap and other linestring buffers, emphasizing automated applications. The pitfalls of existing implementations are introduced, stemming from the lack of a satisfactory flat-cap buffer definition. The buffers' roots in computational geometry are explored. Several candidate methods for robust buffer construction are described, and a novel method is proposed. The specifics and shortcomings of these methods are discussed, outlining a possible path forward.

Keywords. buffer, flat-cap buffer, offset, straight skeleton, weighted straight skeleton

1 Introduction

Buffers are an essential part of geography and cartography. They can be used for *proximity analysis* and visualization alike. As a line or a point has no *inherent thickness* or *dimension*, they would be invisible when drawn on a screen or printed on paper. Lines and points therefore need to be buffered to be visible.

Buffers are generally understood as a zone surrounding a geographic feature, defined by a buffering distance d. Any point closer to the feature than d should be a part of this zone. This general buffer will be referred to as a *regular buffer* throughout this paper.

Regular buffer is a concept rooted in mathematics. It can be defined as the *Minkowski sum* (MS) of the feature in question and a disk with a radius equal to the buffering distance. A Minkowski sum of shapes A and B could be roughly described as the area covered by copies of B placed at all points of A.

Minkowski sum is a well-explored topic in constructional geometry, CAD, and other disciplines. See the Encyclopedia of Mathematics (2011) for its definition, and Wein (2006) for an example of an MS construction using the convolution method.

However, some types of buffers encountered in GIS and computer graphics cannot be defined as a Minkowski sum and can lack a general definition altogether, especially in the case of line buffers. Therefore, the Minkowski sum or offset curve construction methods cannot be used (convolution methods) or must be heavily adjusted (e.g. medial axis or straight skeletons). This paper will refer to these buffers as *irregular* because of this behaviour. See Fig. 1 for a brief overview of some regular and irregular line buffers.

All *regular* line buffers have round caps and joins; *irregular* line buffers can have round, bevel, and mitre joins (with an optional mitre limit) and round, square, or flat caps (also called butt caps), see the *Shapely user manual* (Gillies and Shapely contributors, 2024) as an example. They can also be single-sided or have a variable width, meaning the buffer distance depends on some property of each segment. This wide buffer variety leads to many, often inconsistent, buffer implementations across different tools. This issue is also acknowledged in the W3C Editor's Draft of the SVG 2 specification (SVG Working Group, 2018).

Polygon buffers can also be irregular, but the issues mentioned in this paper never occur in their implementations due to the absence of caps. Point buffers are, on the other hand, always regular since an MS of a point and any shape S results in a translation of S. Therefore, polygon and point buffers are not the main focus of this paper.

1.1 Motivation

The difference in definition aside, some irregular buffer variants lack a fundamental property of the regular buffer: not all points within the buffering distance are interior to the buffer. These variants include flat caps, bevel joins and mitre joins with a small mitre limit.

While irregular buffers certainly have their merits, this behaviour can lead to some hard-to-predict artefacts, making the availability of implementation details for irregular buffers paramount for the user. This is especially relevant when automated processes are of concern, for example:

- *Line visualization* in computer graphics and cartography;

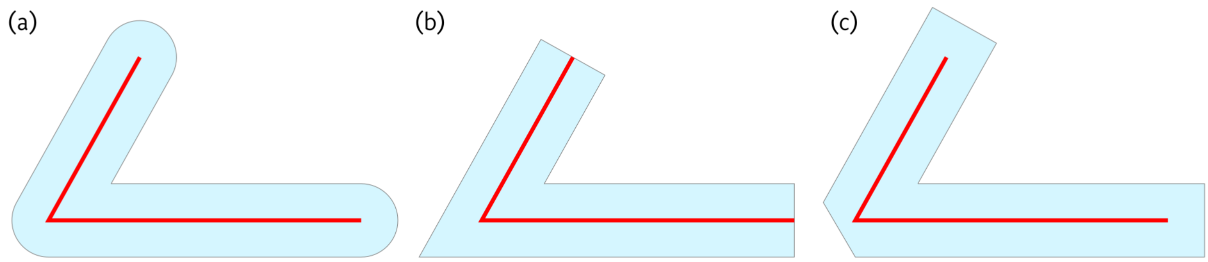

Figure 1. An example of a regular buffer (a) and two irregular buffers, a flat-cap buffer with a mitre join (b) and a square-cap buffer with a bevel join (c).

- *Coverage path planning* for autonomous robots and precision agriculture (e.g. obtaining the area covered by a harvester or a painting robot). Even more specialized approaches than irregular buffers might be required in some cases.

From the experiments done in some of the widely used GIS software, it seems that most of these implementations calculate the offsets of each input segment individually, trimming the offsets at intersection points and detecting self-intersecting loops. This approach often works well but can sometimes produce surprising results, mainly when using flat caps.

Notice that in Fig. 2a, the intermediate segments of a flat-cap buffer form no *closed loops* (unlike regular buffers, Fig. 2b), and a heuristic is necessary to close the boundary and obtain a valid polygon. Since many options for this heuristic can be devised, and none is necessarily more intuitive than the other, each implementation can produce different outputs for the same input (see Fig. 3).

Any advances in irregular buffer construction could also benefit the implementations of *parallel offsets*. Since they often seem to use the same strategies as flat-cap buffer implementations, they also lead to similar issues. Offsets are also used in cartography (e.g. coloured public transport lines) and coverage path planning.

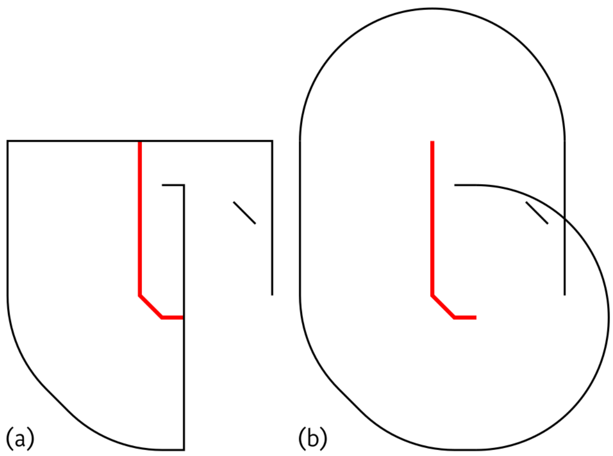

Figure 2. The intermediate offset segments of a flat-cap buffer with round joins (a) and a regular buffer (b).

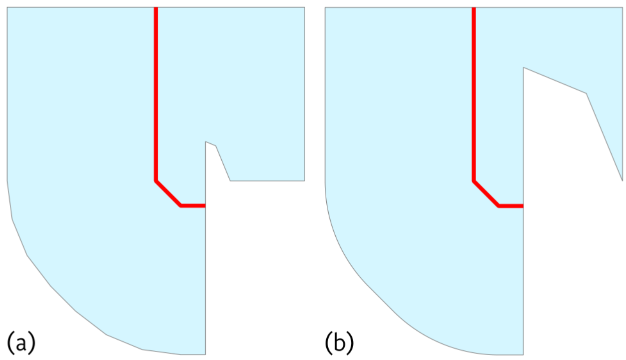

Figure 3. Examples of a flat-cap buffer of the line from Fig. 2 created in QGIS (a) and ArcGIS (b) with the same buffer distance.

1.2 Contribution

This paper describes some of the methods that could be used for constructing flat-cap buffers without the ambiguity caused by these heuristics. Most of these methods are also applicable to regular or other irregular buffers. A novel method is also proposed in section 4, providing an intuitive alternative to the other methods, and expanding upon them.

2 Data and Software Availability

The figures in this paper were created using GeoGebra and Inkscape. QGIS 3.34.2-Prizren and ArcGIS Pro 3.1.0 were used to create the example buffers.

The data used to create the examples in this paper is available in a repository on Zenodo and is accessible via the following DOI: https://doi.org/10.5281/zenodo.10658846. See the attached README.md for the parameters used.

3 Methods overview

Several methods are presented as options for flat-cap buffer construction. The following three methods provide the basis for a new method, described in section 4. Mitre joins and flat caps are used in the examples to demonstrate the specifics of each method.

3.1 Quadrilaterals

A buffer can be trivially constructed by buffering each segment separately and dissolving these shapes to obtain the result. A regular buffer can be produced using circles and rectangles centred at each vertex or segment of the input line, respectively.

Irregular buffers can also be constructed with a similar approach. Angle bisectors are constructed at each vertex of the input line and segment buffers are then bounded by these bisectors and lines parallel to the segment at the specified offset distance d. Buffer caps are formed by extending the set of bisectors with:

1) Lines perpendicular to the terminal segments, passing through the terminal vertices, forming a flat cap (depicted in Fig. 4),
2) Two rays originating at each terminal vertex A perpendicular to each other, their bisector extending in the same direction as the terminal segment $|BA|$, forming a square cap (depicted in Fig. 5).

These boundaries form quadrilaterals that, given a sufficient distance d, become degenerate, forming a triangle instead.

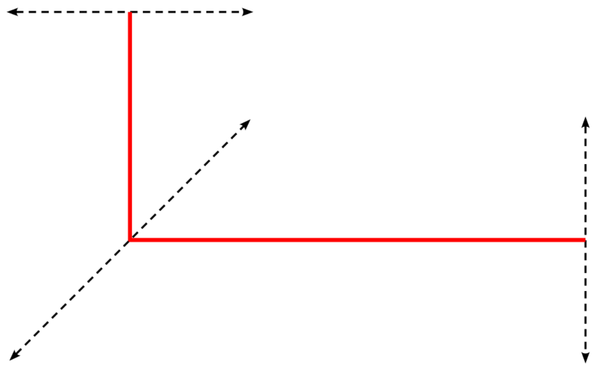

Figure 4. The set of bisectors of a flat-cap buffer.

While this method is simple to implement, the buffer result is not always intuitive since one side of the buffer might stop propagating (Fig. 6), and holes can be formed in some instances of a flat-cap buffer (Fig. 7). No existing buffer implementations seem to use this approach.

3.2 Straight skeleton

A polygon P can be partitioned into cells of a Voronoi diagram, where any point of a cell has a distinct closest point on the boundary of P. A single cell consists of all points closest to a given boundary edge of P. The medial axis is a subset of a Voronoi diagram, consisting of the cell boundaries. The medial axis can be used to obtain an offset of P with round joins (a regular buffer), as the offset vertices lie on the medial axis. Offset construction using the medial axis was explored e.g. by Choi et al. (2008).

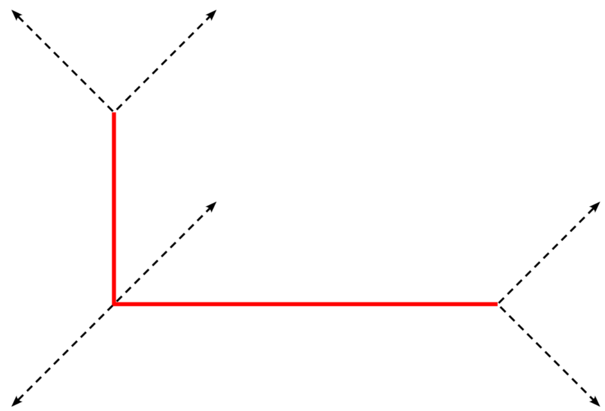

Figure 5. The set of bisectors of a square-cap buffer.

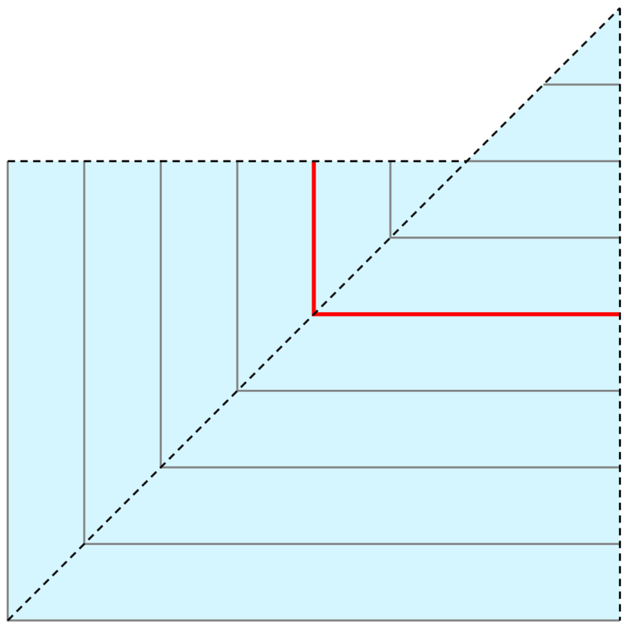

Figure 6. A flat-cap buffer of a two-segment linestring. The input line (red) is buffered along its bisectors (dashed lines) four times (black lines), and the final buffer is the light blue area.

Aichholzer et al. (1996) introduced the *straight skeleton* of a polygon. It is similar to the medial axis but is defined by *wavefront propagation* instead (see Aichholzer and Aurenhammer, 1996, for its description). The straight skeleton can be used to obtain a mitred offset of P.

The construction process is best imagined using the surface of a hip roof. The straight skeleton and medial axis consist of the lines (hips and valleys) where the roof surfaces intersect. While using a straight skeleton, the roof surfaces are always flat, while the medial axis can include curved surfaces.

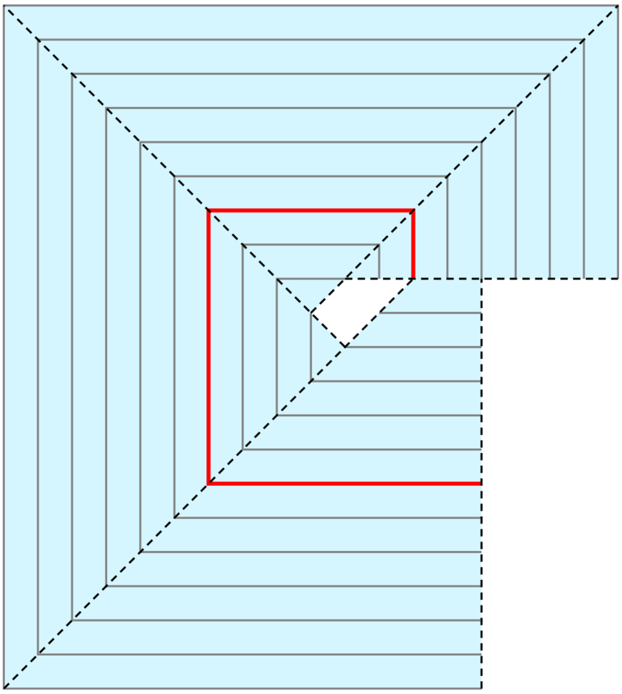

Figure 7. A hole artefact created by the quadrilateral method.

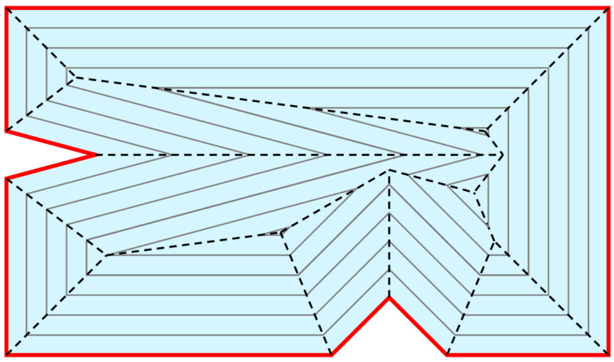

Figure 8. An example of a straight skeleton (dashed segments) with the original polygon in red and incremental offsets in grey.

In this model, an offset of the input feature is resembled by the contour line with a height equal to the offset distance d (provided that there is a 1:1 ratio of height to offset distance, meaning all surfaces intersect their respective edges at a 45° angle). The buffer area is then resembled by all surface points with a height higher than or equal to d.

Aichholzer and Aurenhammer (1996) extended the construction of a straight skeleton from polygons to *planar straight-line graphs* (PSLGs). A PSLG is a collection of non-intersecting straight-line segments in a plane. Therefore, both polygons and linestrings in GIS can be considered PSLGs.

Numerous improvements in straight skeleton construction speed and versatility were made, recently also by Palfrader and Held (2015). While their paper only discusses polygon offsets, their contribution also includes the SURFER library, which handles open linestrings as well. Their implementation produces buffers with square caps and mitre or bevel joins.

3.3 Weighted straight skeleton

The weighted straight skeleton is a modification of the straight skeleton method that allows assigning *weights* to the edges of the PSLG, changing its *propagation speed*. This method allows for constructing flat-cap buffers by adding *hidden edges* with zero length and zero weight as the flat line caps.

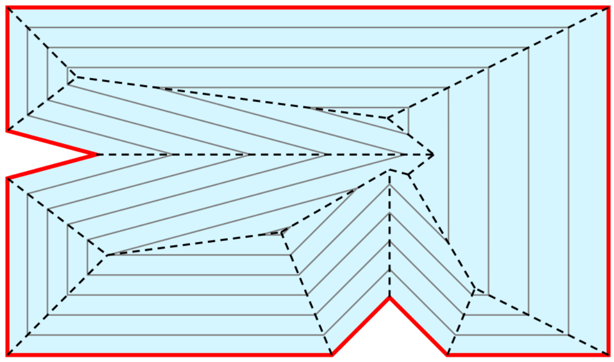

Figure 9. An example of a weighted straight skeleton, modified from Fig. 8. The right-most edge has double the weight of the other edges, making its surface more slanted.

While Biedl et al. (2015) only allow *non-zero weights* for the PSLG edges, and the CGAL implementation only supports strictly positive weights, Held and Palfrader (2017) propose *additively-weighted* straight skeletons that delay the propagation of arbitrary PSLG edges. By setting the edge delay to a very high number, its weight essentially becomes zero, allowing for the flat-cap buffer construction.

One ambiguity is encountered when two parallel wavefront edges with *different weights* become adjacent, for example, after an edge event, as described by Biedl et al. (2015). They argue that stopping the propagation of one of the edges is the only reasonable approach. However, they suggest terminating the lower-weighted edge is as justified as the higher-weighted one.

Considering the flat-cap buffer, this would mean either stopping the propagation of non-zero weight edges altogether or "covering" the area above the flat cap when the flat cap becomes adjacent to a parallel edge. Zero-weight edges are also prone to terminating other edges from propagating at larger buffer distances, leading to results that could be hard to predict.

These properties make the usability of flat-cap buffers constructed with a weighted straight skeleton questionable. However, given the ongoing interest in straight skeleton based offsetting and its weighted variant, novel approaches to resolving these ambiguities might be suggested, potentially making this method worthwhile.

4 Incremental extension

A novel intuitive approach could be described as follows (see Fig. 10 for a visual representation):

1) Offset all edges by a very small distance,

2) Find intersections of neighbouring offsets on the obtuse sides of vertices and trim their interior parts,

3) Extend the offsets on the reflex sides of vertices until they intersect (modifying this step allows for round and bevel joins),

4) Repeat until the desired buffer distance d,

5) Connect the offset sides by the paths traced by the terminal vertices at each step to obtain a closed buffer boundary.

This process advances all edges until they shrink to a single point. If this edge is terminal (the first or last edge of the offset chain), the next edge becomes terminal and stops shrinking on one side. The extension event propagates across any number of edges, until all edges degenerate or a reflex vertex is encountered; then, the extended edge grows infinitely.

This novel approach could most likely be implemented using a *weighted straight skeleton* by modifying the skeleton structure after an edge event involving a zero-weight edge. A new zero-weight edge could be added, perpendicular to the newly adjacent edge, providing a third option for solving the ambiguity mentioned by Biedl et al. (2015). Flat caps no longer stop the propagation of other faces this way, preserving as much of the original geometry as possible.

However, which modifications to *split* and *vertex* events would be needed when implementing this method still needs to be determined.

Some of the artefacts encountered with the quadrilateral method are avoided using this approach, arguably producing a more intuitive result (Fig. 11). However, some artefacts remain (Fig. 12), revealing a fundamental issue with flat-cap buffers. The result becomes hard to predict since the buffer cap can influence other edges.

5 Mitres

Long spikes are formed using mitre joins when two line segments meet at a sharp angle. The length of these spikes approaches infinity as the angle included by the segments approaches zero. If the spike points towards another edge, it might also form self-intersections with the rest of the buffer.

Mitre limits are usually used to prevent very long spikes. However, the limit needs to be chosen manually, as it depends on the scale of the input shape (different inputs might require different mitre limits). Approaches based on the *straight skeleton* prevent self-intersections altogether via *split events*, leading to another discrepancy across the available approaches.

Round or bevel joins might be more desirable for automation for their predictability since they have no parameters that need to be manually chosen.

6 Conclusions

The methods described in this paper represent *some of the options* for buffer construction. Only two of them are suitable for flat-cap buffers without modifications (the quadrilateral and incremental extension methods) and neither are implemented in existing software.

Due to the output differences across the methods and the outputs often being hard to predict, it would seem that using flat-cap buffers is inadvisable. However, there are still use cases for them. Because some require *little to no user input*, they would all benefit from a method with less ambiguity.

The list of methods is by no means complete; other options might be needed for different use cases. The *weighted straight skeleton* and *incremental extension* methods need to be further explored to assess their applicability for flat-cap buffers. If these methods fail to provide satisfactory results, other methods might need be proposed, even on a case-by-case basis.

Buffering and *offsetting* are also closely related (offsets being linestrings parallel to other linestrings), making robust buffering methods useful for offsets as well.

Any revision of the flat-cap buffer and parallel offset implementations available in the JTS and GEOS libraries would benefit all the numerous GIS software that uses these libraries. Access to the *algorithm used* and other implementation details in the GIS software documentation would benefit its users. If this information is unavailable and a reliable output is needed, using round or square caps might be preferable.

Competing interests. The author declares that he has no conflict of interest.

Acknowledgements. The author would like to thank his colleagues and friends for their consultations, patience and understanding during the research leading up to this paper.

References

Aichholzer, O. and Aurenhammer, F.: Straight skeletons for general polygonal figures in the plane, in: Computing and Combinatorics, edited by Cai, J.-Y. and Wong, C. K., pp. 117–126, Springer Berlin Heidelberg, Berlin, Heidelberg, https://doi.org/10.1007/3-540-61332-3_144, 1996.

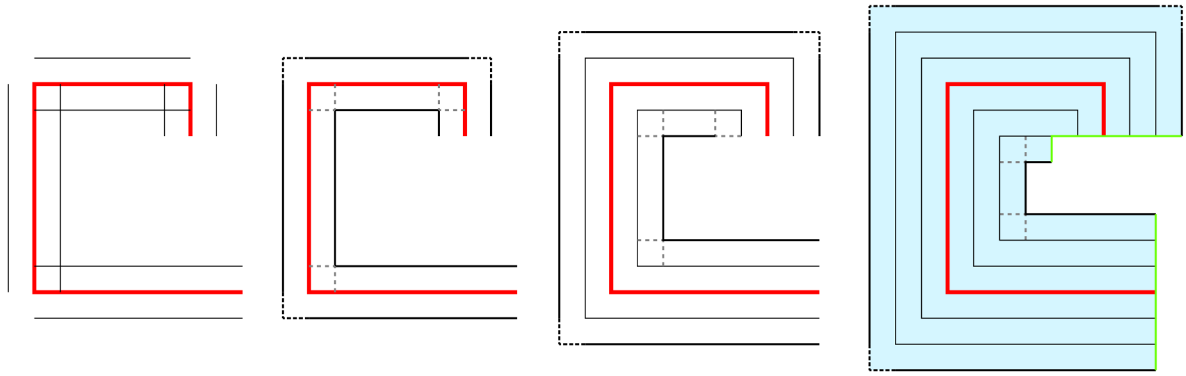

Figure 10. The construction process of the *incremental extension* method. Dashed in gray are the trimmed parts, dashed in black extended parts, traces forming the flat caps are highlighted in green.

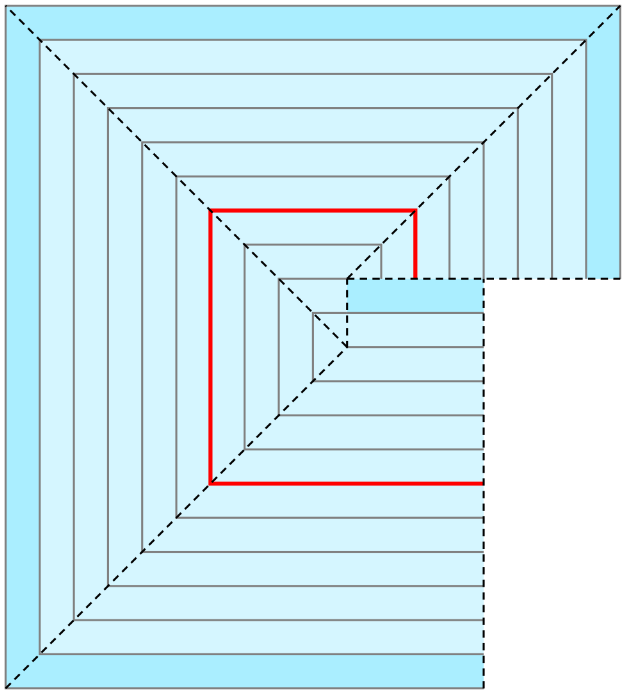

Figure 11. The artefact from Fig. 2 is filled with the *incremental extension* method. One side of the buffer stops propagating after the last iteration (highlighted in blue).

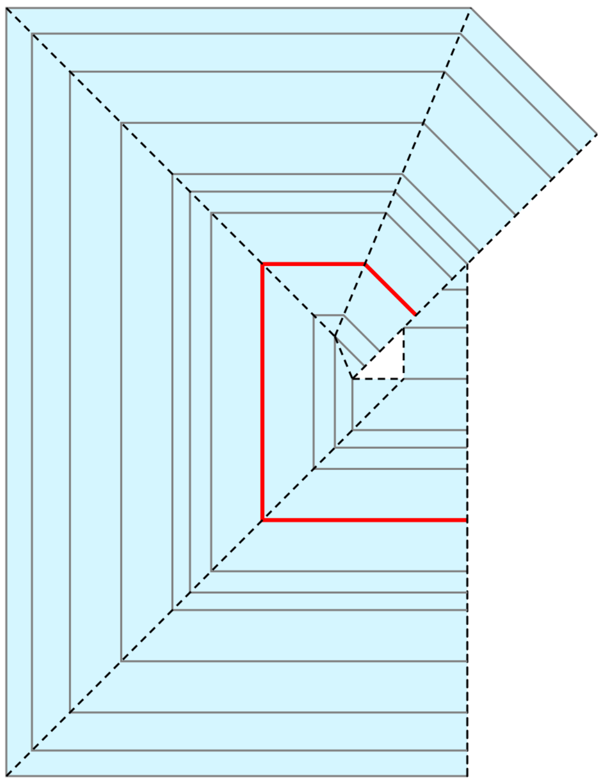

Figure 12. Hole artefacts can still occur with *incremental extension*.

Aichholzer, O., Aurenhammer, F., Alberts, D., and Gärtner, B.: A Novel Type of Skeleton for Polygons, pp. 752–761, Springer Berlin Heidelberg, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-80350-5_65, 1996.

Biedl, T., Held, M., Huber, S., Kaaser, D., and Palfrader, P.: Weighted straight skeletons in the plane, Computational Geometry, 48, 120–133, https://doi.org/10.1016/j.comgeo.2014.08.006, 2015.

Choi, H. I., Choi, S. W., Han, C. Y., Kim, T.-w., Kwon, S.-H., Moon, H. P., Roh, K. H., and Wee, N.-S.: Two-dimensional offsets and medial axis transform, Advances in Computational Mathematics, 28, 171–199, https://doi.org/10.1007/s10444-007-9036-5, 2008.

Encyclopedia of Mathematics: Minkowski addition, available at: http://encyclopediaofmath.org/index.php?title=Minkowski_addition&oldid=49911, last access: 8 April 2024, 2011.

Gillies, S. and Shapely contributors: Shapely 2.0.3 documentation, available at: https://shapely.readthedocs.io/en/stable/manual.html#shapely.BufferCapStyle, last access: 10 April 2024, 2024.

Held, M. and Palfrader, P.: Straight skeletons with additive and multiplicative weights and their application to the algorithmic generation of roofs and terrains, Computer-Aided Design, 92, 33–41, https://doi.org/10.1016/j.cad.2017.07.003, 2017.

Palfrader, P. and Held, M.: Computing mitered offset curves based on straight skeletons, Computer-Aided Design and Applications, 12, 414–424, https://doi.org/10.1080/16864360.2014.997637, 2015.

SVG Working Group: Scalable Vector Graphics (SVG) 2; W3C Candidate Recommendation, available at: https://www.w3.org/TR/2018/CR-SVG2-20181004/painting.html#TermStrokeShape, last access: 12 February 2024, 2018.

Wein, R.: Exact and efficient construction of planar Minkowski sums using the convolution method, in: European symposium on algorithms, pp. 829–840, Springer, 2006.