www.cad-journal.net
Computer-Aided Design AND APPLICATIONS™

# An Experimental Evaluation of Offset Computation for Polygons

Martin Held[1], Stefan de Lorenzo[2], Peter Palfrader[3]

[1]Universität Salzburg, FB Informatik, held@cs.sbg.ac.at
[2]Universität Salzburg, FB Informatik, slorenzo@cs.sbg.ac.at
[3]Universität Salzburg, FB Informatik, peter@palfrader.org

Corresponding author: Martin Held, held@cs.sbg.ac.at

**Abstract.** We present an experimental evaluation of the computation of offset curves of polygons in the plane. Our evaluation involves several leading software packages for computing Boolean operations and for computing Voronoi diagrams. We compare these packages to our own codes CAPOP (for computing Boolean operations on closed curves consisting of straight-line and circular-arc edges) and VRONI (for computing Voronoi diagrams). Our extensive tests show that the differences in the run-times of the packages tested are substantial. In particular, only CAPOP and the Voronoi codes VRONI and OpenVoronoi exhibit a clearly sub-quadratic run-time complexity.

**Keywords:** Offsetting, Boolean operations, Voronoi diagram, Bentley-Ottmann algorithm
**DOI:** https://doi.org/10.14733/cadaps.2024.807-818

## 1 INTRODUCTION

For a set $S$ of (possibly infinitely many) points in the Euclidean plane, the *constant-radius offset* $\mathcal{O}(S,r)$ of $S$ for offset distance $r$ is the set of all points of the plane whose minimum (Euclidean) distance from $S$ equals $r$. Mathematically speaking, such an offset is given by the boundary of the Minkowski sum [11, 24] of $S$ with a disk with radius $r$ centered at the origin. That is, it is the envelope of the offset area $\mathcal{OA}(S,r) := \bigcup_{p\in S} D(p,r)$, where $D(p,r)$ denotes a disk of radius $r$ centered at the point $p$. If $S$ is given by a set of straight-line segments and circular arcs, then such a constant-radius offset consists of one or more closed curves consisting of straight-line segments and circular arcs. See Figure 1a for a sample interior and exterior constant-radius offset of a simple polygon. We call a closed curve that is formed by a (finite) sequence of straight-line segments and circular arcs a *circular-arc polygon*.

Held et al. [16] generalize the concept of multiplicatively weighted Voronoi diagrams (MWVDs) by introducing so-called *variably-weighted straight-line segments*: They assign real-valued weights $w(p) > 0$ and $w(q) > 0$ to the end points $p$ and $q$ of an input straight-line segment $\overline{pq}$. These weights need not be identical. Weights of the points along $\overline{pq}$ can be derived by linearly interpolating between $w(p)$ and $w(q)$. Even for a

fixed offset value, these weights introduce more flexibility: In contrast to conventional constant-radius offsetting, where all parts of the input set expand or shrink uniformly at the same speed, *variable-radius offsetting* induced by the weights allows parts to expand or shrink in a non-uniform manner even when the offset value $r$ is fixed; see Figure 1b. As it can be observed in this figure, the variable-radius offset $\mathcal{O}_v(P, r)$ of a polygon $P$ is a circular-arc polygon, too. We refer to [16] for a formal proof of this property.

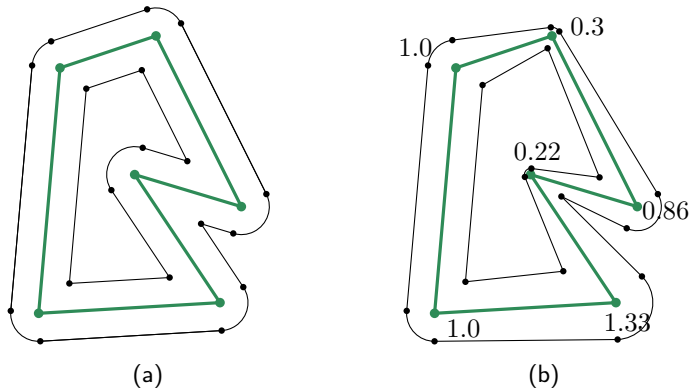



**Figure 1**: Sample interior and exterior (a) constant-radius and (b) variable-radius offsets derived from the same input polygon. The weights used for (b) are written next to their corresponding polygon vertices.

Offsetting is an important task in the manufacturing business and in several other fields. For instance, pocket machining based on traditional contour-parallel or direction-parallel tool paths as well as based on recent spiral tool paths requires the computation of at least one offset curve; see, e.g., [12, 19, 22]. Variable-radius offsets allow to shrink or increase a shape in a non-uniform way, e.g., for garment manufacture or shoe design. Brush stroke generation and the generation of ornamental seams are other sample applications that benefit of variable-radius offsets. In geographic information systems (GIS), offsetting is known as buffering, with both constant-radius and variable buffers being in use [6, 33]. E.g., when modeling noise levels along road networks, iso-contours of noise will correspond to different buffer distances. Similarly, navigation corridors for aviation and shipping need not be uniform offsets along a center route. Wolff et al. [32] use buffers for labelling maps. While 2D offsetting tasks for manufacturing tend to involve shapes of rather moderate complexity with at most a few hundred boundary elements, the complexity of GIS data seems to be ever increasing. For this reason, already twenty years ago Žalik et al. [33] test their implementation for up to $100\,000$ line segments. Hence, the computational efficiency of offsetting applied to large data sets is of genuine practical importance.

By definition, the offset area $\mathcal{OA}(S, r)$ is a union of disks. This union may involve infinitely many disks, though. As shown in Section 3, if $S$ is modeled by a finite number of points and straight-line segments, then such a union of possibly infinitely many disks can be replaced by the union of finitely many disks and rectangles, or by disks and trapezoids in the case of variably-weighted sites. Hence, constant-radius and variable-radius offset areas can be obtained by computing the union of a finite number of circular-arc polygons of low combinatorial complexity. Then an actual offset is given by the boundary of that offset area.

If no weights are assigned to the end points of the straight-line segments of $S$, then there is another alternative: The approach sketched by Persson [23] employs the Voronoi diagram of $S$ to obtain constant-radius offsets of $S$. We refer to Held [13] for a more recent and detailed discussion of the underlying concepts in nowadays terminology. We note that Voronoi diagrams can be generalized appropriately to structures that would admit efficient variable-radius offsetting even for weighted segments once that structure is known, see [14, 16]. Unfortunately, their practical use is somewhat limited since it is much more difficult to compute those generalizations reliably and efficiently than it is to compute standard Voronoi diagrams.

## 2 OUR CONTRIBUTION

In order to probe the computational efficiency of offsetting codes we present an experimental evaluation of run-times consumed by the computation of constant-radius and variable-radius offsets of polygons with up to $2\,000\,000$ vertices. Our evaluation involves several leading software packages for computing Boolean operations on (circular-arc) polygons. Their run-times are compared to the results of our own codes Circular Arc Polygon OPerations (CAPOP) for computing Boolean operations and VRONI (for computing Voronoi diagrams). CAPOP can handle genuine circular-arc polygons without prior (or code-internal) polygonal approximations of circular arcs. Extensive tests show that CAPOP is slower than VRONI but that its performance is clearly superior compared to the other software packages that we tested. However, in contrast to VRONI, CAPOP can also generate variable-radius offsets.

## 3 VARIABLE-RADIUS OFFSETS

Let $p$ be a point of the Euclidean plane $\mathbb{R}^2$ and assign a positive real-valued weight $w(p)$ to it. We call $p$ a weighted point. The weighted distance $d_w(a,p)$ between a point $a \in \mathbb{R}^2$ and $p$ is defined as

$$d_w(a,p) := \frac{d(a,p)}{w(p)},$$

where $d(a,p)$ denotes the standard Euclidean distance between $a$ and $p$. Since $d_w(a,p)$ equals $d(a,p)$ for $w(p) := 1$, this is a genuine generalization of the standard Euclidean distance.

Now consider a straight-line segment $\overline{pq}$ between the weighted points $p$ and $q$. The weights $w(p)$ and $w(q)$ need not be identical. (But, again, we require $w(p) > 0$ and $w(q) > 0$.) For $0 \le \lambda \le 1$, the weight of a point $(1-\lambda)p + \lambda q$ on $\overline{pq}$ is given by $(1-\lambda)w(p) + \lambda w(q)$, i.e., by the matching linear interpolation of the weights of $p$ and $q$. We call $\overline{pq}$ a *variably-weighted straight-line segment*. The weighted distance between a point $a \in \mathbb{R}^2$ and $\overline{pq}$ is given as

$$d_w(a,\overline{pq}) := \min_{b \in \overline{pq}} d_w(a,b).$$

Let $S$ be a set of finitely many weighted points and variably-weighted straight-line segments defined by some pairs of the weighted points. No straight-line segment of $S$ is allowed to contain any weighted point of $S$ except for its two end points, and no pair of segments of $S$ may share a point except for a common end point. We refer to the elements of such a set $S$ as *sites*.

The variable-radius offset area $\mathcal{OA}_v(S,r)$ induced by $S$ relative to the offset value $r \ge 0$ is the set of all points of the plane whose minimum weighted distance from $S$ is at most $r$. More formally,

$$\mathcal{OA}_v(S,r) := \left\{ a \in \mathbb{R}^2 : \min_{s \in S} d_w(a,s) \le r \right\}.$$

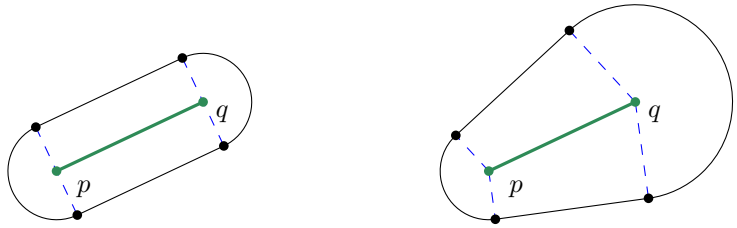
Of course, the variable-radius offset area $\mathcal{OA}_v(S,r)$ matches the constant-radius offset area $\mathcal{OA}(S,r)$ if all weights are set uniformly to 1 (or to any other positive value). The variable-radius offset of $S$ for offset value $r$ is given by the boundary of its offset area:

$$\mathcal{O}_v(S,r) := \partial\, \mathcal{OA}_v(S,r)$$

Hence, the variable-radius offset $\mathcal{O}_v(S,0)$ equals $S$. All variable-radius offsets are exclusively made up of straight-line segments and circular arcs [16].

The definition of $\mathcal{OA}_v(S,r)$ implies for a point $a$ that $a \in \mathcal{OA}_v(S,r)$ if and only if there exists a site $s \in S$ such that $d_w(a,s) \le r$. That is, if and only if $a \in \mathcal{OA}_v(\{s\},r)$. If $s$ is a weighted point, then $\mathcal{OA}_v(\{s\},r)$ is given by the disk $D(s, w(s)\cdot r)$. If $s$ is a variably-weighted straight-line segment $\overline{pq}$, then $\mathcal{OA}_v(\{s\},r)$ is

given by the convex hull of $\mathcal{OA}_v(\{p\}, r)$ and $\mathcal{OA}_v(\{q\}, r)$, i.e., by the convex hull of two disks. Hence, in this case $\mathcal{OA}_v(\{s\}, r)$ is the union of these two disks and a trapezoid defined by the four points in which the two bi-tangents touch the disks. In Figure 2b, the offset of a variably-weighted straight-line segment is shown where the weight of $q$ is twice as large as the weight of $p$.

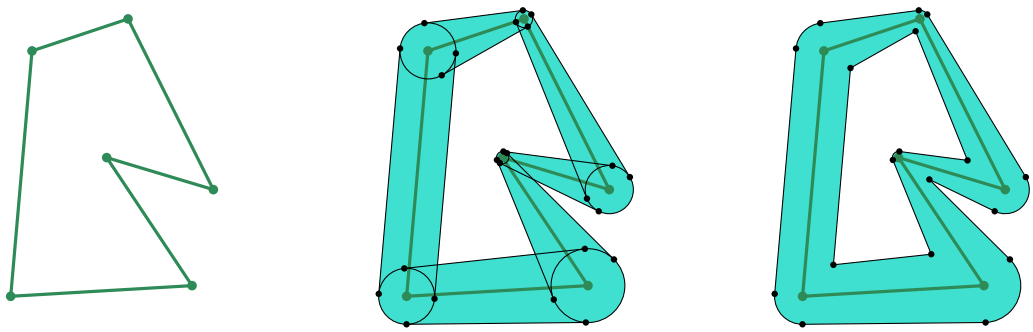



(a) (b)

**Figure 2**: (a) A constant-radius offset and (b) a variable-radius offset of a straight-line segment $\overline{pq}$ is shown.

## 4 BOOLEAN OPERATIONS ON CIRCULAR-ARC POLYGONS

The previous section allows us to conclude that we can compute $\mathcal{OA}_v(S, r)$ by computing the union of the finitely many elementary offset areas $\mathcal{OA}_v(\{s\}, r)$ for all $s \in S$. Since every $\mathcal{OA}_v(\{s\}, r)$ has a simple circular-arc polygon as its boundary, computing the union of circular-arc polygons would suffice to generate a variable-radius offset; see Figure 3.



(a) (b) (c)

**Figure 3**: In (a) the initial polygon is shown. We start with computing the elementary offset areas of the individual (variably-weighted) straight-line segments (b). Finally, in (c) we compute their union and extract the corresponding boundary curves to form the offset curves shown in Figure 2b.

This insight lets us resort to computing Boolean operations for obtaining constant-radius and variable-radius offsets. As a matter of fact, computing Boolean operations among polygons is a well-established research area. Vatti [28] introduced one of the first algorithms for computing the intersection, union, or difference of polygons. Several strategies [5, 20, 21] deal with the more general problem of overlaying two subdivisions of the plane by extending the famous sweep-line approach introduced by Bentley and Ottmann

[2] that is able to find all $k$ intersection among a set of $n$ straight-line segments in $O((n+k)\log n)$ time and $O(n+k)$ space. Chazelle and Edelsbrunner [4] present an algorithm for intersecting $n$ straight-line segments in optimal $O(k+n\log n)$ time and $O(n+k)$ space. Balaban [1] maintains the same time complexity but reduces the space complexity to $O(n)$.

Wein [31] uses a polygonal approximation of the Minkowski sum to offset simple polygons. The corresponding implementation relies on exact rational arithmetic. Van Wyk [27] presents several algorithms for clipping straight-line segments and circular arcs to the boundary of a simple circular-arc polygon. Wang et al. [30] introduce a strategy for computing Boolean operations on two circular-arc polygons of size $m$ and $n$ that runs in $O(m+n+k\log k)$ time using $O(m+n+k)$ space, where $k$ is the number of intersections. They also present an experimental evaluation of their implementation (based on the Computational Geometry Algorithms Library (CGAL)) for problem instances consisting of up to several hundred input edges.

Gong et al. [10] propose a strategy that is able to perform Boolean operations on two conic polygons in $O(nm)$ time. Berberich et al. [3] define a kernel for conic arcs, algorithms for exact computation with low-degree algebraic numbers, and an algorithm for computing the arrangement of conic arcs that immediately leads to a realization of regularized Boolean operations on conic polygons. CGAL makes it possible to perform Boolean operations on circular-arc polygons [9].

The underlying algorithm employed by our own implementation, CAPOP, is a modified and enhanced Bentley-Ottmann approach [2] to compute the union, intersection, or difference of two given subsets $\mathcal{A}_1$ and $\mathcal{A}_2$ of $\mathbb{R}^2$ that are bounded by circular-arc polygons. In the context of GIS, the basics of such an algorithm are described by Žalik et al. [33] and Martínez et al. [20]. However, other than [33], our approach requires only one plane sweep. (This does not change its worst-case complexity but can be expected to reduce the practical run-times of the implementation.) We refer to Martínez et al. [20] for a detailed description of (and pseudo-code for) the main algorithm. Although their algorithm can perform Boolean operations only on purely polygonal data, all of the underlying events can be handled analogously in the case of circular arcs. The only important aspect is that, as a preprocessing step, all circular arcs that show up in the input need to be split into pieces that are monotone: If the line sweep employed by the Bentley-Ottmann algorithm moves from bottom to top, then every circular arc and every full circle needs to be split at its south pole and at its north pole by inserting up to two new vertices. Of course, this preprocessing increases the number of circular arcs by at most a factor of three. A side-effect of this preprocessing is that no new circular arc spans more than $180$ degrees. Theorem 4.1 summarizes the complexity of our implementation.

**Theorem 4.1.** Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two subsets of $\mathbb{R}^2$ that are bounded by circular-arc polygons. The sweep-line algorithm computes $\mathcal{A}_1 \cup \mathcal{A}_2$, $\mathcal{A}_1 \cap \mathcal{A}_2$, $\mathcal{A}_1 \setminus \mathcal{A}_2$, and $\mathcal{A}_2 \setminus \mathcal{A}_1$ in $O((n+k)\log n)$ time and $O(n+k)$ space, where $n$ denotes the overall number of straight-line segments and circular arcs of $\mathcal{A}_1$ and $\mathcal{A}_2$, and $k$ is the overall number of pairwise intersections among the segments and arcs.

For some applications it may be required to distinguish between the inner and the outer offset curves of a polygon $P$. Of course, the inner offset curves could be identified by subtracting $\mathcal{OA}_v(P,r)$ from the area bounded by $P$. (Similarly, the outer offsets would be obtained by subtracting $\mathcal{OA}_v(P,r)$ from the exterior of $P$.) However, this would require a second run of the Boolean-operation code. Classifying closed loops of the offsets by running point-in-polygon tests for each loop could be equally costly. (After all, one offset may consist of several offset loops!)

Fortunately there is a considerably simpler solution to this problem: For the two straight-line segments of the offset of a variably-weighted straight-line segment $\overline{pq}$ we maintain a pointer to $\overline{pq}$. Similarly we maintain a pointer to the appropriate vertex, $p$ or $q$, for the two circular caps of that offset. Of course, if an offset segment is split into two segments during the Bentley-Ottmann plane sweep, then these two new segments inherit the pointers from the original segment. Same for arcs. Once CAPOP has completed the union computations, we loop over all resulting closed offset curves and determine for every offset curve whether it is inside or outside of the input area defined by $P$. For every offset curve $C$ this decision is made by determining the position of

one segment or arc (selected arbitrarily from $C$) relative to the input segment referenced by its pointer. (For an offset arc we inspect the two input segments that share the input vertex referenced by the pointer.) Hence, this decision can be carried out easily in constant time per offset curve.

## 5 EXPERIMENTAL EVALUATION

We evaluated the run-time performance of CAPOP and compared it to several other software packages that support the computation of constant-radius or even variable-radius offsets. In particular, we tested the following libraries:

- The Computational Geometry Algorithms Library (CGAL) [26] supports Boolean operations on circular-arc polygons through its `Arrangement_2` package. Thus, by using CGAL it is possible to derive constant-radius as well as variable-radius offsets from polygons. Following the recommendation given in [8], in our test runs we used the `Exact_predicates_exact_constructions_kernel`.
- The Boost C++ Library (Boost) [18] makes it possible to generate variable-radius offsets by using its geometry buffer functions. However, Boost approximates circular arcs in the input by polygonal chains. By default, Boost approximates a full circle by a polygon consisting of ninety straight-line segments. (Circular arcs are approximated accordingly.)
- Clipper2 [17] is the implementation of an extension of the algorithm described by Vatti [28]. It supports Boolean operations on simple polygons. Additionally, Clipper2 offers a package for deriving constant-radius offsets of polygons. Similar to Boost, Clipper2 needs to approximate input circular arcs by polygonal chains. In our test runs, we chose to approximate a full circle by ninety straight-line segments. (That is, the same resolutions are used by Clipper2 and Boost.)
- VRONI [13] is able to compute Voronoi diagrams of points, straight-line segments, and circular arcs. We refer to [15] for details of its algorithm. Based on a Voronoi diagram it can also compute a family of constant-radius offset curves.
- OpenVoronoi [29] is an open-source library that is able to compute Voronoi diagrams of points and straight-line segments. Similar to VRONI, it uses an incremental topology-oriented algorithm that was introduced by Sugihara and Iri [25]. OpenVoronoi allows to compute constant-radius offset curves, too.

We note explicitly that CGAL is the only software package tested that relies on exact arithmetic. Thus, it does not need to deal with the peculiarities of floating-point arithmetic. More precisely, if the code contains no bug, then its output would be guaranteed to be correct. All other packages tested rely on conventional floating-point arithmetic. Hence, one should expect that they may have to struggle with numerical errors and need not always report the correct output. (While our tests hinted at some problems, we did not attempt to probe the algorithmic or numerical reliability of the codes tested in greater detail.)

In order to be able to include all packages in our test runs we started with timing the computation of constant-radius offsets. The input polygons were taken from the Salzburg Database of Polygonal Data [7]. We tested all packages on about $2700$ different simply-connected and multiply-connected polygonal areas. (That is, polygons without and with holes.) For the sake of brevity, in the sequel we will refer to these polygonal areas simply as "polygons". All tests were carried out on an Intel Core i9-10980XE processor clocked at $3.0\,\mathrm{GHz}$. In order to avoid an excessively long duration of our tests we set a time limit of 15 minutes for the processing of one input polygon.

It is obvious that a square with side length $3r$ will not admit an inner offset for, say, an offset distance $2r$. Hence, relative to the input geometry a reasonable offset distance had to be determined. And, needless to say, such a distance had to be determined for every input polygon in a fully automatic way. We resorted to the following empirically derived value that has been used by VRONI for many years of automated testing

and that has turned out to generate decent results for a very large variety of input geometries:

$$\delta := c \cdot d \cdot \frac{0.03}{\log(\log(n))},$$

where $d$ denotes the maximum of the side lengths of the bounding box of the input, and

$$c := \begin{cases} 10 & n > 2.5 \cdot 10^5, \\ 1 & \text{otherwise.} \end{cases}$$

Of course, the specific value of $\delta$ depends on the actual geometry of the input polygon and, thus, it varies among the different inputs.

Figure 4 shows the run-time results for offset distance $\delta$ as defined above. In the plot the $x$-axis corresponds to the number $n$ of vertices of the input polygons, and the $y$-axis corresponds to the run-time divided by the factor $n \log n$. Both axes are in logarithmic scale. Note that for our CAPOP (apx) test runs, we approximated every input circle by a polygon consisting of ninety straight-line segments. Thus, the inputs for CAPOP (apx) were identical to those used by Clipper2 and Boost. Since they approximate input arcs it is needless to say that Clipper2, Boost and CAPOP (apx) cannot compute the correct offset but only a polygonal approximation of the correct offset.

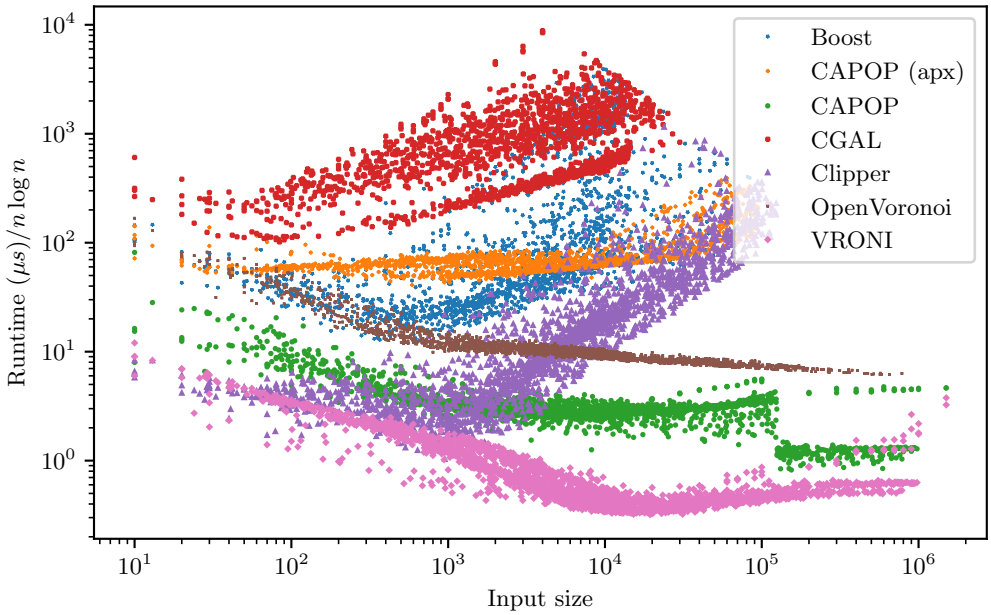



**Figure 4**: Experimental run-time results for constant-radius offsets with offset distance $\delta$.

The plot in Figure 4 indicates that VRONI, CAPOP and OpenVoronoi are the only packages whose run-times seem to be of the order $O(n \log n)$ for this test setting. Studying a similar plot where all run-times are divided by $n^2$ suggests that CGAL, Clipper2 and Boost have a (close-to) quadratic complexity. We note, though, that their run-times started to exceed the 15-minute time limit set for one process once $n$ got larger

than about $10\,000$. And none of them managed to handle input files with substantially more than $100\,000$ vertices within that time limit. OpenVoronoi ended up looping for about eleven percent of our test inputs. (This problem occurred mostly for the more complex input geometries with lots of vertices.) CAPOP (apx) ran out of memory for inputs with more than roughly $100\,000$ vertices. Finally, we remark once again that CGAL is the only code in our tests that used exact arithmetic. Hence, it could be expected to be somewhat slower than the other codes.

The run-times observed for CAPOP suggest that its Bentley-Ottmann line sweep has to handle a number $k$ of intersections (among the boundaries of the elementary offset elements) that is linear in the number $n$ of input vertices. (After all, its worst-case complexity is $O((n+k)\log n)$, recall Theorem 4.1.) Indeed, this is correct: Figure 5 shows that $1.6n \leq k \leq 3.0n$ for all our inputs. On average, $k \approx 2.2n$.

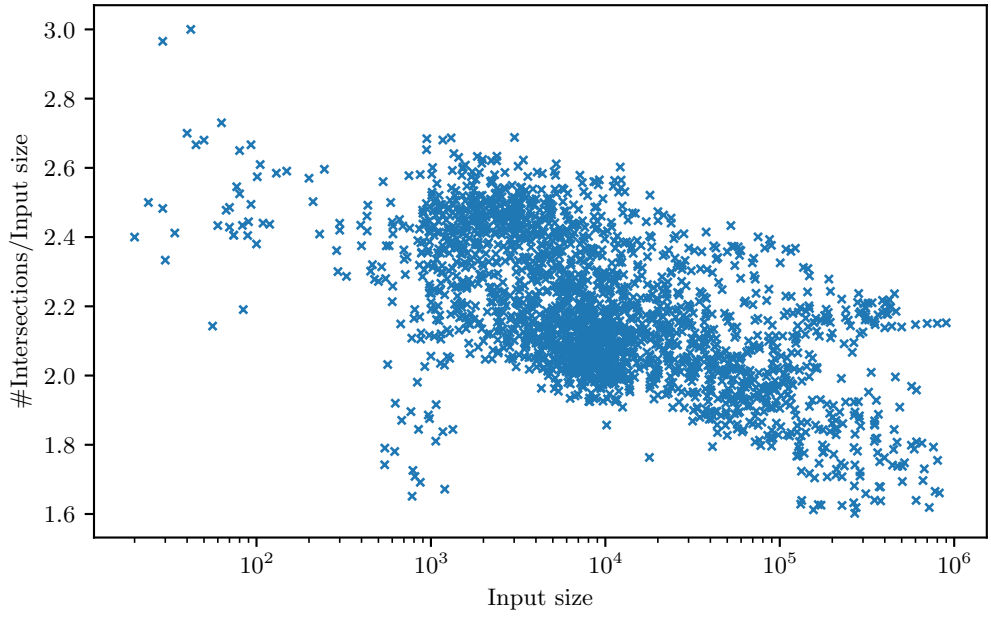



**Figure 5**: Number of intersection points encountered during the line sweep among the boundaries of all elementary offset areas for offset distance $\delta$.

Since one may suspect that the specific value chosen as offset distance has an impact on the codes that are not based on Voronoi diagrams, we ran another series of tests for which the offset distance was set to $10\delta$. In Figure 6 we see the run-time results of test runs in which the offset distance was set to $10\delta$. As one could expect, the timings did not change for VRONI and OpenVoronoi. It is interesting to see that the timings for CAPOP were also not impacted while the other packages ended up with longer run-times. To the converse, CAPOP seems to be slightly faster for $10\delta$ offsets.

This change in run-time becomes even more noticeable when setting the offset distance to $30\delta$, see Figure 7: CAPOP once again was slightly faster while Clipper2 and Boost were substantially slower than when computing offsets for offset distance $\delta$. Since we do not know their algorithms in full detail we refrain from speculating about the reason for this substantial increase in run-time. We note, though, that the version of CAPOP that uses polygonal approximations of circular arcs, CAPOP (apx), gets faster when the offset distance is increased.

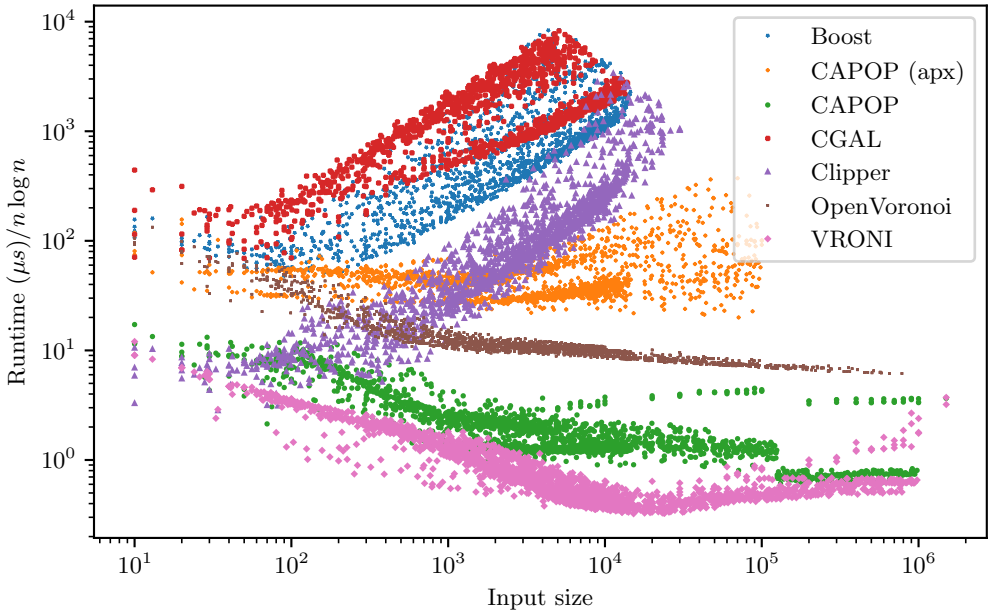



**Figure 6**: Experimental run-time results for constant-radius offsets with offset distance $10\delta$.

Hence, the mere fact that Clipper2 and Boost resort to polygonal approximations of circular arcs does not seem to supply an explanation for this increase in run-time. Concerning CAPOP, an explanation for its increased speed for larger offset values is readily found when studying the number of intersections encountered during the Bentley-Ottmann line sweep: While we have an average of about $2.2n$ intersections for offsets with distance $\delta$, the average number of intersections goes down to about $1.8n$ for $30\delta$ offsets.

Summarizing, our experiments clearly indicate that CAPOP is the fastest code based on Boolean operations for offsetting polygons with about $1000$ or more vertices. It is also evident that it benefits from its ability to handle genuine circular arcs as input elements. Still, the computation of one offset is slightly more costly when using CAPOP than when using VRONI. Of course, the balance would shift even more in favor of VRONI (and OpenVoronoi) if a family of offset curves were to be generated for the same input geometry: While VRONI and OpenVoronoi would simply re-use the Voronoi diagram computed for obtaining a second set of offset curves for a different offset distance, CAPOP would have to start from scratch and carry out the full union-finding computation once again. (And the time spent on obtaining one offset if the Voronoi diagram is known is a tiny fraction of the time spent on computing the Voronoi diagram.)

However, in contrast to VRONI, CAPOP (and the other codes that compute offsets based on Boolean operations) can also generate variable-radius offsets. To probe this issue further, in another series of experiments we let CAPOPcompute variable-radius offsets for vertex weights randomly chosen in the range 1 to 10. (The base offset distance $\delta$ was defined as above.) Our tests suggested that choosing random weights has about the same impact on the run-time of CAPOP as increasing the offset distance. That is, random weights caused its run-time consumption to decrease moderately.

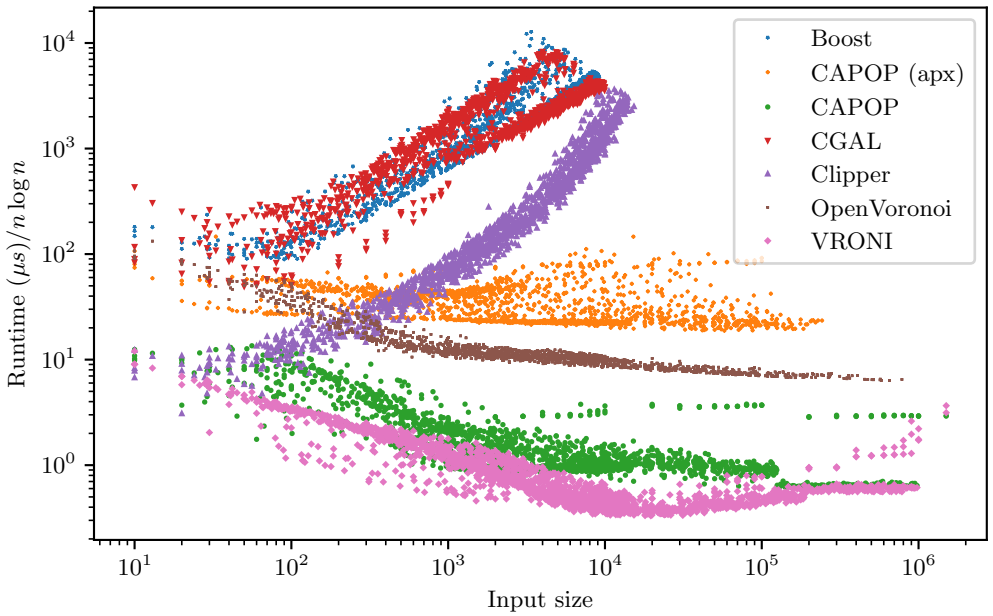



**Figure 7**: Experimental run-time results for constant-radius offsets with offset distance $30\delta$.

## 6 CONCLUSION

We applied our new software package CAPOP for computing Boolean operations on planar areas bounded by curves that consist of straight-line segments and circular arcs to the computation of constant-radius and variable-radius offset curves. A series of extensive tests showed that CAPOP is slower than constant-radius offsetting based on Held's VRONI, but faster than all the other software packages in our test suite, at least for inputs with more than about 1000 vertices. Furthermore, its run-time complexity does not deteriorate in the presence of variably-weighted input segments. Our tests also make it evident that the fact that CAPOP can handle genuine circular arcs without need for any polygonal approximation allows it save a substantial amount of run-time.

## ACKNOWLEDGEMENTS

This work was supported by the Austrian Science Fund (FWF): Grant P31013.

## ORCID

*Martin Held*, http://orcid.org/0000-0003-0728-7545
*Stefan de Lorenzo*, http://orcid.org/0000-0003-4981-805X
*Peter Palfrader*, http://orcid.org/0000-0002-5796-6362

## REFERENCES


[1] Balaban, I.J.: An Optimal Algorithm for Finding Segments Intersections. In Proc. 11th Int. Sympos. Comput. Geom., 211–219, 1995. http://doi.org/10.1145/220279.220302.

[2] Bentley, J.L.; Ottmann, T.A.: Algorithms for Reporting and Counting Geometric Intersections. IEEE Trans. Comput., C-28(9), 643–647, 1979. http://doi.org/10.1109/TC.1979.1675432.

[3] Berberich, E.; Eigenwillig, A.; Hemmer, M.; Hert, S.; Mehlhorn, K.; Schömer, E.: A Computational Basis for Conic Arcs and Boolean Operations on Conic Polygons. In Annu. Europ. Symp. Algorithms, 174–186. Springer-Verlag, 2002. http://doi.org/10.1007/3-540-45749-6_19.

[4] Chazelle, B.; Edelsbrunner, H.: An Optimal Algorithm for Intersecting Line Segments in the Plane. J. ACM, 39(1), 1–54, 1992. http://doi.org/10.1145/147508.147511.

[5] Chen, X.; McMains, S.: Polygon Offsetting by Computing Winding Numbers. In ASME 2005, vol. 4739, 565–575, 2005. http://doi.org/10.1115/DETC2005-85513.

[6] de Smith, M.; Goodchild, M.; Longley, P.: Geospatial Analysis: A Comprehensive Guide. Winchelsea Press, 2018. ISBN 978-1912556038.

[7] Eder, G.; Held, M.; Jasonarson, S.; Mayer, P.; Palfrader, P.: Salzburg Database of Polygonal Data: Polygons and Their Generators. Data in Brief, 31, 105984, 2020. ISSN 2352-3409. http://doi.org/10.1016/j.dib.2020.105984.

[8] Fogel, E.: Personal Communication, 2022.

[9] Fogel, E.; Halperin, D.; Wein, R.: CGAL Arrangements and their Applications: A Step-by-Step Guide, vol. 7. Springer-Verlag, 2012.

[10] Gong, Y.X.; Liu, Y.; Wu, L.; Xie, Y.B.: Boolean Operations on Conic Polygons. J. Comput. Sci. Technol., 24(3), 568–577, 2009. http://doi.org/10.1007/s11390-009-9246-z.

[11] Hadwiger, H.: Minkowskische Addition und Subtraktion beliebiger Punktmengen und die Theoreme von Erhard Schmidt. Math Z, 53, 210–218, 1950. http://doi.org/0.1007/BF01175656.

[12] Held, M.: On the Computational Geometry of Pocket Machining, vol. 500 of Lecture Notes Comput. Sci. Springer-Verlag, 1991. ISBN 3-540-54103-9.

[13] Held, M.: VRONI and ArcVRONI: Software for and Applications of Voronoi Diagrams in Science and Engineering. In Proc. 8th Int. Symp. Voronoi Diagrams in Science & Engineering, 3–12. IEEE, 2011. http://doi.org/10.1109/ISVD.2011.9.

[14] Held, M.; de Lorenzo, S.: Weighted Skeletal Structures for Computing Variable-Radius Offsets. Comput. Aided Design & Appl., 18(5), 875–889, 2021. http://doi.org/10.14733/cadaps.2021.875-889.

[15] Held, M.; Huber, S.: Topology-Oriented Incremental Computation of Voronoi Diagrams of Circular Arcs and Straight-Line Segments. Comput. Aided Design, 41(5), 327–338, 2009. http://doi.org/10.1016/j.cad.2008.08.004.

[16] Held, M.; Huber, S.; Palfrader, P.: Generalized Offsetting of Planar Structures Using Skeletons. Comput. Aided Design & Appl., 13(5), 712–721, 2016. http://doi.org/10.1080/16864360.2016.1150718.

[17] Johnson, A.: Clipper2. http://www.angusj.com/clipper2/Docs/Overview.htm.

[18] Koranne, S.: Boost C++ Libraries. In Handbook of Open Source Tools, 127–143. Springer-Verlag, 2011.

[19] Li, X.; , J.L.; Ni, P.; Wang, Y.; Song, Y.; Tong, L.: Novel Path Generation Algorithm for High-Speed Pocket Milling. Int. J. Prod. Res., 52(2), 397–404, 2014. http://doi.org/10.1080/00207543.2013.828172.

[20] Martínez, F.; Rueda, A.J.; Feito, F.R.: A New Algorithm for Computing Boolean Operations on Polygons. Comput. Geosci., 35(6), 1177–1185, 2009. http://doi.org/10.1016/j.cageo.2008.08.009.

[21] Nievergelt, J.; Preparata, F.: Plane-Sweep Algorithms for Intersecting Geometric Figures. C. ACM, 25(10), 739–747, 1982. http://doi.org/10.1145/358656.358681.

[22] Park, S.; Chung, Y.: Offset Tool-Path Linking for Pocket Machining. Comput. Aided Design, 34(5), 299–308, 2003.

[23] Persson, H.: NC Machining of Arbitrarily Shaped Pockets. Comput. Aided Design, 10(3), 169–174, 1978. http://doi.org/10.1016/0010-4485(78)90141-0.

[24] Schneider, R.: Convex Bodies: The Brunn-Minkowski Theory. Cambridge University Press, 2nd ed., 2013. http://doi.org/10.1017/CBO9781139003858.

[25] Sugihara, K.; Iri, M.: Construction of the Voronoi Diagram for "One Million" Generators in Single-Precision Arithmetic. Proc. of the IEEE, 80(9), 1471–1484, 1992. http://doi.org/10.1109/5.163412.

[26] The CGAL Project: CGAL User and Reference Manual. CGAL Editorial Board, 5.0 ed., 2019. https://doc.cgal.org/5.0/Manual/packages.html.

[27] Van Wyk, C.J.: Clipping to the Boundary of a Circular-Arc Polygon. Comput. Vision Graph. Image Process., 25(3), 383–392, 1984. http://doi.org/10.1016/0734-189X(84)90202-0.

[28] Vatti, B.R.: A Generic Solution to Polygon Clipping. C. ACM, 35(7), 56–63, 1992. http://doi.org/10.1145/129902.129906.

[29] Wallin, A.: OpenVoronoi. https://github.com/aewallin/openvoronoi.

[30] Wang, Z.J.; Lin, X.; Fang, M.E.; Yao, B.; Peng, Y.; Guan, H.; Guo, M.: Re2l: An Efficient Output-Sensitive Algorithm for Computing Boolean Operations on Circular-Arc Polygons and its Applications. Comput. Aided Design, 83, 1–14, 2017. http://doi.org/10.1016/j.cad.2016.07.004.

[31] Wein, R.: Exact and Approximate Construction of Offset Polygons. Comput. Aided Design, 39(6), 518–527, 2007. http://doi.org/10.1016/j.cad.2007.01.010.

[32] Wolff, A.; Knipping, L.; van Kreveld, M.; Strijk, T.; Agarwal, P.: A Simple and Efficient Algorithm for High-Quality Line Labelling. In P. Atkinson, ed., GIS and GeoComputation: Innovations in GIS 7, 147–160. CRC Press, 2000. http://doi.org/10.1201/9781482268263. ISBN 9780429180828.

[33] Žalik, B.; Zadravec, M.; Clapworthy, G.: Construction of a Non-Symmetric Geometric Buffer from a Set of Line Segments. Comput. Geosci., 29(1), 53–63, 2003. http://doi.org/10.1016/S0098-3004(02)00076-6.