Generating Straight Outlines of 2D Point Sets and Holes using Dominant Directions or Orthogonal Projections

Melanie Pohl and Dirk Feldmann
Fraunhofer IOSB, Department Scene Analysis, Ettlingen, Germany

Keywords: Point Set, Outline, Boundary, Hull, Concave, Building, Footprint, Dominant Direction, Preference Angles.

Abstract: Representing the shape of finite point sets in 2D by simple polygons becomes a challenge if the resulting outline needs to be non-convex and straight with only few, distinct edges and angles. Such outlines are usually sought in order to border point sets that originate from man-made objects, e.g., for the purpose of building reconstruction from LIDAR data. Algorithms for computing hulls of point sets obtained from such structures usually yield polygons having too many edges and angles and may thus not capture the actual shape very well. Furthermore, many existing approaches cannot handle empty domains within the boundaries of a point set (*holes*).
In this paper, we present methods that create straight, non-convex outlines of finite 2D point sets and of possibly contained holes. The resulting polygons feature fewer vertices and angles than hulls and can thus faithfully represent objects of angular shapes.

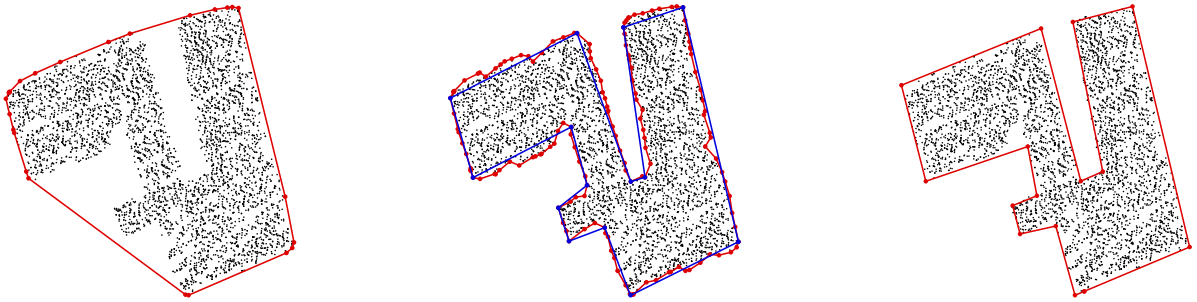
1 INTRODUCTION

The problem of finding simple, planar polygons that outline a finite point set in 2D Euclidean space arises in many practical applications. In order to automatically create polygonal 3D models (*meshes*) of buildings or even entire cities, for instance, it is common practice to employ point data acquired by LIDAR devices or from aerial photographs by means of photogrammetric methods. These points are usually based in a plane section in 2D Euclidean space. The task of creating meshes from such finite point sets comprises the detection of their outlines (also called *footprints*) (Vosselman, 1999). The most notable solutions to this kind of problem are probably *convex hulls*. In the case of constructing 3D models of buildings, convex hulls may be inappropriate, because the outlines of many buildings or building complexes are not convex. A better approach would be the usage of methods for finding non-convex (*concave*) hulls. But since man-made objects like buildings tend to have lots of long, straight edges, which furthermore enclose few angles of discrete measures (e.g., $\pm\frac{\pi}{2}$), true hulls rarely reflect the desired outlines very well as shown in Figure 1: The outline of the building obtained by the Concave Hull Algorihtm (Moreira and Santos, 2007) is clearly preferable over the convex hull, but it has too many vertices and is not as straight as the underlying buildings' exterior boundaries really are. For the task of building reconstruction, having nicely straight outlines with only few vertices is desirable, because it simplifies the process of 3D model generation and the results are more realistic. Straightening outlines obtained from concave hulls using methods like the popular Ramer-Douglas-Peucker Algorithm (Ramer, 1972; Douglas and Peucker, 1973) is not a general solution due to unintentional removal of certain corners as shown in Figure 1(b). Please note that we do not deal with buildings having strongly curved boundaries, even though these can be found in many architectural landmarks.

Furthermore, input data obtained by measurements are always noisy and the distribution of points is likely to be non-uniform. Since hulls must include every point of the set they enclose, unwanted outliers will further diminish the desired quality of the resulting outlines.

The boundaries of finite point sets located in a plane section in 2D Euclidean space may also feature larger domains where no points are present at all (called *holes*). In the process of building reconstruction from LIDAR data, for instance, such empty domains frequently result from buildings or building complexes having inner courtyards or atria. In order to give faithful representations of

Pohl, M. and Feldmann, D.
Generating Straight Outlines of 2D Point Sets and Holes using Dominant Directions or Orthogonal Projections.
DOI: 10.5220/0005720300570069
In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016) - Volume 1: GRAPP*, pages 59-71
ISBN: 978-989-758-175-5



(a) convex hull. (b) concave hull ($k = 100$). (c) our method (orthogonal projection).

Figure 1: Using the convex hull or the Concave Hull Algorithm to generate outlines of 2D point sets obtained from man-made structures, like buildings, leads to shapes that do not necessarily reflect the actual object very well. Straightening the resulting outlines using Ramer-Douglas-Peucker Algorithm (blue line in (b)) only little improves the result. Our method based on orthogonal projection creates outlines that represent such objects more accurately. In Figure 1(b), k denotes the number of neighbors considered (see Section 2.1).

the underlying objects, these holes also need to be bounded by simple polygons.

We present two methods for finding simple outline polygons of 2D point sets which feature fewer vertices and angles than hulls in order to faithfully represent shapes of structures that are typically man-made or originate from technical processes. These structures may contain holes and are characterized by straight edges and may be non-convex. Frequently, they are moreover (near-)rectangular and have only few angles and corners. Our methods are based on the *Concave Hull* algorithm presented in (Moreira and Santos, 2007). Although our application aims at generating outlines for the task of building reconstruction, and we therefore demonstrate and evaluate our method in the context of that application, the proceedings presented in this work are general and versatile.

2 RELATED WORK

Finding outlines of 2D point sets representing structures that have straight edges and enclose only few angles in order to obtain "good" representations of the underlying objects is an inherently ambiguous problem and strongly depends on the application. As pointed out in (Galton and Duckham, 2006), there is no single correct outline or *footprint* of a set of points. Computing outlines by means of convex hulls, for instance, is probably reasonable if the underlying objects are (almost) convex themselves. Convex hulls are well-studied and many algorithms are known for its computation (de Berg et al., 2008), but in case of non-convex objects, convex hulls naturally yield unsatisfactory outlines. Therefore, we do not further enter on methods for their generation.

α-shapes (Edelsbrunner et al., 1983) are a generalization of convex hulls and allow for computing non-convex (*concave*) outlines of 2D point sets. The α-shape of a finite point set $\mathcal{S} \subset \mathbb{R}^2$ is the boundary of the α-hull of $\mathcal{S}$. The latter can be obtained by computing the Delaunay triangulation of $\mathcal{S}$ and removing those k-simplexes, $0 \leq k \leq 2$, whose open circumscribed circles have radii $\geq \frac{1}{\alpha}$ or that contain any other $\mathbf{p} \in \mathcal{S}$. Using $\alpha = 0$, the α-shape of $\mathcal{S}$ is defined to be its convex hull.

Another approach based on Delaunay triangulation are *characteristic shapes* (χ-shapes) as presented in (Duckham et al., 2008). The algorithm repeatedly removes the longest exterior edge whose length is greater than some value L from the initial Delaunay triangulation of $\mathcal{S}$, provided that the remaining exterior edges form a simple polygon. It is also possible to parametrize χ-shapes by a normalized length parameter $\lambda = \frac{l - l_{\min}}{l_{\max} - l_{\min}}$, where l denotes the length of an edge and $l_{\min}$ and $l_{\max}$ are the minimal and maximal edge length from the Delaunay triangulation, respectively.

The resulting outlines produced by α- and χ-shapes depend on the choice of the respective parameters α and l (or λ). Their choice involves a-priori knowledge about the shape of the underlying point set, or appropriate values must be found using heuristics or by trial and error.

The *Concave Hull* Algorithm in (Moreira and Santos, 2007) is based on the idea of Jarvis' March aka. Gift Wrapping Algorithm (Jarvis, 1973) for computing convex hulls: Instead of considering all elements from the set of remaining points $\mathcal{S}$, the Concave Hull Algorithm only considers the $3 \leq k \leq |\mathcal{S}|$ nearest neighbors of the point that has been added to the hull last. Using greater values for k results in larger neighborhoods being considered, and the con-

cave hull becomes "more convex" as k increases. If $k = |\mathcal{S}|$, the algorithm computes the convex hull, because the set of points considered becomes the same as with Jarvis' March. Since our method is based on the Concave Hull algorithm, it is briefly summarized in Section 2.1.

The strength of Concave Hull is its simplicity and that there is no need to compute any (Delaunay) triangulation of which most information about connectivity is not needed at all.

In (Asaeedi et al., 2013), the *Alpha-Concave Hull* is presented, which is a simple polygon whose interior angles are $< \pi + \alpha$. If the parameter $\alpha = \pi$, the Alpha-Concave Hull is the minimal polygon of $\mathcal{S}$; in case of $\alpha = 0$, it is the convex hull.

In the field of photogrammetry, methods for generating rectangular outlines of buildings based on line fitting of contour points, like the one presented in (Vosselman, 1999), seem to be popular. The drawback of these methods is their limitation to the creation of only rectangular outlines. But even in the application of detecting building outlines, there are often boundaries that are not rectangular, at least in some corners, like in cities that grew over centuries or that were not planed from scratch.

There are also approaches based on statistical estimates of the most appropriate outline for a given set of points (Wang et al., 2006).

Furthermore, among the methods presented so far, only α-shapes can create outlines of *holes* in 2D (see Section 1), but the resulting outlines are overly jagged. Representing holes, however, is essential to obtain faithful representations of atria or building complexes enclosing inner courtyards. The solutions presented in (Bendels et al., 2006), (Wang and Oliveira, 2007) and (Wu and Chen, 2014) address the detection (and also filling) of holes in point clouds that represent surfaces in 3D, so that these methods are difficult to apply to our problem.

2.1 Concave Hull Algorithm

To generate *outlines*, also called *boundaries* in the remainder of this work, we adopt the Concave Hull Algorithm in (Moreira and Santos, 2007) which is summarized below, because it is the foundation of our approach. For a more detailed description, we refer to the original article.

Let $\mathcal{S}$ be a finite set of points in 2D Euclidean space where $\mathbf{p}_i \neq \mathbf{p}_j \; \forall \mathbf{p}_i,\ \mathbf{p}_j \in \mathcal{S} \wedge i \neq j$. Let furthermore $\mathbf{p}' \in \mathcal{S}$ be the point having minimal y-coordinate and $\mathcal{C}_n = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n)$ the list of vertices of the outline computed so far.

Given $k \in \mathbb{N},\ 3 \leq k < |\mathcal{S}|$, start with $n = 0,\ n \in \mathbb{N}$, $\mathcal{S}_{-1} = \mathcal{S},\ \mathcal{C}_0 = (\mathbf{v}_0 = \mathbf{p}')$, $\mathbf{p} = \mathbf{p}'$ and $\mathbf{e} = -\mathbf{x}$, where $-\mathbf{x}$ denotes the negative x-axis. Remove $\mathbf{p}$ from the current set of points to obtain $\mathcal{S}_n = \mathcal{S}_{n-1} \backslash \mathbf{p}$ and compute the set $\mathcal{K}_k(\mathbf{p}) \subset \mathcal{S}_n$ of $\mathbf{p}$'s k nearest neighbors. The points $\mathbf{q}_i \in \mathcal{K}_k(\mathbf{p}), 0 \leq i < k$ are sorted according to the clockwise angle φ_i enclosed between the directed edge $\mathbf{e}_i = (\mathbf{q}_i - \mathbf{p})$ and edge $\mathbf{e}$ such that $j = 0$ be the index of the point $\mathbf{p}_j = \mathbf{q}_i$ where φ_i is largest, $j = 1$ the one where φ_i is second largest, etc.

Now let $\mathbf{e}_j = (\mathbf{p}_j - \mathbf{p})$ denote the directed edge from the last vertex of the outline polygon to the point $\mathbf{p}_j \in \mathcal{K}_k(\mathbf{p}),\ j = 0, 1, \ldots k$. Starting at $j = 0$, check if $\mathbf{e}_j$ does not intersect any of the existing edges $\overline{\mathbf{v}_{n-2}\mathbf{v}_{n-3}}, \overline{\mathbf{v}_{n-3}\mathbf{v}_{n-4}}, \ldots, \overline{\mathbf{v}_1\mathbf{v}_0}$ of the polygon. If $\mathbf{e}_j$ intersects an existing edge of the hull, retry using $\mathbf{e}_{j+1}$, if $j < k$. In case no $\mathbf{e}_j$ remains, the outline computed so far is discarded, k is increased and the algorithm is restarted using $n = 0$ again. Otherwise, if no intersection of $\mathbf{e}_j$ with an existing edge is found, set $\mathbf{p}_{j'} = \mathbf{p}_j$ to be the best match in $\mathcal{K}_k(\mathbf{p})$, add $\mathbf{p}_{j'}$ to the list of vertices, i.e., $\mathcal{C}_{n+1} = \mathcal{C}_n \cup \mathbf{p}_{j'}$, and set $\mathbf{e} \leftarrow (\mathbf{v}_n - \mathbf{p}_{j'})$ as the backwards directed edge formed by the last two vertices of the hull. The process is then restarted setting $n \leftarrow n + 1$ and $\mathbf{p} \leftarrow \mathbf{p}_{j'}$ until $\mathbf{p} = \mathbf{p}'$ again or all points in $\mathcal{S}$ haven been processed. In order to ensure that $\mathbf{p} = \mathbf{p}'$ a second time and that the algorithm terminates, $\mathbf{p}'$ must be added to $\mathcal{S}_n$ again after the third iteration. When $\mathbf{p} = \mathbf{p}'$ the second time, it is necessary to check if all remaining points in $\mathcal{S}_n$ are actually inside the resulting polygon $\mathcal{C}_n$ so that it is actually a hull in the proper sense. Otherwise, the outline is also discarded, k is increased and the algorithm is restarted using $n = 0$.

3 COMPUTING OUTER BOUNDARIES

In this section we present two different approaches for computing simple polygons that outline a given finite point set $\mathcal{S} \subset \mathbb{R}^2$. The outline's edges only enclose angles of a few, distinct values, and we call them *angular outlines* in the following.

The first method relies on the concave hull and determination of dominant edge directions. Our second method modifies the Concave Hull Algorithm to directly compute an angular outline.

The advantage of the first method, presented in Section 3.1, is that the hull can be straightened by two predefined main angles. This idea follows the approach, that buildings are mostly oriented along two

directions to guarantee parallel walls. Furthermore, if we also take the idea of perpendicularity into account, we can force the straightened edges to form right angles. Therefore, data that are very noisy or contain holes can be bordered along the main directions or perpendicular to it.

The second approach in Section 3.2 is preferable if the underlying structure is not well approximated by strictly rectangular outlines or if the edges of the shape enclose more than two distinct angles. However, for this method to work, the initial orientation of the underlying shape within a Cartesian coordinate system must be provided.

3.1 Angular Outlines from Dominant Directions

Let $\mathcal{C}_m$ be the list of vertices of a concave hull of $\mathcal{S}$ and $\mathcal{E} = (\mathbf{e}_0, ..., \mathbf{e}_m)$ the list of edges with $\mathbf{e}_i = \overline{\mathbf{v}_i \mathbf{v}_{i+1}}$, $i = 0, 1, \ldots, (m-1)$ and $\mathbf{e}_m = \overline{\mathbf{v}_m \mathbf{v}_0}$. Since we do not consider the orientation of an edge, each difference vector $\mathbf{e}_i$ is set to $\mathbf{e}_i \leftarrow \operatorname{sgn}(e_{i,y}) \cdot \mathbf{e}_i$. For each edge $\mathbf{e}_i$, we compute the angle $\alpha_i \in [0, \pi]$ that is enclosed with the positive $x-$axis. If $\alpha_i = \pi$, we set $\alpha_i \leftarrow 0$. The length of an edge $\mathbf{e}_i$ is denoted by l_i. We analyze the directions and lengths of each edge of the concave hull polygon and compute a length-weighted angle histogram $H(\alpha_i)$. Angle α_i is assigned to the respective bin of width w according to

$$-\frac{w}{2} + (2n-1) \cdot \frac{w}{2} \leq \alpha_i < (2n-1) \cdot \frac{w}{2} + \frac{w}{2}, \quad n \in \{1, 2, ..., m\}.$$

The value of the n-th bin is computed as $\sum_i l_i$.

From this histogram, we compute the two dominant angles β and γ of the concave hull using one of the following two options:

1. Approximate the histogram $H(\alpha_i)$ by a bimodal distribution. Then β and γ correspond to the angles of the two maximums of the approximation.

2. β corresponds to the angle of the maximum value in H and γ is offset by $\pm\frac{\pi}{2}$:

$$\beta = \underset{0 \leq \alpha_i < \pi}{\operatorname{argmax}} \left(H(\alpha_i) \right), \quad \gamma = \begin{cases} \beta + \frac{\pi}{2}, & \beta < \frac{\pi}{2} \\ \beta - \frac{\pi}{2}, & \beta \geq \frac{\pi}{2}. \end{cases}$$

The advantage of option 1 ("peaks" option) is that the edges of the boundary obtained from straightening of the concave hull can enclose an arbitrary but fixed angle that equals $|\beta - \gamma|$. Option 2 ("90°" option) ensures that these edges always enclose right angles.

From β and γ we compute the corresponding dominant directions

$$\mathbf{d}_\varphi = \begin{pmatrix} \cos(\varphi) \\ \sin(\varphi) \end{pmatrix}, \quad \varphi \in \{\beta, \gamma\}$$

and assign labels ρ_φ to all edges of the concave hull according to the dominant angle they belong to:

$$\rho_\varphi = \underset{\varphi}{\operatorname{argmin}} \{ \measuredangle(\mathbf{e}_i, \mathbf{d}_\varphi) \}, \ \varphi \in \{\beta, \gamma\}.$$

To simplify the straightening procedure it is advisable to shift the start point in the sequence of vertices of the concave hull and the entries of the label vector to ensure a tuple of successive vertices having the same label is not split by the start or end vertex. Afterwards, we collect every tuple of successive vertices $\mathcal{T}_{\rho_\varphi} = (\mathbf{t}_1, \mathbf{t}_2, ...)$ from the concave hull that belong to the same label ρ_φ and compute the line ℓ through the center of gravity along the corresponding main direction $\mathbf{d}_\varphi$:

$$\ell : \vec{x} = \frac{1}{|\mathcal{T}_{\rho_\varphi}|} \sum_{\mathbf{t}_i \in \mathcal{T}_{\rho\varphi}} \mathbf{t}_i + \mu \, \mathbf{d}_\varphi.$$

To limit these lines, we have to distinguish two cases: If the foregoing (or subsequent) tuple belongs to none of the dominant angles/directions we project the first (or accordingly the last vertex) onto that line. If the foregoing (or subsequent) tuple belongs to the other dominant angle/direction, the end points are specified by the intersection point of the line corresponding to the foregoing (or subsequent) tuple and the current tuple's line.

3.2 Angular Outlines from Orthogonal Projection

Our second approach to computing angular boundaries modifies the Concave Hull Algorithm in Section 2.1 by projecting the edges of the hull onto orthogonal line segments and adjusting them. First, we explain the process of projection onto orthogonal line segments. This yields a strictly rectangular outline, which is only a "good" representation if the underlying shape is (almost) rectangular, too. The proceeding is then extended by three modifications that improve the shape of the outline in situations where strictly rectangular outlines may be perceived as unsatisfactory representations.

3.2.1 Orthogonal Projection

In this section, we use the same symbols from the description of the Concave Hull Algorithm in Section 2.1. In addition, let $\delta_i = i \cdot \frac{\pi}{2}, i = 0, 1, \ldots, 4$.

When the Concave Hull has determined a candidate $\mathbf{p}_{j'}$ among the sorted k nearest neighbors $\mathcal{K}_a(\mathbf{p})$ of the current point $\mathbf{p} \in \mathcal{S}_n$ that satisfies the requirements for being a new vertex of the outline polygon, we modify the algorithm to additionally compute

$$\delta' = \underset{\delta_i}{\operatorname{argmin}} \left(\left| \delta_i - \varphi_{j'} \right| \right)$$

where φ_j is the angle associated with $\mathbf{p}_j$ such that δ' is the angle that deviates least from $\varphi_{j'}$. If δ' happens to be 0 or 2π, we discard $\mathbf{p}_{j'}$ and proceed with the next candidate in $\mathcal{K}_a(\mathbf{p})$, if there is any left. Otherwise, $\mathbf{p}_{j'}$ is discarded from the set of remaining points, just like with the Concave Hull.

Using δ', we compute the corresponding direction $\mathbf{d}$ and an orthogonal vector $\mathbf{u}$ pointing to the "right-hand side" of the ray R from $\mathbf{p}$ in direction $\mathbf{d}$, away from where the majority of points of $\mathcal{S}$ is located:

$$\mathbf{d} = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}, \quad \mathbf{u} = \begin{pmatrix} \sin(\theta) \\ -\cos(\theta) \end{pmatrix}$$

where $\theta = \theta_n + (\pi - \delta')$ and θ_n is the angle enclosed by the last edge e of the outline computed so far and the x-axis. Please notice that we have to use $\pi - \delta'$ instead of δ' to compute θ, because we chose clockwise angles in $[0, 2\pi]$ with the Concave Hull Algorithm. The outer boundary is thus oriented counterclockwise. The vector $\mathbf{e}_{j'} = (\mathbf{p}_{j'} - \mathbf{p})$, which corresponds to edge $\overline{\mathbf{p}_{j'}\mathbf{p}}$, is then projected onto $\mathbf{d}$ and $\mathbf{u}$ to obtain the lengths $l_d = \mathbf{d} \cdot \mathbf{e}_{j'}$ and $l_u = \mathbf{u} \cdot \mathbf{e}_{j'}$, respectively. Next, the new vertex $\mathbf{q}$ to be added to the current outline $\mathcal{C}_n = (\mathbf{v}_0, \mathbf{v}_1, \ldots, \mathbf{v}_n)$ is computed as

$$\mathbf{q} = \mathbf{p} + |l_d| \cdot \mathbf{d} + \mathbf{w}$$

where

$$\mathbf{w} = \begin{cases} \max(l_u, 0) \cdot \mathbf{u}, & \delta' = \pi \\ l_u \cdot \mathbf{u}, & \text{otherwise} \end{cases} \tag{1}$$

Thus, if an edge is continued (i.e., $\delta' = \pi$), $\mathbf{w}$ is either zero or points towards the right-hand side. Otherwise, if the new edge introduces a turn, it may also point to the left-hand side of R. The outline polygon is then updated by considering the following two cases:

1. If $\mathcal{C}_n$ contains ≥ 2 vertices and the edge is about to be continued, we replace the last vertex by $\mathbf{q}$ and add $\mathbf{w}$ to the second last vertex:

$$\mathbf{v}_n \leftarrow \mathbf{q}, \quad \mathbf{v}_{n-1} \leftarrow (\mathbf{v}_{n-1} + \mathbf{w}) \tag{2}$$

2. Otherwise, if $\delta' = \frac{\pi}{2}$ or $\delta' = \frac{3\pi}{2}$ and the outline is about to make a left or right turn, respectively, or if it contains only one vertex so far, we append $\mathbf{q}$ to $\mathcal{C}_n$ and add $\mathbf{w}$ to $\mathbf{v}_n$:

$$\mathbf{v}_{n+1} = \mathbf{q}, \quad \mathbf{v}_n \leftarrow (\mathbf{v}_n + \mathbf{w}) \tag{3}$$

In the next iteration $n+1$, we proceed with $\mathbf{q}$ instead of $\mathbf{p}_{j'}$, after having removed $\mathbf{p}_{j'}$ from the point set $\mathcal{S}_{n+1}$. As with the Concave Hull algorithm, we check for self-intersections of the resulting polygon. In positive cases, we also discard the polygon and start the algorithm from the beginning using $k+1$. Likewise, we terminate if $\mathbf{p}_{j'} = \mathbf{p}'$ again, but since $\mathcal{C}_n$ is obviously not a hull, it does not necessarily include all points in $\mathcal{S}$. Therefore, we omit the final check of whether all points in $\mathcal{S}$ are located strictly inside $\mathcal{C}_n$. Figure 2 illustrates the proceeding described above.

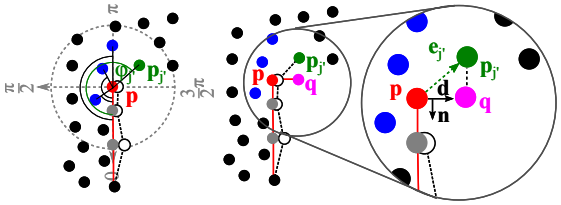
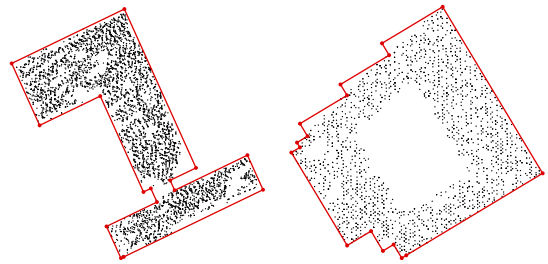


Figure 2: Creating angular outline by means of orthogonal projection is based on the Concave Hull Algorithm, but instead of adding $\mathbf{p}_{j'}$ to the list vertices, its projection $\mathbf{q}$ onto $\mathbf{d}$ is added.

Due to the choice of $\mathbf{w}$ in Equation 1, we ensure that edges are dragged only to the "outside" of the shape and that more points (including $\mathbf{p}_{j'}$) are located "inside" the resulting outline, if an edge is continued. In case of left or right turns, an edge may also become dragged towards the "inside" ($l_u < 0$) in order to enclose the point set $\mathcal{S}$ more tightly. However, this may cause some points of $\mathcal{S}$ to be found again among the k nearest neighbors $\mathcal{K}_{a+1}(\mathbf{p})$ in the next iteration. Depending on the choice of k, this can lead to self-intersections and the rejection of the current k. The results of this proceeding are satisfactory in case of shapes that are (almost) rectangular everywhere. In cases where a shape's corners deviate too much from $\pm\frac{\pi}{2}$, e.g., in case of point sets derived from ancient or architecturally unusual buildings, the outlines are less appealing (see Figure 3).

3.2.2 Improvements for Non-Rectangular Shapes

To make the outline match non-regular shapes more accurately, e.g., if one or more corners deviate too much from $\pm\frac{\pi}{2}$, we introduce some modifications to the proceeding described in Section 3.2.1. First, we can simply omit the addition of $\mathbf{w}$ to the predecessor of the last vertex in Equation 2 when an edge is continued. As a result, the continued edge is not necessarily perpendicular to the previous one anymore, but the outline captures a shape more accurately at corners where two edges join at angles $\varphi > \frac{\pi}{2}$. Corners



(a) (b)

Figure 3: Pure orthogonal projection and dragging continued edges only to the outside yields appealing results if the underlying shape is almost rectangular itself (a), but causes the appearance of unwanted steps if corners are present that are not rectangular (b).

where $\varphi < \frac{\pi}{2}$ remain approximated by steps, because the line segment of a continued edge is only allowed to being dragged outside due to Equation 1. Thus, we modify Equation 1 to obtain Equation 4:

$$\mathbf{w} = \begin{cases} \max(l_u, 0) \cdot \mathbf{u}, & \text{if } \delta' = \pi \wedge \textit{outside} = \textit{true} \\ l_u \cdot \mathbf{u}, & \text{otherwise.} \end{cases} \quad (4)$$

The predicate *outside* is set to *false* if $\delta' = \frac{\pi}{2}$ or $\delta' = \frac{3\pi}{2}$. It is set to *true* if $\delta' = \pi$ and if $l_u > 0$ for any point along the continued edge. In this way, a continued edge may also become dragged towards the inside and enclose $\mathcal{S}$ more tightly, as long as there is no need to drag the edge outside again, i.e., $l_u < 0$ over the whole distance the edge is continued.

Dragging the last vertex of a continued edge towards the inside can cause more points of $\mathcal{K}_u(\mathbf{p})$ to lie on the right-hand side of the edge, because $\mathbf{p}_{j'}$ is not necessarily the first point in the descending order of enclosed, clockwise angles. Those points may be found in the next iteration in $\mathcal{K}_{u+1}(\mathbf{p})$ again, and they may introduce unwanted turns and corners in the outline, or disturb the algorithm by self-intersections. Therefore, we not only remove $\mathbf{p}_{j'}$ from $\mathcal{S}_n$ to obtain $\mathcal{S}_{n+1}$, but also every other point $\mathbf{p}_i \in \mathcal{K}_u(\mathbf{p})$ whose associated edge $(\mathbf{p}_i - \mathbf{p})$ encloses a clockwise angle with the outline's last edge $\mathbf{e} = (\mathbf{p} - \mathbf{v}_{n-1})$ that is larger than the one enclosed between $\mathbf{e}$ and $(\mathbf{q} - \mathbf{p})$.

A further adjustment that affects concavities of a shape is the enforcement of creating steps in the outline by modifying Equation 2 to obtain Equation 5:

$$\begin{aligned} \mathbf{v}_n &\leftarrow \mathbf{q} \\ \mathbf{v}_{n-1} &\leftarrow \begin{cases} \mathbf{v}_{n-1} + \mathbf{w}, & \text{second last trun was right} \\ \mathbf{v}_{n-1}, & \text{otherwise.} \end{cases} \end{aligned} \quad (5)$$

In other words, we add $\mathbf{w}$ to the second last vertex of a continued edge, if the second last turn of the outline we encountered was a right turn, and the edge is thus part of a concavity. In such concave regions, the algorithm then behaves like the version in Section 3.2.1, and problems as shown in Figure 4 are prevented. The benefit of this modification, however, depends on the application and the underlying point set, and to some degree on the user's judgment or personal preferences.

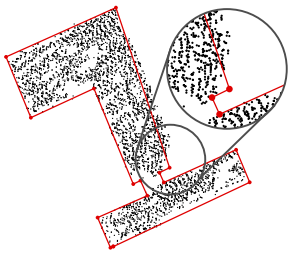


Figure 4: Due to dragging only the last vertex of outlines, edges may be moved into concavities and can unintentionally cut off too many points.

3.2.3 The Choice of Initial Direction

In the proceeding described in Section 3.2.1, the initial direction given by θ_0 is yet unspecified, but it influences the resulting outline as shown in Figure 5. Using $\theta_0 = 0$ is only reasonable if the lowest edge of shape is (near) parallel to the x-axis. But since the orientation of the shape we are trying to approximate by angular outlines may be unknown in advance, we either have to let the user specify θ_0 or obtain it from analysis of the point set's orientation by statistical means, e.g., via principal component analysis (PCA) or the methods employed in Section 3.1.

Figure 5: The shape represented by the point is actuall tilted by $\approx 25°$ against the x-axis. Not accounting for this initial orientation yields unpleasant results.

Starting at point $\mathbf{p}'$ having lowest y-coordinate, we might also run our algorithm using $\theta_0 = 0$. The first time we encounter $\delta' \neq \pi$, we can set θ to the angle enclosed between the first edge and the x-axis. In this way, θ_0 not only depends on $\mathbf{p}'$, but also on the choice of the size of the neighborhood k, which can be unsatisfactory as outliers in the neighborhood of $\mathbf{p}'$ may

yield an inappropriate outline. However, if the point set is sufficiently clean and dense in the area of the outline's first edge, this method works quite well. It can also be used to give the user a hint of the initial orientation of the shape.

3.3 Obtaining Simple Polygons

Due to the straightening process of the concave hull (Section 3.1) or the dragging of edges (Section 3.2.1), the outlines created by our methods may be self-intersecting. To obtain simple polygons, these self-intersections need to be detected, and the line segments between the points of intersection are cut off.

4 OUTLINES OF HOLES

The outlines of point sets in plane sections in 2D Euclidean space may comprise larger areas where no points are present. In the application of building reconstruction from LIDAR data, for instance, this happens if building complexes contain atria or inner courtyards, because the data may only yield information about distinctly elevated structures, like roofs. We call such empty areas *holes* within the point set and define them more formally as in Definition 1.

Definition 1. *Let $\mathcal{S} \subset \mathbb{R}^2$ be a finite point set and $\Omega(\mathcal{S}) \subset \mathbb{R}^2$ be the domain inside a simple polygon that tightly encloses all elements of $\mathcal{S}$ (e.g., one of its hulls or the minimal axis-aligned bounding box). We call a connected region $\mathcal{E} \subset \Omega(\mathcal{S})$ of finite area a* ***hole (in or of $\mathcal{S}$) of size*** *r, if $\mathcal{E}$ is large enough to fully overlap a circle of radius $r > 0$ at some location $\mathbf{c}$ such that $\forall \mathbf{p} \in \mathcal{S} : \ \|\mathbf{p} - \mathbf{c}\| \geq r$.*
r is called the ***size of hole*** *$\mathcal{E}$.*

Our definition of *hole* depends on the choice of polygon to border $\Omega(\mathcal{S})$: it is possible to choose different boundaries for $\mathcal{S}$ that have different numbers of holes, or holes of different sizes. In this work, however, we are only interested in holes that do not border on the enclosing polygons of point sets.

In order to obtain faithful outlines of the objects represented by 2D points, we extend our methods by creating inner boundaries. Since we are only interested in holes that are "large enough", we choose a minimum hole diameter $d_{\min}$ that shall be detected. The choice of $d_{\min}$ depends on the application and the average sampling density of the point set. In the application of building reconstruction, for instance, $d_{\min}$ might be related to the sizes of humans or vehicles, if the holes originate from courtyards, because such structures usually serve a purpose and thus need to be accessible.

In contrast to outer boundaries, inner boundaries shall be oriented clockwise. This is mainly due to technical aspects and conventions, for it allows us to easily determine whether a point is located *inside* or *outside* the inner or outer boundary. Besides, we need to take into account that there might be more than one hole in a given point set.

4.1 Detecting Holes

The basic idea of detecting holes is to find points on their border and to create their outlines by applying modified versions of the methods presented in Section 3. In order to find points on the border of holes, we employ the following heuristic (see also Figure 6):

1. Let h be the extent of point set $\mathcal{S}$ along the y-axis; choose a minimum hole diameter $d_{\min}$. Let furthermore P be a simple, closed outline polygon of $\mathcal{S}$, e.g., a (concave) hull or any other outline obtained by our methods.
2. Split $\mathcal{S}$ along y-axis into m strips of width $w = \frac{d_{\min}}{2}$ in such a way that $m = \left\lceil \frac{2h}{d_{\min}} \right\rceil \in \mathbb{N}$.
3. Obtain disjoint point sets Y_j for each of the $0 < j \leq m$ strips where $\forall a, b \in [0, m-1], a \neq b,\ Y_a \cap Y_{a+1} = \emptyset \wedge \bigcup_{j=0}^{m-1} Y_j = \mathcal{S}$.
4. For each set Y_j, sort the points in Y_j in ascending order of x-coordinates.
5. For each two neighboring points $\mathbf{p}_i$, $\mathbf{p}_{i+1}$ in Y_j, compute the distance $d = |p_{i+1,x} - p_{i,x}|$.
6. If $d \geq 2 \cdot w$ and $\mathbf{m} = (p_{i,x} + w, p_{i,y})$ is located inside P, add the directed line segment $\overline{\mathbf{p}_i \mathbf{p}_{i+1}}$ to the list of candidates Q.

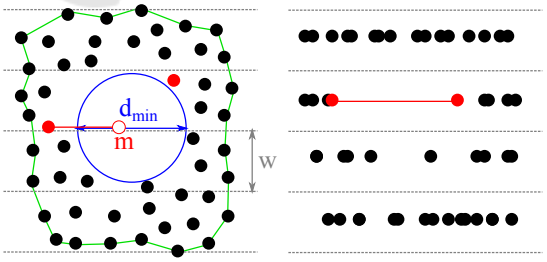


Figure 6: To find locations on the border of a hole from where to start the generation of inner boundaries, the point set is cut into strips perpendicular to the y-axis (left). Neighboring points whose projections onto the x-axis have distances $\geq d_{\min}$ are candidates (red), if m (red circle) is inside the point set's boundary (green).

This approach will find the points on the borders of all holes at once, but we cannot tell the hole a candidate belongs to. To solve this problem, we compute

an outline of the hole for the left vertex of first line segment $\overline{s_0} = \overline{\mathbf{a}_0\mathbf{b}_0}$ in Q as described in Section 4.2 and remove it from Q. For each of the remaining line segments $\overline{s_j} = \overline{\mathbf{a}_j\mathbf{b}_j} \in Q, j > 0$, we check if the midpoint $\mathbf{m}_j$ of $\overline{s_j}$ is located inside any previously computed outline of a hole, and discard $\overline{s_j}$ if $\mathbf{m}_j$ is inside. Furthermore, we check if any edge of a new hole boundary intersects an edge of a previously computed hole outline. If two outlines intersect, we can discard one of them based on a reasonable criterion, e.g., the size of its area or its number of vertices. Another option would be to merge the two outlines in order to obtain a common outline, but we did not implement this approach due to its complexity.

Alternatively, we may suspend the search for further candidates after the first one. After generating the hole boundary, we can fill the hole with phantom data points based on a point density estimation. The next candidate is then detected by restarting the search procedure from the beginning.

With the approach described above, we cannot yet guarantee that the resulting outline borders a hole overlapping a circle of radius $\geq w$ and is therefore consistent with Definition 1: Due to "cutting" the point set perpendicular the y-axis into strips of width $w = \frac{d_{\min}}{2}$, the hole has at least a radius of half the size of $\frac{w}{2}$. Cutting the point set into strips of width $d_{\min}$ might cause holes around this size to be missed, because the strips are projected onto the x-axis and gaps might be concealed from projections of points on the top or bottom of the hole. This is a sampling problem and the strip width w should thus be smaller than $\frac{d_{\min}}{2}$ in a practical implementation.

Verifying the size of the hole found by the heuristic above turns out to be difficult, since we would need to find the center of an empty circle with radius $d_{\min}$ within the hole, but there are potentially infinite many locations. By placing a circle of radius $d_{\min}$ at the centroid of the hole's outline polygon, for example, the hole can be accepted if the circle is empty, but it cannot be safely reject, if the circle is not empty. In practical applications that do not rely on the exact size of a hole, it may still be sufficient to find holes that are known overlap circles with radii the range of $\frac{w}{2}$ and w.

4.2 Bordering Holes

Creating the outlines of holes is done using slightly modified version of the procedures described in Section 3 and the Concave Hull algorithm in Section 2.1. The modifications basically account for changing the orientation of the resulting polygon to clockwise, and the different starting location.

Our modification to the Concave Hull algorithm to obtain a clockwise oriented polygon for outlining holes is as follows: Among all k neighbors $\mathcal{K}_u$ of the current point $\mathbf{p} = \mathbf{v}_n$, we search for the point $\mathbf{q}_i$ with the *smallest* counter-clockwise angle $\varphi_i = \measuredangle(\xi_n, \mathbf{e}_i = \overline{\mathbf{pq}_i})$ where ξ_n is the previous edge directed *backwards*, i.e., $\xi_n = \overline{\mathbf{v}_{n-1}\mathbf{v}_n}$ instead of $\overline{\mathbf{v}_n\mathbf{v}_{n-1}}$. Initially, we use the positive $x-$axis for ξ_0.

By choosing $\mathbf{p}'$ as the first vertex $\mathbf{a}_j$ of the line segments $\overline{s_j} \in Q$ obtained by the heuristic given in Section 4.1, we start at a point that is part of the left border of the hole. The point may furthermore be located on a sharp crease (like a v-shaped corner) such that the smallest φ_i may be associated with a point at the opposite border of that crease. Thus, the outline generation might fail right away, if k was too large. Replacing ξ_n by an edge that is rotated counter-clockwise by $\frac{\pi}{2}$ to point inside the hole yields better results, but if the neighborhood gets even larger, the algorithm can still fail for the same reason.

Due to its position on the left border of the hole, the point $\mathbf{p}' = \mathbf{a}_j$ may not necessarily be found in a k nearest neighborhood a second time. Therefore, we also have to extend the condition for terminating the hole's concave hull computation: For each new line segment added to the hull, we additionally check if it intersects the line segment $\overline{s_j} = \overline{\mathbf{a}_j\mathbf{b}_j} \in Q$. Starting from point $\mathbf{a}_j$, the edges of the outline to be created will enter the "upper" part of the hole. They can only enter the "lower" part by crossing $\overline{s_j}$ on the right border at $\mathbf{b}_j$ or close to it. Once a line segment enters the "upper" part from the hole's "lower" part again, $\overline{s_j}$ is crossed a second time, and we terminate the hull computation, because the polygon has encircled the hole.

Using our method to generate the outline of a hole by means of dominant directions as presented in Section 3.1, we first create a clockwise oriented concave hull as described above, and apply the same proceeding to straighten the concave hull of a hole afterwards. Our method described in Section 3.2 merely needs to be adjusted in the same way as the Concave Hull Algorithm in order to create outlines of holes.

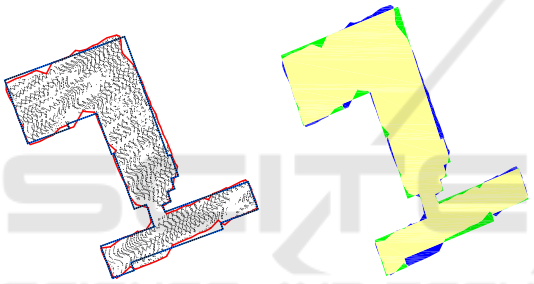
5 RESULTS AND DISCUSSION

We demonstrate the usability of our methods by means of four different data sets $D_1 - D_4$: the point sets already depicted in Figures 3(a) (D_1), 1 (D_2) and 3(b) (D_3), originate from airborne laser scans. The point set depicted in Figure 12(a) (D_4) is a synthetic data set featuring a star-shaped hole. Data sets D_1 –

D_3 represent non-convex building complexes, but D_1 is moreover near-rectangular in contrast to D_2 and D_3. For the purpose of generating figures, we reduced the number of points depicted, but to create outlines, we always employed the original, denser point sets.

5.1 Results

The outline of D_1 generated by means of dominant directions (see Section 3.1) is shown in Figure 7(a). With this data set, both options for determining the dominant directions ("peaks" and "90°") lead to the same rectangular outline that is significantly "straighter" than the concave hull for $k = 60$, but satisfactorily represents the underlying shape. The straightening process causes changes in areas as highlighted in Figure 7(b). These changes are small but help to approximate the building's true outline and compensate for noise and mistaken measurements.



(a) outline dominant directions (b) differences of overlapped areas

Figure 7: The concave hulls (red) of D_1 using $k = 60$ compared to outlines generated by means of our method using dominant directions. Left: the two options "90° angles" (black dotted) and "peaks" (blue) lead to same outline with substantially fewer vertices than the concave hull. Right: areas overlapped by the concave hull (yellow + blue) and the straightened outline (yellow + green).

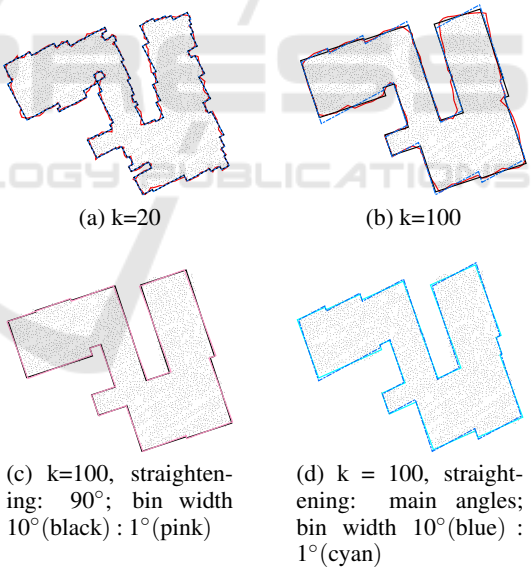
Like with the original Concave Hull algorithm, we can influence the shape of the resulting outlines by altering k. Using larger k yields boundaries that look "more convex" than the ones created using smaller k. This is illustrated in Figure 8 by means of outlines of D_1 obtained by using orthogonal projection (Section 3.2).

Boundaries of D_2 using dominant directions are given in Figure 9. With this data set, the first main direction is almost independent of k, but increasing k also increases the uncertainty in the determination of the second main direction. Since the later is also influenced by the point distribution and data noise, at larger k, the resulting outline looks a somewhat

(a) $k = 60$ (b) $k = 100$

Figure 8: Outlines of D_1 generated by using our method based on orthogonal projection (blue) compared to the concave hull (red) at different values of k.

skewed against the point set's shape. In this case, the outline created by using our "90°" option encloses the points more accurately. The remaining deviations result from the fact that the shape of D_2 is not truly rectangular. Figure 1(c) depicts an outline of this data set generated by means of orthogonal projection. Since the edges of that outline do not necessarily join orthogonally due to the dragging operations performed, it captures the point set's shape very well.



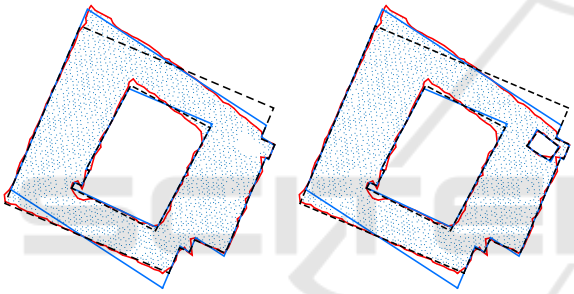
(a) k=20 (b) k=100

(c) k=100, straightening: 90°; bin width 10°(black) : 1°(pink) (d) k = 100, straightening: main angles; bin width 10°(blue) : 1°(cyan)

Figure 9: Straightened building outlines from the concave hull (red) for $k = 20$ (a) and $k = 100$ (b) neighbors. Larger k lead to smoother outlines but increase the uncertainty for computing the main direction. Changing the bin sizes leads to a tilt of the boundary.

D_3 is a real-world example of a point set acquired from a non-rectangular, non-convex building that has an atrium. Outlines of the outer and inner boundaries created by means of our methods are shown in Figure 10. Figures 10(a) and 10(b) show the concave hull and outlines of holes of minimum diameter $d_{\min}$

of 100 units and 30 units, respectively. These boundaries were created by means of our dominant directions method, and the holes were found using our heuristic in Section 4.1. Using a smaller hole size, we are able to find and outline the additional hole in the upper right corner, whereas a larger hole size only allows for handling the atrium. However, the inner and outer boundaries produced by the straightening procedure do not faithfully represent the shape of the underlying building: The lower part of the polygon is approximated quite well when the concave hull is straightened by using the "90°" option, whereas the upper edge is matched unsatisfactorily. Using the "peaks" option, we have the opposite situation, and the upper edge is matched quite well, although the first main direction is equal with both options. Since our approach based on dominant directions only allows to straighten along two dominant directions, it cannot match both edges at once.



(a) $d_{\min} = 100$ (b) $d_{\min} = 30$

Figure 10: Concave hull (red) and straightened outlines obtained from dominant directions and ensuring 90° angles (black dashed) and with respect to histogram peak angles (blue).

Our method based on orthogonal projection succeeds to accurately capture the building's shape, including the upper edge that is tilted by $\approx 12°$ towards the lower edge (see Figure 11).

To put our methods for detecting and outlining holes to the test, we created the synthetic data set depicted in Figure 12. The star-shaped hole was manually cropped and is especially challenging, because it is very sensitive to the choice of k: The edges of the spires are v-shaped and using large values of k causes them to become cut off. Using smaller values of k increases the accuracy, but also increases the uncertainty for determining the dominant directions when straightening the outline (see Figures 12(b) and 12(c)). Since we only consider two dominant directions with this approach, the outline of the star almost completely loses its initial shape. By means of orthogonal projection, we get a more faithful outline of

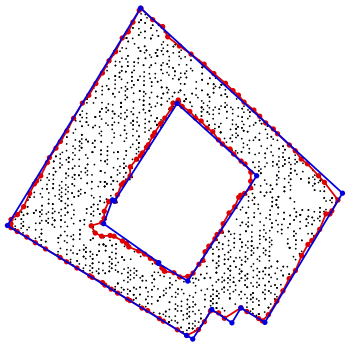
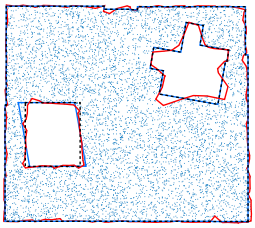
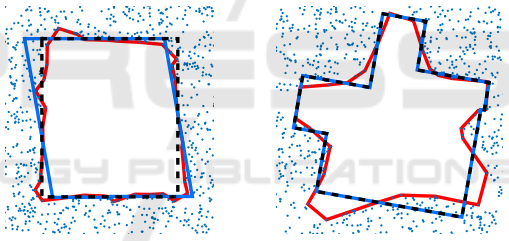


Figure 11: Inner and outer boundaries obtained by orthogonal projection (blue) compared to the concave hull (red) at $k = 20$ of our test data set D_3. The minimum hole diameter is $d_{\min} = 50$ units.



(a) straightened boundaries



(b) straightened lower left hole boundary (c) straightened upper right hole boundary

Figure 12: Outlines created from our approach based on dominant directions of data set D_4 compared to the concave hull (red) at $k = 20$ and $k = 40$ for the inner and outer boundaries, respectively. With the star-shaped hole and the outer boundary, the two dominant directions obtained from the peaks of the angle histogram (blue) are the same as the ones obtained by the "90° angle" option (dashed black). In case of the other hole, the two options yield different dominant directions.

the star-shaped hole, which not perfect, though (see Figure 13).

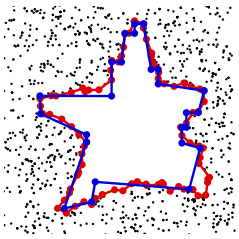


Figure 13: Outline of the star-shaped hole in data set D_4 created by orthogonal projection (blue) using $k = 10$ in comparison to the concave hull (red).

5.2 Discussion

The two methods presented in this work for generating straight outlines have different advantages over each other. Straightening concave hulls by means of dominant directions can accurately capture certain shapes of objects represented by point sets without prior knowledge about the orientation of their edges. However, it relies on the prior computation of concave hulls and is likely to yield unsatisfactory results if the shape's edges run in more than two distinct directions (cf. Figures 10 and 12(c)). Determining the dominant direction is furthermore subject to the width of bins used for the histogram of angles in the concave hull. This influence becomes apparent in Figures 9(c) and 9(d) where the bin widths are 10° and 1°, respectively. Depending on the quality of the concave hull (i.e., mainly on the choice of k) and the point distribution, changing the bin width can tilt the generated outline in an undesirable way. Especially by determining the second dominant direction from the histogram peaks, the choice of bin width may cause a shift in the position of the peaks corresponding to the dominant directions. For example, by enlarging the bin width by 5°, the bin associated with the second maximum may coincide with a neighboring bin. In this way, the second direction is tilted by about the same amount. Of course, the first direction can suffer from the same problem, if the point set is of rather square or circular shape instead of an elongated one.

Our approach based on orthogonal projection and the dragging of edges can create faithful outlines of shapes whose edges run in more than two distinct directions, but it requires hints about the initial orientation of the shape and possibly contained holes. These hints may be obtained automatically (see Section 3.2.3) in some situations, but otherwise they have to be provided by the user. With this method, it is not necessary to compute concave hulls or other intermediate polygons in advance, though.

The choice of the size k of the neighborhood considered is crucial for both our methods, but it is also subject to the distribution of the underlying points. While the Concave Hull Algorithm starts at the smallest $k = 3$ and increases the value if the outline has self-intersections or does not include all points of the set, this proceeding is not guaranteed to succeed with our methods, because our angular outlines may deliberately exclude points. In our experiments, we found that using k of about 5 – 10% of the number of points in the set is a reasonable initial choice, if the point distribution (excluding holes to be detected) is approximately uniform.

The decision which outline matches the underly-

ing object "best" is up to the user who can incorporate background information about the underlying object. In case of the underlying building represented by D_1 (see Figures 7 and 8), for instance, we could not decide whether the object is a single building complex or if there are two separate buildings, possibly linked by a corridor, if we solely rely on the information provided by the point set.

6 CONCLUSION AND FUTURE WORK

In this article, we presented two methods based on the Concave Hull Algorithm to create straight, angular and non-convex outlines of 2D point sets and possibly contained *holes*. The outlines of point sets representing man-made structure, like buildings, created by means of our methods are "better" than concave hulls or boundaries obtained from α-shapes to the effect that our methods capture straight edges more accurately and feature only few, distinct angles.

The employment of straightening concave hulls by means of dominant directions to generate outlines is preferable if the orientation of the underlying shape is unknown or cannot be provided, or if the shape is known to have edges running in only two distinct directions, e.g., in case of rectangular buildings. Additional directions could be obtained by introducing an angle of tolerance $\varphi_{\max}$. If $\measuredangle(\mathbf{e}_i, \mathbf{d}_\varphi) > \varphi_{\max}\ \forall \varphi \in \{\beta, \gamma\}$, the corresponding label vector entry is set to zero (see Section 3.1). These edges would not be affected by the straightening procedure, which is helpful if the building contains a narrowing or constrictions of angles other than the determined dominant ones.

If information about the orientation of the shape is available, our approach based on orthogonal projection can produce more faithful boundary polygons, especially if the underlying shape features edges running in more than two distinct directions. Moreover, this approach forgoes the computation of concave hulls or any other intermediate polygon.

The most challenging task is the reliable detection of holes. We encountered this problem by means of a heuristic that is subject to sampling problems and may have several theoretical limitations. Although we did not encounter major problems due to a rather specific, practical application, we would like to improve on these potential issues. For example, holes might be located close to the outer boundary of the associated point set, and they might thus share boundaries with the exterior outline. This could cause our current algorithms for bordering holes to fail, because the outline might "leave" the hole and intersect the exterior boundary.

Moreover, we only considered "isolated" point sets so far. In the application of building reconstruction, for example, this requires prior segmentation of the source data. We would like to forgo this step and determine spacings or gaps between distinct clusters of points in larger point sets to segment these while creating the corresponding outlines. This may be accomplished by a combination of modifying and constraining the neighborhoods employed by our approaches or the Concave Hull Algorithm, and adjustments of our heuristic for hole detection to this problem, but remains future work to do.

REFERENCES

Asaeedi, S., Didehvar, F., and Mohades, A. (2013). Alpha Convex Hull, a Generalization of Convex Hull. *The Computing Research Repository (CoRR)*, abs/1309.7829.

Bendels, G. H., Schnabel, R., and Klein, R. (2006). Detecting Holes in Point Set Surfaces. *Proceedings of The 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 14.

de Berg, M., Cheong, O., van Krevald, M., and Overmars, M. (2008). *Computation Geometry*. Springer, 3rd edition.

Douglas, D. and Peucker, T. (1973). Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or Its Caricature. *The Canadian Cartographer*, 10(2):112 — 122.

Duckham, M., Kulik, L., Worboys, M., 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, D., and Seidel, R. (1983). On the Shape of a Set of Points in the Plane. *IEEE Transactions on Information Theory*, 29(4):551 – 559.

Galton, A. and Duckham, M. (2006). What is the Region Occupied by a Set of Points? In *Proceedings of the 4th International Conference on Geographic Information Science*, GIScience'06, pages 81–98, Berlin, Heidelberg. Springer-Verlag.

Jarvis, R. A. (1973). On the Identification of the Convex Hull of a Finite Set of Points in the Plane. *Information Processing Letters*, 2(1):18 – 21.

Moreira, A. and Santos, M. Y. (2007). Concave hull: A k-nearest Neighbours Approach for the Computation of the Region Occupied by a Set of Points. *Proceedings of the 2nd International Conference on Computer Graphics Theory and Applications (GRAPP)*, pages 61 – 68.

Ramer, U. (1972). An Iterative Procedure for the Polygonal Approximation of Plane Curves. *Computer Graphics and Image Processing*, 1(3):244 – 256.

Vosselman, G. (1999). Building Reconstruction Using Planar Faces In Very High Density Height Data. In *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 87–92.

Wang, J. and Oliveira, M. M. (2007). Filling Holes on Locally Smooth Surfaces Reconstructed from Point Clouds. *Image and Vision Computing*, 25(1):103 – 113.

Wang, O., Lodha, S. K., and Helmbold, D. P. (2006). A Bayesian Approach to Building Footprint Extraction from Aerial LIDAR Data. In *Proceedings of the Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06)*, 3DPVT '06, pages 192–199, Washington, DC, USA. IEEE Computer Society.

Wu, X. and Chen, W. (2014). A Scattered Point Set Hole-filling Method Based on Boundary Extension and Convergence. In *11th World Congress on Intelligent Control and Automation*, pages 5329 – 5334. IEEE.