AOI-shapes: An Efficient Footprint Algorithm to Support Visualization of User-defined Urban Areas of Interest

MINGZHAO LI and ZHIFENG BAO, RMIT University, Australia
FARHANA CHOUDHURY, University of Melbourne, Australia
HANAN SAMET, University of Maryland
MATT DUCKHAM, RMIT University, Australia
TIMOS SELLIS, Swinburne University of Technology, Australia

Understanding urban areas of interest (AOIs) is essential in many real-life scenarios, and such AOIs can be computed based on the geographic points that satisfy user queries. In this article, we study the problem of efficient and effective visualization of user-defined urban AOIs in an interactive manner. In particular, we first define the problem of user-defined AOI visualization based on a real estate data visualization scenario, and we illustrate why a novel footprint method is needed to support the visualization. After extensively reviewing existing "footprint" methods, we propose a parameter-free footprint method, named AOI-shapes, to capture the boundary information of a user-defined urban AOI. Next, to allow interactive query refinements by the user, we propose two efficient and scalable algorithms to incrementally generate urban AOIs by reusing existing visualization results. Finally, we conduct extensive experiments with both synthetic and real-world datasets to demonstrate the quality and efficiency of the proposed methods.

CCS Concepts: • **Human-centered computing** → **Geographic visualization**; *Visualization techniques*; • **Theory of computation** → **Computational geometry;**

Additional Key Words and Phrases: AOI-shapes, footprint, geographic visualization, incremental algorithm

ACM Reference format:
Mingzhao Li, Zhifeng Bao, Farhana Choudhury, Hanan Samet, Matt Duckham, and Timos Sellis. 2021. AOI-shapes: An Efficient Footprint Algorithm to Support Visualization of User-defined Urban Areas of Interest. *ACM Trans. Interact. Intell. Syst.* 11, 3-4, Article 27 (August 2021), 32 pages.
https://doi.org/10.1145/3431817

1 INTRODUCTION

An urban **area of interest (AOI)** refers to the area within an urban environment that attracts people's attention [22]. Understanding urban AOIs can assist various urban data exploration tasks

This work was partially supported by ARC Grants No. DP180102050 and No. DP200102611, Google Faculty Research Award No. NSFC 91646204, and U.S. National Science Foundation Grants No. IIS-13-20791 and No. IIS-1816889.
Authors' addresses: M. Li, Z. Bao (corresponding author), and M. Duckham, School of Science, RMIT University, Melbourne, Victoria, 3000, Australia; emails: {mingzhao.li, zhifeng.bao, matt.duckham}@rmit.edu.au; F. Choudhury, School of Computing and Information Systems, University of Melbourne, Parkville, Victoria, 3010, Australia; email: farhana.choudhury@unimelb.edu.au; H. Samet, Department of Computer Science, University of Maryland, College Park, MD 20742; email: hjs@umiacs.umd.edu; T. Sellis, Data Science Research Institute, Swinburne University of Technology, Hawthorn, Victoria, 3122, Australia; email: tsellis@swin.edu.au.

2160-6455/2021/08-ART27 $15.00
https://doi.org/10.1145/3431817

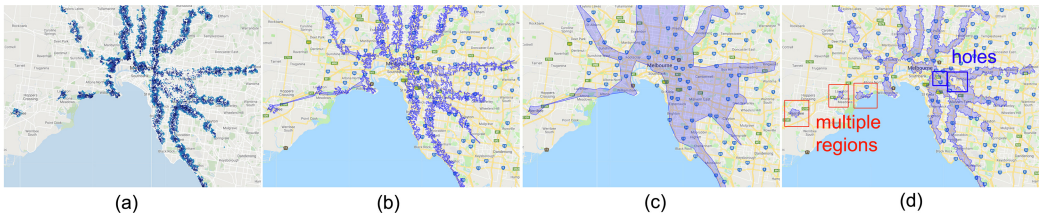


Fig. 1. An example user-defined AOI (properties within a 15-min walk to the nearest train station) presented as: (a) individual property points [28]; (b) a polygon-based region generated using χ-shapes [14] (χ = 0.05); (c) a polygon-based region generated using χ-shapes [14] (χ = 0.2); (d) polygon-based regions generated using our proposed AOI-shapes.

such as house seeking, and decision making for urban planning such as facility deployment of petrol stations, supermarkets, and so on. For example, location is a key factor in house seeking, based on which various AOIs can be defined by users to cater to their respective concerns/interest, as illustrated in Example 1.

Example 1. In our previous work [28], we propose a visual analytics system for Australia's real estate data to help users understand the local real estate market and find their preferred housing properties. At the property level exploration, users can filter out properties based on their own requirements over the multidimensional view. After that, all properties that satisfy the user query will be shown on top of the map. As shown in Figure 1(a), the user prefers to live in a place within a 15-min walk to the nearest train station. Properties that satisfy such a query will result in a user-defined urban AOI.

AOIs could be represented either as polygon-based regions or as a set of individual points (e.g., real estate properties). Previous studies have argued that polygon-based representations have several advantages from both cognitive and computational perspectives [1, 22]. However, unlike most well-defined administrative districts (e.g., suburbs, cities), there are normally no defined boundaries for AOIs. The region (shape) of an urban AOI could be vague [22] and often depends on users' preference.

One of the most popular methods to characterize the shape of an AOI is to generate the region boundaries, which are called "footprints," based on the **points of interest (POIs)** that satisfy the user query. For example, Polisciuc et al. [41] generate the footprints based on people's visited places and visualize the footprints to help understand the land use for urban planning and management. In Example 1, the AOIs shown in Figures 1(b)–1(d) are generated based on the POIs in Figure 1(a) (the housing properties satisfying the user query).

Different footprint algorithms have been proposed, including Convex hulls [21], concave hulls [38], KNN-based concave hulls [36], χ-shapes [14], and so on. However, existing methods either need a parameter that are difficult to define, or could not recognize multiple regions, outliers or inner holes that have significant meanings in a lot of real-life applications. The following example highlights the disadvantages of existing methods and explains why they are important in practical scenarios.

Example 2. We generate different footprints for the properties shown in Figure 1(a) using χ-shapes [56]. The algorithm first constructs a convex hull, and then executes a "digging" process (progressively replace an outer edge with two adjacent inner edges) to get a concave hull (i.e., footprint). The only parameter is a length-related threshold that decides whether each boundary edge is too "long" or not; and the algorithm will terminate if all boundary edges are shorter than the threshold. We use two different parameters to generate the footprints and present both the

results on Google Maps (Figures 1(b) and 1(c)). The results illustrate the weaknesses of existing footprint algorithms and why they are critical in such real-life scenarios: (i) A smaller threshold may make the result too "concave" for users to recognize the region (Figure 1(b)), and a larger threshold may cause the region to be too "big" to characterize the shape of the region (Figure 1(c)); unfortunately, it is hard to find an appropriate threshold (which does not have intuitive real-life meanings) for an unknown set of data points. (ii) Since the algorithm cannot recognize multiple regions, those regions (as highlighted in red in Figure 1(d)) are unexpectedly connected in Figures 1(b) and 1(c). (iii) There are regions in the middle of two train lines (as highlighted in Figure 1(d)) that do not satisfy the user query; however, the χ-shapes algorithm might not detect those holes (Figure 1(c)).

In our application (Example 1), a user's search intention can always be formulated as a query that might have various forms such as boolean queries (e.g., finding all properties that have a swimming pool) or spatial queries (e.g., finding all properties that can reach a train station within 500 m). We note that the output of any types of queries can always be defined as two sets of points: (i) P—the points that satisfy the user query, and (ii) $\overline{P}$—those that do not satisfy the query. Based on the intuition that, "*the points that do not satisfy the user query should not be in the generated AOI*," we propose a parameter-free footprint method, namely, AOI-shapes, to describe the boundary of a user-defined urban AOI. The proposed algorithm is based on a global Delaunay triangulation [49] (Figure 2(a)), followed by a separation of points/regions considering whether the data points meet the user query (inside the AOI, blue points in Figure 2(a)) or not (outside the AOI, orange points in Figure 2(a)). As a result, AOI-shapes is able to support all the four listed features. First, based on whether or not each point satisfies the user query, the region will be naturally separated; we further define boundary edge conditions based on triangle properties (details in Section 5) to make the AOI-shapes more concisely represent the region without additional parameter tuning needed (third feature). Second, multiple regions, outliers and holes are detected based on the points that do not satisfy the query (first and second features). Third, we extend AOI-shapes by exploiting the common result points between two consecutive rounds of user queries, and propose two incremental AOI-shapes algorithms to support efficient visualizations of AOIs (the last feature). The following example illustrates why incremental algorithms are critical in real-life scenarios.

Example 3. Suppose that, after visualizing the result for the properties within 15-min walking distance, the user finds the result area too broad and now wants to refine the query to visualize the properties within 15-min walk to the nearest train station and also within a total of 40-min travel time (by train+walk) to his workplace in the CBD. Such a query will result in a new AOI (Figure 2(b)), which is a sub-region of the previous AOI (Figure 2(a)). If an incremental update is not supported, then the AOI and the boundaries of the AOI need to be re-calculated from scratch for this visualization. As many properties are shared between the example in Figures 2(a) and 2(b), a method that updates the visualization by incrementally considering only the necessary changes is highly important for efficiency and scalability, especially for a large-scale dataset.

To summarize, we make the following contributions:

- We present a comprehensive literature review of the existing footprint methods (Section 2.2) and summarize them based on the construction process and the characteristics of the result (Table 1).
- To the best of our knowledge, our article is the first to propose the problem of user-defined AOI visualization (Section 3).
- We propose a novel footprint method, named AOI-shapes, to effectively capture the region of an urban AOI based on POIs that satisfy the user query and those that do not (Section 5).

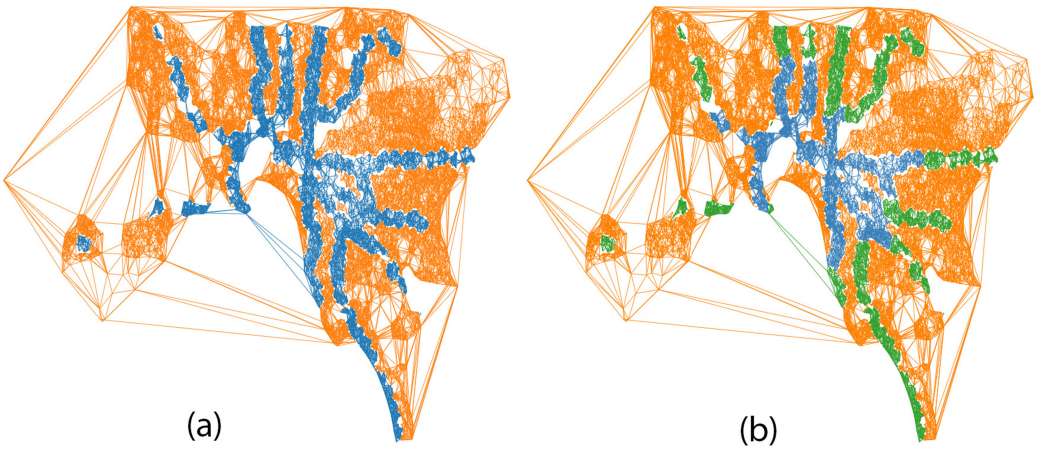


Fig. 2. Illustration of the proposed incremental AOI-shapes based on a global Delaunay triangulation. (a) User query 1: 15-min walk to the nearest train station (blue points satisfying the query, orange points not satisfying the query); (b) a subsequent user query 2: less than 15-min walk to the nearest train station, and 30-min train to the CBD (green points: those satisfying query 1 but not satisfying query 2).

- We propose two efficient and scalable solutions to incrementally generate the AOI-shapes by using existing result after the query is refined by user interactions (Section 6).
- We conduct extensive experiments based on both synthetic and real-world datasets to evaluate the quality and efficiency of the proposed methods (Section 7).

2 RELATED WORK

In this section, we review the closely related work on the visualization of AOIs and the footprint methods for calculating the AOI.

2.1 Visualization of Urban AOIs

The concept of AOIs [22] is closely related to POIs. While a POI represents an individual location (e.g., a property, or a landmark), an AOI usually contains multiple geographic points, such as the restaurants on a pedestrian street [16], nearby landmarks [45], or properties that satisfy users' requirement [28, 51].

In terms of geographic visualization, AOIs are often represented as polygon-based areas [8, 10, 19, 22, 24, 37, 53] instead of as individual points. For example, Hu et al. [22] extracted the urban AOIs using geo-tagged photos and represented the AOI as a polygon, which is generated based on the POIs using the χ-shape algorithm; Jung et al. [24] also used the χ-shape algorithm to generate the "footprint" based on a group of geo-referenced photos (e.g., all the photos with a "park" tag in Manhattan, New York), and then visualized the "footprint" as polygons for its represented AOI on the map. Previous studies have shown the advantages of using polygons to represent AOIs compared to using individual points. First, from the cognitive perspective, polygons can provide simple and accessible representations for areas compared with clustered points [22]. Second, from the computational perspective, it is generally more efficient to perform operations on a polygon than on a set of points [1]. Third, polygons convey information about the shape of an area and can be employed for shape-based analysis [30]. In this article, we also adopt such a polygon-based representation of the AOI.

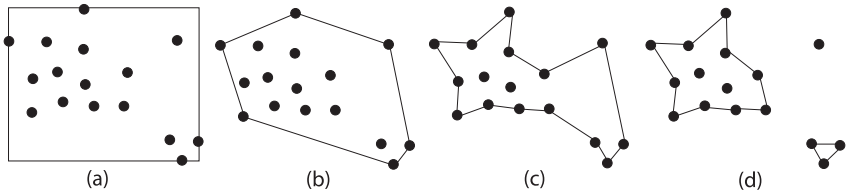


Fig. 3. Example of different kinds of hulls: (a) rectangular box, (b) the convex hull, (c) an example of simple concave hulls, and (d) concave hulls for multiple groups, an example of non-simple concave hulls.

2.2 Footprint Methods: Generating Boundaries for AOIs

Unlike most well-defined administrative districts (e.g., suburbs, states), the boundaries of an urban AOI are often vague [22, 32]. To define its boundary, several algorithms have been proposed to generate a polygonal "footprint" that characterizes the distribution of a set of points P in the two-dimensional plane, i.e., $P \subset R^2$. A trivial approach is to use the minimum bounding rectangle [26] to represent an AOI, as shown in Figure 3(a). There have been some non-trivial footprint methods proposed, such as convex hulls, simple concave hulls, and non-simple concave hulls (see Figures 3(b), 3(c), and 3(d), respectively).

2.2.1 Convex hull. A more sophisticated version of a rectangular box is the rectilinear convex hull, also known as the orthogonal convex hull. A **convex hull (CH)** of a point set P is the smallest convex set, which contains P. Different convex hull algorithms have been proposed to generate the convex hull for a set of points. One well-known method is the Graham Scan [21], which finds all vertices of the convex hull ordered along its boundary. Another method, called Gift wrapping [23], finds the next point that will form the widest angle with the previous two points. Some other notable algorithms include the Divide and Conquer algorithm [42], Andrew's Monotone chain algorithm [3], Quickhull [7, 15], and Chan's method [12]. Since the convex hull for a set of points P is unique, all the aforementioned methods will generate the same result. Chadnov and Skvortsov [11] conducted experiments to compare the efficiency of some of the methods.

2.2.2 Simple concave hull. A simple concave hull is a generalization of the convex hull, which allows any angle between two edges. Same as the convex hull, a simple concave hull is a Jordan-boundary hull that covers all the input points [18]. Any simple shape, which is between the convex hull and the hull that completely minimizes the area, is regarded as a concave hull [47]. Unlike convex hull, concave hulls from a definite set of points are not unique. Concave hulls can vary with different algorithms, and/or different parameters. More than 30 concave hull methods have been proposed in the past decade. To our best knowledge, there is no existing study that categorizes different concave hull methods. Here, we divide them based on how the concave hull is generated.
(1) Convex hull +"digging". This group of methods generate the convex hull for the given data points and then progressively replace a "longer" outer edge with two inner edges (an iterative "digging" process) until the concave hull meets the requirement. On the one hand, some works focus on the "digging" process of convex hulls. For example, Park and Oh [38] proposed a method that identifies a convex hull first and then "digs" the hull to produce a concave hull with appropriate depth, and the digging process is determined by a threshold value N, which corresponds to the edge length. Polisciuc et al. [41] proposed a similar method that constructs a convex hull first but then pushes "long" edges inside to generate new boundary edges. Rosén et al. [47] proposed a parallelized algorithm to further improve performance. Simple-shape [20] starts reconstruction from the convex hull and makes it concave step by step based on a new hybrid selection criterion

that is built on human beings' visual perception. On the other hand, many researchers explored the "digging" process subsequent to the convex hull generated based on **Delaunay triangulation (DT)** [49]. One of the most popular and widely adopted methods is χ-shapes [14], which starts with the DT and iteratively removes the longest exterior edge of the triangulation that does not cause "irregularity" (defined in Reference [14]). Also, a parameter-free method called *ec*-shapes [48] constructs the DT of a given data set, and then successively remove exterior edges subject to proposed "circle" and regularity constraints to compute a resultant boundary.

(2) Boundary detection. Another set of methods try to directly detect the boundary. Galton et al. proposed the swinging arm method [18], which generalises the gift wrapping algorithm [23] for concave hull generation. While finding the next point that will form the widest angle with the previous two points, it only finds the point within a distance of r (a fixed parameter) of the current point, instead of finding it in the entire set of points. The **K-nearest neighbour (kNN)** method [36] also generalises the gift wrapping algorithm [23]. At each iteration, the kNN-based algorithm finds the next point from the kNN of the current point, instead of processing the entire point set P used in the gift-wrapping algorithm. Chaudhuri et al. [13] defined an s-shape of a point set P similar to the process of pixelization, i.e., P is partitioned into a grid of square cells of side-length s. Then, an r-shape of P is defined to smooth the s-shape. Based on the observation that boundary points tend to have fewer reverse k-nearest neighbors (RkNNs, where the RkNN of an object p is the set of points that have p as one of their k-nearest neighbors), Xia et al. [52] proposed the BORDER method to detect boundary points in a point set. However, it is beneficial only for data sets that are well-clustered.

(3) Exploiting auxiliary points. In addition to the original point set P, several methods take a set of auxiliary points A to construct the concave hull. A-shapes [33, 34] first computes the **Voronoi Diagram (VD)** of $P \cup A$, then the A-shape of P is defined by joining any pair of points $p_i, p_j \in P$ whose Voronoi cells share at least a border with some other Voronoi cells containing a point of A. Arampatzis et al. [4] used a similar method to generate the region boundaries based on DT instead of VD. Li et al. [29] grouped points based on their regions and then computed their region boundaries based on DT with a similar method to A-shapes. Alani et al. [2] introduced a Voronoi diagram method for generating approximate regional extents from sets of centroids that are, respectively, inside and external to a region.

(4) Based on a mathematical definition. One of the most famous methods is called α-shapes [44]: the α-hull of P is the intersection of all closed discs with radius $1/\alpha$ that contains all the points of P ($\alpha > 0$). If $\alpha < 0$, then the α-hull of P is defined as the intersection of all closed complements of discs (where these discs have radius - l/α) that contain all the points of P. The α-shape of P is the straight line version of the α-hull. When $\alpha = 0$, the α-shape of P is its convex hull. Reference [5] defined α-Concave Hull, which is a generalization of the convex hull. The parameter α determines the smoothness level of the constructed hull on a set of points, where $\alpha + \pi$ is the largest inner angle of the calculated concave hull. After proving that computing an α-Concave hull with a given area on a set of points is NP-complete for any $\alpha \in (0, \pi)$, the authors present an approximation algorithm to compute an α-Concave hull. ω-hull [54] is similar to α-Concave Hull, where ω refers to the smallest outer angle of the hull. Based on the definition, a convex hull is a class ω-hull of $\omega \geq \pi$. The authors proposed an algorithm to calculate ω-hull by extending the Graham Scan algorithm [21] for computing the convex hull.

2.2.3 Non-simple concave hull. In this article, a non-simple concave hull refers to the concave hull that has non-Jordan boundaries (i.e., the boundary of the concave hull is not a Jordan curve [6]), or allows multiple regions and outliers.

Table 1. A Summary of Footprint Methods

	convex hull + "digging"	boundary detection	auxiliary points	mathematical definition	based on DT	based on VD	holes	outliers	multiple regions	parameter free	incremental method
	Construction process						Characteristics				
Rectangular box [26]				✓						✓	
Rectilinear convex hull [46]				✓						✓	
Convex hull [21]				✓						✓	✓
Concaveman⋆ [38]	✓										
Polisciuc et al. [41]	✓										
Rosén et al.⋆ [47]	✓										
Simple-shape [20]	✓										
χ-shapes⋆ [14]	✓				✓						
ec-shapes [35]	✓				✓						
Swinging arm [18]		✓									
kNN-based method⋆ [36]		✓									
s-shape [13]		✓									
r-shape [13]		✓									
BORDER [52]		✓									
A-shapes [33, 34]			✓			✓	✓		✓		
Arampatzis et al. [4]			✓		✓					✓	
Li et al. [29]			✓		✓					✓	
Alani et al. [2]			✓			✓				✓	
α-shapes⋆ [44]				✓			✓				
α-Concave Hull [5]				✓							
ω-hull [54]		✓		✓							
LDA-α-shapes [31]				✓							
Kolingerová and Žalik [25]					✓		✓				
Pohl et al. [40]							✓				
Peethambaran et al. [39]					✓		✓			✓	
Methirumangalath et al. [35]					✓					✓	
Zhu et al. [57]							✓				
DBScan + χ-shapes [55]	✓				✓		✓		✓		
χ-outlines⋆ [55]	✓				✓			✓	✓		
Incremental χ-shapes [56]	✓				✓						✓
Our proposed AOI-shapes			✓		✓		✓	✓	✓	✓	✓

⋆ Methods in gray are used as alternatives in the experiment (Section 7).

(1) Concave hull with hole detection. Several methods have been proposed that can effectively detect holes inside a hull. One of the most famous examples is the α-shape of a point set P ($\alpha < 0$). Variants of α-shapes such as k-order α-hulls and LDA-α-shapes [31] effectively use the local density variation found in the point sets for detecting the hollow regions. [25] also defined a method based on DT to detect the domain boundaries of a given set P. It first constructs a DT for P and

then removes those "unsatisfactory shapes of triangles," which are defined as triangles with too "big" or too "small" angles and triangles with too "long" or too "short" edges. As a result, the algorithm can detect both outer boundaries and inner holes. Reference [40] presented two methods based on the KNN-based Concave Hull Algorithm [36] to create straight, angular and non-convex outlines of 2D point sets and possibly contained holes. Reference [39] proposed a fully automatic Delaunay-based sculpting algorithm to approximate the shape of a finite set of points P. It uses a combination of a local measure (circumcenter of Delaunay triangles) and a global measure (size of the Delaunay triangles quantified by the circumradius) for the filtration of the DT. As a result, the algorithm is able to generate a simplicial complex, and effectively detect holes inside the shape. [57] proposed an algorithm to detect cavities and holes from planar points based on density. The algorithm conceptualizes cavity and hole as "a structure with a low-density region surrounded by the high-density region."

(2) Concave hulls for multiple groups. Different algorithms have been proposed to characterize more complicated point distributions with disconnected shapes and outliers. One straightforward approach is to pre-process a spatial clustering algorithm such as DBScan [17], followed by any simple concave hull method (presented at Section 2.2.2). One main disadvantage of such a two-step method is that any pre-clustering algorithm necessarily requires additional parameterization. Reference [55] proposed χ-outlines, which extends χ-shapes by dealing with situations of disconnected shapes and excluding outliers.

2.2.4 A summary of existing footprint methods. We summarize different footprint methods in Table 1 based on the main construction process of each method and characteristics of the generated hull. Particularly, since many methods are based on DT or VD, we summarize such information in Columns 6 and 7; the last column indicates whether an incremental method has been proposed.

To summarize, existing footprint methods fail to support at least one of the following features when we use them to generate user-defined urban AOIs based on the points that satisfy the user query: (1) the method can recognize multiple regions and/or outliers; (2) the method can detect holes; (3) the algorithm does not need any additional parameter tuning; (4) the method supports incremental generation of the footprint.

3 PROBLEM ILLUSTRATION

In this section, we illustrate our problem based on a real-world application related to user-defined urban AOIs. In particular, we first define the visualization tasks using a real estate dataset as an example. After describing the visualization idioms, we illustrate why a novel footprint method is needed to support the visualization.

3.1 Visualization Tasks

For ease of explanation, we describe our work using Australia's real estate data [28] as an example. The data lists 1.42 million properties sold between 2007 and 2016. Each property is associated with 72 dimensions including its geo-spatial information (latitude and longitude), categorical dimensions (e.g., property type), and numerical dimensions (e.g., price, distance to the nearest train station, supermarket, etc.).

Recall Example 1, each user query results in a user-defined urban AOI, and such AOI will change when the user refines her individual requirements. Therefore, our main visualization task is to visually present the user-defined urban AOI and allow users to gradually refine their requirements, in which case the visualization will need to be updated without latency.

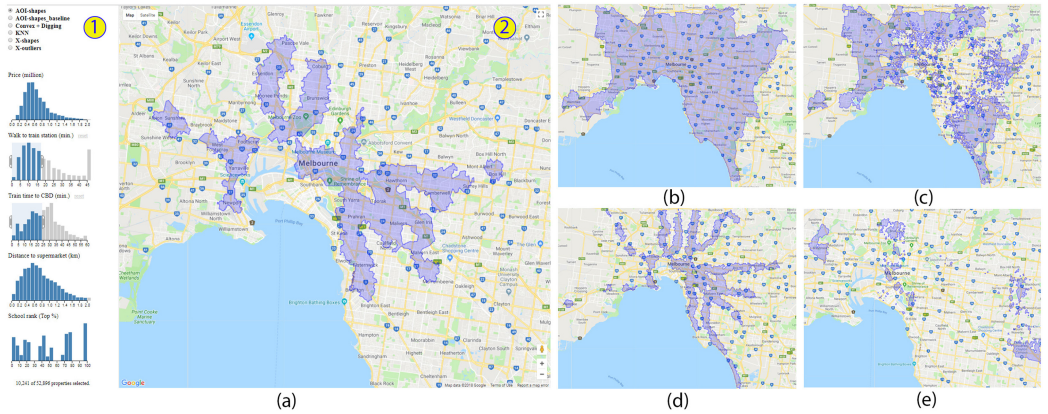


Fig. 4. The interface and some examples of the visualization of user-defined urban areas of interest. (a) The interface, which includes (1) the multidimensional view, where users can define their queries based on multiple attributes; and (2) the Google maps view, where the user-defined AOI is shown on top of the map. Examples of the results: (a) the area in Melbourne within a 15-min walk to the nearest train station and a 25-min train to the CBD; (b) the Melbourne metropolitan area; (c) the Melbourne metropolitan area with house prices less than 1 million dollars; (d) the Melbourne metropolitan area within a 15-min walk to the nearest train station; (e) the Melbourne metropolitan area with house prices less than 1 million dollars and in a top-10% public secondary school zone.

3.2 Visualization Idioms

We use a straightforward composite visualization design to visualize the user-defined AOIs. As shown in Figure 4, the design includes two coordinated views: a multidimensional view and a Google maps view.

Multidimensional view. As shown in Figure 4(a-1), a multidimensional view allows users to define the urban areas of their interest in an interactive way. Users can first select attributes (i.e., dimensions) of interest from the selection box on top of the view. Suppose that a user selects m attributes, then m coordinated histograms will be initialized to present the data distribution on the m selected dimensions. Users can filter out the dataset by directly brushing on top of each histogram. It is worth noting that, many other visualization idioms (e.g., parallel coordinates, static histograms) can be used to support multidimensional filtering. While each of them has its advantages and disadvantages, we choose dynamic histograms (CrossFilter[1]) in our implementation, since the main purpose of this view is to support user filtering (selections), and the changes on the remaining histograms might give users a better sense of their filtering actions.

Google maps view. As discussed in Section 2.1, we adopt the most straightforward polygonal representation to visualize the urban AOI. As shown in Figure 4(a-2), the AOI can include multiple regions, and each region is visualized as a polygon on the Google maps. It is worth noting that, holes could also exist in real-life application, and such region is visualized as empty areas inside the polygon (Figure 4(a-2)).

3.3 Why a Novel Footprint Method Is Needed?

After users filter out the data from the multidimensional view, the following steps are sequentially executed: (i) the data (POIs) that satisfy user's query are returned, (ii) the region (AOI) is calculated using a footprint method based on the updated data, and (iii) the calculated region is updated on the Google maps.

[1]http://square.github.io/crossfilter/.

While the first step can be effectively implemented using **relational database management system (RDBMS)**, and the last step should not be a problem if we only need to draw a limited number of regions on Google maps, so the main bottleneck is at the second step, which is the main technical contribution of our work.

Both the effectiveness and efficiency could be a problem when we use existing footprint method to calculate the urban AOIs. On the one hand, as we have summarized in Table 1, existing methods suffer from at least one of the drawbacks related to detecting multiple regions, outliers and holes, as well as parameter-tuning. On the other hand, although existing algorithms (e.g., χ-shapes [14]) can achieve the time complexity of O($n$log$n$), it will take more than one second if the data size is larger than 200k (for more details, see Section 7 in the experiment part), which is considered as high latency for an interactive visualization application [9]. Therefore, a novel footprint method is needed to solve both the effectiveness and efficiency problems of existing methods.

To summarize, the proposed footprint method should satisfy the following criteria to address the limitation of existing methods and support interactive visualization of user-defined AOIs:

- Parameter-free: users do not need to tune any parameters for the AOI;
- Multiple regions and outliers need to be recognized;
- Inner holes could be detected; and
- Incremental generation of AOIs should be supported.

4 A BASELINE METHOD

In this section, we propose a baseline method to overcome the drawbacks of existing footprint methods (as summarized in Table 1). Specifically, the proposed baseline method is (i) parameter-free: users do not need to tune any parameter for the AOI, (ii) separable: both the data points that satisfy the user query and those that do not are considered to generate the AOI, and (iii) multiple regions, outliers and holes can be recognized.

4.1 Preliminaries

Given a set of data points D, let $P \subseteq D$ be the points satisfying the user query (i.e., the points inside the user-defined AOI), and $\overline{P} = D \backslash P$ be the points outside the AOI. Our aim is to compute the boundary of the regions that are formed by the points in P, including the outliers.

4.1.1 Main idea of the proposed baseline method. The main idea of our proposed baseline method is to separate the regions that satisfy the user query from those that do not (Figure 1(d)). The method consists of an offline preprocessing step and a query time processing step. In the preprocessing part, we build a DT [49] based on the whole dataset D. In the query processing part, once the user issues a query, the set of points P that satisfies the query will be obtained. Then we will try to find the region boundaries formed by P.

4.1.2 Labels of vertices and edges. To better illustrate the algorithm, we first define the following labels of vertices (points) and edges based on the DT result. The examples are all illustrated in Figure 5.

For points (i.e., vertices), we divide them into the following types based on whether they are in P and whether they will form a boundary:

- ***Outside/inside***. p is an *outside* point if $p \in \overline{P}$. p is an *inside* point if $p \in P$.
- ***Boundary***. A *boundary* point p satisfies the user query and is on the final boundary of a region (i.e., the resulting footprint). For example, p_1–p_4 are *boundary* points that form a disclosed region as $p_1p_2p_3p_4$ in Figure 5.

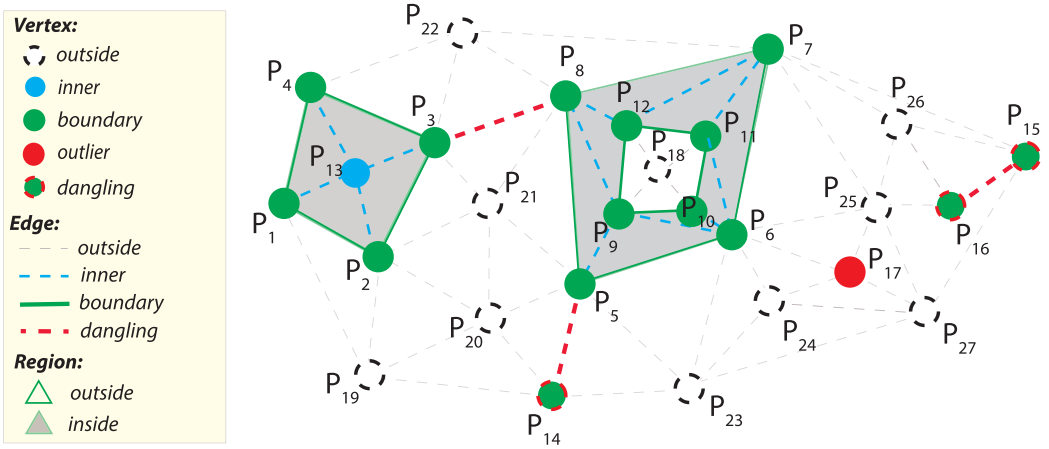


Fig. 5. Illustration of different types of points, edges and regions. (1) p_1–$p_{17} \in P$: p_1–p_{12} are *boundary* points, where p_9–p_{12} form a hole boundary; p_{13} is an *inner* point; p_{14}–p_{16} are *dangling* points; p_{17} is an *outlier* point; (2) p_{18}–$p_{27} \in \overline{P}$ (i.e., they are *outside* points).

- ***Inner***. An *inner* point p satisfies the user query (i.e., $p \in P$) but is inside the final boundary of a region. For example, p_{13} is an *inner* point.
- ***Outlier***. A point p is an *outlier* if $p \in P$, but p does not form triangles with any other points in P. For example, p_{17} is an *outlier* point.
- ***Dangling***. A point p is a *dangling* point if (1) $p \in P$, and (2) p is not an *inner*, *boundary*, or *outlier* point; i.e., for any two points p' and p'' that form a triangle with a point p from P, only one of them (e.g., p') is in P, while the other point p'' is an *outside* point.

It is worth noting that, a point can either be an *outside* or *inside* point. *Inside* points will be further recognized as *boundary*, *inner*, *outlier* points or *dangling* points during the algorithm process. Similar to the labels of vertices, edges are also divided into the following labels:

- ***Outside/Inner***. pp' is an *outside* (*inner*) edge if either p or p' is an *outside* (*inner*) point. For example, p_1p_{19}, $p_{23}p_{24}$ and $p_{12}p_{18}$ are *outside* edges; p_1p_{13} is an *inner* edge.
- ***Boundary***. pp' is a *boundary* edge if both p and p' are *boundary* points. For example, p_1p_4 and p_9p_{10} are *boundary* edges.
- ***Dangling***. Suppose that p_m forms a triangle with pp', and p_n forms another triangle with pp'. pp' is a *dangling* edge if and only if $p \in P$, $p' \in P$, and $p_m \in \overline{P}$, $p_n \in \overline{P}$. For example, p_3p_8, p_5p_{14} and $p_{15}p_{16}$ are *dangling* edges.

4.2 The Proposed Baseline Algorithm

Algorithm 1 presents the whole process of the baseline method, which includes two parts. At the pre-processing part, the global DT is built and edge labels are initialized (Lines 1.3–1.6). At the query processing part, we first initialize the edge/points status based on the user query, and then sequentially (i) find *boundary* edges, *dangling* edges and *outlier* points (Lines 1.10–1.30) based on the definition of different labels in the DT result, (ii) recognize multiple regions by connecting *boundary* edges as loops (Lines 1.32–1.42), and (iii) detect holes (Lines 1.43–1.45). Since the first two steps are straightforward, we refer readers to check Algorithm 1 for more details.

ALGORITHM 1: Baseline (D, P)

Input: A set D of geographical points, and a set P that are within the user-defined urban AOI (i.e., satisfy the user query). Each $d \in D$ has attributes {latitude, longitude, label}, and $d.label = outside$.
Output: A list of regions ($Regions$) and a list of outliers ($Outliers$)
Delauney_triangulation(D); ▷ Start of pre-processing
$S_T \leftarrow$ Triangle set of the DT result;
for *each edge* $e \in S_T$ **do**
  $e.label \leftarrow outside$;
$BE, DE, OP \leftarrow \varnothing$; ▷ Start of query processing
for *each point* $p \in P$ **do**
  $p.label \leftarrow inside$;
for *each triangle* t *in* S_T **do**
  $tp_{in} \leftarrow \varnothing$; // triangle points in P
  for *each point* p *in* t **do**
    if $p.label \neq outside$ **then**
      Add p to tp_{in};
      if $p.label = outlier$ **then**
        Remove p from OP;
  if $tp_{in}.length = 2$ **then**
    $e \leftarrow Edge(tp_{in}[0], tp_{in}[1])$;
    if $e.label = boundary$ **then**
      Delete e from BE;
      Add e to DE;
    else
      Add e to BE;
      $e.hole\_check\_point \leftarrow$ the incenter of t;
      $e.label \leftarrow boundary$;
      $tp_{in}[0].label \leftarrow boundary$;
      $tp_{in}[1].label \leftarrow boundary$;
  else if $tp_{in}.length = 1$ & $tp_{in}[0].label \neq boundary$ **then**
    $tp_{in}[0].label \leftarrow outlier$;
    Add $tp_{in}[0]$ to OP;
$Regions, edges \leftarrow \varnothing$;
while $BE \neq \varnothing$ **do**
  $e \leftarrow BE[0]$;
  while $e \neq null$ **do**
    Add e to $edges$;
    Remove e from BE;
    $e \leftarrow$ Adjacent(e, BE);
    $position \leftarrow$ Intersect(e, $edges$);
    if $position \neq$ -1 **then**
      $loop \leftarrow edges\{position, edges.\text{length-}1\}$;
      $edges \leftarrow edges$ - $loop$;
      Add $loop$ to $Regions$;
for *each* r *in* $Regions$ **do**
  if Point_in_Polygon($r[0].hole\_check\_point, r$) = True **then**
    $r.label \leftarrow hole$;
$Outliers \rightarrow$ Get_Outliers(OP, DE);
return $Regions$, $Outliers$;

Detecting holes. After step 2, we have recognized multiple regions, each of which is presented as a single loop formed by boundary edges. For example, Figure 5 will be recognized as three loops (regions) ($p_1p_2p_3p_4$, $p_5p_6p_7p_8$ and $p_9p_{10}p_{11}p_{12}$). It is worth noting that, the loop might also form a negative region. For example, the loop $p_9p_{10}p_{11}p_{12}$ in Figure 5 is a hole. To recognize such

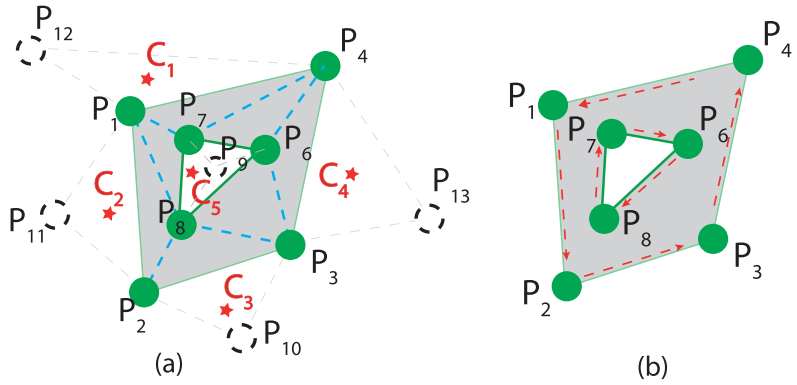


Fig. 6. Illustration of detecting holes: (a) the baseline method, which uses the *hole_check_point* (note: not all *hole_check_point*s are shown in the figure due to space limit); (b) the AOI-shapes algorithm, which is based on the direction of the *boundary* darts.

hole regions, when we traverse each triangle (t) in DT to recognize *boundary* edges, we store the incenter (i.e., the point forming the origin of a circle inscribed inside the triangle t, which is always inside the triangle t) of t as a *hole_check_point* (Line 1.23), as shown in Figure 6. Finally, to detect whether a region is a hole, we only need to check if the *hole_check_point* of one *boundary* edge of the region is inside the region or not. If it is inside, then we label the region as a hole (Lines 1.43-1.45). For example, in Figure 6, when we recognize p_7p_8 as a boundary edge based on triangle $p_7p_8p_9$ ($p_7, p_8 \in P$ but p_9 in $\overline{P}$), we store the incenter of $p_7p_8p_9$ (c_5) as the *hole_check_point* of *boundary* edge p_7p_8, which is part of the region $p_6p_7p_8$. Since c_5 is inside the region $p_6p_7p_8$, so the region is a hole region.

4.3 Time Complexity of the Baseline Method

Suppose that the number of points in D and P are N and n, respectively. For the pre-processing part, the Delauney triangulation costs O(Nlog$N$); since the number of edges in the triangulation result, which is proved to be no larger than $3N$ [50], the initialization of the labels of points and edges will cost O(N). Therefore, the pre-processing part is O(Nlog$N$). For the query processing part (Algorithm 1), finding boundary edges (step 1) needs to traverse all triangles, which is O($2N$); recognizing multiple regions need to traverse all boundary edges (and finding an adjacent edge for each boundary edge): it costs O($s$log$s$), where s is the number of *boundary* edges, and $s \ll n$; detecting holes (step 2) takes a linear O(m), where m is the total number of regions, and $m \ll s$. Therefore, the complexity of the query processing part is O(N).

5 THE PROPOSED AOI-SHAPES

In this section, we propose our AOI-shapes method to improve the efficiency of the baseline method by using a modified DCEL structure to store the triangulation result. In this subsection, we first give a brief description of the DCEL data structure, and then explain how DCEL could possibly improve the efficiency of the baseline method. Finally, we describe the proposed AOI-shapes algorithm and analyse the complexity in the end.

5.1 The DCEL Data Structure

Our proposed AOI-shapes algorithm is based on the **Doubly Connected Edge List (DCEL)** data structure [43], which is widely adopted in computational geometry and solid modelling. DCEL links three sets of records, i.e., vertices (points), darts (edges), and faces (triangles): each point is

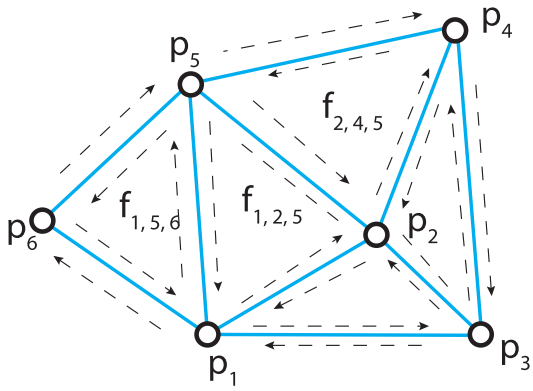


Fig. 7. Illustration of the DCEL structure, where there are 5 faces, 20 darts (half edges), and 6 vertices.

a vertex, the line connecting two points in the Delaunay triangulation is an edge, and a triangle is a face that consists of three darts. Each edge is represented as two "darts" (half edges) with a counter-clockwise direction.

The following terms will be helpful in explaining our algorithm.

- **Twin dart**. Since an edge is represented by two darts, each one of the two darts will have a twin dart in DCEL. As shown in Figure 7, the twin dart of $\langle p_1, p_5 \rangle$ in face $f_{1,5,6}$ is $\langle p_5, p_1 \rangle$ in face $f_{1,2,5}$.
- **Opposite vertex**. The opposite vertex of a dart is the other vertex that is within the same face with it. The opposite vertex of $\langle p_5, p_1 \rangle$ is p_2 (Figure 7). They share face $f_{1,2,5}$.
- **Start/end vertex**. A dart is a half-edge from a start vertex to an end vertex. As shown in Figure 7, the start vertex of $\langle p_5, p_1 \rangle$ is p_5, and its end vertex is p_1.

The following functions can be effectively implemented using the basic DCEL data structure.

- Twin_Dart(d), which returns the twin dart of dart d.
- Opposite_Vertex(d), which returns the opposite vertex of a given dart d.
- Start_Vertex(d), which returns the start vertex of d.
- End_Vertex(d), which returns the end vertex of d.

We also implement the following functions based on DCEL for efficiency purposes (details will be explained in later sections).

- Darts_Start_With(v), which returns all the darts starting from a given vertex v.
- Obtuse(d), which returns $True$ if the opposite angle of a dart d is obtuse.

5.2 Advantage of Using DCEL

We adopt the DCEL structure to store the DT result in our proposed AOI-shapes, because DCEL can link all three sets of records, i.e., edges, vertices, and faces. DCEL could possibly improve the efficiency of the baseline method (i.e., Algorithm 1) from the following aspects:

- For step 1 of Algorithm 1, where we find the *boundary* edges by traversing all the triangles (Lines 1.10–1.30): since DCEL links vertices and triangles (faces), for a give P, we only need to access those triangles that are connected with p ($p \in P$). Also, since DCEL links edges (darts) and faces, we can directly determine whether a candidate *boundary* edge is a *dangling* edge or not (Section 5.3.1).

ALGORITHM 2: AOI-SHAPES(D, P)

```
2.1  Input & Output: Same as Algorithm 1.
2.2  Lines 1.3-1.6 in Algorithm 1 ;                          ▷ Start of pre-processing
2.3  Build the DCEL structure based on S_T;
2.4  Same as Lines 1.9-1.11 in Algorithm 1. ;                ▷ Start of query processing
2.5  for each point p ∈ P do
2.6      Darts_p ← DARTS_START_WITH(p);
2.7      outlier_flag ← True;
2.8      for each dart dt ∈ Darts_p do
2.9          if END_VERTEX(dt).label = inside then
2.10             outlier_flag = False;
2.11             dt' ← TWIN_DART(dt);
2.12             if OPPOSITE_VERTEX(dt).label = inside then
2.13                 if OPPOSITE_VERTEX(dt').label = inside then
2.14                     dt.label=inner;
2.15                 else
2.16                     dt.label=in_boundary;
2.17                     Add dt to BE;
2.18             else
2.19                 if OPPOSITE_VERTEX(dt').label = inside then
2.20                     dt.label=out_boundary;
2.21                 else
2.22                     dt.label=dangling;
2.23                     Add dt to DE;
2.24     if outlier_flag = True then
2.25         Add p to OP;
2.26 for each edge e ∈ BE do
2.27     if e.obtuse = True then
2.28         BE ← DIGGING(e, BE);
2.29 Same as Lines 1.33-1.36 in Algorithm 1;
2.30 e ← ADJACENT_DCEL(e, BE);
2.31 Same as Lines 1.38-1.42 in Algorithm 1;
2.32 for each r in Regions do
2.33     if DIRECTION(r) = clockwise then
2.34         r.label ← hole;
2.35 RETURN Regions, Outliers;
```

- Since DCEL links vertices and edges (darts), the INTERSECT function (line 1.38) can be implemented in a more efficient way (Section 5.3.2).
- Since darts (edges) have direction information in DCEL, a hole can be more easily detected without storing *hole_check_point* (Section 5.3.3).
- With DCEL, a DIGGING function (which makes the final shape more "fit") would be efficient to implement (more details, see Section 5.3.4).

5.3 Constructing the AOI-shapes

Algorithm 2 presents the complete process of the proposed AOI-shapes method. Its input and output are the same as the baseline, and it also includes two steps: an offline pre-processing part and an on-line query processing part. The pre-processing part is the same as the baseline except that we build the DCEL structure after calculating the DT (Line 2.3). At the query processing step, besides those three steps in the baseline method, the AOI-shapes method has an extra step:

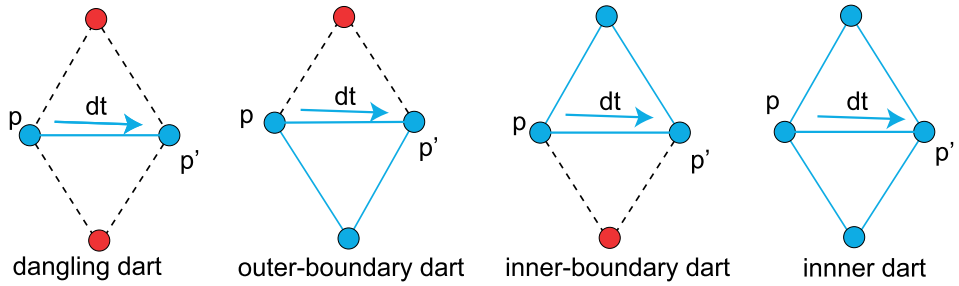


Fig. 8. Illustration of step 1 of the AOI-shapes algorithm.

a "digging" process, which makes the final boundary more "fit" to the region. Next, we illustrate how we implement each step of the AOI-shapes method using DCEL.

5.3.1 Step 1: finding boundary edges. In this first step, we try to find all the boundaries, i.e., *boundary* edges, *dangling* edges and outliers. In our baseline method (Algorithm 1), we traverse all the triangles in the resulting DT to find such boundaries. It is worth noting that, all the *boundary* edges and *dangling* edges connect two vertices within the user query (P), and the outliers are also within (P). Since DCEL could connect the vertex set and the edge (dart) set, we could possibly obtain all the *boundary* edges, *dangling* edges, and outliers, by traversing only those triangles that are connected with p ($p \in P$), instead of all the triangles in DT.

Lines 2.6–2.15 in Algorithm 2 present how we detect *inner* darts (Figure 8(a)), *boudary* darts (Figures 8(b) and 8(c)) and *dangling* darts (Figure 8(d)) in a single traversal of P. It is worth noting that the twin dart of an *inner-boundary* dart is an *outer-boundary* dart, and vice versa. For storage efficiency purposes, we only need to store one of them. In our algorithm, we store the *inner-boundary* darts (Line 2.17). All *boundary* darts refer to *inner-boundary* darts in the rest of the article, unless specified otherwise.

5.3.2 Step 2: recognizing multiple regions. The second step is to recognize multiple regions from the *boundary* darts. The whole process is similar to the baseline method: we connect those *boundary* darts via the intersection vertices. Our AOI-shapes method improves the efficiency of the baseline in the following aspect. In the baseline method (Line 1.39, Algorithm 1), we find one of the *adjacent* edges as the next connected edge using the function ADJACENT(), which traverses all the edges in BE to find an adjacent edge. In the AOI-shapes method (Line 2.30), with DCEL, we first find the end vertex (v_1) of the current edge (dart) e, and then use the function DARTS_START_WITH() to find all the darts (*darts*) starting from v_1. Then, for each dart (*dt*) in *darts*, if it is in BE, we will use it as the next edge e.

5.3.3 Step 3: detecting inner holes. In the baseline method, we detect whether or not a region is a hole based on the *hole_check_point*. In the AOI-shapes method, since darts in DCEL have directions, each region formed by the *boundary* darts will be either clockwise or counter-clockwise. As mentioned in Section 5.3.1, all the *boundary* darts stored from step 1 are *inner-boundary* darts. Therefore, all the non-hole regions should have the same direction as the DCEL face (counter-clockwise). Apparently, hole regions will have an opposite direction (clockwise), as shown in Figure 6(b). Lines 2.32–2.34 in Algorithm 2 demonstrate how we recognize the regions as hole regions based on the direction of the region (which is formed by *boundary* darts).

5.3.4 A "digging" process. Until now, the proposed AOI-shapes will return the same results as the baseline method. One problem of the resulting shape is: the region boundaries are formed at

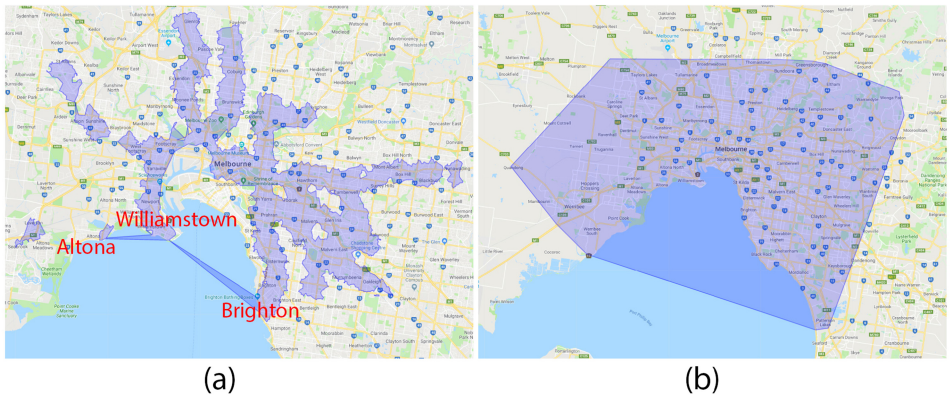


Fig. 9. Illustration of the problem of the shapes generated from the baseline method. (1) The AOI of properties within a 15-min walk to train stations and a 30-min train ride to the city; (2) the AOI of all properties within the Melbourne Metropolitan area.

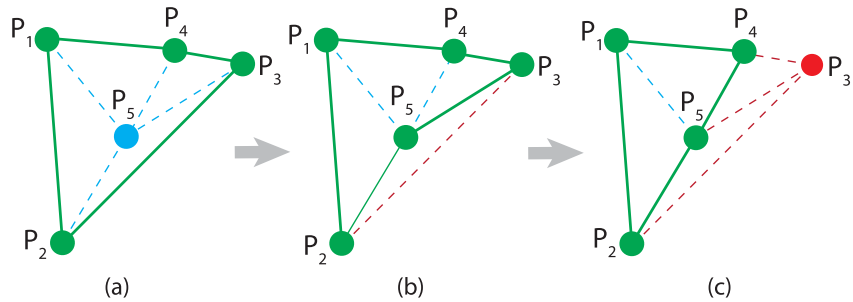


Fig. 10. Illustration of the "digging" process to construct the AOI-shapes. (a) The original *boundary* darts p_1p_2, p_2p_3, p_3p_4, p_4p_1, and an *inner* point p_5. (b) Since $\angle p_2p_5p_3$ is an obtuse angle, the "digging" process replaces the *boundary* dart p_2p_3 with p_2p_5 and p_5p_3, and p_2p_3 becomes a *dangling* dart. (c) Since $\angle p_5p_4p_3$ is obtuse, the *boundary* dart p_5p_3 will be replaced by p_5p_4 and p_4p_3; however, since p_4p_3 was originally a *outer-boundary* dart, both p_5p_3 and p_4p_3 will become *dangling* darts and p_3 becomes an *outlier* point.

the process where we separate those points in $\overline{P}$, so if the set of $\overline{P}$ is very "small," or at the worst case, $\overline{P}$ is $\varnothing$, then the resulting shape from the baseline method could be too "big" to characterize the boundary of region. Example 4 illustrates such a problem.

Example 4. We present two examples in Figure 9 using the baseline method. In Figure 9(a), we can see that, since there are no data points in $\overline{P}$ in the water area, the region between Altona and Williamstown, and the region between Williamstown and Brighton are unexpectedly connected. In Figure 9(b), where $\overline{P}$ is $\varnothing$, the resulting shape is the convex hull of D (P).

Existing work solves the above problem from two aspects. On the one hand, χ-shapes [14] and Jin et al. [38] use a "digging" process, which progressively replaces a "long" *boundary* edge with two "shorter" *inner* edges (as we have introduced in Example 2); however, this needs a parameter (which is hard to define) to determine whether a *boundary* edge is "short" enough to terminate the "digging" process. On the other hand, Peethambaran et al. [39] introduce a parameter-free method, which defines complex regularity and circle constraints to construct the "footprint" shape; however, the process will introduce an extra O($n$log$n$) time complexity.

Here, we combine the advantages of those two kinds of methods, and introduce a simple "digging" process (which is a simpler version of the complex regularity and circle constraints defined in Reference [39]). We consider a *boundary* dart to be too "long" if the opposite angle of the dart is obtuse (for details about why the obtuse angle is used, please refer to the circle constraint defined in Reference [39]); then, we will replace the dart with another two darts (Lines 2.26–2.28). This process happens between the step of identifying boundary edges (Step 1) and recognizing multiple regions (Step 2), and can be regarded as a process of redefining the boundary edges. Figure 10 illustrates the "digging" process. Example 5 presents the results of the AOI-shapes method compared to the baseline.

Example 5. Figure 4(a) presents the AOI-shapes result using the same user input as that in Figure 9(a) (baseline). We can see that those unexpected connections are removed in the "digging" process. Figure 4(b) presents the AOI-shapes result with the same empty $\overline{P}$ as that in Figure 9(b), we can see that, the AOI-shapes is more accurate.

5.4 Time Complexity of the AOI-shapes method

Compared to the baseline, the AOI-shapes method has an extra process to build the DCEL structure in the pre-processing part. Since the time of building the DCEL index depends on the number of edges in the triangulation result (O(3N)), the time complexity of the whole pre-processing part is still O(Nlog$N$), where N is the number of points in D. For the query processing part in Algorithm 2, we only need to traverse the points in P, whose complexity depends on the darts being visited: since each edge corresponds to two darts, finding boundary edges (step 1) has a complexity of O(6n), where n is the number of points in P. In the worst case, if $\overline{P} = \varnothing$, $n = N$, the cost of step 1 is O(6N). Similar to the baseline, the second and third steps cost O($s$log$s$) and O(m), respectively. The digging process needs to traverse all *boundary* darts, and on average it has a time complexity of O(s); in the worst case, if all *inner* darts are changed to *boundary* darts, it will cost O(6n). Therefore, the average time complexity of the query processing part is O(n), and in the worst case, the complexity is O(N). It is worth noting that, the AOI-shapes will return the same result for different traversing orders, and the time complexity remains the same, since all triangles to be visited will always be and only be visited once regardless the traversing orders.

6 THE INCREMENTAL AOI-SHAPES

AOI-shapes improves the efficiency of existing footprint methods by building a global DT at the pre-processing part. In this section, we propose two incremental methods to further improve the efficiency of the AOI-shapes by reusing the calculations at the query process part of the previous user query. The first incremental method reuses the labels of boundary edges (the result of Step 1, Section 5.3.1), and the second method reuses the final region results from the previous AOI-shapes.

6.1 Incremental Method I

Algorithm 3 presents the pseudocode for the first incremental method. We first calculate a difference set, and store the points set as P_{delete} and P_{new}. We then update the labels of all points in P_{delete} as *outside* and all points in P_{new} as *inside* (Line 3.8). Then, for each point in P_{delete} and P_{new} (Lines 3.9-3.12), we store all the following darts whose labels will be possibly changed: (1) all the darts connected with p (whose end point is also an *inside* point), and (2) each dart (dt) whose opposite point is p, and its twin dart (dt').

For each dart in *Darts*, we use the same condition check (shown in Figure 8) to label them as *dangling*, *boundary*, *inner* or *outsie* (Lines 3.13–3.14). Specifically, (1) if a dart d is changing from *dangling* to *outside*, then we check whether one of the points connected with d becomes an *outlier*;

ALGORITHM 3: INCRE_AOI-SHAPES_1 (P_{new}, P_{pre})

3.1 **Input**: the new point set P_{new}, the point set P_{pre} from the previous user query.
3.2 **Output**: A list of footprint regions $Regions$.
3.3 $P_{delete} \leftarrow P_{pre} - P_{new}$;
3.4 $P_{add} \leftarrow P_{new} - P_{pre}$;
3.5 **if** $P_{new}.size \leq P_{delete}.size + P_{add}.size$ **then**
3.6 | RETURN AOI-SHAPES(D, P_{new});
3.7 $Dt \leftarrow \varnothing$;
3.8 Update the label of points in P_{delete} and P_{add};
3.9 **for** *each point* $p \in P_{delete} \cup P_{add}$ **do**
3.10 | **for** *each dart* $dt \in$ (OPPOSITE_DARTS(p) $\cup$ START_VERTEX(p)) **do**
3.11 | | $dt' \leftarrow$ TWIN_DART(dt);
3.12 | | Add dt and dt' to Dt;
3.13 **for** *each dart* $dt \in Dt$ **do**
3.14 | Lines 2.12-2.23 in Algorithm 2;
3.15 CHECK_OUTLIERS();
3.16 Lines 2.26-2.34 in Algorithm 2;
3.17 RETURN $Regions$, $Outliers$;

(2) if a new point does not connect with any darts whose end point is also in P_{new}, then the new point becomes an *outlier* point (Line 3.15).

After finding the new boundary edges, the incremental algorithm uses the same steps to dig the boundary edges, recognize multiple regions and detect inner holes as the AOI-shapes algorithm (Line 3.15), and finally gets the region and outlier list (Line 3.16).

It is worth noting that, If the number of points in the new point set (n) is smaller than the sum of the number of points to be added and the number of points to be deleted, then we will directly consider the new point set to generate a new footprint using the AOI-shapes method (Lines 3.5–3.6).

6.2 Incremental Method II

Here, we reuse the final boundary and outlier information from the previous user query, and update each individual point one by one. Algorithm 4 presents the whole process. Same as Algorithm 3, we first calculate a difference set, and store the point set as P_{delete} and P_{new} (Line 4.2). Adding a new point might result in multiple existing separated regions connected, and deleting a point might result in a region splitting into multiple regions. To avoid a newly merged region being split in the removal process (Lines 4.3–4.10), we remove points first before we add new points (Lines 4.11–4.18).

6.2.1 Removing existing points. Removing an *outlier* point (e.g., p_{15} in Figure 11) does not need further operations. Removing a *dangling* point (e.g., p_{13} in Figure 11) might result in the other connected *dangling* point (p_{14}) becoming an *outlier* point.

To remove a boundary point p, we only need to find the connected vertices of p, and consider the following: (1) if no connected vertices are *inside* points, and there are more than three points in the hull boundary, then we only need to remove p, and connect the other two vertices as a new *boundary* edge (e.g., removing p_2 from Figure 11(a)); (2) if p connects with an *inside* point, then the *inside* point is labelled as *boundary*, and will be connected with the other two vertices to make two new *boundary* edges (e.g., removing p_7 from Figure 11(a)); (3) if there are only two *boundary* vertices left for the affected hull after p is removed, then the two *boundary* vertices will become *dangling* points (e.g., removing p_{10} from Figure 11(a)).

It is worth noting that, removing a *boundary* point (p) may split a hull into multiple disconnected components. In such a case, we re-label all the affected darts (those darts connect p and those darts

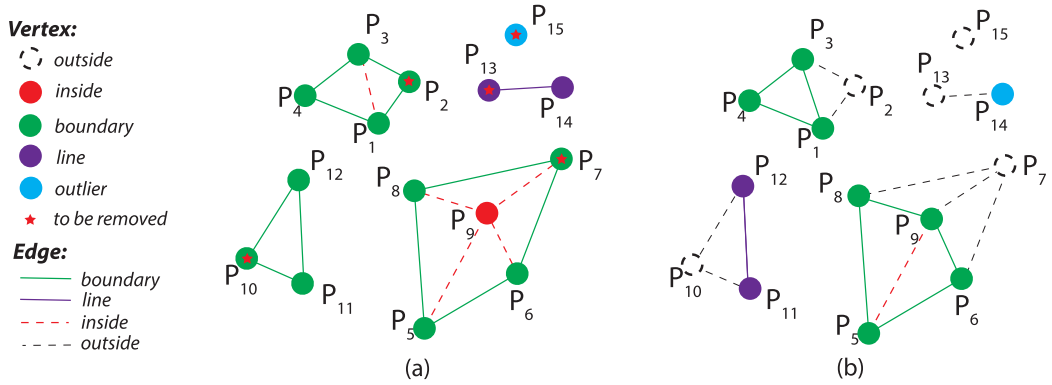


Fig. 11. Example of removing *boundary* points (p_2, p_7, and p_{10}), *line* points (p_{13}), and *outlier* points (p_{15}): (a) the original footprints, with the points to be removed labelled with stars; (b) the result after removing the starred points.

ALGORITHM 4: INCRE_AOI-SHAPES(P_{new}, P_{pre})

```
4.1  Input & output: Same as algorithm 3.
4.2  Same as Lines 3.3-3.6 in algorithm 3.
4.3  while P_delete ≠ ∅ do
4.4      while ∃p : p ∈ P_delete & p.label = boundary do
4.5          REMOVE_BOUNDARY_POINT(p);
4.6      while ∃p : p ∈ P_delete & p.label = dangling do
4.7          REMOVE_LINE_POINT(p);
4.8      while ∃p : p ∈ P_delete & p.label = outlier do
4.9          REMOVE_OUTLIER(p);
4.10     REMOVE_INSIDE_POINT(p);
4.11 while P_add ≠ ∅ do
4.12     while ∃p : p ∈ P_add & p.label = transition do
4.13         while ∃p : TRANSITION(p).label = outlier do
4.14             ADD_OUTLIER(p);
4.15         while ∃p : TRANSITION(p).label = dangling do
4.16             ADD_DANGLING_POINT(p);
4.17         while ∃p : TRANSITION(p).label = boundary do
4.18             ADD_BOUNDARY_POINT(p);
4.19 RETURN Regions, Outliers;
```

that have p as opposite points), then we connect those *boundary* edges (related to p) and get the region boundary.

Removing an *inner* point p will result in either holes or *dangling* edges. On the one hand, if all the opposite darts of p are *inside* darts, then there will be a hole formed by those opposite darts. On the other hand, if one or more of the opposite darts of p are *boundary* darts, then those *boundary* darts will become *dangling* darts, and there might be new regions formed by other *darts*.

6.2.2 Adding new points. When adding a new point to the boundary, we need to traverse the current boundary and detect whether there are *boundary* darts connecting with the edge. To avoid such a traversal, we add another label for *outside* points: *transition*, and consider those *transition* points first when updating the AOI-shapes.

- ***Transition***. A *transition* point is a point in $\overline{P}$, but that shares a triangle with a *boundary* edge, a *dangling* edge or an *outlier* point (i.e., it is the opposite vertex of the twin dart of a *boundary* or a *dangling* edge, or it directly connects with an *outlier* point).

Since *transition* points are the closest to existing *boundary* edges, *dangling* edges or *outlier* points, we first consider such points. It is obvious that: (1) adding a transition point p' to an *outlier* point p results in a *dangling* edge pp'; (2) adding a transition point p'' to a *dangling* edge pp' results in a new triangle $pp'p''$ (if the three points are non-collinear): p, p', and p'' will become three *boundary* points.

A major challenge while adding new points is that multiple disconnected components may become connected, and we need to merge such components. In such a case (i.e., a new point p connects with multiple regions), we re-label all the affected darts (those darts connect p and those darts that have p as opposite points), connect those *boundary* edges (related to p) and obtain the region boundary.

6.3 Time Complexity of the Incremental Algorithms

Suppose that, the number of points in the new point set is n, the numbers of points to be added and deleted are a and d, respectively. For the first incremental method, the process of finding boundary edges take O(MIN(n, $a + d$)) time, and the rest of the process takes the same time as the original AOI-shapes method, i.e., O($slogs + m$), where s is the number of *boundary* edges, and m is the total number of regions. Therefore, the overall time-complexity in the worst case is O(n). For the second incremental method, in the worst case adding (removing) a single point p could result in connecting (disconnecting) O(n) disconnected (connected) boundaries, in which case this incremental step takes O(n) time. The number of such loops is O(MIN(n, $a + d$)). Therefore, in the worst case, the time complexity of the second incremental AOI-shapes methods is O(MIN(n^2, $(a + d)n$)).

7 EXPERIMENTS

We have implemented an online demo system[2] using Australia's real estate data [28] to demonstrate the usefulness of the proposed AOI-shapes for interactive visualizations of urban AOIs. Besides the AOI-shapes, we have also implemented several state-of-the-art alternatives. In the demo, we allow users to switch freely among different methods and compare both the quality and efficiency of them. We also provide a slider for users to change the parameter if the alternative method (e.g., χ-shapes [14]) needs one. For demonstrations using real-life scenarios and how AOI-shapes supports human-in-the-loop visual analytics of the data, please refer to our demo paper [27].

In this section, we conduct experiments to evaluate the quality and efficiency of our proposed AOI-shapes based on both synthetic and real-world datasets and compared it with the state-of-the-art alternatives shown in Table 2. All the methods are implemented in JavaScript, which is open-sourced and available online.[3] It is worth noting that, the implementations of all methods use the same libraries if needed (e.g., all methods that are based on Delaunay triangulation used the same library named *Delaunator*[4]). All the experiments are done in Chrome (Version 74.0.3729.169) on a quad-core i7-5500U CPU (2.4 GHz) with 8 GB RAM.

[2] http://aoishapes.com.
[3] http://aoishapes.com/js/.
[4] https://github.com/mapbox/delaunator.

Table 2. Alternative Methods Compared in the Experiment

	Parameter range	Default value	Complexity
KNN-based concave hull [36]	[3,n]	$\lceil n/10 \rceil$	O(n^3)
Convex+"digging" [47]	[0,1]	0.2	O($n$log$n$)
ConcaveMan [38]	[0,1]	1	O($n$log$n$)
α-shapes [44]	[−1,1]	0.2	O($n$log$n$)
χ-shapes [14]	[0,1]	0.1	O($n$log$n$)
χ-outliers [55]	[0,2]	0.2	O($n$log$n$)

n: number of points in P.

(a) AOI-shapes: 0.06s **(b)** AOI-baseline: 0.12s **(c)** α-shapes (α = 100): 3.8s **(d)** χ-outlier (χ=0.15): 24s

(e) KNN (k=2270): 58s **(f)** Digging (λ=0.05): 3.8s **(g)** Concaveman(c=1): 1.4s **(h)** χ-shapes (χ=0.1): 0.1s

Fig. 12. Results of the "footprint" generated based on each method and the construction time (P is properties within 15 min to the nearest train stations in Melbourne).

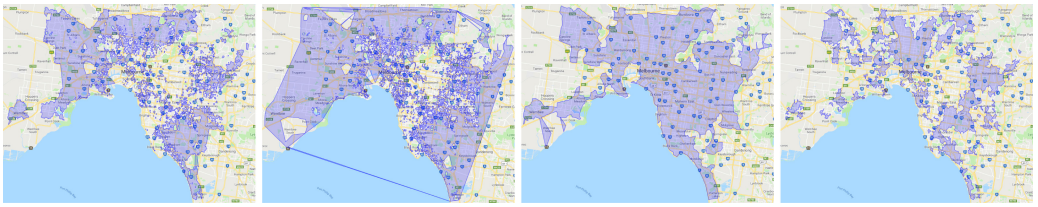
7.1 Quality of the AOI-shapes

We first evaluate the quality of our proposed AOI-shapes and compare it with the state-of-the-art alternatives (Table 2). We present both the qualitative and quantitative comparisons in this subsection.

Since most alternative methods need one parameter, for each method, we have either tried 5–10 different values within the range or chosen the optimal value suggested by the original authors. For example, for χ-shapes, the authors suggested that a parameter in [0.05, 0.2] out of the range [0, 1] often provides optimal or near-optimal characteristic shapes; in our experiments, we have used different χ values within the suggested range.

7.1.1 Qualitative evaluation. Based on the real estate data, we have presented several real-world AOI-visualization scenarios supported by AOI-shapes in our demo paper [27]. Here, we present the footprint results and the construction time for eight different methods, using a user query (properties within a 15-min walk to the nearest train station) as an example.

As shown in Figure 12, the proposed AOI-shapes can recognize multiple regions and inner holes and is one of the most efficient methods. On the one hand, only those four methods on the top row (AOI-shapes, AOI-baseline, α-shapes, and χ-outliers) can recognize multiple regions and outliers. While those four methods present similar results, the construction of AOI-shapes and the



(a) AOI-shapes: 0.12s **(b)** AOI-baseline: 0.40s **(c)** α-shapes (α=100): 10.8s **(d)** χ-outlier (χ=1.0): 57.6s

Fig. 13. Results and construction time of the "footprint" generated based on four methods that is able to recognize multiple regions (P is properties sold within 1 million dollars in Melbourne).

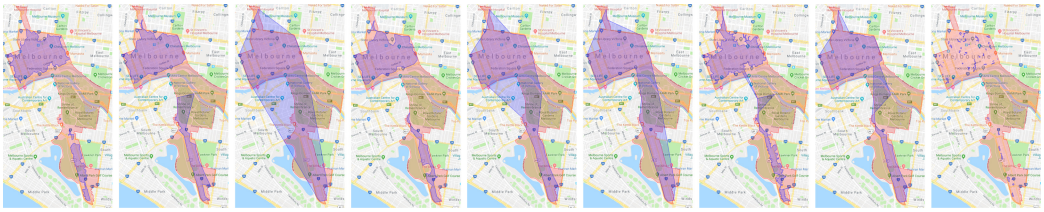
AOI-baseline are much more efficient compared to the other two methods (more details are given in section 7.2). On the other hand, the AOI-shapes and AOI-baseline can detect holes inside the region (e.g., those regions in the middle of two train lines as shown in Figure 1(d)); α-shapes with a negative α are also possible for hole detection, however, the sign of α needs to be pre-defined based on whether the final shape has a hole or not (which might be unknown to system designers); all the other five methods cannot detect holes.

We further compare all the methods based on another example, where the AOI query is defined as "regions with properties sold below 1 million dollars." The result of AOI-shapes (Figure 13(a)) reveals that: house prices in most of the areas in South and East Melbourne are above 1 million dollars. Figure 12(b) presents similar patterns, but there are unexpected region connections on top of the water, and the region boundary is not optimal. χ-outliers (Figure 13(d)) also presents similar results but it takes much longer time (1 min vs 0.1 s) to construct. χ-shapes (Figure 13(c)) are not able to recognize the empty areas. Similar to that in Figure 12, all the four other methods present similar results to χ-shapes; readers can check the results by exploring the online demo.[2]

7.1.2 Quantitative evaluation. Since the boundary of AOI is usually vague and unknown, there is no ground truth of AOI for quantitative evaluation. To quantitatively compare our algorithms with existing methods, we conduct experiments by reconstructing "footprints" derived from known "ground-truth" regions.

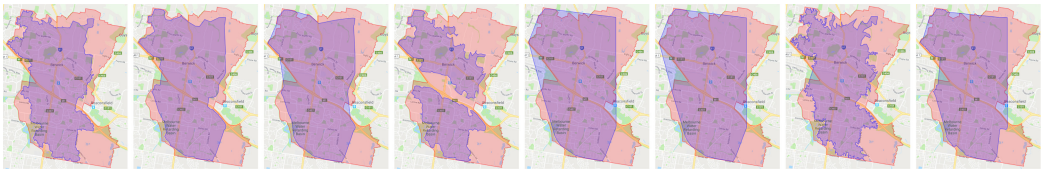
Experimental setup. Since state-of-the-art methods cannot detect inner holes and only two of them can recognize multiple regions, the "ground-truth" regions have to be designed to be simple hulls (no holes). In other words, none of the state-of-the-art methods can be an appropriate baseline for a fair comparison. The only reason we still conduct the above experiment of "comparing our methods with state-of-the-art methods on constructing simple hulls" is for the purpose of comprehensive evaluation (although it is not fair to our method and constructing simple hulls is not the focus of this article). Particularly, we designed the following datasets.

- **Dataset 1: administrative districts (Melbourne CBD) + real estate data points**. The first dataset is based on the region of Melbourne CBD and the real estate data [28] mapped in Melbourne CBD (1,500 properties are included after removing duplicate locations). The "ground-truth" shape is the geometry boundaries of Melbourne CBD, and the real estate data in the region form the P.
- **Dataset 2: administrative districts (Berwick) + real estate data points**. Similar to dataset 1, we select the suburb of Berwick (which has one of the highest numbers of properties sold in the regional Victoria, Australia) as the ground-truth shape; and the properties mapped in the region are denoted as P (9,350 properties).
- **Dataset 3: synthetic letters + synthetic points**. We adopt the letter-based dataset defined in Reference [55], where six hole-free letters were arbitrarily chosen: "H," "E," "X," "C," "S," and "W"



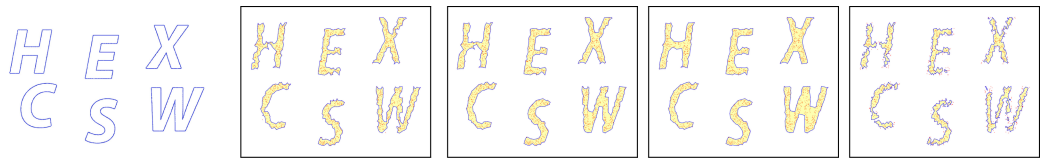
(a) AOI-shapes: 5ms **(b)** AOI-baseline: 15ms **(c)** α-shapes (α =100): 183ms **(d)** χ-outlier (χ=1.5): 998ms **(e)** KNN (k=100): 1104ms **(f)** Digging (λ=0.05): 50ms **(g)** Con-caveman (c=1): 78ms **(h)** χ-shapes (χ=0.1): 56ms **(i)** χ-outliers (χ=0.15): 1205s

Fig. 14. Properties in Melbourne CBD (purple: the generated footprint; yellow: the ground-truth shape).



(a) AOI-shapes: 16ms **(b)** AOI-baseline: 33ms **(c)** α-shapes (α = 100): 471ms **(d)** χ-outlier (χ=1.5): 1517ms **(e)** KNN (k=100): 5007ms **(f)** Digging (λ=0.05): 108ms **(g)** Concave-man (c=1): 115ms **(h)** χ-shapes (χ=0.1): 85ms

Fig. 15. Properties in Berwick, Victoria (purple: the generated footprint; yellow: the ground-truth shape).



(a) Orginal synthetic letters **(b)** AOI-shapes: 25ms **(c)** AOI-baseline: 56ms **(d)** α-shapes (α=0.5): 130ms **(e)** χ-outlier (χ=0.2): 240ms

Fig. 16. Footprint results based on the letter dataset.

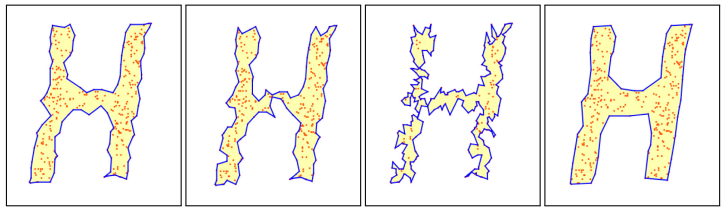
are placed on a 2×3 grid, which is the "ground-truth" shape (Figure 16(a)). It is worth noting that, since only the proposed methods and α-shapes with appropriate parameter settings can recognize holes, it is not fair to other methods if we generate letters with holes. We then generate the point set as described in Reference [55], which are P in our algorithm.

- **Dataset 4: individual letter + synthetic points**. Since half of the methods cannot recognize multiple regions, we also generate six datasets with each individual letter as the ground-truth shape.

Evaluation metric. Same as existing work (e.g., References [5, 14, 55]), we use the area-based F1-score, to evaluate the differences of the footprints based on the input points from the "ground-truth" shape. The score is calculated as the harmonic mean of area-based precision and recall:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \times 100\%, \tag{1}$$

where precision is the proportion of the footprint area that intersects the true shapes, and the recall is the proportion of the true area that is captured by the footprint.



(a) KNN (k=100): 17ms **(b)** Digging (λ=0.05): 49ms **(c)** Concaveman (c=1): 37ms **(d)** χ-shapes (χ=0.2): 85ms

Fig. 17. Footprint results based on an individual letter "H."

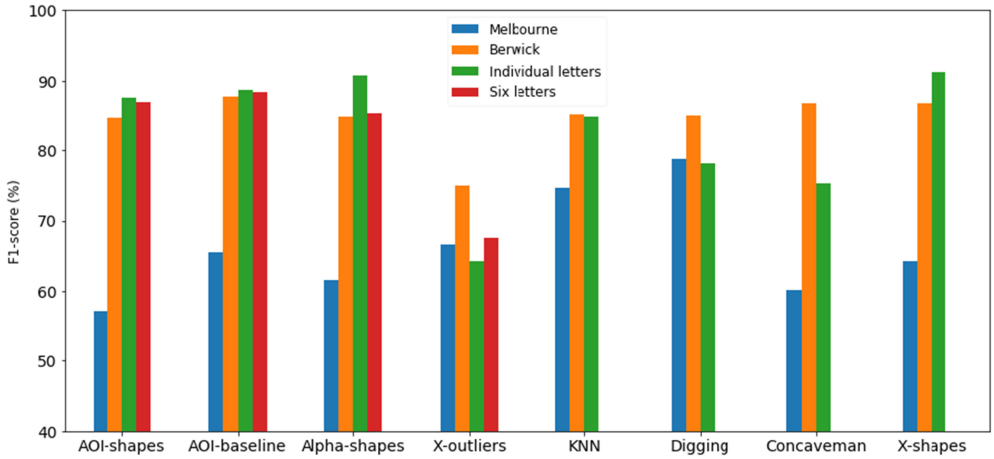


Fig. 18. Region-based F1-score for quantitative evaluation.

Experimental result. We visualize the results of all generated footprints (compared with the ground truth) in Figures 14–17 for each of the datasets. The F1-scores are presented in Figure 18. Since our method considers $\overline{P}$, we can see that almost all the boundaries from AOI-shapes are within the actual ground-truth shapes, corresponding to a better precision score. The reasons why our proposed AOI-shapes does not have significant advantages regarding F1-scores (Figure 18) include: (1) all the datasets do not have multiple regions and/or inner holes (which are the main strengths of our methods), (2) the ground truth of suburb geometry boundary has a significant proportion of gardens where there are no real estate properties; although the AOI-shapes correctly excludes those regions (shown in Figures 14(a) and 15(a)), it results as a worse recall score compared to other methods.

It is worth noting that parameter-free is one of the biggest strengths of the proposed AOI-shapes, since all the alternatives need a parameter, which is hard to specify for users. What is worse is that the optimal values vary in different applications and even different scenarios of the same application [14, 55]. For example, for χ-outliers, the range of parameter χ is [0, 2]. We have chosen an optimal χ as 0.15 in Figure 12(d) and 0.10 in Figure 13(d). However, as shown in Figure 14(i), a χ of 0.15 will result that most of the data points become outliers and thus give us multiple small regions; and after comparison of the F1-score, a χ of 1.5 gives us the near-best result, as shown in Figures 14(d) and 15(d).

7.2 Efficiency of the AOI-shapes

We evaluate the efficiency of the proposed AOI-shapes by comparing the construction time of AOI-shapes with the alternatives (Table 2). Since AOI-baseline is the only method whose complexity is

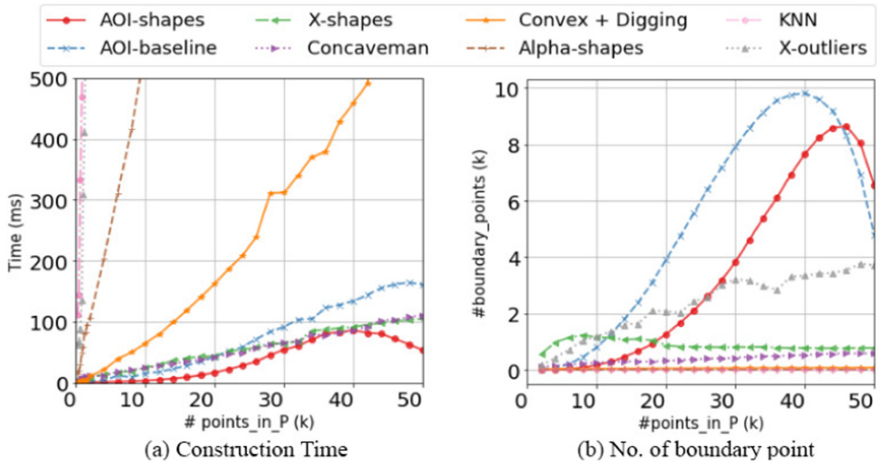


Fig. 19. 50k real estate dataset in Melbourne (query: distance to train stations).

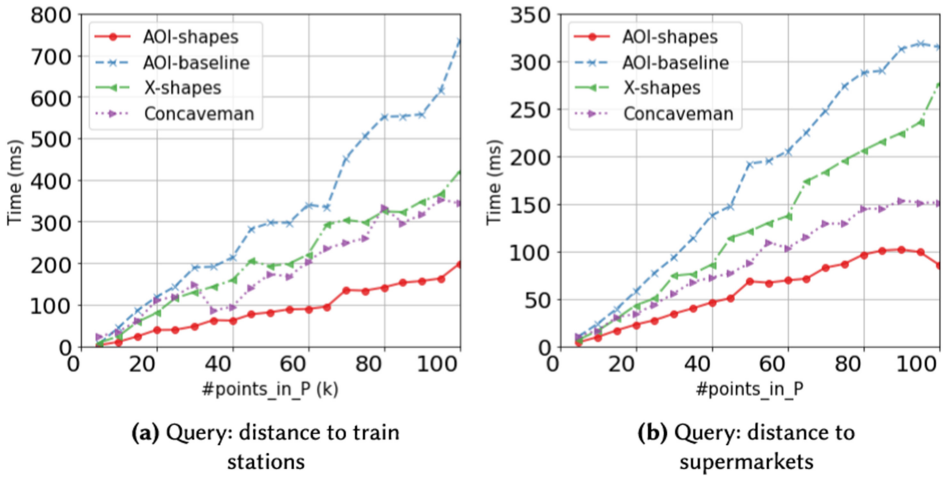
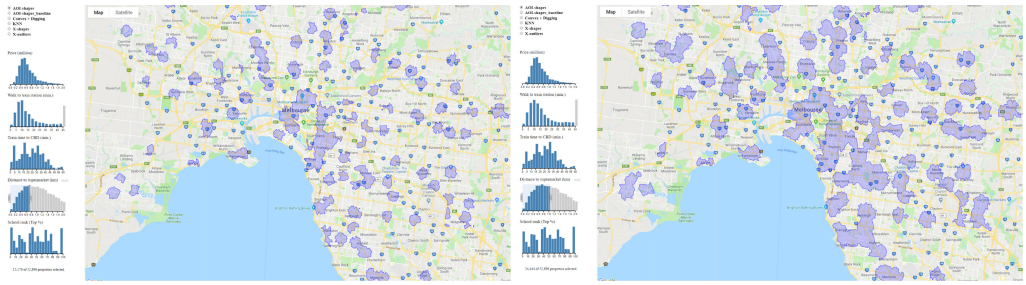


Fig. 20. Construction time (500k real estate dataset).

based on N instead of n, in our implementation of AOI-baseline, we store a mapping from points to triangles in DT to make the baseline step 1 more efficient, so the AOI-baseline can also have a time complexity of O(n), instead of O(N).

Simulated user query. As shown in Figure 4(a), the AOI is often generated based on range queries (for quantitative attributes). Therefore, for efficiency evaluation, we simulated user queries based on the attribute range. For example, for a real estate dataset with 50k properties (Figure 19), we first sort the data based on the distance to the nearest train station. Simulated query (P) is selected as the first 2k, 4k, . . . , 50k rows of the sorted data. Each of the user queries will correspond to a specific user query (e.g., 20k corresponds to a 15-min walk to the nearest train station (Figure 12)).

For those alternative footprint methods, which all need parameters, the choice of parameters might influence their construction time. For example, for χ-shapes, a larger χ will terminate the algorithm earlier than a smaller χ, and thus takes a shorter time to construct the footprint. As discussed in Example 2, the choice of χ will also influence the complexity of the resulted shape, which can be measured by the number of boundary points. Our experiment shows that: the



(a) Regions within 600m to supermarkets **(b)** Regions within 800m to supermarkets

Fig. 21. Illustration of why the AOI-shapes curve in Figure 19(b) goes down when there are more points in P: Panel (b) has more numbers of points in P but results in fewer regions, since some of the regions are merged.

AOI-shapes can deliver a more complex shape (with more boundary points) in the least construction time, compared to the alternatives with default parameters. As shown in Figure 19(a), four methods (AOI-shapes, χ-shapes, Concaveman, and AOI-baseline) are the most efficient ones, the construction of which are much shorter than the other four methods; however, both χ-shapes and Concaveman only generate simple shapes (shown in Figure 19(b)). The only method that can generate complex shapes as AOI-shapes and AOI-baseline is χ-outliers; however, its construction time is at least a thousand times more compared to AOI-shapes. For AOI-shapes and AOI-baseline, both the number of boundary points and the construction time decrease after around 40k, since different regions might merge. Also, AOI-shapes takes about half of the construction time of AOI-baseline thanks to the DCEL structure, although AOI-shapes has an extra digging process. It is worth noting that, in Figure 19(b), the number of boundary points in both AOI-shapes and AOI-baseline decreases once there are more than 40–45k points in P. X-outliers has a similar phenomenon (but not stable). The reason is illustrated in Figure 21: when we increase the distance to train stations/supermarkets, some separated regions might be merged, which will result in fewer regions (fewer boundary points). Other methods do not have such a phenomenon, since they can only recognize a single region.

Among those four more efficient methods, we further evaluate their scalability on a larger dataset, which has 500k real estate properties. The query is constructed based on the distance to the nearest train station (Figure 20(a)) and the distance to the nearest supermarket (Figure 20(b)). Both results reveal that the AOI-shapes is the most efficient one. Compared to Figures 20(a) and 20(b), we find that with *distance_to_supermarket* as input, the construction time of all footprints are shorter than the one using *distance_to_train_station* as input. The reason is that the resulted region of the query related to the supermarket is less complex with the same number of points in P. Also, if we compare Figures 19(a) and 20, then the size of D has no significant effect on the construction time of AOI-shapes, since only the edges connected with P will be visited for constructing the AOI-shapes (the size of D affects the construction time of the global DT, which is not included here).

7.3 Efficiency of the Incremental AOI-shapes

We further evaluate and discuss whether the proposed two incremental methods can improve the efficiency compared with the original AOI-shapes. For evaluation purposes, we skip the process of comparing P_{new} and p_{pre} for the incremental methods (Lines 3.5–3.7 in Algorithms 3 and 4).

Similar to Section 7.2, we simulated the user query based on data ranges with the real estate data. Here, we use the 50k real estate dataset as D. We select five attributes to define the range query: *price*, *distance_to_train_station*, *distance_to_supermarket*, *distance_to_hospital*,

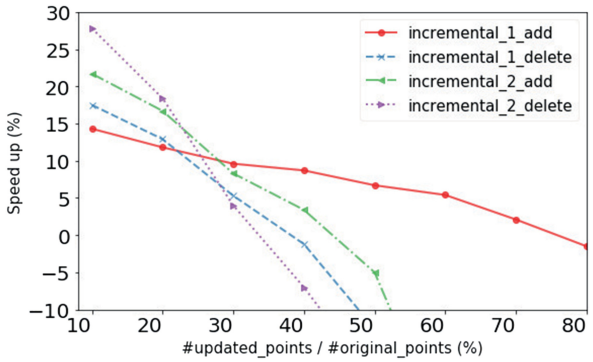


Fig. 22. Effeciency of the two incremental methods compared to the original AOI-Shapes (based on 50k properties).

school_rank. We sort the data based on each attribute, and then choose the first $n\%$ (n is a random number between 10 and 50) of the data as the original P. Then, we gradually add or delete data based on P and report whether the two incremental methods will speed up the construction process based on the ratio of updated point number and original point number (the speed-up percentage is average score based on the five chosen attributes).

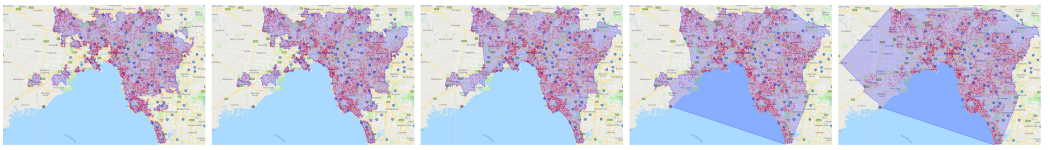
As shown in Figure 22, when there are more than 35%–40% of data to delete, both of the incremental methods are worse than the original method. The second incremental method (which updates the final shapes) works well when there are only a few data points (less than 20%) to be updated, while the first incremental method (which improves the efficiency at the labeling process) works better when there are more data points to update. The incremental methods will not speed up the process when there are more than about 35% (incremental_2) and 80% (incremental_1) of data points to be updated. The reason is that the updating process based on the final points (incremental_2) takes much longer time than the original construction process; for incremental_1, more added data points might also result in more original point labels to be revisited.

7.4 Discussion

In this subsection, we discuss the strengths and weaknesses of the proposed AOI-shapes, applications and extensions.

Strengths. Based on the experimental results, the proposed AOI-shapes has several advantages over the state-of-art footprint methods. First, all the alternatives need a parameter, which often has no real-life meaning and is hard to define, and the optimal parameter setting for the same method varies for different applications. Second, compared with χ-outliers and α-shapes, which possibly present similar results with an appropriate choice of parameters, the construction of AOI-shapes is normally thousands of times faster. Third, compared with the proposed AOI-baseline, AOI-shapes is faster and more accurate on boundary detection (especially when $\overline{P}$ is sparse, as shown in Figure 13). Finally, compared to all the other methods, AOI-shapes takes at least a similar time as the most efficient state-of-the-art χ-shapes, but can recognize multiple regions and inner holes.

Limitations. One possible limitation of AOI-shapes is the need for $\overline{P}$. Without $\overline{P}$, the AOI-baseline will return a convex hull (Figure 9(b)), and the AOI-shapes, which includes an additional "digging" process, might return a result that is "better" than the convex hull (Figure 4(b)), but there is no guarantee of the result quality. As described in Section 5.3.4, our "digging" process is a simpler vision of the complex "circle" and "regularity" constraints defined in Reference [39]. For applications with no available $\overline{P}$, the parameter-free method defined in Reference [39] and other parameter-based



(a) *angle* = 80 **(b)** *angle* = 85 **(c)** *angle* = 90 **(d)** *angle* = 95 **(e)** *angle* = 100

Fig. 23. Illustration of varying the *angle* in the "digging" process ($\overline{P}$=∅). The optimal *angle* in this application might be between 85 and 90.

methods such as χ-shapes and χ-outliers might return much "better" results than AOI-shapes. One possible extension of the AOI-shapes is to introduce a parameter related to the angle in the "digging" process. Figure 23 presents a comparison of different angle choices when $\overline{P} = \varnothing$. Another limitation of the method is the pre-computation part of DT. Since DT computation is based on the global dataset D, the space needed to store the DT result and DCEL is O(N), where N is the number of points in D. This means that more memory consumption is needed by AOI-shapes compared to the alternative methods (whose space complexity is O(n), where n is the number of points in P). Generally speaking, with a pre-computed global DT stored in memory, AOI-shapes sacrifices the space for better time efficiency.

Applications. AOI-shapes is more suitable for applications where both P and $\overline{P}$ are available. Besides the real estate domain presented throughout the article, there are many other possible applied scenarios of the method. For example, in traffic analysis, those road points that have traffic jams can form a P, while other uncongested points can form a $\overline{P}$; for facility deployment, P and $\overline{P}$can also be easily obtained based on online geo-tags.

Extensions. Based on our experiments, AOI-shapes works well over dataset under 500k records (most of the query can be returned within 0.5 s). There are several directions to extend the AOI-shapes to support a dataset that is above 500k and avoid interactive latency. On the one hand, we could divide the global D into subsets based on either the administrative districts (e.g., states, suburbs) or using geospatial data structures such as Quadtree [49], then we can compute and store the DT for each subset of D. On the other hand, only a subset of the AOI might be of interest to the user even when the actual number of points in P is large. For example, the map returned to the user might only contain 10% of the points in P when the map is zoomed in; while when the map is zoomed out, the details of the boundaries might not be important to the user anymore.

8 CONCLUSION AND FUTURE WORK

In this article, we studied the problem of visualizing user-defined urban areas of interest (AOIs). First, we summarized the existing "footprint" methods. To address the limitations of the existing studies, we proposed a novel parameter-free footprint method, namely, AOI-shapes, to generate the region boundary of user-defined urban AOIs to visualize on the map. Specifically, the proposed AOI-shapes is able recognize multiple regions, outliers and inner holes of the AOI. To effectively update the AOI as per user's update of query, we proposed two efficient and scalable algorithms to generate the AOI in an incremental manner. Experiments on both synthetic and real-world datasets confirmed the quality and efficiency of our proposed methods.

Besides the extensions and applications of AOI-shapes presented in Section 7.4, we also propose the following directions for future work. On the one hand, we would like to conduct user studies to compare different footprint methods in different applications. On the other hand, based on the summary table of different footprint algorithms (Table 1), we plan to build an online footprint system that allows users to upload their own data and helps them choose the most appropriate footprint method in their own applications.

REFERENCES

[1] Fatih Akdag, Christoph F. Eick, and Guoning Chen. 2014. Creating polygon models for spatial clusters. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. 493–499.

[2] Harith Alani, Christopher B. Jones, and Douglas Tudhope. 2001. Voronoi-based region approximation for geographical information retrieval with gazetteers. *Int. J. Geogr. Info. Sci.* 15, 4 (2001), 287–306.

[3] Alex M. Andrew. 1979. Another efficient algorithm for convex hulls in two dimensions. *Info. Process. Lett.* 9, 5 (1979), 216–219.

[4] Avi Arampatzis, Marc van Kreveld, Iris Reinbacher, Christopher B. Jones, Subodh Vaid, Paul Clough, Hideo Joho, and Mark Sanderson. 2006. Web-based delineation of imprecise regions. *Comput. Environ. Urban Syst.* 30, 4 (2006), 436–459.

[5] Saeed Asaeedi, Farzad Didehvar, and Ali Mohades. 2017. A generalization of convex hull. *Theor. Comput. Sci.* 702 (2017), 48–59.

[6] G. Berg, W. Julian, R. Mines, and Fred Richman. 1975. The constructive Jordan curve theorem. *Rocky Mount. J. Math.* (1975), 225–236.

[7] Alex Bykat. 1978. Convex hull of a finite set of points in two dimensions. *Info. Process. Lett.* 7, 6 (1978), 296–298.

[8] Zechun Cao, Sujing Wang, Germain Forestier, Anne Puissant, and Christoph F. Eick. 2013. Analyzing the composition of cities using spatial clustering. In *Proceedings of the ACM SIGKDD International Workshop on Urban Computing*. 1–8.

[9] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. 1991. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference*. 181–186.

[10] Emre Celikten, Geraud Le Falher, and Michael Mathioudakis. 2017. Modeling urban behavior by mining geotagged social data. *IEEE Trans. Big Data* 3, 2 (2017), 220–233.

[11] R. V. Chadnov and A. V. Skvortsov. 2004. Convex hull algorithms review. In *Proceedings of the Russian-Korean International Symposium on Science and Technology*. 112–115.

[12] Timothy M. Chan. 1996. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete and Computational Geometry* 16, 4 (1996), 361–368.

[13] A. Ray Chaudhuri, Bidyut Baran Chaudhuri, and Swapan K. Parui. 1997. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *Comput. Vision Image Understand.* 68, 3 (1997), 257–275.

[14] Matt Duckham, Lars Kulik, Mike Worboys, and Antony Galton. 2008. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recogn.* 41, 10 (2008), 3224–3236.

[15] William F. Eddy. 1977. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.* 3, 4 (1977), 398–403.

[16] Birgit Elias. 2003. Extracting landmarks with data mining methods. In *Proceedings of the International Conference on Spatial Information Theory*. 375–389.

[17] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the SIGKDD Conference*. 226–231.

[18] Antony Galton and Matt Duckham. 2006. What is the region occupied by a set of points? In *Proceedings of the GIScience Conference*. 81–98.

[19] Song Gao, Krzysztof Janowicz, and Helen Couclelis. 2017. Extracting urban functional regions from points of interest and human activities on location-based social networks. *Trans. GIS* 21, 3 (2017), 446–467.

[20] Amin Gheibi, Mansoor Davoodi, Ahmad Javad, Fatemeh Panahi, Mohammad M. Aghdam, Mohammad Asgaripour, and Ali Mohades. 2011. Polygonal shape reconstruction in the plane. *IET Comput. Vision* 5, 2 (2011), 97.

[21] Ronald L. Graham. 1972. An efficient algorith for determining the convex hull of a finite planar set. *Info. Process. Lett.* 1, 4 (1972), 132–133.

[22] Yingjie Hu, Song Gao, Krzysztof Janowicz, Bailang Yu, Wenwen Li, and Sathya Prasad. 2015. Extracting and understanding urban areas of interest using geotagged photos. *Comput. Environ. Urban Syst.* 54 (2015), 240–254.

[23] Ray A. Jarvis. 1973. On the identification of the convex hull of a finite set of points in the plane. *Info. Process. Lett.* 2, 1 (1973), 18–21.

[24] Donggi Jung, Hyunwoo Park, Ryong Maeng, and Sangki Han. 2010. A geometric pattern-based method to build hierarchies of geo-referenced tags. In *Proceedings of the IEEE International Conference on Privacy, Security, Risk and Trust*. 546–551.

[25] Ivana Kolingerová and Borut Žalik. 2006. Reconstructing domain boundaries within a given set of points, using Delaunay triangulation. *Comput. Geosci.* 32, 9 (2006), 1310–1319.

[26] James A. Leonard, Mark A. Kramer, and Lyle H. Ungar. 1992. Using radial basis functions to approximate a function and its error bounds. *IEEE Trans. Neural Netw.* 3, 4 (1992), 624–627.

[27] Mingzhao Li, Zhifeng Bao, Farhana Choudhury, and Timos Sellis. 2019. Interactive visualization of urban areas of interest: A parameter-free and efficient footprint method. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*. 782–785.

[28] Mingzhao Li, Zhifeng Bao, Timos Sellis, Shi Yan, and Rui Zhang. 2018. HomeSeeker: A visual analytics system of real estate data. *J. Visual Lang. Comput.* 45 (2018), 1–16.
[29] Mingzhao Li, Farhana Choudhury, Zhifeng Bao, Hanan Samet, and Timos Sellis. 2018. ConcaveCubes: Supporting cluster-based geographical visualization in large data scale. In *Computer Graphics Forum*, Vol. 37. Wiley Online Library, 217–228.
[30] Wenwen Li, Tingyong Chen, Elizabeth A. Wentz, and Chao Fan. 2014. NMMI: A mass compactness measure for spatial pattern analysis of areal features. *Ann. Assoc. Amer. Geogr.* 104, 6 (2014), 1116–1133.
[31] Yvan Maillot, Bruno Adam, and Mahmoud Melkemi. 2010. Shape reconstruction from unorganized set of points image analysis and recognition. In *Proceedings of the Conference on Image Analysis and Recognition*. 274–283.
[32] Grant McKenzie, Krzysztof Janowicz, Song Gao, Jiue-An Yang, and Yingjie Hu. 2015. POI pulse: A multi-granular, semantic signature-based information observatory for the interactive visualization of big geosocial data. *Cartographica* 50, 2 (2015), 71–85.
[33] Mahmoud Melkemi. 1997. A-shapes of a finite point set. In *Proceedings of the Symposium on Computational Geometry*. 367–369.
[34] Mahmoud Melkemi and Mourad Djebali. 2000. Computing the shape of a planar points set. *Pattern Recogn.* 33, 9 (2000), 1423–1436.
[35] Subhasree Methirumangalath, Amal Dev Parakkat, and Ramanathan Muthuganapathy. 2015. A unified approach towards reconstruction of a planar point set. *Comput. Graph.* 51 (2015), 90–97.
[36] Adriano Moreira and Maribel Yasmina Santos. 2007. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. In *Proceedings of the International Conference on Computer Graphics Theory and Applications*. 61–68.
[37] Eli Packer, Peter Bak, Mikko Nikkilä, Valentin Polishchuk, and Harold J. Ship. 2013. Visual analytics for spatial clustering: Using a heuristic approach for guided exploration. *IEEE Trans. Visual. Comput. Graph.* 19, 12 (2013), 2179–88.
[38] Jin-Seo Park and Se-Jong Oh. 2012. A new concave hull algorithm and concaveness measure for n-dimensional datasets. *J. Info. Sci. Eng.* 28 (2012), 587–600.
[39] Jiju Peethambaran and Ramanathan Muthuganapathy. 2015. A non-parametric approach to shape reconstruction from planar point sets through Delaunay filtering. *CAD Comput. Aided Design* 62 (2015), 164–175.
[40] Melanie Pohl and Dirk Feldmann. 2016. Generating straight outlines of 2D point sets and holes using dominant directions or orthogonal projections. In *Proceedings of the Conference on Computer Vision*. 59–71.
[41] Evgheni Polisciuc, Ana Alves, and Penousal Machado. 2015. Understanding urban land use through the visualization of points of interest. In *Proceedings of the 4th Workshop on Vision and Language*. 51–59.
[42] Franco P. Preparata and Se June Hong. 1977. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM* 20, 2 (1977), 87–93.
[43] Franco P. Preparata and Michael I. Shamos. 2012. *Computational Geometry: An Introduction*. Springer Science & Business Media.
[44] John D. Radke. 1983. On the shape of a set of points. *IEEE Trans. Info. Theory* 29, 4 (1983), 551–559.
[45] Martin Raubal and Stephan Winter. 2002. Enriching wayfinding instructions with local landmarks. In *Proceedings of the International Conference on Geographic Information Science*. 243–259.
[46] Gregory J. E. Rawlins and Derick Wood. 1987. Optimal computation of finitely oriented convex hulls. *Info. Comput.* 72, 2 (1987), 150–166.
[47] Emil Rosén, Emil Jansson, and Michelle Brundin. 2014. *Implementation of a Fast and Efficient Concave Hull Algorithm*. Technical Report. Uppsala University. 1–34.
[48] A. Saalima and Dimple A. Shajahan. 2016. Shape reconstruction by analysing all the inner holes. In *Proceedings of the International Conference on Emerging Technological Trends*. 1–6.
[49] Hanan Samet. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.
[50] Raimund Seidel. 1995. The upper bound theorem for polytopes: An easy proof of its asymptotic version. *Comput. Geom.* 5, 2 (1995), 115–116.
[51] Di Weng, Heming Zhu, Jie Bao, Yu Zheng, and Yingcai Wu. 2018. Homefinder revisited: Finding ideal homes with reachability-centric multi-criteria decision making. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–12.
[52] Chenyi Xia, Wynne Hsu, Mong Li Lee, and Beng Chin Ooi. 2006. BORDER: Efficient computation of boundary points. *IEEE Trans. Knowl. Data Eng.* 18, 3 (2006), 289–303.
[53] Hanfa Xing, Yuan Meng, Dongyang Hou, Jie Song, and Haibin Xu. 2017. Employing crowdsourced geographic information to classify land cover with spatial clustering and topic model. *Remote Sens.* 9, 602 (2017), 1–20.
[54] Junyi Xu, Yanping Feng, Zuoya Zheng, and Xihong Qing. 2010. A concave hull algorithm for scattered data and its applications. In *Proceedings of the International Congress on Image and Signal Processing*, Vol. 5. 2430–2433.

[55] Xu Zhong and Matt Duckham. 2016. Characterizing the shapes of noisy, non-uniform, and disconnected point clusters in the plane. *Comput. Environ. Urban Syst.* 57 (2016), 48–58.
[56] Xu Zhong and Matt Duckham. 2017. An efficient incremental algorithm for generating the characteristic shape of a dynamic set of points in the plane. *Geogr. Info. Sci.* 31, 3 (2017), 569–590.
[57] Jie Zhu, Yizhong Sun, and Yueyong Pang. 2017. A density based algorithm to detect cavities and holes from planar points. *Comput. Geosci.* 109, 1 (2017), 178–193.

Received November 2019; revised September 2020; accepted October 2020