Interactive Visualization of Urban Areas of Interest: a Parameter-free and Efficient Footprint Method

Mingzhao Li, Zhifeng Bao, Farhana Choudhury
RMIT University, Australia

ABSTRACT
Understanding urban areas of interest (AOIs) is essential to decision making in various urban planning and exploration tasks. Such AOIs can be computed based on the geographic points that satisfy the user query. In this demo, we present an interactive visualization system of urban AOIs, supported by a parameter-free and efficient footprint method called AOI-shapes. Compared to state-of-the-art footprint methods, the proposed AOI-shapes (i) is parameter-free, (ii) is able to recognize multiple regions/outliers, (iii) can detect inner holes, and (iv) supports the incremental method. We demonstrate the effectiveness and efficiency of the proposed AOI-shapes based on a real-world real estate dataset in Australia. A preliminary version of the online demo can be accessed at http://aoishapes.com/.

ACM Reference Format:

1 INTRODUCTION
An urban area of interest (AOI) refers to the area within an urban environment which attracts people’s attention [2]. Understanding urban AOIs can assist various urban data exploration tasks such as house seeking, and decision making for urban planning such as facility deployment of petrol stations, supermarkets, and general practitioners.

Taking house seeking as an example, location is a key factor, based on which various AOIs can be defined by users to cater for their respective concerns/interest. A user may want to find a region to live which is reachable from the nearest train station, supermarket and one of the top-10 public/private schools by at most 15-minute walk. The region that satisfies such a user need can be considered as a user-defined AOI.

AOIs could be represented as either polygon-based regions or individual points (e.g., real estate properties). Previous studies have shown that the polygon-based representation performs better from both cognitive and computational perspectives [2]. However, unlike most well-defined administrative districts (e.g., suburbs, cities), the region (shape) of an urban AOI could be vague [2] 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 POIs (points of interest) that satisfy the user query (e.g., in Example 1, the POIs are those real estate properties (Fig. 1(a)) that satisfy the user query). Different footprint algorithms have been proposed, including concave hulls [6], \( \chi \)-shapes [1], etc. However, footprints from a definite set of points are usually not unique, i.e., the constructed footprints can be different in different algorithms and/or different parameters; and there is no consensus on the effectiveness of the footprint algorithms.

We summarize the problem of existing footprint algorithms in Table 1 (a more complete summary can be found in our research paper [3]). We find that, the state-of-the-art footprint algorithms suffer from at least one of the following drawbacks: (1) additional parameter tuning is needed from the user; (2) multiple regions and/or outliers cannot be recognized; (3) inner holes cannot be detected; and (4) incremental generation of the footprint cannot be easily supported. The following example highlights the first drawback of existing methods (all other drawbacks and how we address them will be illustrated in Section 4).

Example 1. We generate footprints for those properties in Fig. 1(a) using \( \chi \)-shapes [9]. The algorithm first constructs a convex hull, and then executes a “digging” process (which progressively replaces the longest boundary edge with two close inner edges) to get a concave hull (i.e., footprint). The only parameter is a length-related threshold to decide whether each boundary edge is too “long” or not. We use two different parameters to generate the footprints, and present both the results on Google Maps (Fig. 1(b & c)). The results illustrate the first drawback of existing footprint algorithms: a smaller threshold might make the result too “concave” for users to recognize the region (Fig. 1(b)), and a larger threshold may cause the region too “big” to characterize the shape of the region (Fig. 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.

### Table 1: A summary of footprint methods

<table>
<thead>
<tr>
<th>Footprint Method</th>
<th>Holes</th>
<th>Outliers</th>
<th>Multiple Regions</th>
<th>Parameter-free</th>
<th>Incremental Method</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concave Hull [6]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>Methirumangalath et al. [5]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>(Incremental) ( \chi )-shapes [1, 9]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td></td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>( \chi )-outlines [8]</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td>✔️</td>
<td></td>
<td>O(nlogn)</td>
</tr>
<tr>
<td>The proposed AOI-shapes</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
Different from other footprint-related applications, in our application, an AOI is generated by a user query, which separates the global data points ($D$) into two sets of points: those that satisfy the user query ($P$) and those do not ($\overline{P}$). 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, named AOI-shapes, to compute the region boundaries of the user-defined urban AOI based on $P$ and $\overline{P}$. The details of how we address the problems of existing footprint methods can be found in Section 4.

We have implemented an online system\footnote{http://aoishapes.com} for visualizing AOIs which is supported by our proposed AOI-shapes. Besides, we have implemented various existing footprint methods (Table 1) for user to compare, evaluate and find their preferred choices.

## 2 VISUALIZATION DESIGN

Recall Example 1, each user query results in a user-defined urban AOI, and such AOI will change when the user refines his/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 needs to be updated without latency. To this end, we use a composite visualization design to visualize user-defined AOIs. As shown in Fig. 2, our design includes the following two coordinated views.

- **Multidimensional view.** As shown in Fig. 2(a), the 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. Suppose that a user selects $m$ attributes, then $m$ coordinated histograms will be initialized to present the data distribution on each dimension selected. Users can filter out part of the data by directly brushing on top of a certain histogram. After the user filters out the data on one dimension, all histograms of the remaining dimensions will be refreshed accordingly.

- **Google maps view.** We adopt the most straightforward polygonal representation to visualize the urban AOI. As shown in Fig. 2(b), an AOI can include multiple regions, and each region is visualized as a polygon on the Google maps. It is worth noting that, holes could exist in real-life applications, and such holes are visualized as empty areas inside the polygon (Fig. 2(b)).

After users filter out part of the data, 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. The first step can be effectively implemented using relational database management system (RDBMS). The last step is not a problem if we only need to draw a limited number of regions on the Google maps. We will solve the bottleneck of the visualization system (the second step) in the next section by proposing an effective and efficient footprint method, AOI-shapes.

## 3 THE AOI-SHAPES ALGORITHM

In this section, we propose an algorithm called AOI-shapes to overcome the drawbacks of existing footprint methods. Specifically, it is (i) parameter-free: users do not need to tune any parameters for the AOI, (ii) separable: both the data points that satisfy the user query and those that do not are taken into account in AOI generation, and (iii) able to recognize multiple regions, outliers, and holes.

### 3.1 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 urban AOI), and $\overline{P} = D \setminus P$ be the points outside the user-defined urban AOI. Our goal is to compute the boundary of the regions that are formed by the points in $P$, including outliers. Figure 3 illustrates different labels of points, edges, and regions which will be used in our algorithm.
Figure 3: Different types of points, edges and regions.

The proposed AOI-shapes algorithm constructs the AOI based on the Delaunay triangulation [7] of D. It can be implemented efficiently with the DCEL (Doubly Connected Edge List [7]) data structure. DCEL links three sets of records, i.e., vertices (points), darts (edges), and faces (triangles). Specifically, each edge is represented as two ‘darts’ (half-edges) with a counter-clockwise direction.

3.2 Constructing the AOI-shapes

Algorithm 1 presents the workflow of AOI-shapes. It includes two parts: an offline pre-processing part and a query processing part. At the pre-processing part, we build a Delaunay triangulation (DT) based on the whole dataset (D) (Lines 1.3-1.4), and then construct the DCEL structure to represent the triangulation result (Line 1.5). At the query-processing step, the set of points (P) is obtained based on the user query. We initialize the lists of boundary edges (DE), dangling edges (BE), which are edges that connect two inside points (p ∈ P) but are not part of the final boundaries, e.g., P5P8, P5P14 and P5P16 in Fig. 3, and outlier points (OP); for each point (p) satisfying the user query (p ∈ P), we label it as inside (Lines 1.6-1.7). Then the following steps are processed:

Step 1: Finding boundary edges. To find all the boundary edges (darts), we traverse all the points that satisfy the user query (P) (Lines 1.8-1.29). For each point p (p ∈ P), if it does not connect to any other points in P, it becomes an outlier (Lines 1.27-1.28). For dart dt (which starts at p and whose end point is also in P), suppose that the twin dart of dt is dt’. We label dt as (1) a boundary edge if only one of the opposite points of dt and dt’ is in P, or (2) a dangling dart if both the opposite points of dt and dt’ are in P, or (iii) otherwise an inner dart.

Step 2: A “digging” process. Note that, region boundaries are formed at the process where we exclude those points in P. If the set of P is very small, or at the worst case P is ∅, then the resulting shape from the baseline method could be too “big” to characterize the region boundary. Here, we introduce a parameter-free “digging” process (Lines 1.28-1.31). We consider a boundary dart is too “long” if the opposite angle of the dart is obtuse, and we will replace the dart with the other two darts in the same triangle. If a new dart is originally a boundary dart, then it becomes a dangling dart.

Step 3: Recognizing different regions from boundary edges. For dangling and outlier points, they are all treated as outliers (individual points) in the final visualization (Line 1.32). For boundary edges, they will form multiple disclosed regions. We can recognize multiple regions by sequentially connecting boundary edges via the intersection points. In the function Connect_Edges_via_Points (Line 1.33), we traverse all the boundary edges stored in BE. More details about the function can be found in our research paper [3].

Algorithm 1: AOI-shapes(D, P)

11 Input: A set D of geographical points, and a set P which are within the user-defined urban AOI (i.e., satisfy the user query). Each d ∈ D has attributes [latitude, longitude, label], and d.label = outside.
12 Output: A list of regions (Regions) and a list of outliers (Outliers)
13 Delaunay_Triangulation(D); ▷ Start of pre-processing
14 St ← Triangle set of the DT result;
15 Build the DCEL structure based on St;
16 BE, DE, OP ← ∅; ▷ Start of query processing
17 for each point p ∈ P do
18    Dartsp ← DARTS_START_WITH(p);
19    outlier_flag ← True;
20    for each dart dt ∈ Dartsp do
21        if END_VERTEX(dt).label = inside then
22            outlier_flag = False;
23            dt’ ← TWIN_DART(dt);
24            if OPPOSITE_VERTEX(dt).label = inside then
25                dt.label = inner;
26            else
27                dt.label = in_boundary;
28                Add dt to BE;
29        else
30            if OPPOSITE_VERTEX(dt’).label = inside then
31                dt.label = out_boundary;
32            else
33                dt.label = dangling;
34                Add dt to DE;
35            if outlier_flag = True then
36                Add p to OP;
37    for each edge e ∈ BE do
38        if e.obtuse = True then
39            BE ← Digging(e, BE);
40        Outliers → GET_OUTLIERS(OP, DE);
41        Regions → CONNECT_EDGES_VIA_POINTS(BE);
42        for each r in Regions do
43            if DIRECTION(r) = clockwise then
44                r.label ← hole;
45        return Regions, Outliers;
46
47 Step 4: Detecting holes. Until now, we have recognized multiple regions, each of which is presented as a single loop (formed by the boundary edges). For example, Fig. 3 will be recognized as three loop-based regions (P1P2P3P4, P5P6P7P8 and P9P10P11P12). It is worth noting that, the loop might also form a negative region (hole). For example, the loop P9P10P11P12 in Fig. 3 is a hole. Since darts in DCEL have directions, the non-hole regions should have the same direction as the DCEL face (counter-clockwise), and hole regions will have the opposite direction (clockwise). Lines 1.34-1.36 in Algorithm 1 present how we recognize each hole region based on the direction of those boundary darts that form the region.

The time complexity of the state-of-the-art footprint algorithms [1, 8] depends on the process of building the DT O(nlogn). Our AOI-shapes improves the efficiency by building a global DT at the offline pre-processing part. Each time the user refines the user query, the AOI-shapes will find and update those affected points, and then go through the four steps (only for those new and deleted points) in the query-processing part O(n). More details about our incremental algorithms and the complexity analysis can be found in our research paper [3].
We demonstrate our AOI-shapes (available online\(^1\)) using Australia’s real estate data \([4]\). The data lists 1.42 million properties sold between 2007 and 2016. Each property is associated with 72 dimensions including its geospatial information (latitude and longitude), categorical dimensions (e.g., property type), and numerical dimensions (e.g., price, distance to the nearest train station, supermarket, etc.). In particular, we first demonstrate the AOI visualization system based on a real-life scenario. Then we illustrate the effectiveness and efficiency of the proposed AOI-shapes by comparing it with the state-of-the-art footprint methods.

### 4.1 Demo scenario: finding houses

Our demonstration is based on the scenario that a user, John, tries to find a place to live in Melbourne. Our system helps him explore and understand different urban areas to cater for his own preferences on various dimensions before he tries to find a particular house.

**Example 2.** John first plays with the system to check where he could buy a house within a budget of one million dollars. He gradually filters the properties on the “price” dimension (Fig. 2(a)), and the regions on the map view (Fig. 2(b)) keep changing as he moves the slider. He finds that most of the regions in North and West Melbourne satisfy his query (i.e., he could afford houses in those regions); but if he wants to buy houses in the South and East Melbourne, he might need to live quite far away from the city (where he works).

**Example 3.** Living far away from the city is not a big problem since John could take the train. He then issues a new query and selects the region within 15-minute walk to the nearest train station (Fig. 1(d)). There are a large number of regions satisfying his requirement, so he refines his requirements by limiting the travel time to the city within half an hour (Fig. 4(a-1)). As he keeps checking the changes of the region, he notices that the dimension “school_rank” does not follow a Normal distribution (Fig. 4(a-2)). Considering his child’s education, he continues to refine his query and selects those regions within a top-20% secondary schools (Fig. 4(b)). After that, he applies his budget-limitation (1 million dollars) again and checks those regions in Fig. 4(c). Finally, John checks the individual properties within the urban area of his interest (Fig. 4(d)).

### 4.2 Quality and efficiency of AOI-shapes

To evaluate the proposed AOI-shapes and compare it with the state-of-the-art footprint algorithms, we implemented four baseline methods (Table 1). In our demo system\(^1\), 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 baseline method (e.g., \(\chi\)-shapes \([1]\)) needs one.

**Quality.** Beyond parameter-free, AOI-shapes also outperforms the state-of-the-art methods in term of the quality of the generated footprints: the AOI-shapes can recognize multiple regions, outliers, and inner holes. For example, as shown in Fig. 1(d), the AOI-shapes recognizes multiple disconnected regions around the train stations in West Melbourne (as highlighted in red), while those regions are unexpectedly connected at the result of \(\chi\)-shapes (Fig. 1(b & c)); also, the AOI-shapes detects the hole regions in the middle of two train lines (as highlighted in blue in Fig. 1(d)) while the baseline methods all fail to do so.

**Efficiency.** Among all the baseline methods, \(\chi\)-shapes \([1]\) is the most efficient one. Using our real estate data as an example, when the size of \(P\) is 100k, it takes about 0.5s and 0.15s to generate the \(\chi\)-shapes and AOI-shapes, respectively. During the demonstration, we will encourage the audiences to try the system using different methods (and in different parameters), so they can compare the latency. More experiments about the efficiency comparison can be found in our research paper \([3]\).

**Acknowledgements.** This work was partially supported by ARC DP170102726, DP180102050, and NSFC 61728204, 91646204. Zhifeng Bao is supported by a Google faculty award.

### REFERENCES


