An efficient incremental algorithm for generating the characteristic shape of a dynamic set of points in the plane

Article *in* International Journal of Geographical Information Science · August 2016

DOI: 10.1080/13658816.2016.1216995

CITATION
1

READS
67

2 authors, including:

Xu Zhong
IBM Research
20 PUBLICATIONS **112** CITATIONS

SEE PROFILE

International Journal of Geographical Information Science
Vol. 31, No. 3, July 2016, 1–26

RESEARCH ARTICLE

An efficient incremental algorithm for generating the characteristic shape of a dynamic set of points in the plane

Xu Zhong[a*] and Matt Duckham[b]

[a]*IBM Research Australia, Victoria, Australia*;
[b]*Department of Mathematical and Geospatial Sciences, School of Science, RMIT University, Victoria 3001, Australia*

()

Several algorithms have been proposed to generate a polygonal "footprint" to characterize the shape of a set of points in the plane. One widely-used type of footprint is the χ-shape. Based on the Delaunay triangulation, χ-shapes guaranteed to be simple (Jordan) polygons. This paper presents for the first time an incremental χ-shape algorithm, capable of processing point data streams. Our incremental χ-shape algorithm allows both insertion and deletion operations, and can handle streaming individual points and multiple point sets. The experimental results demonstrated that the incremental algorithm is significantly more efficient than the existing, batch χ-shape algorithm for processing a wide variety of point data streams.

Keywords: incremental algorithm; pattern recognition; footprint; non-convex; χ-shape

1. Introduction

The χ-shape (Duckham *et al.* 2008) algorithm is a widely-used, efficient method for creating potentially non-convex "footprints" that characterize the shape of a set of points in a plane. The χ-shape algorithm is different from other footprints algorithms in that it is guaranteed to generate simple, regular (Jordan) polygons, i.e., without holes, isolated points, lines, or disconnected components.

The χ-shape algorithm starts with the Delaunay triangulation (DT) of a set of input points P and iteratively removes the longest exterior edge of the triangulation that does not cause irregularity. However, because of this method of construction, χ-shapes are more difficult than other footprint algorithms to calculate incrementally. Once an change occurs in the sequence of removed edges, it can lead to a cascade effect on the removal order of exterior edges. A change may cause some previously non-removable edges to

*Corresponding author. Email: peter.zhong@au1.ibm.com

ISSN: 1365-8816 print/ISSN 1362-3087 online

DOI: 10.1080/13658816.2016.1216995
http://www.informaworld.com

become removable, and some previously removed edges to be become non-removable and retained in the new χ-shape. In turn, these changes may lead to further changes in exterior edges. As a consequence of this "edge cascade," even though incremental DT algorithms have been well-established (Guibas *et al.* 1992, Lischinski 1994), calculating χ-shape incrementally remains a challenge.

The main contribution of this paper is to solve the "edge cascade" problem in the χ-shape algorithm, and to enable incremental generation of χ-shapes. A tree structure is proposed to represent the χ-shape. Then an algorithm is developed to incrementally generate the updated χ-shape by making the minimum number of modifications to the tree. By only making the necessary changes, which require far fewer operations than recomputing the χ-shape for the entire data set, the incremental algorithm affords significantly improved computational efficiency. The performance of the incremental χ-shape algorithm is evaluated empirically. The results demonstrate the valuable properties of the incremental χ-shape algorithm to guide usage.

Following a review of the background literature in Section 2, the incremental χ-shape algorithm is described in Section 3. Empirical evaluation experiments are designed in Section 4 to investigate the efficacy of the incremental χ-shape algorithm in different circumstances. Section 5 discusses the highlights of the incremental χ-shape algorithm. Finally, Section 6 provides the final conclusions.

2. Background

2.1. *Motivation*

Data sets of spatial points in a plane $P \subset \mathbb{R}^2$ are common in many real-life applications, such as geo-reference social media data (Sakaki *et al.* 2010, Vieweg *et al.* 2010, Yates and Paquette 2011, Crooks *et al.* 2013), emergency call data (Zhong *et al.* 2016a), volunteered geographic information (Goodchild and Glennon 2010, Poser and Dransch 2010, Schade *et al.* 2013), and species distribution data (Busby 1991, Guo *et al.* 2005, Leathwick *et al.* 2005, Phillips *et al.* 2006). The points are often observations of spatial regions. Hence many algorithms have been proposed to reconstruct the spatial regions from the points by constructing polygonal shapes that characterize the distribution of the points. Such regions are called "footprints" of P. Non-convex footprints were proposed to handle cases where the distribution of points is markedly non-convex. In these cases, there can be no single "best" footprint. Rather, the accuracy of a footprint may depend on the specific application, or on human cognition and preference. Hence there is a variety of non-convex footprint algorithms which can be parameterized to generate the desired footprint.

In many cases, the point set P can be large and highly dynamic. For example, geo-reference social media data is extremely plentiful and generated rapidly (Manovich 2011, Tufekci 2014). Data warehouses are increasingly large and need to process frequent insertion and deletion operations (Ester *et al.* 1998). Spatiotemporal data streams processing often needs to aggregate data tuples in a window (Chandrasekaran and Franklin 2002, Li *et al.* 2005). The high-volume and high-velocity nature of spatiotemporal data streams implies the data tuples in the window can get massive and change rapidly (Babcock *et al.* 2002, Motwani *et al.* 2003, Nittel *et al.* 2004, Mokbel *et al.* 2005). In these cases where a large dynamic set of data frequently includes or evicts a relatively small set of data, algorithm efficiency can be dramatically improved if incremental computation is possible. Incremental computation can alleviate the need to recompute the entire solution at each iteration. Hence, it should be no surprise that

incremental computation is widely used to mine data warehouses (Ester *et al.* 1998) and process spatiotemporal data streams (Acharya and Lee 2014, Mokbel *et al.* 2005, Ünal *et al.* 2006, Zhang *et al.* 2008, 2014, Zhong *et al.* 2016b).

2.2. *Edge cascade*

The core challenge faced in designing an incremental χ-shape algorithm is the "edge cascade," where new points can cause previously retained edges to be removed, or previously removed edges to be retained. The edge cascade problem also exists for some other non-convex footprint algorithms. Peethambaran and Muthuganapathy (2015) proposed a non-parametric approach to non-convex footprints reconstruction. The algorithm first filters the DT of the input points in a similar fashion to χ-shapes and then reconstructs holes if detected. In the filtering step, the circumradii of the triangles (instead of edge length in χ-shapes) are used to determine the order of edge removal under the regularity constraint. Hence this approach has the same edge cascade problem with the χ-shape algorithm.

The recursive voids method (Şeref and Zobel 2013) finds empty regions within an area that encloses all input points. The empty regions are detected recursively and stored in a tree structure. Changes in the input points may influence the detection of empty regions in the current iteration and sequentially affect the later recursions, which can lead to the edge cascade.

Garai and Chaudhuri (1999) proposed a "split-isolate-merge" procedure to construct non-convex footprints. The split step successively inserts extra edges to carve off pieces of the convex hull of P, based on the k-nearest-neighbor (kNN) of the vertices of the current polygon. Then the isolation step separates the components in the pattern of P, again based on kNN of the vertices of the current polygon. Finally, the merge step smooths the zig-zagged polygon formed in the split and isolation step, according to the angle between the two edges incident to each vertex and the area of the triangle formed by the vertices on those two edges. When the set P is changed, the split and isolate steps can be conducted incrementally using incremental kNN search. However, the merge step can lead to an edge cascade, because the order in which angles of the polygon are smoothed matters.

The kNN based method proposed by Moreira and Santos (2007) may also lead to edge cascades. The kNN based method generalizes the gift-wrapping convex hull algorithm (Jarvis 1973). At each iteration, the kNN-based algorithm finds the next point from the kNN of current points, instead of processing the entirety of P used in the gift-wrapping algorithm. Each subsequent point produces the largest right-hand turn from the current point without resulting in self-intersection. Although the kNN of the points can be searched incrementally, the selection of the next edge depends on the current edge because of the no-self-intersection constraint. Hence when an edge is changed due to the updates of P, sequential changes may happen to the later selection of edges.

Parker and Downs (2013) proposed a footprint algorithm based on fuzzy-neighborhood clustering. The input points are clustered using the Fuzzy-Neighborhood (FN)-DBSCAN algorithm. The final footprint is formed by the union of the footprint of each cluster, generated using a contouring method. The edge cascade happens in the clustering process, which has been solved by Ester *et al.* (1998).

Unlike χ-shapes, some non-convex footprint algorithms do not lead to edge cascade problems. The α-shape (Edelsbrunner *et al.* 1983) is another DT-based footprint. A new DT can be calculated incrementally following insertion or deletion, and the new

α-shape can be obtained directly by checking the circumradius of triangles that appear or disappear.

The $\mathscr{A}$-shape (Melkemi 1997, Melkemi and Djebali 2000) is a Voronoi diagram (VD) based footprint. The $\mathscr{A}$-shape firstly takes a set of auxiliary negative points P_a (points that must lie outside the footprint) and calculates the VD of $P \cup P_a$. Then any pair of points $p, q \in P$ whose Voronoi cells are adjacent to each other and to a common Voronoi cell of a point in P_a are connected, forming the $\mathscr{A}$-shape. Similar to the α-shape, once the new VD is calculated incrementally, the new $\mathscr{A}$-shape can be determined directly by checking if the changes in the Voronoi cells create or destroy any new connections between points. The same principle can also be applied to the DT-based footprint proposed by Arampatzis *et al.* (2006) and the VD based footprint proposed by Alani *et al.* (2001).

Footprints such as the covering disks method (Galton and Duckham 2006), s-shape and r-shape (Chaudhuri *et al.* 1997) are constructed by assigning an influence region to each point and integrating the influenced regions of all points to form the footprint. When new points join the data set or some points are removed from the data set, an incremental algorithm needs only consider how the regions associated with these points affect the footprint. For example, for the covering disks method, the new footprint can be obtained by three simple steps: 1) calculating the intersection between the regions that are unassociated with the deleted points and the regions associated with the deleted points; 2) calculating the intersection between the old footprint and the complement of the regions assigned to the deleted points; and 3) calculating the union of the shape obtained in step 2 with the regions obtained in step 1 and the regions assigned to the inserted points.

Nevertheless, while footprint algorithms such as α-shapes and $\mathscr{A}$-shapes can be constructed incrementally, these algorithms don't offer the regularity guarantees of χ-shapes. Hence, in this paper we solve the edge cascade problem with a tree-based data structure and associated algorithm to enable efficient, incremental construction of χ-shapes.

3. Method

Our new incremental χ-shape algorithm is described in this section. In essence, the χ-shape is represented as a tree data structure. Then based on the changes in the DT caused by insertion or deletion of points, the incremental χ-shape algorithm modifies the tree to calculate the new χ-shape more efficiently.

3.1. *Preliminaries*

Given a point set P and its χ-shape, our incremental algorithm can efficiently compute the χ-shape for the insertion or deletion of a (possibly singleton) set of points S. The following terms are helpful in explaining our algorithm, also depicted in Figure 9.

Definition 3.1 The **adjacent triangles** of an edge e in a triangulation is the set of one (if e is an exterior edge) or two (if e is an interior edge) triangles in the triangulation that are incident with (i.e., adjacent to) e.

Definition 3.2 The **arms** of an exterior edge e in a triangulation are the other two edges incident with the (unique) adjacent triangle of e.

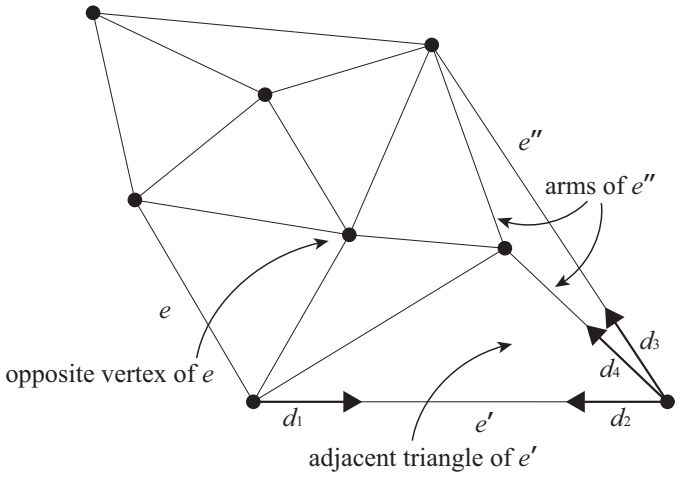


Figure 1.: Examples of adjacent triangle, arms, opposite vertex, and darts in a triangulation.

Definition 3.3 The **opposite vertex** of an exterior edge e in a triangulation is the (unique) third vertex of the (unique) adjacent triangle of e, i.e., such that the opposite vertex is not incident with e.

The incremental χ-shape algorithm proposed in this paper can be implemented efficiently using the doubly connected edge list (DCEL) data structure (De Berg *et al.* 2000), widely used in computational geometry. In the DCEL, each edge is represented as a pair of half edges or "darts" (e.g., edge e' represented by darts $\{d_1, d_2\}$ in Figure 9). For each dart, we may efficiently retrieve the "twin" (opposite pair) of that dart, represented using the function θ_0 (e.g., $\theta_0 d_1 = d_2$ and $\theta_0 d_2 = d_1$ in Figure 9). Finally, the counterclockwise cyclic order of darts around a vertex is stored using the function θ_1, e.g., $\theta_1 d_2 = d_3$, $\theta_1 d_3 = d_4$, $\theta_1 d_4 = d_2$ in Figure 9. Note that we can combine functions θ_0 and θ_1 to move efficiently around the triangulation, e.g., $\theta_0\theta_1\theta_0\theta_1 d_1 = d_4$. Using the DCEL, the pairs, arms, opposite vertex of an exterior edge of a planar triangulation can be found in constant time.

3.2. *Representing the χ-shape algorithm as a tree*

The idea behind of the χ-shape algorithm is to remove exterior edges of the Delaunay triangulation (DT) of a set of input P in descending order of length, subject to a regularity constraint. The regularity constraint of χ-shapes requires that at all times the exterior edges of the triangulation form the boundary of a simple (Jordan) polygon (the χ-shape itself). Every time an exterior edge is marked for removal, the polygon formed by the new set of exterior triangulation edges is checked for topological irregularities to the polygon (i.e., namely isolated points, point connections, lines or disconnected components).

Based on this distinction between regular and irregular polygons formed by the exterior edges of a triangulation, we can define two types of exterior edges:

Definition 3.4 An exterior edge of a triangulation is termed **removable** if removing that edge would result in the exterior edges of the resulting triangulation forming an simple polygon. Otherwise an exterior edge of a triangulation is termed **non-removable**.

The removability of an exterior edge can be determined by checking if the opposite vertex of the edge is on the boundary of the triangulation. The "$v-boundary$" function

defined in (Duckham *et al.* 2008) returns true if the input vertex is a boundary vertex, and false otherwise. The $v-boundary$ function is initialized by assigning the boundary vertices of the DT with a true value and the internal vertices with a false value. Then when a removable exterior edge is removed from the triangulation, the opposite vertex of the exterior edge becomes a boundary vertex. Thus the $v-boundary$ function of the opposite vertex is set to be true. Maintaining this $v-boundary$ function enables the removability of an exterior edge to be determined in constant time.

The polygon formed by the exterior edges of the triangulation at any iteration of the χ-shape algorithm is guaranteed to be a simple polygon that contains all the input points. The algorithm ceases if there is no removable exterior edge longer than the length threshold λ. The exterior edges of the final triangulation form the χ-shape ($\boldsymbol{\chi}(P, \lambda)$) of P.

The order of edge removal and the relationship between an exterior edge and its arms are critical for the χ-shape algorithm. Thus it is necessary to find out a method to maintain these two pieces of information of $\boldsymbol{\chi}(P, \lambda)$ to calculate $\boldsymbol{\chi}(P \cup S, \lambda)$ or $\boldsymbol{\chi}(P \setminus S, \lambda)$ incrementally (where S is the possibly singleton set of points to be added to or removed from P). Note every time an exterior edge e is removed in the χ-shape algorithm, the two arms of e are exposed. According to this rule, the χ-shape algorithm can be represented as a tree structure. The root of the tree is a virtual "null" node. Each node except the root represents an edge removed in the χ-shape algorithm, along with the length and opposite vertex of that edge. The children of the root correspond to the initially exterior edges of $DT(P)$. Each internal node of the tree other than the root has exactly two children, which store the arms of the edge corresponding to that node. The leaves of the tree correspond to non-removable edges or edges shorter or equal to the length threshold λ, which form $\boldsymbol{\chi}(P, \lambda)$. This tree is referred as χ-*tree* ($\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$) in this paper. As shown in Algorithm **1**, the χ-tree can be constructed as the χ-shape algorithm scans the list of exterior edges. An auxiliary list ($O(P, \lambda)$) is also created, which saves the pointers to nodes of the χ-tree in the order of being scanned in the χ-shape algorithm. In addition, a new column is added to the DCEL, which stores the pointers to the tree-nodes that correspond to each dart. For darts not included in the χ-tree, this column is left empty. The function "$node(d)$" returns the value of this column for the dart d. For brevity we may also write $node(e)$ in place of $node(d)$, where e is an edge incident with dart d.

Figure 10(b) illustrates the χ-tree of an example χ-shape shown in Figure 10(a). The nodes of the tree are labelled with their corresponding triangulation edge labels. The leaves of the χ-tree correspond to the edges that form the χ-shape shown in Figure 10(a).

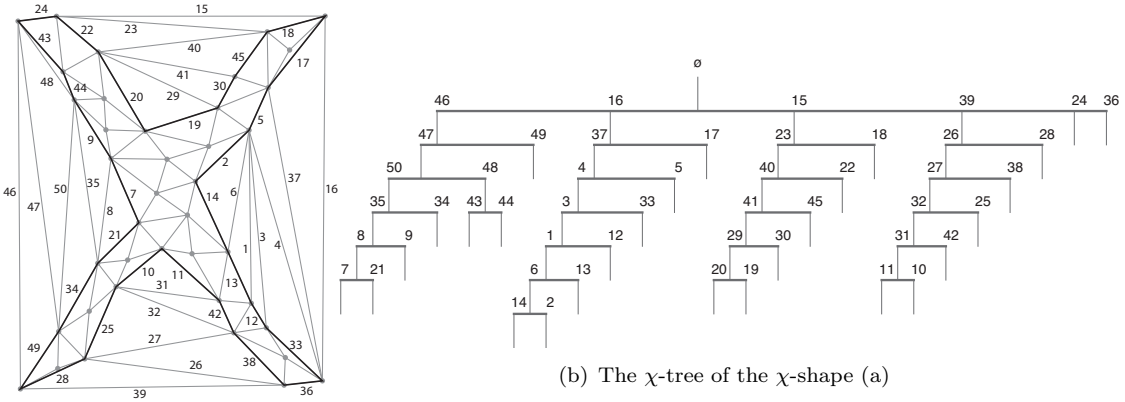
The DT for insertion ($DT(P \cup S)$) or deletion ($DT(P \setminus S)$) can be obtained incrementally from $DT(P)$ using established incremental DT algorithms. The next step is to find out the minimum modifications to the χ-tree to handle insertion or deletion incrementally and efficiently. Then the χ-shape after insertion or deletion can be simply extracted from the leaves of the modified χ-tree.

3.3. *Insertion*

Inserting a set of points S into P only affects the triangles of $DT(P)$ that are locally relevant to the points in S (Anglada 1997). Let $\bar{\Delta}_S$ be the sub-triangulation of $DT(P)$ that disappears as a result of inserting S, and Δ_S be the sub-triangulations of $DT(P \cup S)$ that are created by S. $\bar{\Delta}_S$ and Δ_S can be obtained using an incremental DT algorithm such as Guibas *et al.* (1992), Lischinski (1994). Figure 11 shows an example of inserting S into P and the resultant $\bar{\Delta}_S$ and Δ_S.

Algorithm 1 χ-shape algorithm with preparation for future insertion or deletion

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$ and one parameter $\lambda \in \mathbb{R}$
1: Construct the Delaunay triangulation $DT(P)$ of P
2: $\Delta \leftarrow DT(P)$
3: Construct the list B of exterior edges of $DT(P)$
4: Sort the list B in descending order of edge length
5: Initialize the $v-boundary$ function
6: Set the root (r) of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ to be {edge = $\emptyset$, oppositeVertex = $\emptyset$, length = $\emptyset$}
7: Construct the list of parent nodes (PN) for the elements in B
8: Set each element in PN to be r
9: $O(P, \lambda) \leftarrow \emptyset$
10: **while** B is not empty **do**
11: $e = \{d_1, d_2\} \leftarrow \text{pop}(B)$
12: $p \leftarrow \text{pop}(PN)$
13: $o \leftarrow$ opposite vertex of e in Δ
14: $N \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
15: Insert N in $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of p
16: $node(d_1) \leftarrow N$
17: $node(d_2) \leftarrow N$
18: Append N to $O(P, \lambda)$
19: **if** $||e|| > \lambda$ **and** $v-boundary(o) = false$ **then**
20: Remove e from Δ
21: $v-boundary(o) = true$
22: Insert the arms of e in Δ into B in order of edge length
23: Insert N into PN at the corresponding position of the arms of e in B
24: **end if**
25: **end while**
26: **Return** $\boldsymbol{\chi}(P, \lambda)$ formed by leaves of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$



(a) An example χ-shape

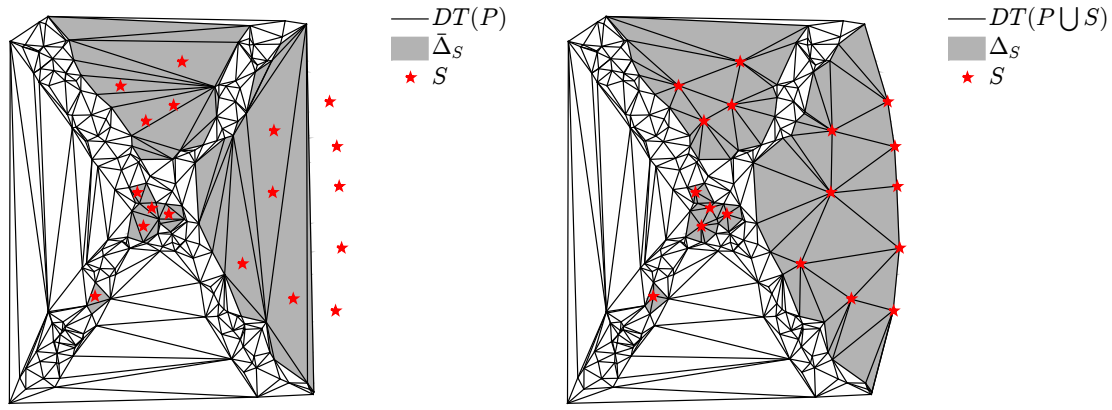
(b) The χ-tree of the χ-shape (a)

Figure 2.: The χ-tree (b) of an example χ-shape (a). Labeled nodes in the χ-tree (b) correspond to edges of the χ-shape (a). The leaves of the χ-tree form the χ-shape (thicker edges) shown in (a).

The incremental χ-shape for insertion is presented in Algorithm **2**. First, the triangles that disappear ($\bar{\Delta}_S$) are handled in lines 2–14. Next, the changes in the exterior edges of the DT are solved in lines 15–23. Finally, the nodes in the χ-tree are scanned in lines 24–58 to generate the χ-shape.

3.3.1. Handling triangles that disappear ($\bar{\Delta}_S$)

After inserting S, the triangles in $\bar{\Delta}_S$ are replaced with the new ones in Δ_S. The nodes of which the edge and opposite vertex are both on a triangle in $\bar{\Delta}_S$ will not exist in the new χ-tree. Let $N_d = \{N \mid N = node(e)$, e is on $\bar{\Delta}_S$, $N.oppositeVertex$ is on $\bar{\Delta}_S\}$ denote such nodes. For a node $N \in N_d$, if $N.edge$ is on the boundary of Δ_S, we only need to change $N.oppositeVertex$ to the opposite vertex of $N.edge$ in Δ_S. When the opposite



(a) Before insertion of S (b) After insertion of S

Figure 3.: Changes to DT caused by inserting a set of points S into P.

vertex of N is changed, the children of N (arms of $N.edge$) must be changed in the new χ-tree. Thus the children of N are disconnected from N if N is an internal node, which will be reconnected or deleted in the later steps (see Section 3.3.3). If $N.edge$ is not on the boundary of Δ_S, N needs to be deleted from the χ-tree and the auxiliary list of the χ-tree. The χ-tree can become a disconnected graph (a forest) after deleting such nodes.

3.3.2. Fixing exterior edges of DT

When S contains points outside the convex hull ($conv(P)$) of P, the exterior edges of DT can be changed after the insertion. The DT after the insertion has exterior edges which are incident to the points in S. Let E_S^b be such exterior edges. New nodes corresponding to these new exterior edges need to be added to the χ-tree as the children of the root of the χ-tree. We also need to determine the position of the newly added nodes in the auxiliary list of the χ-tree. A list W is created to save the newly created nodes waiting to be inserted into the auxiliary list of the χ-tree. The newly created nodes are inserted in W in descending order of the length of the nodes. The nodes in W are inserted into the auxiliary list of the χ-tree in the later step (see lines 26–28 in Algorithm **2**).

In addition, some exterior edges of $DT(P)$ may become internal in $DT(P \cup S)$ (covered by the exterior edges of $DT(P \cup S)$ that are incident to points in S). The nodes corresponding to such previous exterior edges are firstly disconnected from the null root of the χ-tree, since the children of the null root of the χ-tree must correspond to the exterior edges of DT. The resultant disconnected component will be reconnected or deleted in the later steps (see Section 3.3.3).

$N_d = \emptyset$ and $E_S^b = \emptyset$ implies that the points in S do not affect the χ-tree at all. In this case, the χ-tree $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda)$ and its auxiliary list will be identical with $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and the auxiliary list of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, respectively. Hence the incremental χ-shape algorithm stops in this situation. Otherwise, the nodes in the χ-tree are scanned in the next step.

3.3.3. Scanning the χ-tree

After the previous initial changes to the χ-tree and its auxiliary list, the nodes of the χ-tree can be scanned in the order given by the auxiliary list. Recall that some nodes of the χ-tree in W are waiting to be inserted into the auxiliary list of the χ-tree. When a node N is scanned in an iteration, it is compared with the head of W ($H = head(W)$). Note the χ-shape algorithm always process the longest exterior edge. Hence if $H.length > N.length$, H is inserted into the auxiliary list of the χ-tree before N and removed from

Algorithm 2 Incremental χ-shape algorithm (insertion)

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$, a finite set of two-dimensional points $S \notin P$ to be inserted, one parameter $\lambda \in \mathbb{R}$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$

Calculate $DT(P \cup S)$ incrementally from $DT(P)$ and obtain $\bar{\Delta}_S$, Δ_S and E_S^b
$N_d \leftarrow \{N \mid N = node(e)$, e is on $\bar{\Delta}_S$, $N.oppositeVertex$ is on $\bar{\Delta}_S\}$
if $N_d = \emptyset$ **and** $E_S^b = \emptyset$ **then**
    Return $DT(P \cup S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \cup S, \lambda) \leftarrow O(P, \lambda)$
end if
for all $N \in N_d$ **do**
    if $N.edge$ is on the boundary of Δ_s **then**
        $N.oppositeVertex \leftarrow$ opposite vertex of $N.edge$ in Δ_S
        Disconnect the children of N from N if N is an internal node
    else
        $node(N.edge) \leftarrow null$
        Delete N from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
    end if
end for
$W \leftarrow \emptyset$
$E_{\Delta_S}^b \leftarrow \{e \mid e \in \Delta_s, e$ is an internal edge of $DT(P \cup S)$ and an exterior edge of $DT(P)\}$
Disconnect $node(E_{\Delta_S}^b)$ from the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$
for all $e \in E_S^b$ **do**
    $o \leftarrow$ opposite vertex of e in $DT(P \cup S)$
    $N \leftarrow \{$edge $= e$, oppositeVertex $= o$, length $= ||e||\}$
    Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
    $node(e) \leftarrow$ N
end for
$N \leftarrow head(O)$
while $N.next \neq null$ **do**
    if $W \neq \emptyset$ **and** $N.length < head(W).length$ **then**
        Insert $head(W)$ into $O(P, \lambda)$ before N
        $N \leftarrow pop(W)$
    end if
    if $N.parent = null$ **then**
        $N_b \leftarrow N$ and its descendants
        $node(N_b.edge) \leftarrow null$
        Delete N_b from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
    end if
    if $N.length > \lambda$ **then**
        if N is an internal node **and** $v{-}boundary(N.oppositeVertex) = false$ **then**
            $v{-}boundary(N.oppositeVertex) = true$
        else if N is an internal node **and** $v{-}boundary(N.oppositeVertex) = true$ **then**
            $N_b \leftarrow N$ and its descendants
            $node(N_b.edge) \leftarrow null$
            Delete N_b from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
        else if N is a leaf **and** $v{-}boundary(N.oppositeVertex) = false$ **then**
            $v{-}boundary(N.oppositeVertex) = true$
            $A \leftarrow$ arms of $N.edge$ using Equation 1
            for all $e \in A$ **do**
                if $node(e) \neq null$ **and** $node(e).oppositeVertex$ is not on $N.edge$ **then**
                    Connect $node(e)$ to N as a child of N
                else
                    $o \leftarrow$ opposite vertex of e using Eq 2
                    $N' \leftarrow \{$edge $= e$, oppositeVertex $=$ o, length $= ||e||\}$
                    Insert N' into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of N and into W in descending order of length
                    $node(e) \leftarrow N'$
                end if
            end for
        end if
    end if
    $N \leftarrow N.next$
end while
Return $DT(P \cup S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \cup S, \lambda) \leftarrow O(P, \lambda)$

W. And in this iteration H is scanned instead of N.

The initial changes cause the χ-tree to become a disconnected graph. Some of the connected components will be reconnected to the χ-tree in the Case 4 below. If a node N being scanned in the current iteration is a root of one of the connected components

($N.Parent = null$), it means the connected component has not been reconnected to the χ-tree. The nodes in the connected component are deleted from the χ-tree and its auxiliary list. Then the next node in the auxiliary list of the χ-tree is scanned. The edges corresponding to these deleted nodes either rejoin the χ-tree later or are no longer contained in the χ-tree. If $N.Parent \neq null$ and $N.length \leq \lambda$, N must be a leaf of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and will still be a leaf in $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda)$. Hence no further processing is needed. If $N.Parent \neq null$ and $N.length > \lambda$, N can either be a leaf or an internal node, and $N.edge$ can either be removable or non-removable. Hence each node falls in one of the following four cases.

Case 1: N is a leaf and $N.edge$ is non-removable. $N.edge$ was not removed in the χ-shape algorithm on P and is non-removable in the χ-shape algorithm on $P \cup S$. In this case, no further processing is needed.

Case 2: N is internal and $N.edge$ is non-removable. $N.edge$ was removed in the χ-shape algorithm on P but becomes non-removable in the χ-shape algorithm on $P \cup S$. In this case, the descendants of N should be deleted from the χ-tree and its auxiliary list.

Case 3: N is internal and $N.edge$ is removable. $N.edge$ was removed in the χ-shape algorithm on P and is still removable in the χ-shape algorithm on $P \cup S$. In this case, we just need to set the $v-boundary$ function of $N.oppositeVertex$ to be true.

Case 4: N is a leaf and $N.edge$ is removable. $N.edge$ was non-removable in the χ-shape algorithm on P and becomes removable after the insertion. In this case, two nodes, corresponding to the arms of $N.edge$, should become the children of N. These two nodes can be existing nodes in the χ-tree or newly created nodes. In addition, the $v-boundary$ function of $N.oppositeVertex$ needs to be set to be true.

Let $N.edge$ be $\{d_1, d_2\}$. As the opposite vertex of N must be on the arms of $N.edge$, the two arms $A = \{A_1 = \{d_{A_1}, \theta_0 d_{A_1}\}, A_2 = \{d_{A_2}, \theta_0 d_{A_2}\}\}$ of $N.edge$ can be found by

$$d_{A_1} = \begin{cases} \theta_1 d_1 & \text{if } vertex(\theta_0\theta_1 d_1) = N.oppositeVertex \\ \theta_0\theta_1\theta_0\theta_1\theta_0 d_1 & \text{otherwise} \end{cases}$$

$$d_{A_2} = \begin{cases} \theta_1 d_2 & \text{if } vertex(\theta_0\theta_1 d_2) = N.oppositeVertex \\ \theta_0\theta_1\theta_0\theta_1\theta_0 d_2 & \text{otherwise.} \end{cases} \quad (1)$$

If A_1 or A_2 is not contained in the χ-tree (i.e., $node(d_{A_1}) = null$ or $node(d_{A_2}) = null$), a new node is created and added to the χ-tree as a child of N. A_1 and A_2 can have two adjacent triangles. The opposite vertices (o) of A_1 and A_2 are on the triangle that does not contain $N.edge$, which can be obtained by

$$o(A_1) = \begin{cases} vertex(\theta_0\theta_1 d_{A_1}) & \text{if } vertex(\theta_0\theta_1 d_1) = N.oppositeVertex \\ vertex(\theta_1\theta_0\theta_1\theta_0 d_{A_1}) & \text{otherwise} \end{cases}$$

$$o(A_2) = \begin{cases} vertex(\theta_0\theta_1 d_{A_2}) & \text{if } vertex(\theta_0\theta_1 d_2) = N.oppositeVertex \\ vertex(\theta_1\theta_0\theta_1\theta_0 d_{A_2}) & \text{otherwise.} \end{cases} \quad (2)$$

If A_1 or A_2 is already contained in a node N_i in the χ-tree, we need to check whether the opposite vertex of N_i is on $N.edge$. If so, $N_i.edge$ is visited by the χ-shape algorithm on $P \cup S$ in a different direction from that on P. In this case, a new node is created as

above and added to the χ-tree as a child of N. The branch that originally contains N_i will be captured and chopped off in the Case 2 above. Otherwise, we connect N_i to N as a child of N. When a new node is created and added to the χ-tree, the node is also inserted in W in descending order of the length of the nodes, waiting to be inserted into the auxiliary list of the χ-tree.

After scanning the χ-tree, disconnected components are either reconnected or deleted. The graph again contains a single connected component, i.e., the new χ-tree $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda)$. As the changes to the DT caused by inserting S occur locally relevant to S, the cases 1 and 3 happen much more frequently than the cases 2 and 4 do. The cases 1 and 3 are also much less time consuming than the cases 2 and 4. Hence this incremental algorithm can afford improved efficiency.

3.4. *Deletion*

The same principles used to handle insertion can also be used to deal with deletion. Reading Figure 11 from right to left shows the modifications to the DT caused by deletion. For deletion, $\bar{\Delta}_S$ is the sub-triangulation of $DT(P)$ that disappears after the deletion. And Δ_S is the sub-triangulation of $DT(P \setminus S)$ that is created by the deletion operation. The algorithm described in Section 3.3.1 and lines 6–14 in Algorithm **2** can be used to handle the triangles that disappear ($\bar{\Delta}_S$).

The difference between the deletion algorithm and the insertion algorithm is how the changes to the exterior edges of the DT are handled. There is no exterior edge of $DT(P)$ being covered after the deletion. Hence we do not need to disconnect any nodes from the null root of the χ-tree. In addition, the new exterior edges of the DT are a result of two sources. The first source is that some internal edges of $DT(P)$ are exposed after deletion and become exterior edges of $DT(P \setminus S)$. Let $E_S^{b_1} = \{e \mid e$ is an internal edges of $DT(P)$, e is an exterior edge of $DT(P \setminus S)\}$ denote the new exterior edges created by this source. These edges may be already included in the χ-tree. For each $e \in E_S^{b_1}$, if $node(e) \neq null$, $node(e)$ should be connected to the null root of the χ-tree as one of its children. Otherwise, e is not included in the χ-tree, a new node is created and inserted into the χ-tree as a child of the null root. The node is also inserted into W in descending order of length. The second source is that some newly created edges of Δ_S are exterior edges of $DT(P \setminus S)$. Let $E_S^{b_2} = \{e \mid e$ is on Δ_S and is not on $\bar{\Delta}_S$, e is an exterior edge of $DT(P \setminus S)\}$ denote the new exterior edges created by this second source. Every edge in $E_S^{b_2}$ must not be contained in the χ-tree. Hence a new node is created for each edge in $E_S^{b_2}$ and inserted into the χ-tree as a child of the null root. The node is also inserted into W in descending order of length.

After the initial modifications to the χ-tree and the auxiliary list of the χ-tree, the algorithm described in Section 3.3.3 and lines 24–58 in Algorithm **2** can be used to scan the χ-tree and generate the final χ-tree after deletion. The incremental χ-shape algorithm for deletion is given in Algorithm **3**, where the same steps as the incremental χ-shape algorithm for insertion are omitted and quoted from Algorithm **2**.

4. Results

The efficiency of the incremental χ-shape algorithm proposed in the previous section was evaluated through experiments. The execution time of the incremental χ-shape algorithm is dominated by the step of scanning the χ-tree. This step is identical in the algorithms for

Algorithm 3 Incremental χ-shape algorithm (deletion)

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$, a finite set of two-dimensional points $S \in P$ to be deleted, one parameter $\lambda \in \mathbb{R}$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$

1: Calculate $DT(P \setminus S)$ incrementally from $DT(P)$ and obtain $\bar{\Delta}_S$, Δ_S, $E_S^{b_1}$ and $E_S^{b_2}$
2: $N_d \leftarrow$ {$N \mid N = node(e)$, e is on $\bar{\Delta}_S$, $N.oppositeVertex$ is on $\bar{\Delta}_S$}
3: **if** $N_d = \emptyset$ **and** $E_S^{b_1} = \emptyset$**and** $E_S^{b_1} = \emptyset$ **then**
4: **Return** $DT(P \setminus S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \setminus S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \setminus S, \lambda) \leftarrow O(P, \lambda)$
5: **end if**
6: Lines 6 – 14 in Algorithm **2**
7: $W \leftarrow \emptyset$
8: **for all** $e \in E_S^{b_1}$ **do**
9: **if** $node(e) = null$ **then**
10: Connect $node(e)$ to the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as one of its children
11: **else**
12: $o \leftarrow$ opposite vertex of e in $DT(P \setminus S)$
13: $N \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
14: Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
15: $node(e) \leftarrow N$
16: **end if**
17: **end for**
18: **for all** $e \in E_S^{b_2}$ **do**
19: $o \leftarrow$ opposite vertex of e in $DT(P \setminus S)$
20: $N \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
21: Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
22: $node(e) \leftarrow N$
23: **end for**
24: Lines 24 – 58 in Algorithm **2**
25: **Return** $DT(P \setminus S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \setminus S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \setminus S, \lambda) \leftarrow O(P, \lambda)$

insertion and for deletion. Hence, the performance of the algorithm for insertion was only tested in these experiments—identical patterns of performance are found for deletion.

4.1. *Single-point insertion*

The first experiment tested the performance of the incremental χ-shape algorithm when dealing with the most simple insertion case, i.e., insertion of a single point. In this experiment, we investigated the impact of three factors on the efficiency of the incremental χ-shape algorithm: the number of points in P, the length threshold λ, and the location of the inserted point.

4.1.1. Experimental design

To investigate how the incremental χ-shape algorithm scales under the increasing number of input points, the number of points in P (denoted as $|P|$) was varied across nine incremental levels: $\{2^2 \times 10^2, 2^3 \times 10^2, \cdots, 2^{10} \times 10^2\}$.

Rather than the absolute length threshold λ, a normalized length threshold $\lambda' \in [0, 100]$ was the second factor tested in our experiments. The absolute length threshold is calculated as the length of the edge that is shorter than $\lambda'\%$ of the edges of the DT. By Euler's formula, the number of edges of the DT is linearly proportional to the number of input points. Thus, the normalized length threshold λ' ensures the number of removed edges is proportional to the number of input points, enabling comparison across input points sets of different cardinalities. The normalized length threshold was tested across three levels: {5, 10, 15}, to explore the impact of parameter choice on the efficacy of the incremental χ-shape algorithm.

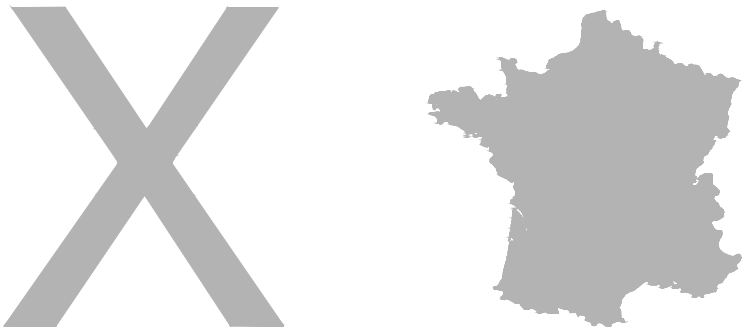
Finally, the location of the inserted point is expected to affect to what extent the DT and the χ-tree are changed after the insertion. The convex hull and χ-shape of P divide the plane into three non-overlapping regions: (1) within the χ-shape of P, (2) between

the χ-shape and the convex hull of P, and (3) outside the convex hull of P. Each inserted point was drawn randomly from one of these three regions.

A full factorial design of experiment (DoE) was conducted on the above three factors. A total number of 100 independent and random replications of P and inserted singleton point set S were generated and processed for each combination of factors. The χ-shape of $P \cup S$ was calculated using both the original and the incremental χ-shape algorithms. The ratio of the execution time of the original χ-shape algorithm to that of the incremental χ-shape algorithm ("speed-up") provided the measure of the increase in performance of the incremental algorithm.

To highlight the efficiency gain resulting from the core of the incremental χ-shape algorithm, the execution time of the incremental DT algorithm was excluded from the execution time of the original and the proposed incremental algorithms (which may both use an existing incremental DT algorithm). In addition, we found the total execution time of both algorithms is dominated by scanning the exterior edges of the triangulation. Hence excluding the execution time of the incremental calculation of DT has an extremely small effect upon the speed-up factor. The execution time of the original χ-shape algorithm was measured as the elapsed time from lines 3–26 of Algorithm **1**. The execution time of the incremental χ-shape algorithm was measured as the elapsed time from lines 2–59 of Algorithm **2**.

The experiments were conducted on two typical types of non-convex shapes shown in Figure 12: (a) a shape of letter "X" and (b) a shape of the border of France. In each replications of the experiment, the points in P were randomly drawn from these two shapes.



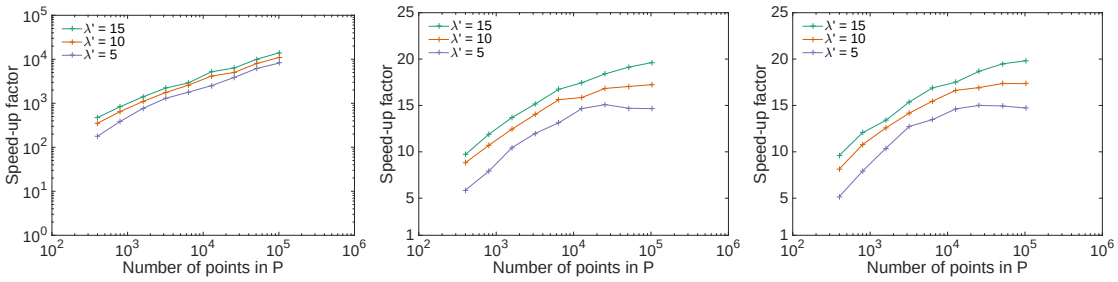
(a) letter "X" (b) France

Figure 4.: Two non-convex shapes where points in P are drawn in the experiment.

4.1.2. Experimental results

The median speed-up in experiments on the shape of letter "X" is illustrated in Figure 13, categorized by the region where the inserted point S was drawn. Each sub-figure depicts the effect of the number of points in P (horizontal axis) and the normalized length threshold λ' (three different colors) on the speed-up factor. A logarithmic scale is used in Figure 13(a), because the speed-up obtained when S is inside the χ-shape of P was several orders of magnitude greater than those obtained in other two regions. When S is inside the χ-shape of P and when the length threshold is not too small (as in our experimental setup), the changes in the DT caused by the insertion often do not affect the χ-tree. In this case, the incremental χ-shape algorithm terminates at line 4 of Algorithm **2**, resulting in the significantly greater speed-up in Figure 13(a)

when compared with Figures 13(b) and (c). When S is outside the χ-shape of P, the insertion is certain to affect the χ-tree. In this situation, the level of speed-up observed was unaffected by whether S is inside (Figure 13(b)) or outside (Figure 13(c))the convex hull of P.



(a) S inside $\boldsymbol{\chi}(P, \lambda)$ (b) S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$ (c) S outside $conv(P)$

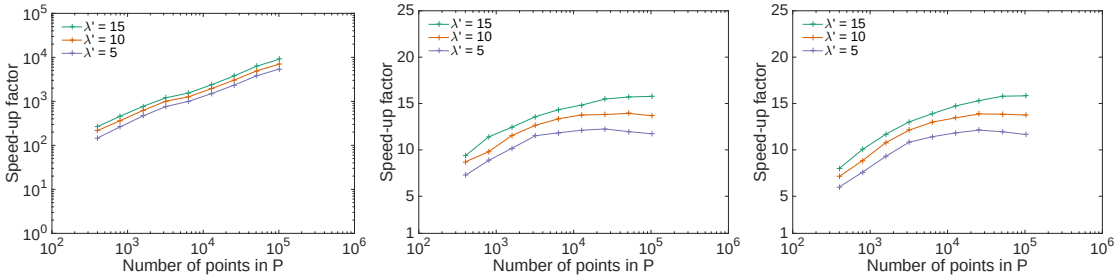
Figure 5.: Relationship between the speed-up factor and the number of points in P, under different λ' values (differently colored lines) and different regions to draw S (a-c) for the shape of letter "X".

In all experimental settings, the speed-up observed was greater than 1, i.e., the incremental χ-shape algorithm was faster than the original χ-shape algorithm. The increase in algorithm speed up was more pronounced with increasing number of points in P. The original and the incremental algorithms have the same worst case time complexity $O(n \log n)$, where n is the number of points in P. To study the average case time complexity, a polynomial regression of the form $T_e = an^b$ was fitted to the median execution time curves, where T_e is the median execution time. As mentioned in Section 4.1.1, T_e excludes the execution time of incremental DT to evaluate the core contribution of this paper. In all cases, the coefficient of determination (R^2) indicated a good fit ($R^2 > 0.99$). The exponent of the curve b provides an empirical estimate of the average case time complexity. Table 3 lists the constant factor a and the exponent b for the original and the incremental χ-shape algorithms, calculated for different experimental settings. The average case time complexity of the original χ-shape algorithm was about $O(n^{0.68})$ under all the experimental settings. The incremental χ-shape algorithm had a nearly constant average case time complexity $O(n^{0.07})$ when S was inside the χ-shape of P. Hence in this case, the speed-up factor reached as high as 1.4×10^4 in our experiment. When S was drawn outside the χ-shape of P, the average case time complexity of the incremental χ-shape algorithm was only slightly lower than that of the original χ-shape algorithm. In this circumstance, the average case time complexity tended to decrease as more edges were removed (i.e., greater λ'). As a consequence, the speed-up factor was highest when $\lambda' = 15$.

The results of the experiment on the shape of French border are illustrated in Figure 14 and Table 4. The impacts of the experimental factors on the speed-up factor demonstrated by this experiment were the same as those exhibited from the experiment on the shape of letter "X". The speed-up factors of the shape of French border were lower than those of the shape of letter "X". The "X" shape exhibits greater concavity than the French border, and the benefits of the incremental χ-shape algorithm are more pronounced when dealing with increasingly non-convex points distributions.

Table 1.: Coefficients for regressions of the form $T_e = a|P|^b$ upon scalability (growth in execution time in seconds as a function of $|P|$), calculated under different λ' values and different regions to draw S for the shape of letter "X".

λ'		S inside $\boldsymbol{\chi}(P, \lambda)$		S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$		S outside $conv(P)$	
		a	b	a	b	a	b
15	Original	3.5×10^{-4}	0.69	3.68×10^{-4}	0.68	3.36×10^{-4}	0.69
	Incremental	3.11×10^{-5}	0.05	3.95×10^{-5}	0.62	3.82×10^{-5}	0.62
10	Original	3.1×10^{-4}	0.67	2.97×10^{-4}	0.68	2.9×10^{-4}	0.68
	Incremental	2.97×10^{-5}	0.05	2.97×10^{-5}	0.63	2.6×10^{-5}	0.64
5	Original	2.16×10^{-4}	0.68	2.19×10^{-4}	0.68	2.13×10^{-4}	0.68
	Incremental	2.61×10^{-5}	0.07	1.81×10^{-5}	0.66	1.81×10^{-5}	0.66



(a) S inside $\boldsymbol{\chi}(P, \lambda)$ (b) S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$ (c) S outside $conv(P)$

Figure 6.: Relationship between the speed-up factor and the number of points in P, under different λ' values (differently colored lines) and different regions to draw S (a-c) for the shape of French border).

Table 2.: Coefficients for regressions of the form $T_e = a|P|^b$ upon scalability (growth in execution time in seconds as a function of $|P|$), calculated under different λ' values and different regions to draw S for the shape of French boder.

λ'		S inside $\boldsymbol{\chi}(P, \lambda)$		S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$		S outside $conv(P)$	
		a	b	a	b	a	b
15	Original	1.11×10^{-4}	0.75	1.27×10^{-4}	0.74	1.18×10^{-4}	0.74
	Incremental	2.55×10^{-5}	0.07	1.15×10^{-5}	0.70	1.25×10^{-5}	0.70
10	Original	1.08×10^{-4}	0.73	1.09×10^{-4}	0.73	1.02×10^{-4}	0.73
	Incremental	2.69×10^{-5}	0.07	8.57×10^{-6}	0.72	9.56×10^{-6}	0.71
5	Original	8.43×10^{-5}	0.73	7.65×10^{-5}	0.74	7.81×10^{-5}	0.73
	Incremental	2.55×10^{-5}	0.07	5.89×10^{-6}	0.74	6.6×10^{-6}	0.73

4.2. *Multi-point insertion*

The single-point insertion experiment demonstrated the performance of the incremental χ-shape algorithm when handling the most simple type of insertion. The next experiment studied the efficiency of our algorithm under the increasing number of points to be inserted, i.e., as $|S|$ increases, as well as changing the spatial distribution of S.

4.2.1. Experimental design

The shape of letter "X" was used to generate the points in P in the multi-point experiment, as based on the single point insertion experiment, it was clear that strongest signal was achieved from more concave shapes. Also based on representative settings

from the single point experiment, the number of points in P was fixed to be $2^8 \times 10^2$; and λ' was set to be 10. The number of points to be inserted had ten incremental levels: $|S| \in \{2^2, 2^3, \cdots, 2^{11}\}$.

The "X" shape contains four non-convex regions (gray regions in Figure 15). When the "X" shape is filled with adequate points, inserting points in one of the non-convex regions does not affect the DT in other non-convex regions. Hence we can control the extent to which the DT is changed by the inserted points through randomly drawing the inserted points from different numbers of the non-convex regions. In this experiment, inserted points were drawn from a single non-convex region (Region 1 in Figure 15), two non-convex regions (Region 1 and 2 in Figure 15), and four non-convex regions (Region 1 – 4 in Figure 15). When inserted points were drawn from multiple regions, the same number of points was generated in each region.

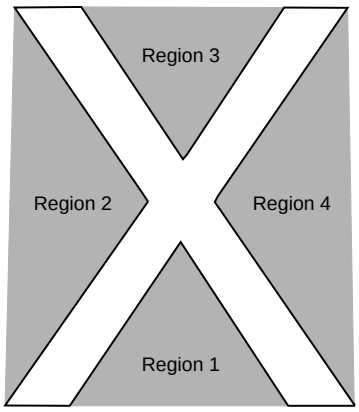


Figure 7.: Four non-convex regions of the shape of letter "X".

As with the single-point insertion experiment, a full factorial experimental design was conducted. The response of the experiment was again the increase in speed relative to the (non-incremental) χ-shape algorithm. A total of 100 independent and random replications were tested for each combination of experimental settings.

4.2.2. Experimental results

Figure 16 illustrates how the median speed-up factor changed with the increasing number of points in S, and when points in S were drawn from different regions. As the number of points in S increased, more changes occurred in the DT, and therefore the speed-up factor decreased. Nevertheless, the incremental χ-shape algorithm was still twice as fast as the original χ-shape algorithm when more than one thousand points were inserted at once.

The distribution of the points in S also played a significant role on the efficacy of the incremental algorithm. As anticipated, the increase in speed was greatest when S was concentrated in a single non-convex region of the shape of "X", because fewer triangles of the DT were affected in this circumstance. In contrast, when S occupied all the four non-convex regions of the shape of "X", the largest number of triangles of the DT were affected, and consequently the increase in efficiency of the incremental algorithm was at its least.

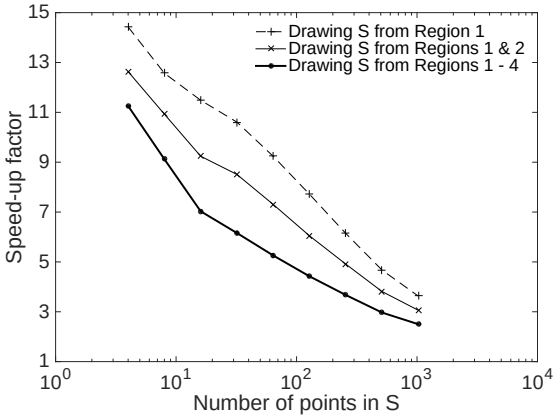


Figure 8.: Relationship between the speed-up factor and the number of points in S. Points in S were drawn from Region 1 (dashed line), Regions 1 & 2 (thinner solid line), and Regions 1 – 4 (thicker solid line) in Figure 15.

5. Discussion

The incremental χ-shape algorithm proposed in this paper was evaluated empirically against the original χ-shape algorithm, when dealing with a dynamic set of points. Two typical kinds of non-convex shapes, a highly non-convex letter shape and a less convex country shape, commonly used in existing literature to evaluate non-convex footprint algorithms, were tested in the single-point experiment. The incremental χ-shape algorithm yielded significant increases in efficiency for both kinds of non-convex shapes. The experimental results demonstrated that the more non-convex the distribution of input points, the greater the increase in efficiency afforded by the incremental algorithm.

The speed-up factor also increases with the increasing size of P. Hence the incremental χ-shape algorithm can be used to efficiently mine the spatial patterns of big dynamic sets of spatial points, such as social media data. The incremental χ-shape algorithm can also efficiently delete or insert multiple points at once. As might be expected from an incremental algorithm, the larger the number of points to be inserted or deleted at once, the lower the increase in efficiency when compared with the non-incremental algorithm. However, our incremental algorithm still provided increases in processing speed even when faced with thousands of points inserted at once. Such performance characteristics are important in stream processing, where batches of points rather than individual points may often need to be processed within a temporal window.

The incremental algorithm efficiency also depends on the distribution of inserted points. Each non-leaf branch attached to the root of the χ-tree captures a distinct non-convex subregion (between χ-shape and convex hull) of the distribution of the input points. The multi-point experiment demonstrated that the incremental χ-shape algorithm works better when fewer such branches are affected by the points to be inserted or deleted. For dynamic, evolving regions, expansion may frequently be in just one direction between two consecutive epochs. In such cases, fewer non-convex regions are likely to be affected, and the benefits of the incremental algorithm increased. In contrast, a spatial region expand in all directions simultaneously, is likely to require insertion or deletion in multiple branches of the tree (subregions), with likely lesser improvements in performance. However, many real-life spatial events may be expected to exhibit anisotropic change, as in the former case. Wildfires, for example, typically propagate in one direction (decided by wind direction, topography, and fuel conditions) over shorter timescales of minutes to hours.

6. Conclusion

An incremental algorithm was proposed in this paper for constructing χ-shapes for dynamic, streaming sets of points. When a large dynamic set of points adds or removes a relatively small number of points, the χ-shape does not need to be recomputed entirely. By representing the χ-shape as a χ-tree, our incremental algorithm can make minimum modifications to the χ-tree and obtain the new χ-shape more efficiently as new points are inserted or deleted.

Our evaluation demonstrated experimentally how the efficiency of our new incremental algorithm is affected by factors including the size of the dynamic data set, the distribution and the size of the inserted or deleted set of points, and the parameterization of the χ-shape. These results provide a guide to the expected increase in performance of our incremental algorithm in a range of circumstances.

References

Acharya, S. and Lee, B.S., 2014. Incremental causal network construction over event streams. *Information Sciences*, 261, 32–51.

Alani, H., Jones, C.B., and Tudhope, D., 2001. Voronoi-based region approximation for geographical information retrieval with gazetteers. *International Journal of Geographical Information Science*, 15 (4), 287–306.

Anglada, M.V., 1997. An improved incremental algorithm for constructing restricted Delaunay triangulations. *Computers & Graphics*, 21 (2), 215–223.

Arampatzis, A., *et al.*, 2006. Web-based delineation of imprecise regions. *Computers, Environment and Urban Systems*, 30 (4), 436–459.

Babcock, B., *et al.*, 2002. Models and issues in data stream systems. *In*: *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, 1–16.

Busby, J.R., 1991. BIOCLIM-a bioclimate analysis and prediction system. *Plant Protection Quarterly (Australia)*.

Chandrasekaran, S. and Franklin, M.J., 2002. Streaming queries over streaming data. *In*: *Proceedings of the 28th international conference on Very Large Data Bases*, 203–214.

Chaudhuri, A.R., Chaudhuri, B.B., and Parui, S.K., 1997. A novel approach to computation of the shape of a dot pattern and extraction of its perceptual border. *Computer Vision and Image Understanding*, 68 (3), 257–275.

Crooks, A., *et al.*, 2013. # Earthquake: Twitter as a distributed sensor system. *Transactions in GIS*, 17 (1), 124–147.

De Berg, M., *et al.*, 2000. The Doubly-Connected Edge List. *Computational geometry*. Springer, chap. 2, 29–33.

Duckham, M., *et al.*, 2008. Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41 (10), 3224–3236.

Edelsbrunner, H., Kirkpatrick, D.G., and Seidel, R., 1983. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29 (4), 551–559.

Ester, M., *et al.*, 1998. Incremental clustering for mining in a data warehousing environment. *In*: *International Conference on Very Large Data Bases*, Vol. 98, 323–333.

Galton, A. and Duckham, M., 2006. What is the region occupied by a set of points?. *Geographic Information Science*. Springer, 81–98.

Garai, G. and Chaudhuri, B., 1999. A split and merge procedure for polygonal border detection of dot pattern. *Image and Vision Computing*, 17 (1), 75–82.

Goodchild, M.F. and Glennon, J.A., 2010. Crowdsourcing geographic information for disaster response: a research frontier. *International Journal of Digital Earth*, 3 (3), 231–241.

Guibas, L.J., Knuth, D.E., and Sharir, M., 1992. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, 7 (1-6), 381–413.

Guo, Q., Kelly, M., and Graham, C.H., 2005. Support vector machines for predicting distribution of Sudden Oak Death in California. *Ecological Modelling*, 182 (1), 75–90.

Jarvis, R.A., 1973. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2 (1), 18–21.

Leathwick, J.R., *et al.*, 2005. Using multivariate adaptive regression splines to predict the distributions of New Zealand's freshwater diadromous fish. *Freshwater Biology*, 50 (12), 2034–2052.

Li, J., *et al.*, 2005. Semantics and evaluation techniques for window aggregates in data streams. *In*: *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 311–322.

Lischinski, D., 1994. Incremental delaunay triangulation. *Graphics gems IV*, 47–59.

Manovich, L., 2011. Trending: The promises and the challenges of big social data. *Debates in the digital humanities*, 2, 460–475.

Melkemi, M., 1997. A-shapes of a finite point set. *In*: *Proceedings of the thirteenth annual symposium on Computational geometry*, 367–369.

Melkemi, M. and Djebali, M., 2000. Computing the shape of a planar points set. *Pattern Recognition*, 33 (9), 1423–1436.

Mokbel, M.F., *et al.*, 2005. Continuous query processing of spatio-temporal data streams in place. *GeoInformatica*, 9 (4), 343–365.

Moreira, A. and Santos, M.Y., 2007. Concave hull: A k-nearest neighbours approach for the computation of the region occupied by a set of points. .

Motwani, R., *et al.*, 2003. Query processing, resource management, and approximation in a data stream management system. *In*: .

Nittel, S., Leung, K.T., and Braverman, A., 2004. Scaling Clustering Algorithms for Massive Data Sets using Data Streams.. *In*: *ICDE*, Vol. 4, p. 830.

Parker, J.K. and Downs, J.A., 2013. Footprint generation using fuzzy-neighborhood clustering. *Geoinformatica*, 17 (2), 285–299.

Peethambaran, J. and Muthuganapathy, R., 2015. A non-parametric approach to shape reconstruction from planar point sets through Delaunay filtering. *Computer-Aided Design*, 62, 164–175.

Phillips, S.J., Anderson, R.P., and Schapire, R.E., 2006. Maximum entropy modeling of species geographic distributions. *Ecological modelling*, 190 (3), 231–259.

Poser, K. and Dransch, D., 2010. Volunteered geographic information for disaster management with application to rapid flood damage estimation. *Geomatica*, 64 (1), 89–98.

Sakaki, T., Okazaki, M., and Matsuo, Y., 2010. Earthquake shakes Twitter users: real-time event detection by social sensors. *In*: *Proceedings of the 19th international conference on World wide web*, 851–860.

Schade, S., *et al.*, 2013. Citizen-based sensing of crisis events: sensor web enablement for volunteered geographic information. *Applied Geomatics*, 5 (1), 3–18.

Şeref, O. and Zobel, C.W., 2013. Recursive voids for identifying a nonconvex boundary of a set of points in the plane. *Pattern Recognition*, 46 (12), 3288–3299.

615 Tufekci, Z., 2014. Big questions for social media big data: Representativeness, validity
616 and other methodological pitfalls. *arXiv preprint arXiv:1403.7400*.
617 Ünal, A., Saygın, Y., and Ulusoy, Ö., 2006. Processing count queries over event streams
618 at multiple time granularities. *Information sciences*, 176 (14), 2066–2096.
619 Vieweg, S., *et al.*, 2010. Microblogging during two natural hazards events: what twitter
620 may contribute to situational awareness. *In*: *Proceedings of the SIGCHI conference*
621 *on human factors in computing systems*, 1079–1088.
622 Yates, D. and Paquette, S., 2011. Emergency knowledge management and social media
623 technologies: A case study of the 2010 Haitian earthquake. *International Journal of*
624 *Information Management*, 31 (1), 6–13.
625 Zhang, X., *et al.*, 2014. Data Stream Clustering With Affinity Propagation. *Knowledge*
626 *and Data Engineering, IEEE Transactions on*, 26 (7), 1644–1656.
627 Zhang, X., Furtlehner, C., and Sebag, M., 2008. Data streaming with affinity propagation.
628 *Machine Learning and Knowledge Discovery in Databases*. Springer, 628–643.
629 Zhong, X., *et al.*, 2016a. Real-time estimation of wildfire perimeters from curated
630 crowdsourcing. *Scientific reports*, 6.
631 Zhong, X., Kealy, A., and Duckham, M., 2016b. Stream Kriging: Incremental and
632 recursive ordinary Kriging over spatiotemporal data streams. *Computers &*
633 *Geosciences*, 90, 134–143.

Algorithm 1 χ-shape algorithm with preparation for future insertion or deletion

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$ and one parameter $\lambda \in \mathbb{R}$
Construct the Delaunay triangulation $DT(P)$ of P
$\Delta \leftarrow DT(P)$
Construct the list B of exterior edges of $DT(P)$
Sort the list B in descending order of edge length
Initialize the $v-boundary$ function
Set the root (r) of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ to be {edge = $\emptyset$, oppositeVertex = $\emptyset$, length = $\emptyset$}
Construct the list of parent nodes (PN) for the elements in B
Set each element in PN to be r
$O(P, \lambda) \leftarrow \emptyset$
while B is not empty **do**
    $e = \{d_1, d_2\} \leftarrow \text{pop}(B)$
    $p \leftarrow \text{pop}(PN)$
    $o \leftarrow$ opposite vertex of e in Δ
    $N \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
    Insert N in $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of p
    $node(d_1) \leftarrow N$
    $node(d_2) \leftarrow N$
    Append N to $O(P, \lambda)$
    if $||e|| > \lambda$ **and** $v-boundary(o) = false$ **then**
        Remove e from Δ
        $v-boundary(o) = true$
        Insert the arms of e in Δ into B in order of edge length
        Insert N into PN at the corresponding position of the arms of e in B
    end if
end while
Return $\boldsymbol{\chi}(P, \lambda)$ formed by leaves of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$

Algorithm 2 Incremental χ-shape algorithm (insertion)

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$, a finite set of two-dimensional points $S \notin P$ to be inserted, one parameter $\lambda \in \mathbb{R}$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$

Calculate $DT(P \cup S)$ incrementally from $DT(P)$ and obtain $\bar{\Delta}_S$, Δ_S and E_S^b
$N_d \leftarrow$ {$N \mid N = node(e)$, e is on $\bar{\Delta}_S$, $N.oppositeVertex$ is on $\bar{\Delta}_S$}
if $N_d = \emptyset$ **and** $E_S^b = \emptyset$ **then**
    Return $DT(P \cup S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \cup S, \lambda) \leftarrow O(P, \lambda)$
end if
for all $N \in N_d$ **do**
    if $N.edge$ is on the boundary of Δ_s **then**
        $N.oppositeVertex \leftarrow$ opposite vertex of $N.edge$ in Δ_S
        Disconnect the children of N from N if N is an internal node
    else
        $node(N.edge) \leftarrow null$
        Delete N from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
    end if
end for
$W \leftarrow \emptyset$
$E_{\Delta_S}^b \leftarrow \{e \mid e \in \Delta_s, e$ is an internal edge of $DT(P \cup S)$ and an exterior edge of $DT(P)\}$
Disconnect $node(E_{\Delta_S}^b)$ from the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$
for all $e \in E_S^b$ **do**
    $o \leftarrow$ opposite vertex of e in $DT(P \cup S)$
    $N \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
    Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
    $node(e) \leftarrow$ N
end for
$N \leftarrow head(O)$
while $N.next \neq null$ **do**
    if $W \neq \emptyset$ **and** $N.length < head(W).length$ **then**
        Insert $head(W)$ into $O(P, \lambda)$ before N
        $N \leftarrow pop(W)$
    end if
    if $N.parent = null$ **then**
        $N_b \leftarrow N$ and its descendants
        $node(N_b.edge) \leftarrow null$
        Delete N_b from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
    end if
    if $N.length > \lambda$ **then**
        if N is an internal node **and** $v{-}boundary(N.oppositeVertex) = false$ **then**
            $v{-}boundary(N.oppositeVertex) = true$
        else if N is an internal node **and** $v{-}boundary(N.oppositeVertex) = true$ **then**
            $N_b \leftarrow N$ and its descendants
            $node(N_b.edge) \leftarrow null$
            Delete N_b from $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$
        else if N is a leaf **and** $v{-}boundary(N.oppositeVertex) = false$ **then**
            $v{-}boundary(N.oppositeVertex) = true$
            $A \leftarrow$ arms of $N.edge$ using Equation 1
            for all $e \in A$ **do**
                if $node(e) \neq null$ **and** $node(e).oppositeVertex$ is not on $N.edge$ **then**
                    Connect $node(e)$ to N as a child of N
                else
                    $o \leftarrow$ opposite vertex of e using Eq 2
                    $N' \leftarrow$ {edge = e, oppositeVertex = o, length = $||e||$}
                    Insert N' into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of N and into W in descending order of length
                    $node(e) \leftarrow N'$
                end if
            end for
        end if
    end if
    $N \leftarrow N.next$
end while
Return $DT(P \cup S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \cup S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \cup S, \lambda) \leftarrow O(P, \lambda)$

Algorithm 3 Incremental χ-shape algorithm (deletion)

Require: A finite set of two-dimensional points $P \subset \mathbb{R} \times \mathbb{R}$, a finite set of two-dimensional points $S \in P$ to be deleted, one parameter $\lambda \in \mathbb{R}$, $DT(P)$, $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and $O(P, \lambda)$

1: Calculate $DT(P \setminus S)$ incrementally from $DT(P)$ and obtain $\bar{\Delta}_S$, Δ_S, $E_S^{b_1}$ and $E_S^{b_2}$
2: $N_d \leftarrow \{N \mid N = node(e)$, e is on $\bar{\Delta}_S$, $N.oppositeVertex$ is on $\bar{\Delta}_S\}$
3: **if** $N_d = \emptyset$ **and** $E_S^{b_1} = \emptyset$**and** $E_S^{b_1} = \emptyset$ **then**
4: **Return** $DT(P \setminus S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \setminus S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \setminus S, \lambda) \leftarrow O(P, \lambda)$
5: **end if**
6: Lines 6 – 14 in Algorithm **2**
7: $W \leftarrow \emptyset$
8: **for all** $e \in E_S^{b_1}$ **do**
9: **if** $node(e) = null$ **then**
10: Connect $node(e)$ to the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as one of its children
11: **else**
12: $o \leftarrow$ opposite vertex of e in $DT(P \setminus S)$
13: $N \leftarrow \{$edge $= e$, oppositeVertex $= o$, length $= ||e||\}$
14: Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
15: $node(e) \leftarrow N$
16: **end if**
17: **end for**
18: **for all** $e \in E_S^{b_2}$ **do**
19: $o \leftarrow$ opposite vertex of e in $DT(P \setminus S)$
20: $N \leftarrow \{$edge $= e$, oppositeVertex $= o$, length $= ||e||\}$
21: Insert N into $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ as a child of the null root of $\mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$ and into W in descending order of length
22: $node(e) \leftarrow N$
23: **end for**
24: Lines 24 – 58 in Algorithm **2**
25: **Return** $DT(P \setminus S)$, $\mathbf{T}_{\boldsymbol{\chi}}(P \setminus S, \lambda) \leftarrow \mathbf{T}_{\boldsymbol{\chi}}(P, \lambda)$, $O(P \setminus S, \lambda) \leftarrow O(P, \lambda)$

Table 3.: Coefficients for regressions of the form $T_e = a|P|^b$ upon scalability (growth in execution time in seconds as a function of $|P|$), calculated under different λ' values and different regions to draw S for the shape of letter "X".

λ'		S inside $\boldsymbol{\chi}(P, \lambda)$		S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$		S outside $conv(P)$	
		a	b	a	b	a	b
15	Original	3.5×10^{-4}	0.69	3.68×10^{-4}	0.68	3.36×10^{-4}	0.69
	Incremental	3.11×10^{-5}	0.05	3.95×10^{-5}	0.62	3.82×10^{-5}	0.62
10	Original	3.1×10^{-4}	0.67	2.97×10^{-4}	0.68	2.9×10^{-4}	0.68
	Incremental	2.97×10^{-5}	0.05	2.97×10^{-5}	0.63	2.6×10^{-5}	0.64
5	Original	2.16×10^{-4}	0.68	2.19×10^{-4}	0.68	2.13×10^{-4}	0.68
	Incremental	2.61×10^{-5}	0.07	1.81×10^{-5}	0.66	1.81×10^{-5}	0.66

Table 4.: Coefficients for regressions of the form $T_e = a|P|^b$ upon scalability (growth in execution time in seconds as a function of $|P|$), calculated under different λ' values and different regions to draw S for the shape of French boder.

λ'		S inside $\boldsymbol{\chi}(P, \lambda)$		S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$		S outside $conv(P)$	
		a	b	a	b	a	b
15	Original	1.11×10^{-4}	0.75	1.27×10^{-4}	0.74	1.18×10^{-4}	0.74
	Incremental	2.55×10^{-5}	0.07	1.15×10^{-5}	0.70	1.25×10^{-5}	0.70
10	Original	1.08×10^{-4}	0.73	1.09×10^{-4}	0.73	1.02×10^{-4}	0.73
	Incremental	2.69×10^{-5}	0.07	8.57×10^{-6}	0.72	9.56×10^{-6}	0.71
5	Original	8.43×10^{-5}	0.73	7.65×10^{-5}	0.74	7.81×10^{-5}	0.73
	Incremental	2.55×10^{-5}	0.07	5.89×10^{-6}	0.74	6.6×10^{-6}	0.73

Figure 9.: Examples of adjacent triangle, arms, opposite vertex, and darts in a triangulation.

(a) An example χ-shape (b) The χ-tree of the χ-shape (a)

Figure 10.: The χ-tree (b) of an example χ-shape (a). Labeled nodes in the χ-tree (b) correspond to edges of the χ-shape (a). The leaves of the χ-tree form the χ-shape (thicker edges) shown in (a).

(a) Before insertion of S (b) After insertion of S

Figure 11.: Changes to DT caused by inserting a set of points S into P.

(a) letter "X" (b) border of France

Figure 12.: In the experiments, the points in P (dots) are drawn from two non-convex shapes (gray regions): (a) shape of letter "X" and (b) shape of French border. The χ-shapes of the above points are depicted by the polygons in solid line.

(a) S inside $\boldsymbol{\chi}(P, \lambda)$ (b) S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$ (c) S outside $conv(P)$

Figure 13.: Relationship between the speed-up factor and the number of points in P, under different λ' values (differently colored lines) and different regions to draw S ((a): S inside $\boldsymbol{\chi}(P, \lambda)$, (b): S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$, (c): S outside $conv(P)$) for the shape of letter "X".

(a) S inside $\boldsymbol{\chi}(P, \lambda)$ (b) S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$ (c) S outside $conv(P)$

Figure 14.: Relationship between the speed-up factor and the number of points in P, under different λ' values (differently colored lines) and different regions to draw S ((a): S inside $\boldsymbol{\chi}(P, \lambda)$, (b): S inside $conv(P)$ and outside $\boldsymbol{\chi}(P, \lambda)$, (c): S outside $conv(P)$) for the shape of French border.

Figure 15.: Four non-convex regions of the shape of letter "X".

Figure 16.: Relationship between the speed-up factor and the number of points in S. Points in S were drawn from Region 1 (dashed line), Regions 1 & 2 (thinner solid line), and Regions 1 – 4 (thicker solid line) in Figure 15.