A new algorithm for finding polygonal voids in Delaunay triangulations and its parallelization

José Ojeda[1], Rodrigo Alonso[1], and Nancy Hitschfeld-Kahler[1]

1 **Department of Computer Science, FCFM, University of Chile**
jojeda@dcc.uchile.cl, ralonso@dcc.uchile.cl, nancy@dcc.uchile.cl*

Abstract

A known geometrical problem is to find low density zones (voids) in planar point sets and to represent them as polygons. In this paper we recall the concept of terminal-edge region to identify subvoid candidates over a triangulation, present a linear algorithm to find subvoids taking as input a Delaunay triangulation, and show that this new strategy can be naturally parallelized using GPU computing. We also show preliminary experimental results.

1 Introduction

A real geometrical problem is to find underdense zones in planar point sets and represent their shapes by polygons. This problem is particularly relevant in astronomy, where the regions almost empty of bright galaxies are known as cosmological voids. In computational geometry, a similar and well studied problem is to find large convex holes in a planar point set P. A convex hole is represented by a convex polygon that contains no point of P in its interior. The key question is the expected size (number of vertices) of the existent convex polygons [2, 8]. Convex polygons solutions would not generally apply to the problem in astronomy, where the cosmological voids are usually not convex and may contain a few galaxies in its interior. The computation of the α-shape of a set of points [3], which is a generalization of the convex hull of it can also be seen as a related problem. The original goal of the α-shape algorithm is to represent the shape of the set of points and not to compute the empty spaces inside the convex hull of the point set but it has been also used to study the topology of the cosmic web [11].

In the last years, a new algorithm was developed to detect and build polygons representing underdense regions or voids on a planar point set [1, 4]. The algorithm starts from terminal-edges (local longest-edges) in a Delaunay triangulation and builds the largest possible low density terminal-edge regions around them. A terminal-edge region can represent either an entire void or part of a void (subvoid). The algorithm is divided in two main steps: (1) *Building subvoids* and (2) *Joining subvoids*. In the first step, each triangle is attached to the terminal-edge region (a triangle set) it belongs to. In the second step, fragmented voids, represented by several terminal-edge regions, are joined to build whole voids. Each void is described by a polygon defined by the edges which appear only once in the region.

In this paper, we (a) summarize the foundations of the *Building subvoids* step introduced in [1], (b) present the design of a linear algorithm to find terminal-edge regions starting from a Delaunay triangulation, and (c) show that this new solution can be naturally parallelized using GPU computing. We also show preliminary experimental results.

2 Problem Statement and Solution

In this section we summarize the concepts and some theoretical foundations of the algorithm to find underdense regions (voids) published in [1]. These concepts are required to understand

34th European Workshop on Computational Geometry, Berlin, Germany, March 21–23, 2018.
This is an extended abstract of a presentation given at EuroCG'18. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

the new algorithm presented in § 3. It is worth to mention that a region whose interior point density is lower than the background density is considered a void candidate. Each application must define the corresponding threshold density value.

2.1 Main Approach

A Delaunay triangulation is geometric structure that allows one to find naturally the input points (sites) that are (locally) close or far from each other. By definition, an edge is a Delaunay edge, if there exists an empty circle that passes through its endpoints. If the empty circles are large enough, they are candidates to be part of a void. That is why a Delaunay triangulation (in time $O(n \log n)$) over the input points is generated first and used later as input to look for subvoids. As basic strategy, the algorithm described in [1] builds empty regions around local longest-edges. Each two triangles that share "large enough" local longest-edge will be part of a subvoid and so define the boundaries of initial empty polygons. The question is now which other triangles should be added to these two initial triangles to form a larger empty polygon. The strategy includes triangles that are adjacent to the two initial triangles and that share their longest-edge with these two initial triangles. Then the empty polygon will be now defined by the two initial triangles plus the neighbor triangles that fulfill this criterion. This process is repeated for the newly added triangles until no other triangle can be added. Notice that this criterion avoids going from a large empty region to another large empty region trough a zone like a tunnel or a region with higher point density [4].

2.2 Basic Definitions

Let T be a conforming triangulation of n points and m triangles. Let t_i be a triangle, $i = 0, ..., m-1$. Each triangle has a unique *longest-edge* and a *shortest-edge* except for isosceles and equilateral triangles. We assume that there are neither isosceles nor equilateral triangles but if they would exist, the equal length edges are arbitrarily ordered. The terms *Longest-edge propagation path* and *terminal-edge* were first introduced by Rivara in the context of new algorithms for the refinement/improvement of triangulations [9]. In [1] these terms are used to define and characterize the *terminal-edge regions*, that is, the subvoid candidates. That is why they are also recalled here.

▶ **Definition 2.1.** Terminal-triangles and -edges [10]: Two triangles are called *terminal triangles* if they share their longest-edge. This shared longest-edge is called *terminal-edge*.

▶ **Definition 2.2.** Longest-edge propagation path [10]: For any triangle t_0 in any conforming triangulation T, the *Longest-Edge Propagation Path* of t_0 (Lepp(t_0)) is the ordered list of all the triangles t_0 , t_1, t_2, ..., t_{l-1}, t_l, such that t_i is the neighbor triangle of t_{i-1} by the longest-edge of t_{i-1}, for $i = 1, 2, \ldots, l$. The longest-edge shared by t_{l-1} and t_l is a terminal-edge and t_{l-1} and t_l are terminal-triangles.

▶ **Definition 2.3.** Boundary edge and boundary terminal-edge: The edges that belong to only one triangle of a triangulation are called *boundary edges*. If a boundary edge is the longest-edge of the triangle, it will be called *boundary terminal-edge*.

2.3 Algorithm Foundations

In this section we recall the definitions and theorems published in [1] required to understand the new sequential algorithm proposed in § 3.1 and the parallel algorithm presented in § 3.2. Theorem 2.9 was extended to show that terminal-edges regions cover the whole space.

▶ **Definition 2.4.** Terminal-edge region: A *terminal-edge region* R is a region formed by the union of all triangles t_i such that Lepp(t_i) has the same terminal-edge. In case the terminal-edge is a boundary-edge the region will be called *boundary terminal-edge region*. As illustration see Figure 1.

▶ **Definition 2.5.** Frontier-edge: A *frontier-edge* is an edge that is shared by two triangles, each one belonging to a different terminal-edge region.

▶ **Lemma 2.6.** *Let t_i and t_j be two triangles that share the edge e. If the edge e is a frontier-edge, then e is neither the longest-edge of t_i nor of t_j.*

Proof: See [1].
Note that the opposite is false. It is possible that two triangles t_i, t_j whose shared edge is not the longest-edge of any of them and these two triangles belong to the same terminal-edge region R_i. e is still a particular case frontier-edge in the sense that the region R_i can not grow through it. Since e does not separate two different regions, it was called a *barrier-edge*.

▶ **Definition 2.7.** Barrier-edge: A *barrier-edge* of a region R is an edge shared by two triangles of R which is not the longest-edge of any of them.

▶ **Definition 2.8.** Internal-edge: An *internal-edge* of a region R is an edge that is neither a terminal-edge, a frontier-edge, a barrier-edge nor a boundary-edge.

▶ **Theorem 2.9.** *Let T be a conforming triangulation of any set of points P and let CH be the convex hull of P. Then T can be partitioned into a set of terminal-edge regions covering the whole convex hull area as shown in Figure 1, and without overlapping.*

Proof: In [1] was demonstrated that terminal-edge regions do not overlap, then we only need to show that terminal-edge regions cover P. Let us assume that the triangulation T has n triangles and m terminal-edges $e_i, i = 0, \ldots, m-1$. e_i can also be a boundary terminal-edge. By definition 2.4, each terminal-edge e_i has associated a terminal-edge region R_i. Let assume that there exists a triangle $t_j, j = 0, \ldots, n-1$ which does not belong to any terminal-edge region $R_i, i = 0, \ldots, m-1$. By definition 2.2 each triangle t_j has a Lepp(t_j) and so an associated terminal-edge e. Then t_j must be included in the terminal-edge region associated to e and this fact contradicts our assumption. □

3 The New Algorithm

The algorithms presented in this section decrease the computational cost of the *Building subvoids* step published in [1]. The original algorithm sorts the triangles of the Delaunay triangulation by their longest-edge, and so computes the terminal-edge regions associated to the largest-terminal edges first. In this way, the *Building subvoids* step is done in $O(n \log n)$, where n is the number of points. The new algorithm works over a graph representation $G = (V, E)$ of a triangulation [7] and builds the terminal-edge regions in $O(n)$. In G each node $v \in V$ represents a triangle and E contains the adjacency relations. More specifically, arc $(u, v) \in E$ if and only if the triangles represented by the nodes u and v are adjacent in the actual triangulation.

3.1 Sequential Algorithm

Algorithm 1 partitions the adjacency graph $G = (V, E)$ into terminal-edge regions by removing arcs of E associated with adjacencies between triangles sharing frontier-edges. The graph is

divided into terminal-edge regions, each one a subvoid candidate. The regions are classified into boundary subvoids or subvoids. Algorithm 2 uses a simple navigation strategy to compute the area and to check if it is a boundary component or not. The *Building subvoids* step is now in $O(n)$.

Input: A adjacency graph $G = (V, E)$ of the Delaunay triangulation
Input: A *threshold_value*
foreach $v \in V$ **do**
  Label the longest-edge in triangle related to v
foreach $(u, v) \in E$ **do**
  if the triangles related to u and v share a frontier edge **then**
    $E \leftarrow E \setminus \{(u, v)\}$
foreach $v \in V$ **do**
  if v is not visited yet **then**
    $total\_area, touches\_border \leftarrow$ **Visit**(G, v)
    if $total\_area \geq threshold\_value$ **then**
      if *touches_border* **then**
        Component type of $v \leftarrow$ Boundary subvoid candidate
      else
        Component type of $v \leftarrow$ Subvoid candidate
return $G = (V, E)$

Algorithm 1: *Building subvoids* algorithm

3.2 Parallel Algorithm

The sequential algorithm is naturally parallelizable on GPU because the computation of terminal-edge regions can be partitioned into $O(n)$ smaller tasks and solved by threads per arc and threads per nodes using similar data structures as the ones described in [5, 6]. Currently, the tasks per thread are $O(1)$ except for the area computation (Algorithm 2), which in the worst case is $O(n)$. The steps of the kernels are:

1. Kernel Initialization
   - Create a thread per node v (triangle)
   - Each thread labels its triangle longest-edge
   - Each thread computes its triangle area
2. Kernel Generation of terminal-edge regions
   - Create a thread per arc e
   - Each thread eliminates arc e if the shared edge is a frontier-edge
   - Each thread labels the shared edge as terminal-edge if corresponds
3. Kernel Terminal-edge regions classification
   - Create a thread per terminal-edge region
   - Each thread computes the area of its terminal-edge region
   - Each thread labels its region as subvoid, boundary subvoid or null according to a threshold value.

Input: $G = (V, E)$ the graph
Input: A node $v \in V$ belonging to a terminal-edge region
1 Mark v as visited
2 $total\_area \leftarrow$ Area of the triangle related to v
3 $touches\_border \leftarrow$ Triangle related to v is a boundary triangle
4 **foreach** $(u, v) \in E$ **do**
5 **if** u is not visited yet **then**
6 $u\_total\_area, u\_touches\_border \leftarrow$ **visit**(G, u)
7 $total\_area \leftarrow total\_area + u\_total\_area$
8 $touches\_border \leftarrow touches\_border \vee u\_touches\_border$
9 **return** *total_area*,*touches_border*

Algorithm 2: The **Visit** subroutine

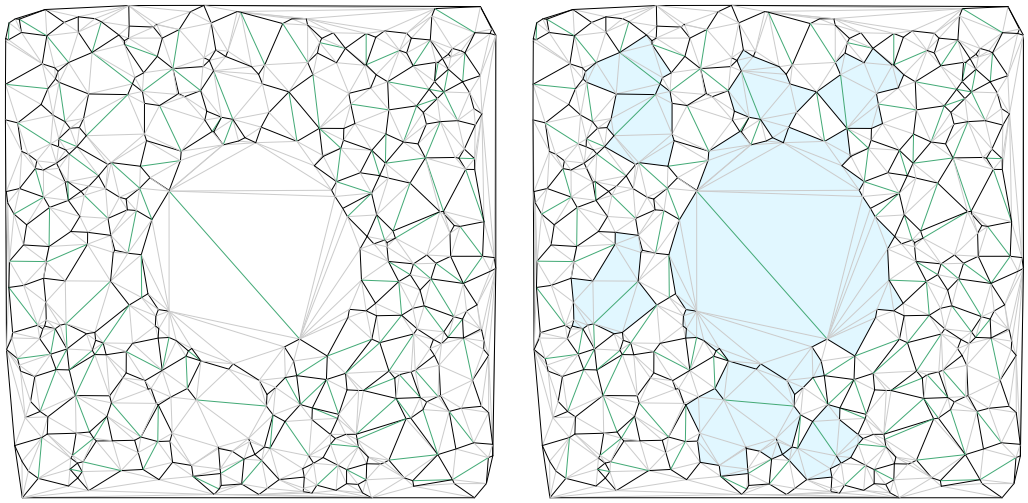

Figure 1 Terminal-edge regions in a Delaunay triangulation. The green edges are terminal-edges, the light blue are internal edges and the black edges are frontier edges. At the left, each terminal-edge region is delimited by the black edges, and at the right, the light blue regions are the ones with area greater than a threshold value. These are subvoids candidates.

4 Empirical Evaluation

Figure 2 shows a comparison of the three implementations of the *Building subvoids* step. The x-axis shows the number of points and the y-axis the classification time. The green line represents the performance of the original solution written in Python [1], the blue line the time of Algorithm 1 in § 3.1 and the red line, the performance of the parallel implementation described in § 3.2. Both new implementations are written in C and Opencl.

5 Conclusions and Ongoing work

We have presented a new algorithm to find terminal-edge regions (subvoids candidates) in planar points sets. This new strategy allowed us to propose sequential and parallel algorithms which have a lower time complexity than the approach published in [1]. The parallel strategy is still in the worst case $O(n)$ due to the linear computation of each terminal-edge region area, but this computation can also be improved by parallelizing inside each terminal-edge

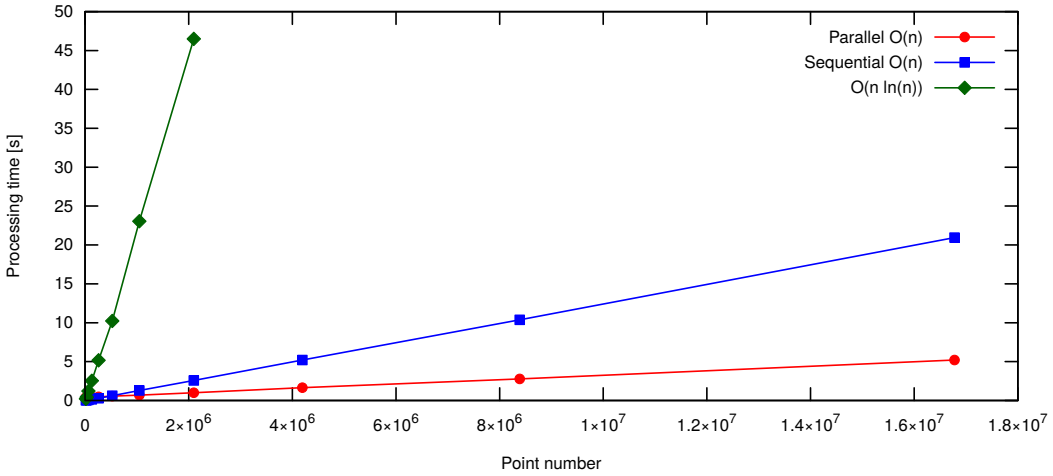

Figure 2 Performance comparison on a multicore architecture.

region. We also plan to compare our approach with prior work in related problems [11].

References

1 R. Alonso, J. Ojeda, N. Hitschfeld, C. Hervías, and L.E. Campusano. Delaunay based algorithm for finding polygonal voids in planar point sets. *Astronomy and Computing*, 22:48 – 62, 2018.

2 József Balogh, Hernán González-Aguilar, and Gelasio Salazar. Large convex holes in random point sets. *Comput. Geom.*, 46(6):725–733, 2013.

3 Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–558, 1983.

4 Carlos Hervías, Nancy Hitschfeld-Kahler, Luis E. Campusano, and Giselle Font. On finding large polygonal voids using Delaunay triangulation: The case of planar point sets. In *Proceedings of the 22nd International Meshing Roundtable*, pages 275–292, 2013.

5 C. A. Navarro, N. Hitschfeld-Kahler, and E. Scheihing. A GPU-based method for generating quasi-Delaunay triangulations based on edge-flips. In *GRAPP/IVAPP*, pages 27–34, 2013.

6 Cristobal Navarro, Nancy Hitschfeld-Kahler, and Eliana Scheihing. A parallel GPU-based algorithm for Delaunay edge-flips. In *Abstracts from 27th European Workshop on Computational Geometry (EUROCG2011)*, pages 75–78. Switzerland, March 28-30, 2011.

7 Joseph O'Rourke. *Computational Geometry in C.* Cambridge University Press, New York, NY, USA, 2nd edition, 1998.

8 Rom Pinchasi, Rados Radoicic, and Micha Sharir. On empty convex polygons in a planar point set. *J. Comb. Theory, Ser. A*, 113(3):385–419, 2006.

9 M. C. Rivara. New longest-edge algorithms for the refinement and/or improvement of unstructured triangulations. *Int. Jour. for Num. Meth. in Eng.*, 40:3313–3324, 1997.

10 M. C. Rivara, N. Hitschfeld, and R. B. Simpson. Terminal edges Delaunay (small angle based) algorithm for the quality triangulation problem. *CAD journal*, 33:263–277, 2001.

11 Rien van de Weygaert, Gert Vegter, Herbert Edelsbrunner, Bernard J. T. Jones, Pratyush Pranav, Changbom Park, Wojciech A. Hellwing, Bob Eldering, Nico Kruithof, E. G. P. (Patrick) Bos, Johan Hidding, Job Feldbrugge, Eline ten Have, Matti van Engelen, Manuel Caroli, and Monique Teillaud. Alpha, betti and the megaparsec universe: On the topology of the cosmic web. *Trans. Computational Science*, 14:60–101, 2011.