Direct α-Shape on the GPU

Nathan Tihon Jonathan Lambrechts Jean-François Remacle

December 20, 2024

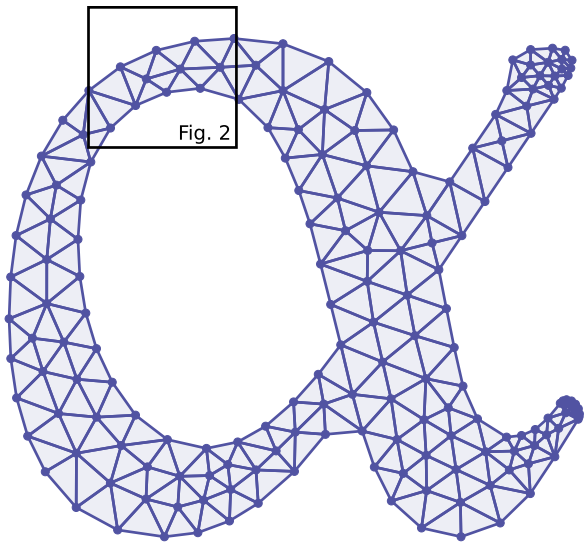


Figure 1: *The α-shape $C_\alpha(S)$ (in light blue) of a set of points S (blue dots) is the union of all triangles of the Delaunay triangulation DT(S) whose circumradius is less than α (blue lines).*

Abstract

In this note, we describe an innovative method for computing the triangulation of the α-shape of a point set displaying two benefits. First, it avoids the computation of the Delaunay triangulation – hence *directly* computing the α-shape. Second, it is entirely local and efficient on the GPU. Our method is two orders of magnitude faster than CGAL's implementation.

1 Introduction

Let $S = \{s_1, \ldots, s_n\}$ be a set of n points of $\mathbb{R}^2$ and α be a positive real number which has the dimension of a length. The α-shape $C_\alpha(S)$ of S [3] generalizes the notion of convex hull and gives a formal definition to the concept of *shape* of a given point set. In 1983, Edelsbrunner claimed this concept would find applications in pattern recognition and cluster analysis. Since then, α-shapes have been successfully applied to molecular biology [6], surface reconstruction, ecology [10] and even astronomy. Our main interest is with the *Particle Finite Element Method* (PFEM)[1, 5], a numerical method requiring the **computation of the α-shape at every time step** of a finite element simulation. The efficiency of this calculation therefore plays a critical role in the overall performance of the PFEM.

The current algorithms allowing one to compute α-shapes work essentially as described in the original paper [3]: compute the Delaunay triangulation DT(S) and subsequently filter out the triangles whose circumradiuses are larger than α. In [7], authors have shown that it is possible to calculate DT(S) quite efficiently. Unfortunately, these results were followed by a dreary conclusion: **using the classical Bowyer-Watson algorithm to compute the Delaunay triangulation does not allow good parallel scaling** when more than about 10 cores were involved.

Some authors [9] have proposed to compute the Voronoi Diagram VD(S) in parallel on a GPU. Their approach requires to know *a priori* the maximum number of neighboring points involved in the construction of the Voronoi cells, and this number can be large. It turns out it is possible to reformulate these ideas to obtain $C_\alpha(S)$ by constructing local triangulations. To the best of our knowledge, authors of [8] leveraged problem-specific information to provide the only parallel implementation of α-shapes without filtering Delaunay triangulations.

In this note, we develop a provably correct algorithm that constructs $C_\alpha(S)$ directly i.e. without neither computing DT(S) nor VD(S). Our algorithm is more lightweight and faster than [8] thanks to theoretical guarantees derived directly from the definition of the α-shape.

2 A local Bowyer-Watson algorithm to compute $C_\alpha(S)$

The concept of α-shape is closely linked to the Delaunay triangulation DT(S).

Definition 2.1. *We refer to the triangle with vertices s_i, s_j, s_k as Δ_{ijk}. The α-complex $C_\alpha(S)$ of S is the set*

of triangles of DT(S) whose circumradiuses R_{ijk} are smaller than α.

$$C_\alpha(S) := \{\Delta_{ijk} \in DT(S) : R_{ijk} \leq \alpha\} \tag{2.1}$$

Definition 2.1 is an intuitive way of understanding α-shapes and leads to the algorithm described in [3]. Notice that the quantity R_{ijk} is dependent on the triangles, hence it cannot be evaluated *a priori*. This is why classical algorithms first compute $\mathrm{DT}(S)$. As mentioned in section 1, computing $\mathrm{DT}(S)$ classically does not yield good scaling. To address this problem, **we propose a method that is completely local**. Let us first introduce the definition of *local α-Delaunay*:

DEFINITION 2.2. *A triangle Δ_{ijk} is said to be locally Delaunay if its circumcircle is empty i.e. it does not contain points of S. Additionally, we say that a triangle is locally α-Delaunay if it is locally Delaunay and if $R_{ijk} \leq \alpha$.*

Intuitively, our algorithm computes a local triangulation T_i around each point $s_i \in S$. Each triangulation is made of all locally α-Delaunay triangles Δ_{ijk} having s_i as a vertex. A graphical representation is seen in black lines on fig. 2 around point s.

$$T_i := \{\Delta_{ijk} \in \mathrm{DT}(S) : R_{ijk} \leq \alpha\} \subseteq C_\alpha(S) \tag{2.2}$$

In this way, T_i represent the subset of $\Delta_{ijk} \in C_\alpha(S)$ which have s_i as vertex. Hence, taking the union of all local triangulations leads to the α-shape!

Let us now give some results to ensure the correctness of those local triangulations.

PROPOSITION 2.1. *Suppose $e_{\max}$ is the longest edge in $C_\alpha(S)$, we have :*

$$e_{\max} \leq 2\alpha \tag{2.3}$$

Proof. Let $R_{\max}$ be the largest circumradius of the triangles in $C_\alpha(S)$. By definition of eq. (2.1) we have $R_{\max} \leq \alpha$. Coupling this with the triangular inequality, we find :

$$e_{\max} \leq R_{\max} + R_{\max} \leq 2\alpha \tag{2.4}$$

See fig. 3a for a graphical representation. ∎

DEFINITION 2.3. *We say two points $s_i, s_j \in S$ are neighbors if they share an edge of $C_\alpha(S)$. Following proposition 2.1 we define N_i as the set of **potential** neighbors of s_i in $C_\alpha(S)$:*

$$N_i := \{s_j : \|s_j - s_i\|_2 \leq 2\alpha\} \tag{2.5}$$

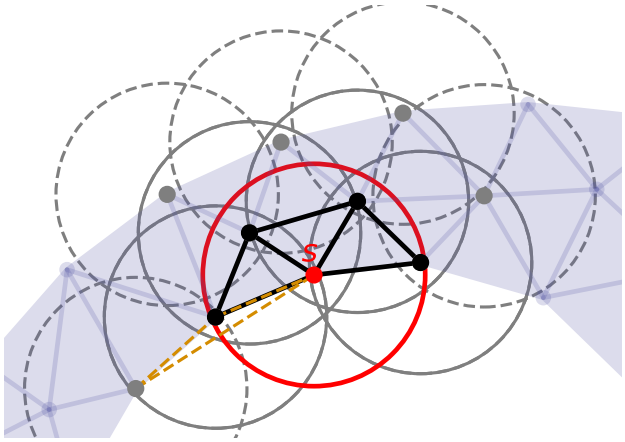


Figure 2: *Local triangulation around point s (red dot). All circles have radius α. In the α-shape, s can only be connected to points that are at a maximum distance 2α (gray and black points). This condition is equivalent to a collision problem: find all gray circles that intersect the red circle. Black dots are neighbors kept in the final local α-shape whereas gray dots are cut-off neighbors. The dashed orange triangle is an example of invalid triangle with only valid neighbors.*

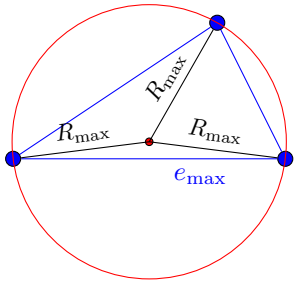
Using proposition 2.1 and eq. (2.5), we can guarantee that given a point s_i, any triangle Δ_{ijk} constructed with a neighbor $s_j \in N_i$ and a non-neighbor $s_k \notin N_i$ is not locally α-Delaunay.

$$s_j \in N_i, s_k \notin N_i \Longrightarrow R_{ijk} > \alpha \tag{2.6}$$

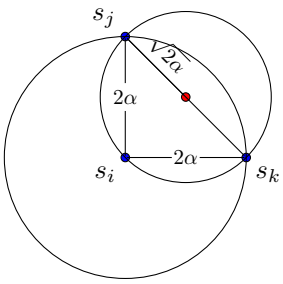
Indeed, the edge between s_i and s_k is longer than 2α, resulting in a circumradius $R_{ijk} > \alpha$. **Notice that the converse is not true:** it is possible to construct a triangle Δ_{ijk} with $R_{ijk} > \alpha$ using only neighbors $s_j, s_k \in N_i$ as shown in fig. 3b and in dashed orange in fig. 2. In consequences, we still have to ensure that the circumradiuses of the generated triangles respect the α-shape condition.

Furthermore, it is impossible for a point $s_k \notin N_i$ to violate the local Delaunay property of locally α-Delaunay triangles. In other words, $s_k \notin N_i$ cannot induce edge flips in T_i. Indeed, the farthest point s_k of s_i inside the circumcircle of a locally α-Delaunay triangle Δ_{ijk} is located at a distance 2α of s_i. Therefore any point farther than 2α cannot be inside the circumcircle of a locally α-Delaunay triangle. This property is illustrated in fig. 3c.

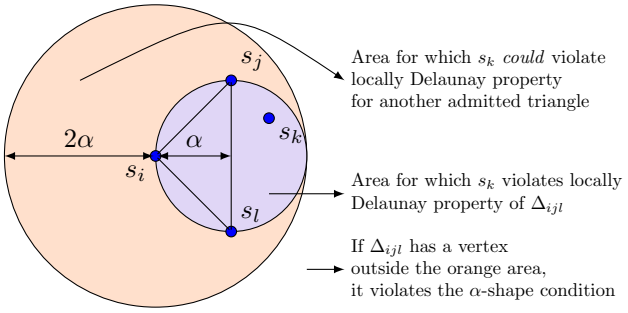
Using these two results, we can guarantee that locally α-Delaunay triangles $\Delta_{ijk} \in T_i$ (i) only use vertices of the set N_i and (ii) cannot be invalidated by points outside of N_i. Therefore, the local triangulation T_i around s_i can be constructed by using the points in N_i.



(a) *Relating α to $e_{\max}$.*



(b) *Illegal triangle ($R_{ijk} \geq \alpha$) generated by accepted neighbors (blue dot)*



(c) *Points outside of the disk of radius 2α centered on s_i cannot violate the locally Delaunay property of triangles in the local triangulation.*

Figure 3: *Graphical representation of properties detailed in section 2.*

3 Algorithm

Practically, there are 3 steps to our algorithm. The first is to compute the neighbors N_i for each point s_i, after which we compute the locally Delaunay triangles Δ_{ijk}. Finally, there remains to suppress the triangles violating the α-shape condition. In the end, only the locally α-Delaunay triangles containing s_i as vertex remain, forming the local triangulation T_i.

3.1 Neighbor Detection. In this step, we are interested by efficiently computing N_i. To do this, we reformulate eq. (2.5) as a collision problem.

$$N_i = \{s_j : B_\alpha(s_j) \cap B_\alpha(s_i) \neq \emptyset\} \tag{3.7}$$

In other words, instead of searching for all points at a maximum distance 2α from s (eq. (2.5)), we search disks of radius α centered around each point ($B_\alpha(s_j)$) intersecting $B_\alpha(s_i)$. Figure 2 displays all collisions for a given point s_i as gray circles.

The formalism of eq. (3.7) has two advantages over the one of eq. (2.5). First, it can be solved efficiently on the GPU by using a *Linear Bounding Volume Hierarchy* (LBVH) [4]. Secondly, it guarantees the symmetry of the neighboring relation[1] **even for non-constant** α, which is not the case of eq. (2.5). We will exploit this in our next research to handle space-varying α, particularly useful for the *Particle Finite Element Method* [5].

3.2 Local Triangulations. The algorithm proposed in [9] allows to compute every cell of the Voronoi Diagram in parallel on the GPU. For a detailed explanation we encourage the reader to look at sections 3.1-3.3 of [9]. The idea is to initialize each cell with the Axis-Aligned Bounding-Box (AABB) of S and iteratively clip it by adding the bisector line of the segments linking point s_i and its neighbors N_i. **By reformulating some key steps of this algorithm, we are able to compute our local triangulations entirely in parallel.**

We adapt their algorithm as follows. First, [9] use a sorted K-Nearest Neighbor (K-NN) structure to compute N_i. As highlighted by the authors, the choice of the parameter K has an impact on both runtime and correctness of the algorithm. In our case we replace the K-NN by a LBVH. Practically this is equivalent to an automatic per-point fine-tuning of the number of neighbors K required for the α-shape. Secondly, the predicate employed in the original algorithm to decide if a line participates in the construction of the Voronoi cell is given by definition 3.1 and illustrated in fig. 4.

DEFINITION 3.1. *Considering the cell of point d and given 2 bisecting lines d_1, d_2 corresponding to neighbors e, f, does their intersection lie above, on or below the bisecting line d_3 corresponding to a new neighbor g ?*

The predicate given in definition 3.1 is equivalent to the classic `incircle`. Figure 4 gives a graphical intuition of the equivalence of the two problems, an interactive plot as well as the proof of equivalence can be found in the supplementary materials. Using this equivalence, we can reformulate the original algorithm to decide if a point participates in the local triangulation.

First, we add 3 points to the set S that we call "infinity points" and initialize each triangulation T_i by a triangulation of the 3 infinity points and s_i. This step is represented in top-left on fig. 5. The major part of the algorithm is then to perform the following iterations: for each neighbor s_j of point s_i (i) identify the circumcircles containing s_j (the *cavity* in Bowyer-Watson terms). (ii) Identify the boundary of this cavity. (iii) Re-triangulate the boundary. Once the iterations are done, we suppress the triangles violating the α-shape condition. The result is T_i, the set of locally α-Delaunay triangles with vertex s_i. Figure 5 shows the

[1] i.e. if i is neighbor to j then j is neighbor to i

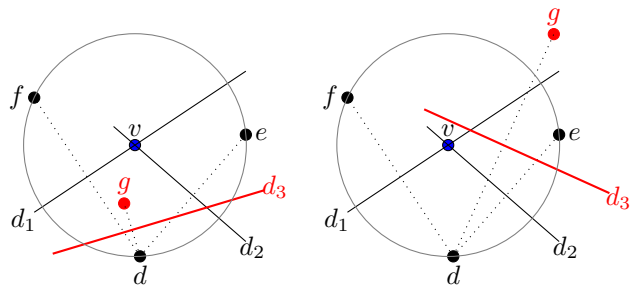


Figure 4: *Intersection-orient and incircle problem equivalence. (Left) point g is in the circle therefore v is above d_3. (Right) point g is out of the circle therefore v is below d_3*

initialization and an iteration of the algorithm applied on the neighbors of s taken from fig. 2. Whereas fig. 6 shows the suppression phase. The rest of the iterations can be found in the supplementary materials. Although performing a check of the α-shape condition at the end causes some additional work, **it actually allows us to perform automatic boundary detection**. Simply put, an edge is on the boundary if one of the triangles it belongs to have been removed during the last phase. This additional feature of the algorithm makes it even more attractive for the PFEM, but it could also be useful for surface rectonstruction.

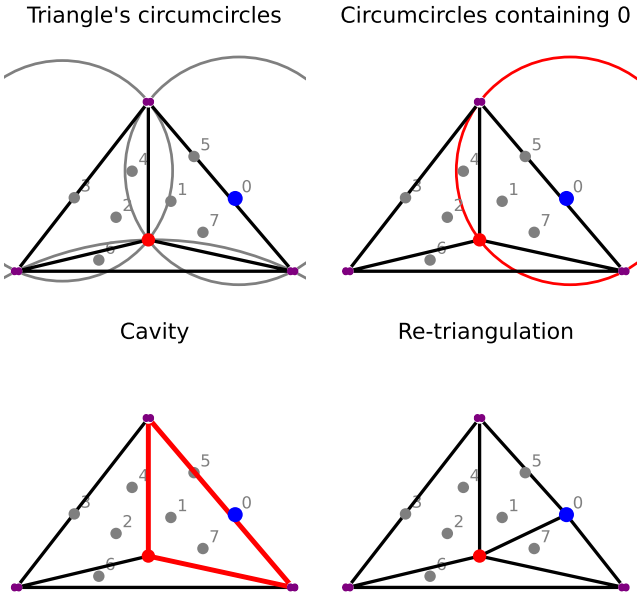


Figure 5: *Initialization of the local triangulation (top left, 3 triangles made by the infinity points and the red point) and insertion of point 0. The point set is extracted from fig. 2*

Figure 7 displays the time taken for our algorithm to compute the α-shape of a random point set S of

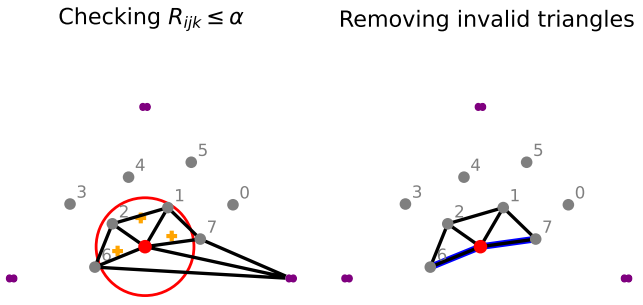


Figure 6: *Suppression of large triangles. As a bonus, we can detect boundaries (blue lines) on the fly.*

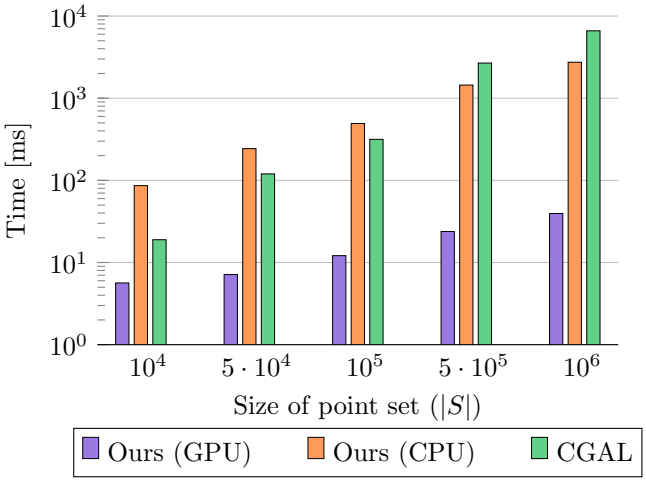


Figure 7: *Time to compute the α-shape of a set of uniformly distributed points in $[0,1]^2$, supposed to be in general position. This constitutes : array initialization, LBVH construction and traversal, computing T_i. Note the logarithmic axis.*

increasing size. The testcases are run on a laptop equipped with an RTX4060 and an Intel Ultra 7 155H, all values are averaged over 10 runs. We compare the runtime of our algorithm, both in GPU ( ) and CPU ( ) against the implementation of α-shape provided by CGAL [2]( ). We observe in this figure that **our GPU version is two orders of magnitude faster than CGAL for large point sets.** The code will be published once it has reached a more mature state, but can nonetheless be sent upon request to the authors.

Acknowledgements

This project has received funding from the European Research Council (ERC) under the European Union's Horizon research and innovation program (Grant agreement No. 101 071 255).

References

[1] M. Cremonesi, A. Franci, S. Idelsohn, and E. Oñate, *A state of the art review of the particle finite element method (pfem)*, Archives of Computational Methods in Engineering, 27 (2020), p. 1709–1735.

[2] T. K. F. Da, *2D alpha shapes*, in CGAL User and Reference Manual, CGAL Editorial Board, 6.0.1 ed., 2024.

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

[4] T. Karras, *Maximizing parallelism in the construction of bvhs, octrees, and k-d trees*, in Proceedings of the Fourth ACM SIGGRAPH / Eurographics conference on High-Performance Graphics, EGGH-HPG'12, Goslar, DEU, 2012, Eurographics Association, p. 33–37.

[5] T. Leyssens, M. Henry, J. Lambrechts, and J.-F. Remacle, *A delaunay refinement algorithm for the particle finite element method applied to free surface flows*, International Journal for Numerical Methods in Engineering, (2024), p. e7554.

[6] J. Liang, H. Edelsbrunner, P. Fu, P. V. Sudhakar, and S. Subramaniam, *Analytical shape computation of macromolecules: I. molecular area and volume through alpha shape*, Proteins: Structure, Function, and Bioinformatics, 33 (1998), p. 1–17.

[7] C. Marot, J. Pellerin, and J.-F. Remacle, *One machine, one minute, three billion tetrahedra*, (2018). arXiv:1805.08831.

[8] T. B. Masood, T. Ray, and V. Natarajan, *Parallel computation of alpha complex for biomolecules*, (2020). arXiv:1908.05944.

[9] N. Ray, D. Sokolov, S. Lefebvre, and B. Lévy, *Meshless voronoi on the gpu*, ACM Trans. Graph., 37 (2018), pp. 265:1–265:12.

[10] J. Vauhkonen, T. Tokola, P. Packalén, and M. Maltamo, *Identification of scandinavian commercial species of individual trees from airborne laser scanning data using alpha shape metrics*, Forest Science, 55 (2009), p. 37–47.