Largest triangulation and minimal enclosing triangle of a convex polygon

Dirksen Maxime, Dupuis Emma, Renard Simon

1 Introduction

The largest triangle in a polygon problem consists in finding the maximum-area triangle that can be contained in a given convex polygon in the plane. In order to solve this problem, we have used the article written by *Keikha, Löffer, Urhausen and van der Hoog*[1]. The minimum-area enclosing triangle problem requires finding the smallest triangle that contains a given convex polygon. *Pârvu, Ovidiu and Gilbert, David* [2] present an algorithm to compute such a triangle.

To illustrate these problems and their corresponding algorithms, we made an interactive website to create geometric artworks. The user draws some points, then the convex hull is computed, thus we obtain a convex polygon. The biggest inscribed triangle in this polygon is computed to be colored. When this triangle is colored, we can get at most 3 convex polygons on which we apply the same steps on which we apply the same steps until we only get colored triangles. In addition, the smallest triangle that contains the original polygon is plotted. An example of the result is shown in Figure 1.

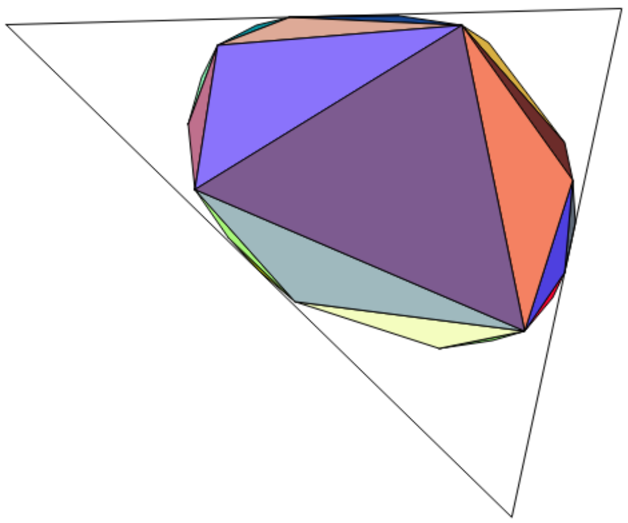


Figure 1: Artwork example

2 Maximum-area triangle in a convex polygon

2.1 Problem and main notions definitions

Let P be a convex polygon with n vertices. A triangle T is *P-aligned* if the vertices of T are a subset of P. All polygons have at least one P-aligned triangle because a polygon has at least 3 vertices. Let $a \in P$ be a root of T, every P-aligned triangle which contains this vertex a is said to be *rooted* on a. Two P-aligned triangles T1 and T2 are *interleaving* if between each vertex of $T1$, we can find a vertex of $T2$ by x-coordinate and not radially (if they share a vertex, it does not break the interleaving) [3].

In order to find the largest triangle $T = \triangle abc$ inscribed in P, where a, b, c are three (different) points of P, we need to find a 3-stable triangle. A triangle is *3-stable*, if by moving only one vertex, we never get a bigger triangle. Thus $max\,(\triangle abc, \triangle a'bc, \triangle ab'c, \triangle abc') = \triangle abc$. Every 3-stable triangle is also *2-stable* which mean that if we fix a root a of the triangle $\triangle abc$, then there are no bigger triangle $\triangle ab'c$ or $\triangle abc'$. We can say that b and c are stable [4].
By definition, the largest triangle inscribed in a convex polygon is 3-stable, thus also 2-stable.

2.2 Dobkin and Snyder's algorithm

Dobkin and Snyder gave an algorithm to find the maximum-area triangle inscribed in a convex polygon in $O(n)$ [4]. The algorithm 1 resumed Dobkin and Snyder's solution.

Algorithm 1: $O(n)$ Dobkin and Snyder's algorithm

```
Input   : $P$: convex polygon,$r$: a vertex of $P$
Output  : $T$: a triangle
Legend  : Operation next means the next vertex in clockwise order of $P$
$a \leftarrow r$
$b \leftarrow$ next($a$)
$c \leftarrow$ next($a$)
$m \leftarrow \triangle abc$
while True do
    while True do
        if $\triangle ab\textbf{next}(c) \geq \triangle abc$ then
            $c \leftarrow$ next($c$)
        end
        else if $\triangle a\textbf{next}(b)c \geq \triangle abc$ then
            $b \leftarrow$ next($b$)
        end
        else
            break
        end
    end
    if $\triangle abc \geq m$ then
        $m \leftarrow \triangle abc$
    end
    $a \leftarrow$ next($a$) if $a = r$ then
        return m
    end
    if $b = a$ then
        $b \leftarrow$ next($a$)
    end
    if $c = b$ then
        $c \leftarrow$ next($b$)
    end
end
```

The algorithm correctness assumes the correctness of several statements.

Statement 1. *If $\triangle abc > \triangle ab\textbf{next}(c)$ and $\triangle abc > \triangle a\textbf{next}(b)c$, then $\triangle abc$ is 2-stable.*

Statement 2. *For a given root a, there exists only one 2-stable triangle $\triangle abc$.*

Proof 1 (proof of Statement 1)**.** *Initially, we have $T = \triangle abc$ with $b = \textbf{next}(a)$, $c = \textbf{next}(b)$.
If Ts' area increases when we move the vertex c forward, the distance between the segment ab and the vertex c increases. After a few steps, we will get $c = c^*$ such that c^* is the vertex that maximizes the height. As we are in a convex polygon, we know that $\triangle abc^* > \triangle ab\textbf{next}(c^*)$.
After finding c^*, we can apply the same reasoning to find b^*. By applying these steps several times, we get a triangle $T = \triangle abc$ such that $b = \textbf{next}(a)$, $c = \textbf{next}(b)$. By construction, this triangle rooted on a is 2-stable.*

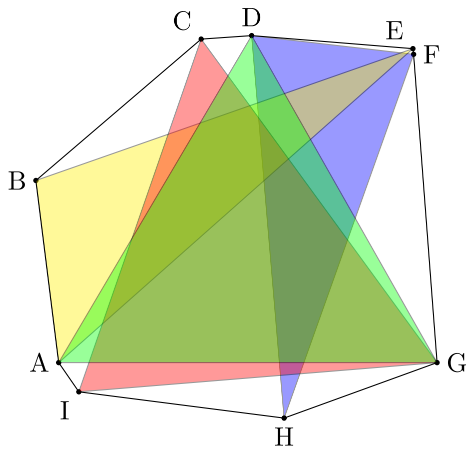


Figure 2: Counter example of Dobkin and Snyder's algorithm

We can demonstrate that the Statement 1 is true, but unfortunately Statement 2 is false. To prove this, we give in Section 2.2.1 a convex polygon such that, for a rooted vertex a, there exists two 2-stable triangles. More generally, the number of 2-stable triangles with a fixed root is bounded by $O(n)$ [1].

2.2.1 Counter example

Vahideh Keikha et al.[1] found a convex polygon such that the Algorithm 1 doesn't return the biggest inscribed triangle. The polygon is shown in Figure 2. Its vertices coordinates are: $A = (1000, 1000)$, $B = (759, 2927)$, $C = (2506, 4423)$, $D = (3040, 4460)$, $E = (4745, 4322)$, $F = (4752, 4262)$, $G = (5000, 1000)$, $H = (3383, 413)$, $I = (1213, 691)$.

With this polygon, Algorithm 1 found that the $\triangle CGI$ is the biggest. In reality, the biggest one is the triangle $\triangle ADG$. As it's the biggest, $\triangle ADG$ is 3-stable and so it's 2-stable whatever the root is.

Even if $\triangle ADG$ is 2-stable, Algorithm 1 will never find it. This mistake appears because for each vertex v of the real biggest triangle, there are two 2-stable triangles rooted on v and Algorithm 1 finds a smaller one than $\triangle ADG$. These 2-stable triangles rooted on the same vertices as $\triangle ADG$ are:

- When the root is A, $\triangle ABE$ (yellow)
- When the root is D, $\triangle DFH$ (blue)
- When the root is G, $\triangle GIC$ (red)

So, Algorithm 1 defined by *Dobkin and Snyder* is incorrect. In the next section will study another algorithm that is correct.

2.2.2 Biggest 4-gon

Dobkin and Snyder also defined an algorithm to find the biggest 4-gon in a convex polygon that runs in $O(n)$. This second algorithm is very similar to Algorithm 1. It consists of starting from an arbitrary root a, then defining b, c, d as the three next vertices after a. After that, we have to move forward d as long as the area of $abcd$ increases. Then, we do the same by moving c forward and then the same for b. We repeat these steps as long as at least one vertex is moved.

When neither b, c nor d can move forward, we move a forward and remember the current biggest 4-gon encountered. This is done until a steps on each vertex of P. As a is moved n times and b, c and d move at most $2n$ times, this algorithm runs in $O(n)$.

Unfortunately, like the Algorithm 1, this algorithm is not correct. A counter example has been found by *Vahideh Keikha et al.*[1] that is built similarly as it's done in 2.2.1.

2.3 $O(n^2)$ algorithm

The idea is to find for each vertex $p_i \in P$ all 2-stable triangles by moving the two other points, where p_i is the root. Moreover, we must find the largest 3-stable triangle among these 2-stable triangles, recall that the largest triangle is 3-stable and all 3-stable triangles are 2-stable. In contrast to the Dobkin and Snyder's algorithm, each time the root is moved, the two other points are reset [1].

The Algorithm 2 shows the pseudocode of the program presented in [1]. An error has been made in the second "while". Instead of having the condition **while** $c \neq a$ **do**, it should be **while next**$(c) \neq a$ **do**. Algorithm 2 contains the correction. Indeed, if the condition is $c = a$ in the most nested loop, c will never move as the area of $\triangle ab\mathbf{next}(c)$ will at some point take the value of $\triangle aba$ for which the area is equal to 0. Thus, the condition $\triangle aba \geq \triangle abc$ cannot be true for any value of b, thus the code will loop infinitely.

Algorithm 2: $O(n^2)$ algorithm

```
Input    : $P = p_0, p_1, ..., p_n$: convex polygon
Output   : $T = p_a, p_b, p_c$: largest P-aligned triangle
Legend   : Operation next means the next vertex in clockwise order of $P$
$a \leftarrow p_0$
$b \leftarrow$ next$(a)$
$c \leftarrow$ next$(b)$
$m \leftarrow \triangle abc$
while $True$ do
    while next$(c) \neq a$ do
        while $\triangle ab$next$(c) \geq \triangle abc$ do
            $c \leftarrow$ next$(c)$
        end
        if $\triangle abc \geq m$ then
            $m \leftarrow \triangle abc$
        end
        $b \leftarrow$ next$(b)$
    end
    $a \leftarrow$ next$(a)$
    if $a = p_0$ then
        return m
    end
    $b \leftarrow$ next$(a)$
    $c \leftarrow$ next$(b)$
end
```

We can analyze the Algorithm 2 by breaking down the three while loops :

- **while** $\triangle ab\mathbf{next}(c) \geq \triangle abc$ **do**: tries to move c forward as much as possible to increase the area.
- **while next**$(c) \neq a$ **do**: we are sure that all 2-stable triangles have been found for the root a as we cannot move c in order to find a larger triangle. After each c augmenting loop, b moves to try to move c further successfully. — These two loops have a total complexity of $O(n)$ because we can shift c of at most $n-1$ positions and b can't be moved more than c.
- **while** $True$ **do**: set iteratively each vertex as the root of a 2-stable $\triangle abc$ triangle, where a is the root. — $O(n)$, total : $O(n^2)$

In this way, all 2-stable triangles are found due to the fact that all 3-stable triangle are 2-stable. We can therefore find the largest triangle contained in a convex polygon [1].

2.4 Largest triangulation

Our aim is to perform the largest triangulation, we just need to recursively run the Algorithm 2 on each new polygon created by removing the largest triangle found in the actual polygon. There are at most 3 recursions, the figure 3 takes up the 4 possible cases after the algorithm has found the largest triangle. If the new created polygons are triangles, there is no need to execute the largest triangle algorithm on them.

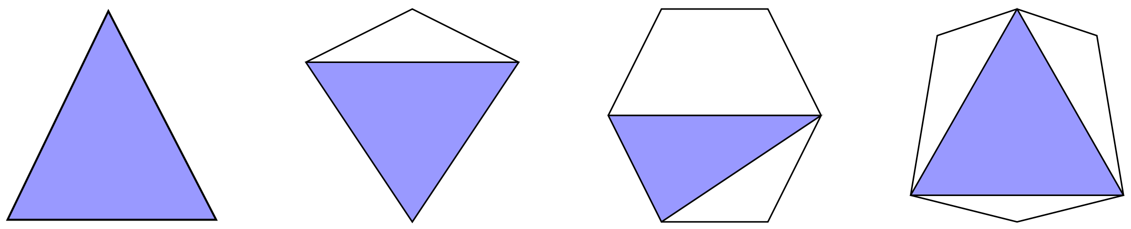
Let n be the number of vertices of a convex polygon P. The largest triangle, divide the boundary of P into 3 intervals of vertices. Note that an interval is never empty, it at least contains two triangle's vertices. And an interval can at most contains $n-1$ vertices. So, we recursively call the algorithm on the sub-polygons formed by the intervals. Note that it's pointless to do this if the interval is less than or equal to 3, i.e. when the triangle is immediately find or when there is no sub-polygon created (the size of the interval is equal to 2 when the boundary of the triangle is the boundary of the polygon).

The procedure can be expressed by the following equation, let a, b, c be the vertices of the largest triangle and n the total number of vertices of the current polygon:

$$T(n) = O(n^2) + T(\alpha) + T(\beta) + T(\gamma)$$

where $\alpha \in \{\#$ vertices in the interval $[a, b]\}$, $\beta \in \{\#$ vertices in the interval $[b, c]\}$, $\gamma \in \{\#$ vertices in the interval $[c, a]\}$. The recursion stop when $n = 3$ or $n = 2$, then $T(3) = 1$ and $T(2) = 1$. Note that $\alpha + \beta + \gamma = n + 3$.

Hence, the total running time of our custom triangulation is bounded by $O(n^2 \log n)$.



(a) 0 new polygon created

(b) 1 new polygon created

(c) 2 new polygons created

(d) 3 new polygons created

Figure 3: The 4 cases of polygons created by removing the largest triangle. In blue the largest triangle to remove from the polygon, in white the polygons that will be created (not necessarily triangles).

2.5 $O(n \log n)$ algorithm

The general idea of the Divide-and-Conquer algorithm of [1] is to reduce the polygon to the largest triangle. It rests on the two following lemmas :

Lemma 1. *([3], Lemma 2.2) A globally largest-area k-gon and a larest-area rooted k-gon interleave.*

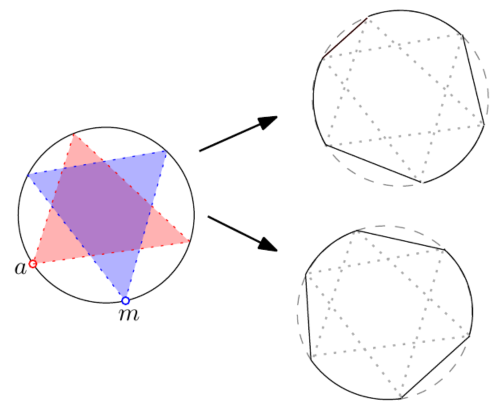
Lemma 2. *([1], Lemma 9) The largest-area root triangle can be found in linear time.*

The proof of this second lemma have been explained in the subsection 2.3.

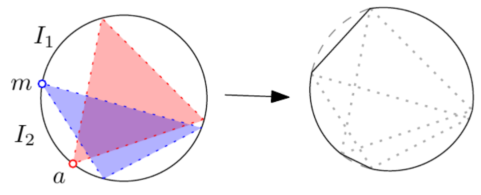
Let a be an arbitrary vertex of P and let T_a be the largest triangle rooted at a. As we saw in the figure 3, the largest triangle decomposes the boundary of a polygon in many intervals. Let m be the median vertex of the interval with the most vertex, and T_m be the largest triangle rooted at m. By the Lemma 1, the largest triangle that we are looking for interleaves with T_a and T_m.

This means that the 3 points of the largest triangle are in 3 of these intervals (3 distinct). Because 3 intervals (figure 3) are created by T_a (an interval contains at least two of the triangle's vertices), there are exactly 6 intervals create by T_a and T_m (note: if T_a and T_m overlaps on some vertex, this creates an interval with only this point).

Case 1 : T_a and T_m interleave, since the largest triangle vertices are in 3 intervals, if we remove the 3 unused interval, there is only 2 different possible set of intervals that can be chosen in order to form a new polygon which contains T_a, T_m and the largest triangle which all interleaves. Let P' and P'' be these two polygons. The figure 4 a) show this case.



(a) Interleaving dividing triangles construct two subproblems.



(b) Non-interleaving dividing triangles construct one subproblem.

Figure 4: ([1], Figure 9) Dividing triangles and the resulting subproblems.

Case 2 : T_a and T_m don't interleave, there are therefore just one configuration where the largest triangle interleave with the two others, as shown in Figure 3 b).

If we are in case 1, the algorithm will be recursively executed on P' and P''. And only on the single polygon created in the case 2. We repeat the procedure until there is only 3 vertices in the polygon, which mean that the largest triangle have been found for the previous sub-polygon and if the calls are traced back, it is only when the recursion have been applied on P' and P'' (case 1) that we have to only keep the largest triangle, because we found two largest triangles but for two different sub-polygon. So at the end of the calls back, we obtain the largest triangle for the initial polygon.

Algorithm 3: Divide-and-Conquer $O(n \log n)$ algorithm

```
Procedure: LARGEST-TRIANGLE($P$)
Input    : $P = p_0, p_1, ..., p_n$: convex polygon
Output   : $T = p_a, p_b, p_c$: largest P-aligned triangle
if $|P| = 3$ then
    return $P$
end
$a \leftarrow$ arbitrary vertex of $P$
$T_a \leftarrow$ largest-area triangle rooted at $a$
$m \leftarrow$ median point on the largest interval on $P$ between two vertices of $T_a$
$T_m \leftarrow$ largest-area triangle rooted at $m$
$P', P'' \leftarrow$ sub-polygons constructed by interleaving intervals using $T_a$ and $T_m$
if $T_a$ and $T_m$ are interleaving then
    return max(LARGEST-TRIANGLE($P'$), LARGEST-TRIANGLE($P''$))
end
else if $P'$ can include the largest-area triangle then
    return LARGEST-TRIANGLE($P'$)
end
else
    return LARGEST-TRIANGLE($P''$)
end
```

This algorithm performs the largest triangle finding problem in $O(n \log n)$.

Lemma 3. *([1], Lemma 10) Let P be a convex polygon with n vertices. The (one or two) subproblems induced by P have size at most $\frac{5}{6}(n+6)$.*

Without redoing the proof of the lemma 3 [1], the idea is that T_a and T_m decompose P into 6 intervals. Hence, the two possible sub-polygon P' P'' have therefore a size between $\frac{1}{6}$ and $\frac{5}{6}$ times the number of vertices of the actual polygon. Furthermore, if we are in the case 1, the total number of vertices be equals to $|P'| + |P''| = |P| + 6$ (see the figure 4 to be convinced).

We now have all what we need to compute the Divide-and-Conquer equation :

$$T(n) = max\{T(\alpha(n+6)) + T((1-\alpha)(n+6)) + O(n+6), T(\alpha(n+6) + O(n+6))\}$$

Where α is the size of the created sub-polygon : by the lemma 3 we know that $\frac{1}{6} <= \alpha <= \frac{5}{6}$. The recursion stop when $|P| = 3$ then we have $T(3) = 1$. By using mathematical artifice of Akra and Bazzi [5], the equation can be rewritten as

$$T(m) = T(\alpha m) + T((1-\alpha)m) + m$$

m is in $O(n)$ which leads to an upper bound $O(n \log n)$ for $T(n)$. [1]

3 Minimum-area enclosing triangle of a convex polygon

3.1 Problem and main notions and definitions

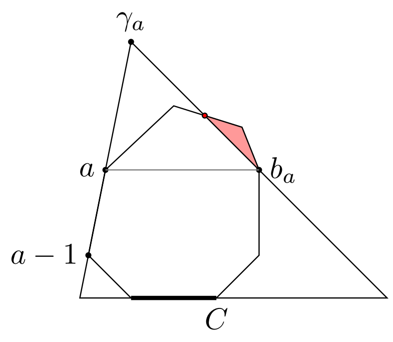
Let P be a convex polygon with n vertices. A triangle T is a *minimum-area enclosing triangle* if T corresponds to the smallest area triangle containing every vertex of P. This triangle T is represented by three vertices and three sides. vertexA, vertexB, vertexC $\in$ vertices of T and A, B, C $\in$ sides of T. $a, b\&c$ are the indexes used to represent the vertices used to form the triangles' sides. $vertex + 1$ means the next vertex of P in clockwise order.

A side S of T is said to be *flush* with an edge e of P if $S \supseteq e$. A side S of T is said to be *tangent* to the polygon P on vertex v if $v \in e$ where e is an edge of the polygon.

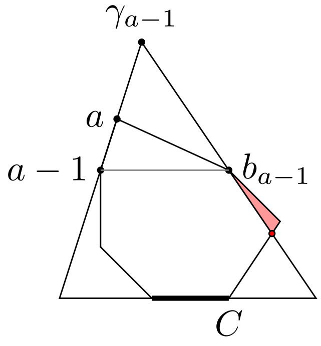
Let $h(p)$ be the distance between p and side C. The *right chain* is the set of vertices of P where $h(p) > h(p+1)$. The *left chain* is the set of the remainder of vertices from P. γ_p is the point on A flush to $[a, a-1]$ such that $h(\gamma_p) = 2h(p)$ on the ray $[a-1, a)$.

Let's assume a is a vertex of P on the left chain and b_a is a vertex of P on the right chain with $h(b_a) \geq h(a)$. The edge $[a-1, a]$ is said to be *low* if $\gamma_a b_a$ intersects P above b_a (Figure 5a), *high* if $\gamma_{a-1} b_{a-1}$ intersects P below b_{a-1} (Figure 5b) and *critical* if it is neither low nor high.

Note To avoid finding b_a, we can also say that if $h(b) > h(a)$ and $\gamma_a b$ cuts P above or tangent to b, then $[a-1, a]$ is low. Moreover, if $h(b) > h(a)$ and $\gamma_a b$ cuts P below b , then $[b-1, b]$ is high.



(a) Edge $[a, a-1]$ is low



(b) Edge $[a, a-1]$ is high

Figure 5: High/Low examples

3.2 Important theorems

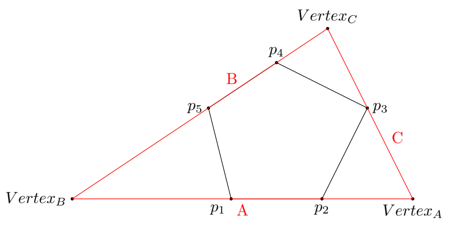
We base our algorithms on several important theorems introduced in [2].

Theorem 1. *The midpoint M of each side of a minimum-area enclosing triangle must touch the polygon.*

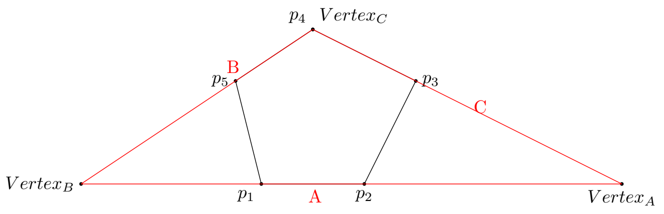
Theorem 2. *For any P, a minimum-area enclosing triangle has at least two sides are flush and a third side which can be either flush or tangent to P.*

3.3 Brute force ($O(n^3)$)

One way of finding this minimum-area enclosing triangle T is by brute force, seen in Algorithm 4. To do this, we analyze each triangle possible following theorems 1 and 2. Each one of these triangles are found by first choosing two vertices i and j of the polygon to form the two first flush sides: A flush to $[i, i+1]$ and B flush to $[j, j+1]$ of the polygon. We then choose the last vertex k of the polygon to be either the midpoint of the tangent side or the flush side of the triangle to the polygon, depending on the area of these triangles.



(a) Example of triangle with tangent side



(b) Example of triangle with 3 flush sides

Figure 6: 2 types of possible enclosing triangles

Algorithm 4: Brute force algorithm

```
Input  : $P = p_1, p_2, ..., p_n$: convex polygon
Output : $T$: the minimum enclosing triangle
Legend : Operation **next** means the next vertex in clockwise order of $P$
for $1 \leq i, j, k \leq n$ do
    $A \leftarrow [p_i, next(p_i)]$
    $B \leftarrow [p_j, next(p_j)]$
    if $A \neq B$ then
        if $k \notin A \wedge k \notin B$ then
            $flushTriangle \leftarrow getTriangleAllFlushSides(A, B, k)$
            $tangentTriangle \leftarrow getTriangleWithTangentSide(A, B, k)$
            $minEnclosingTriangle \leftarrow$
              $min(minEnclosingTriangle, flushTriangle, tangentTriangle)$
        end
    end
end
return $minEnclosingTriangle$;
```

Algorithm 5: getTriangleAllFlushSides

```
Input  : A, B, k: Flush sides A & B, vertex k of polygon
Output : $T$: the enclosing triangle
Legend : Operation **next** means the next vertex in clockwise order of $P$
$C \leftarrow [k, next(k)]$
$vertexA \leftarrow intersection(A, B)$
$vertexB \leftarrow intersection(B, C)$
$vertexC \leftarrow intersection(C, A)$
return Triangle(vertexA, vertexB, vertexC)
```

Finding the tangent side

Let A and B be two flush sides with known equations. We must find the third side C of the triangle tangent to P's vertex k as seen in Figure 6a. Theorem 1 indicates that k must be the midpoint of C, as it is the only intersection between C and P. We then know that the distance between the x coordinate of $vertex_A$ and the x coordinate of p_3 is the same as the distance between the x coordinate of $vertex_C$ and the x coordinate of p_3. This is the same for the y coordinate. We can now solve this problem with a two equations system to find the equation of $vertex_C$ where $vertex_C$ is tangent to the polygon by its midpoint.

3.4 O'Rourke et al.'s algorithm ($\theta(n)$)

This algorithm [6] will use the same theorems (1 and 2) seen for the brute force algorithm. Instead of finding all combinations of possible sides of the triangle, for each side C, the algorithm will find the second flush side and set the third flush/tangent side, forming the minimum enclosing triangle having this fixed flush side C.

Finding γ

We know C and $h(\gamma_p) = 2h(p)$. To find γ_p, we use a line parallel and at a distance $2h(p)$ of C. γ_p is the intersection of the line and the ray $[a-1, a)$.

The algorithm

The main algorithm is a loop on each vertex c of P to set the first flush side C. C is set to $[c, c-1]$. We set $a = 2$ and $b = 3$. We now want to find the minimum enclosing triangle containing this flush side C.

Finding the positions of a, b & c First, we want to position our index points $a, b\&c$ such that these points can form the three sides of the minimum enclosing triangle. To do this, we move the index b to the right chain. This results in advancing b of one position as long as $h(b+1) > h(b)$. We then, adjust the positions of a and b such that both are either critical or high as we do not want T to pass through P. This means that, until b is as far as a from c, we advance a if the edge $[a, a-1]$ is low and advance b if the edge $[b, b-1]$ is high. $[a-1, a]$ is now high or critical. To find the tangency of side B, we keep advancing b as long as $[b-1, b]$ is high and $h(b) \geq h(a-1)$. Once this is done, tangency for side B has been achieved.

Updating sides We can now define the sides A and C by computing their vertices, where A is flush with $[a, a-1]$ and C is flush with $[c, c-1]$. If we were not able to find a tangency of side B, both sides A and B must be updated. However, if we were, only side B must be updated.

Updating sides A and B means that we do not find any possible triangles with a tangent B side. Thus, we must set B as our second flush side instead of A and set A as the third flush or tangent side. To do this, we must compute the middle point of side B. As B is set to be flush to $[b, b-1]$, we can compute side $B = [vertex_A, vertex_C]$ where $vertex_A = intersection(B, C)$ and $vertex_C = intersection(A, B)$. We can determine the nature of side A to P (flush or tangent) by looking at the distance of $vertex_{a-1}$ and Bs' midpoint to $]c, c-1[$ as we want to find the minimum area enclosing triangle. In other words, if $h(Bs'midpoint) < h(a-1)$, side A is tangent to P by the $vertex_{a-1}$ or else side A is flush to P. B and C are both flush sides to P.

Updating side B indicates that a tangency for B has been found. A and C may remain our two flush sides. To set our final side B, we first want to find the intersection between A and B which is $vertex_C$. As B is tangent to P in a vertex v, we know B's midpoint is v. Using the same method as finding γ_v, $vertex_C$ is the intersection of the ray $[a-1, a) \in A$ with the line parallel and at a distance $2h(v)$ of C.

Defining the local minimal enclosing triangle Finally, we must compute our found triangle's area. The minimal area enclosing triangle is the smallest local minimal enclosing triangle found.

Time Complexity

All we do is increment our three indexes on the polygon used to form our triangle. Thus, this algorithm executes in $\theta(3n) = \theta(n)$ steps.

Algorithm 6: O'Rourke et al.'s algorithm

```
Input   : $P = p_1, p_2, ..., p_n$: convex polygon
Output  : $T$: the minimum enclosing triangle
Legend  : Operation next means the next vertex in clockwise order of $P$
$a \leftarrow 1$
$b \leftarrow 2$
for $c \leftarrow 1, n$ do
    //Finding the positions of a,b&c
    while $h(next(b)) \geq h(b)$ do
        $b \leftarrow next(b)$
    end
    while $h(b) > h(a)$ do
        if isHigh(b) then
            $b \leftarrow next(b)$
        end
        else
            $a \leftarrow next(a)$
        end
    end
    while isHigh(b) and $h(b) > h(a)$ do
        $b \leftarrow next(b)$
    end
    //Updating sides
    if $tangencyFound(B)$ then
        UpdateSidesAB()
    end
    else
        UpdateSideB()
    end
    //Defining minimal enclosing triangle
    if $Area(ABC) \leq Area(currentEnclosingTriangle)$ then
        $minEnclosingTriangle \leftarrow ABC$
    end
end
return $minEnclosingTriangle$
```

4 Implementation

An implementation of our work is available online here `https://sirenard.github.io/maximalTriangulationArt`. On the main page, you can draw your own artwork from your polygon and if you wish, you can visualize step by step the triangulation. You also have access to another web page, on which you can see each steps of Algorithm 1 on a custom polygon or on the polygon presented in Section 2.2.1 to convince you of the non correctness of this algorithm.

5 Work contribution

- Simon: subsection 2.2, understanding the proofs of the theorems exposed in subsection3.2 and implementation of the Algorithm 1
- Maxime: subsections 2.3, 2.4, 2.5 and implementation of the Algorithm 2.
- Emma: section 3 and the implementation of the Algorithm 4

References

[1] V. Keikha, M. Löffler, J. Urhausen, and I. van der Hoog, "Maximum-area triangle in a convex polygon, revisited," *CoRR*, vol. abs/1705.11035, 2017. [Online]. Available: http://arxiv.org/abs/1705.11035

[2] O. Pârvu and D. Gilbert, “Implementation of linear minimum area enclosing triangle algorithm,” *Computational and Applied Mathematics*, vol. 35, no. 2, pp. 423–438, 2016.

[3] J. E. Boyce, D. P. Dobkin, R. L. S. Drysdale, and L. J. Guibas, “Finding extermal polygons,” *SIAM Journal on Computing*, vol. 14, no. 1, pp. 134–147, 1985. [Online]. Available: https://dx.doi.org/10.1137/0214011

[4] D. P. Dobkin and L. Snyder, “On a general method for maximizing and minimizing among certain geometric problems,” *20th Annual Symposium on Foundations of Computer Science*, pp. 9–17, 1979. [Online]. Available: https://dx.doi.org/10.1109/SFCS.1979.28

[5] M. Akra and L. Bazzi, “On the solution of linear recurrence equations.” *Computational Optimization and Applications 10*, vol. 10, no. 1, pp. 195—-210, 1998. [Online]. Available: https://doi.org/10.1023/A:1018373005182

[6] J. O’Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, “An optimal algorithm for finding minimal enclosing triangles,” *Journal of Algorithms*, vol. 7, no. 2, pp. 258–269, 1986. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0196677486900076