Exercise 12

Discrete and Algorithmic Geometry - MAMME - UPC

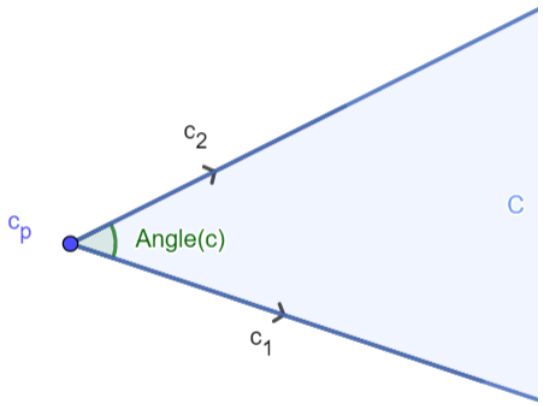
Ernest Sorinas, Xavi Arnal

January 21, 2022

Convention and notation

We will consider all angles to be in $\mathbb{R}/2\pi\mathbb{R}$. For some vector v, $Angle(v)$ denotes the angle of v counterclockwise from $(1, 0)$. For any two vectors (v, u), $\text{Angle}(v, u)$ is the angle between v and u, counterclockwise in that order, which can be defined as $\text{Angle}(u) - \text{Angle}(v)$.

We will use $\langle v\rangle^+$ to denote the set of positive multiples of v. This is useful to denote rays — for instance, $p + \langle v\rangle^+$ is the ray starting at p going in the direction of v. A cone is a plane shape of the form $c = p + \langle v_1\rangle^+ + \langle v_2\rangle^+$. We will say that c is fixed at p, which we will denote as c_p for some arbitrary cone. Likewise, we will denote by $\{c_1, c_2\}$ the vectors corresponding $\{v_1, v_2\}$, but always relabeling such that (c_1, c_2) is positively oriented (meaning that $\det(c1\ c2) > 0$). We define that $\text{Angle}(c) = \text{Angle}(c_1, c_2)$.



We say that some point in the plane p *sees* some plane shape S with angle α iff there exists some cone K fixed at p, with $\text{Angle}(K) \leq \alpha$, such that $S \subseteq K$. We will also say that p *sees* some $s \in \partial S$ iff the segment joining p and s does not intersect S (other than in s).

Preliminary results

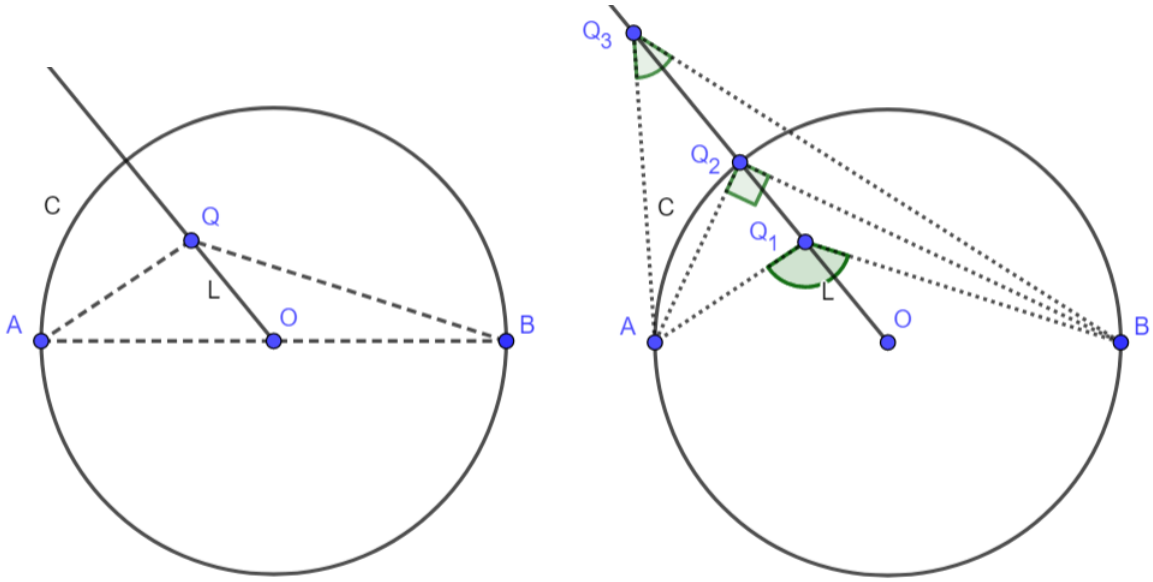
We will begin by giving a couple of results, which will be necessary for this section. The first is a very slight strengthening of Thale's theoremn:

Proposition 1. *Let A, B, P be three non-colinear points in the plane, and let D be the unique disk having AB as a diameter, and $C = \partial D$ its boundary. Then:*

- *If $P \in D^o$, then $\angle APB < \frac{\pi}{2}$.*
- *If $P \in C$, then $\angle APB = \frac{\pi}{2}$.*

- *If $P \notin D$, then $\angle APB > \frac{\pi}{2}$.*

Proof. Let O be the center of D, and consider any ray l from O, not going through A or B. Let Q be an arbitrary point in L. Note that both $\angle BAQ$ $(= \angle OAQ)$ and $\angle QBA$ $(= \angle QBO)$ are larger the farther away Q is from O, meaning that $\angle AQB$ is *smaller* the farther away Q is from O. Moreover, if $Q \in C$, then $\angle AQB = \frac{\pi}{2}$, by Thale's theorem. From these two facts, it follows that $\angle AQB$ is (strictly) greater or smaller than $\frac{\pi}{2}$ depending on whether Q appears in L before or after its intersection with C, that is, whether Q is in D^o or outside D.



This shows that the proposition holds for Q in L — to extend it to any arbitrary P (as long as it isn't co-linear with A, B), one need only pick L such that it goes through P. □

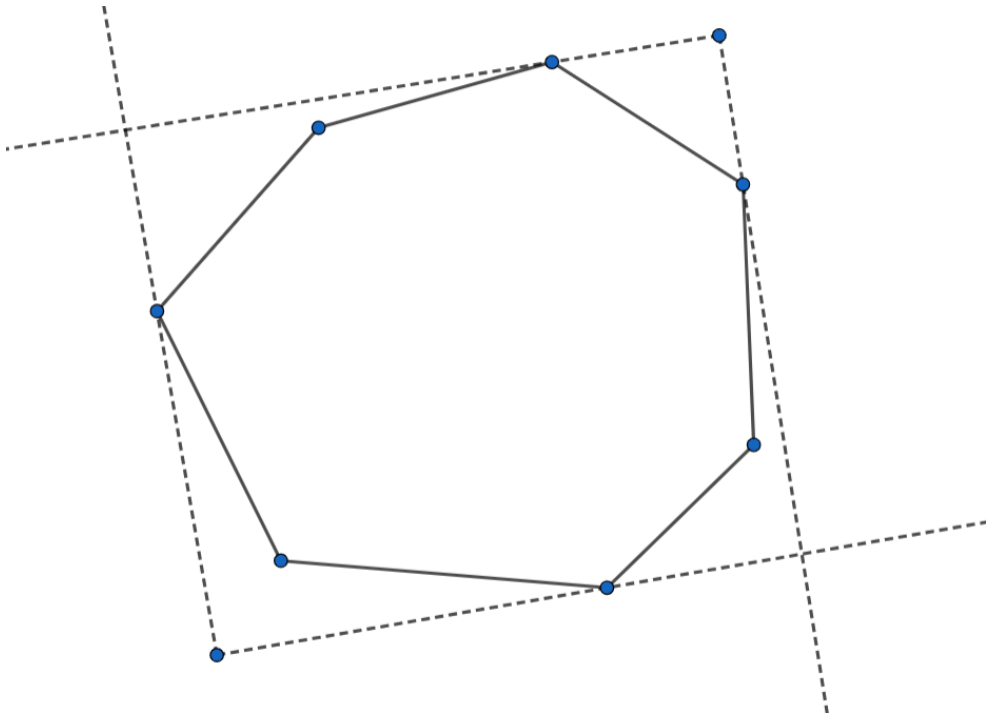
The second statement is to do with cones. We will not prove this, as the statement is fairly intuitive but does not seem to admit a non-cumbersome proof.

Proposition 2. *Let A and B be cones, both fixed at some point p, such that $A \subseteq B$. Then, Angle$(A) \leq$ Angle(B). If the inclusion is strict, so is the inequality (and vise versa).*

We are now equipped to start the exercise:

> Let Q be a convex polygon in the plane
>
> a) Study and describe the locus L of all points in the plane that see Q with a right angle

Let $v_1, \ldots, v_n$ be the counterclockwise-ordered vertices of Q. For each pair i, j with $1 \leq i < j \leq n$ let $C_{i,j}$ be the (unique) circumference with diameter $d(v_i, v_j)$ that passes through v_i and v_j, and let $D_{i,j}$ be the disk resulting from filling in this circumference (such that $\partial D_{i,j} = S_{i,j}$).



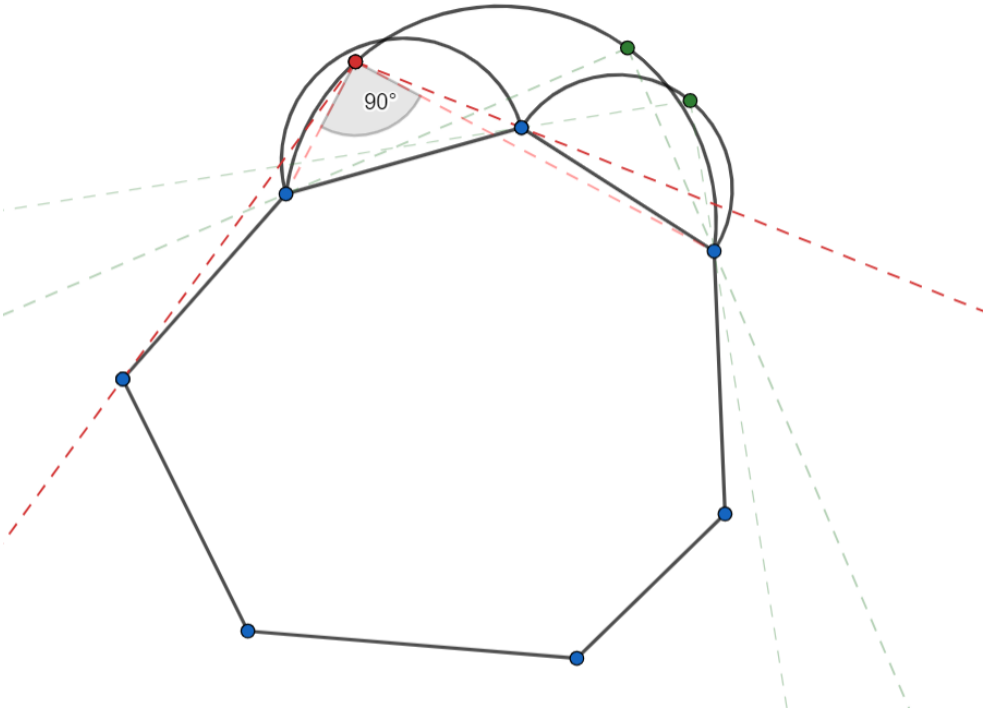
Claim. All points lie outside L if and only if they lie in the interior of some $D_{i,j}$, i.e. $L^C = \bigcup_{i,j} D^o_{i,j}$.

Proof. Let p be a point in the plane that is not inside Q. We know that the supporting rays from p to Q pass through two vertices of Q, say v_a and v_b. Let $K = p + \langle v_a - p\rangle^+ + \langle v_b - p\rangle^+$ — by definition, this is a cone containing Q. Note that $\alpha := \text{Angle}(K)$ is the smallest possible angle with which p can see Q — so $p \notin$ Ł $\iff \alpha > \frac{\pi}{2}$. We would like to show that this is equivalent to $p \in \bigcup_{i,j} D^o_{i,j}$. We cover both implications:

- Assume that $\alpha > \frac{\pi}{2}$. By Proposition 1, this means that $p \in D^o_{i,j} \subseteq \bigcup_{i,j} D^o_{i,j}$.
- Assume that $p \in \bigcup_{i,j} D^o_{i,j}$. This means that there must be must be some i, j such that $p \in D^o_{i,j}$, and therefore the angle $\angle v_i p v_j$ must be greater than $\frac{\pi}{2}$. However $K' = p + \langle v_i - p\rangle^+ + \langle v_j - p\rangle^+$ — clearly $\text{Angle}(K') = \angle v_i p v_j$, but because v_i, v_j are vertices of Q, we have that $K' \subseteq K \implies \text{Angle}(K') \leq \alpha \implies \frac{\pi}{2} \leq \alpha$. □

Moreover, we claim ∂L (i.e. the points that see Q with exactly than $\frac{\pi}{2}$ radians, and the vertices of acute angles of Q) lies in the outer border of $\bigcup_{i,j} D_{i,j}$. This is because, the moment that there is an intersection in the outer border, then the points that define the supporting lines change.

This can be seen in the following image. There, the green points are in L but the red one is not, because even though it sees two vertices with a right angle, these two vertices are not the ones that define the supporting line.



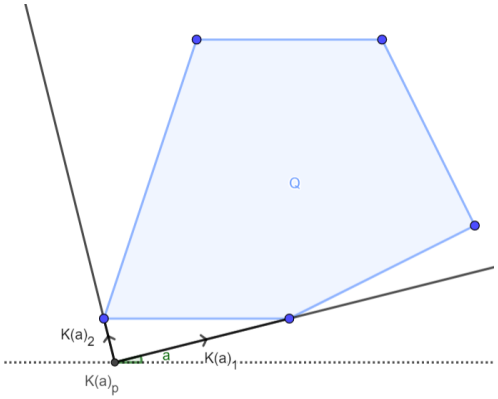
Our conclusion is that the points of ∂L are exactly the boundary of the union of the circles $D_{i,j}$.

b) Propose an algorithm to compute L

We are asked to find an algorithm to describe L. Of course, there is some ambiguity here, as we will have to agree on some representation of L — for this, recall that ∂L will be composed of a series of circular arcs. We will give an algorithm that describes these arcs - specifically, it will provide their centers and extremes. However, there is some ambiguity as to whether this is a "good" description of L, since it doesn't seem to facilitate any useful type of computation (e.g. testing inclusion).

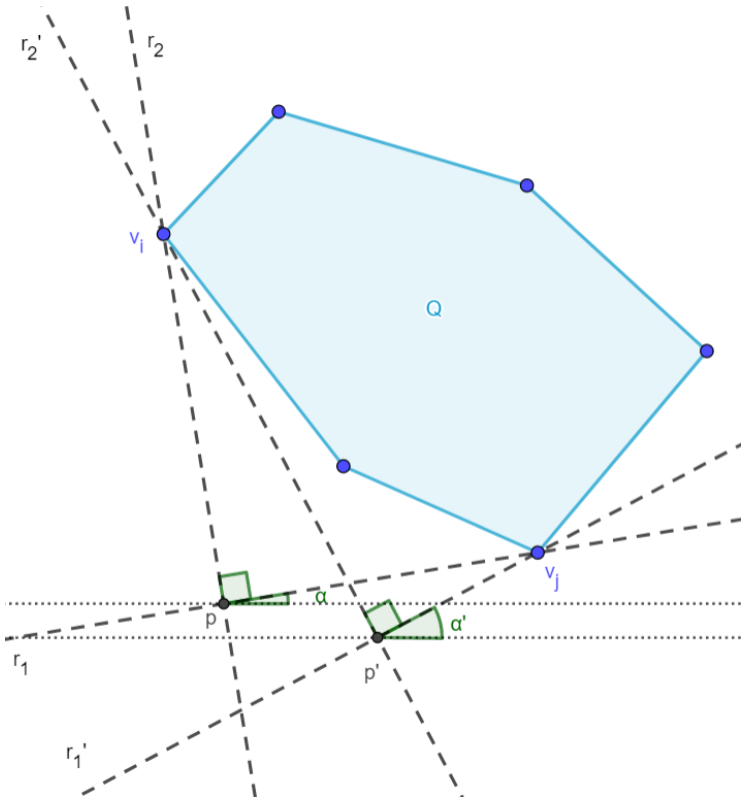
Parametrizing ∂L

Let α be some angle. Let $\overline{K}(\alpha)$ denote the set of cones k, satisfying $Q \subseteq k$ and $\text{Angle}(k) = \frac{\pi}{2}$, with $\text{Angle}(k_1) = \alpha$. Note that this set is closed under intersection, meaning that it has a unique minimal element with respect to inclusion. Let $K(\alpha)$ denote this minimal element. Then, $\alpha \mapsto K(\alpha)_p$ acts as a function parametrizing ∂L — it is precisely the set of points p seeing Q at a right angle that cannot be brought any closer to Q (by inclusion-minimality). So it would be useful to know, for any α, the value of $p := K(\alpha)_p$. Denote by $r_1 := p + \langle K(\alpha)_1 \rangle$, and likewise for r_2.



Note that r_1 and r_2 must[1] be tangent to Q. So if we want to find r_1, it is enough to find a line of the form $p+\langle v_1\rangle$, with $\text{Angle}(v_1) = \alpha$, tangent to Q, such that Q is oriented positively with respect to $(p, p+r_1)$.

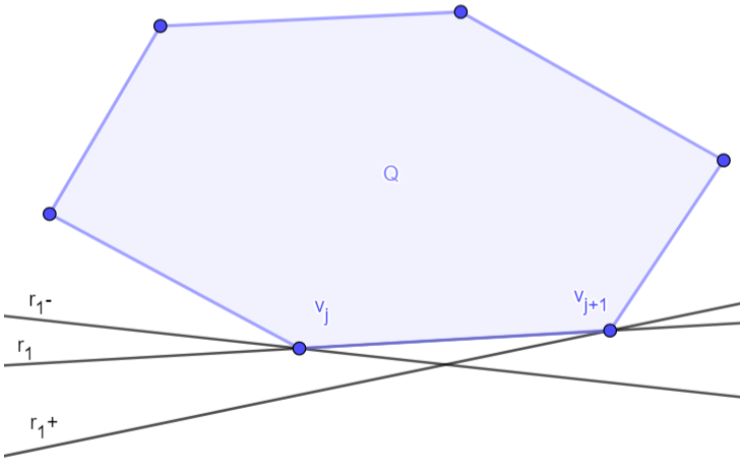
Similarly for r_2: we will need to find a line of the form $p + \langle v_2\rangle$, with $\text{Angle}(v_2) = \alpha + \frac{\pi}{2}$, tangent to Q, such that Q is oriented negatively with respect to $(p, p + r_1)$.
The key observation is as follows: Suppose we have found r_1, r_2 for some α, and they are tangent to Q, including some vertices v_i, v_j respectively. Then (unless r_1 or r_2 are parallel to some edge in Q), for $\alpha' > \alpha$, with $\alpha' - \alpha$ sufficiently small, finding r_1', r_2' will be really easy — they will be tangent to Q at the same vertices v_i, v_j.



This corresponds to $K(\alpha)_p$ and $K(\alpha')_p$ both belonging to C_{ij}, in particular to the arc of C_{ij} included in ∂L. For the moment, we will not concern ourselves with exactly *what* arc of C_{ij} is included in ∂L, but rather what circumferences C_{ab} appear at all in ∂L.

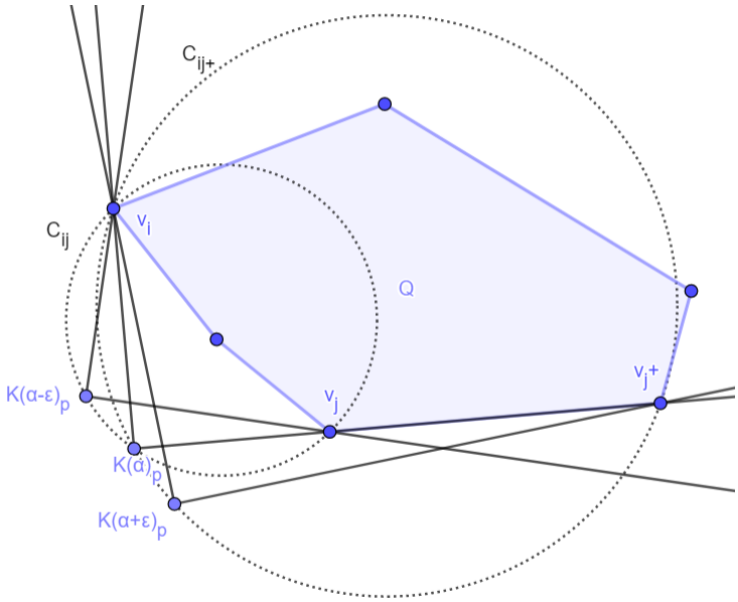
[1] otherwise, they could be moved closer to Q, defining a different element from $\overline{K}(\alpha)$ included in $K(\alpha)$ — this is a contradiction.

We have noted that C_{ij} appearing in ∂L is verified by the existence of a perpendicular pair of lines, r_1 and r_2, tangent to Q at v_i and v_j respectively. Moreover, varying the angle of r_1 and r_2 slightly, while keeping them tangent to Q, will not change the vertices that they are tangent at (unless r_1 or r_2 are tangent through an entire edge). Note aswell that if we increment the angle of r_1, eventually it will switch its tangent point from v_j to v_j^+:

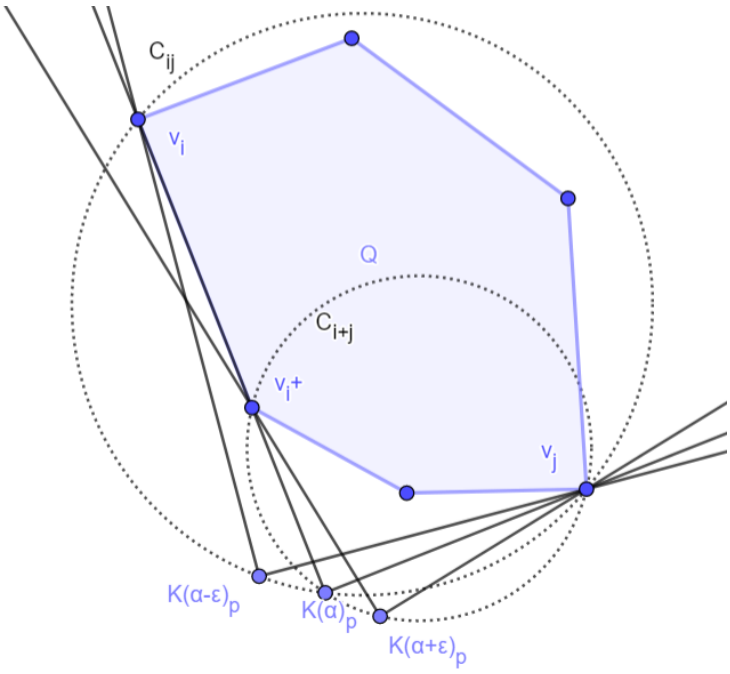


And likewise for r_2 and v_i^+. Instead of concerning ourselves with at what specific angle this changes occurr, we will ask which change occurs first, because this will tell us that the next (counterclockwise) arc to appear in ∂L is from

- $C_{i,j+1}$, if r_1 switches from v_j to v_j^+ before r_2 switches from v_i to v_i^+:



- C_{i+1}, j, otherwise:



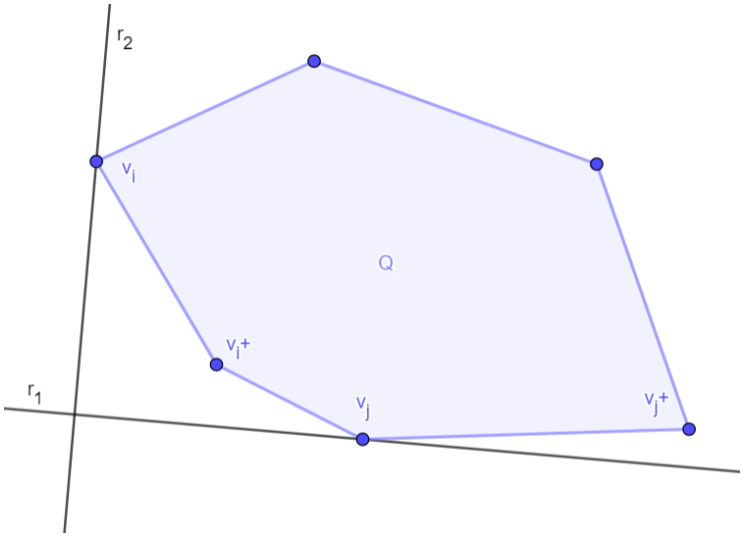
So we want to figure out which of these two switches happens earlier. Note that we can simply order the angles at which these lines switch (so, the angles of the oriented edges of Q), with the angles at which r_2 switches offset by $\frac{\pi}{2}$, and this yields the basic structure of an algorithm for knowing which C_{ij} are in ∂L:

Algorithm 1: FIND_L_V1

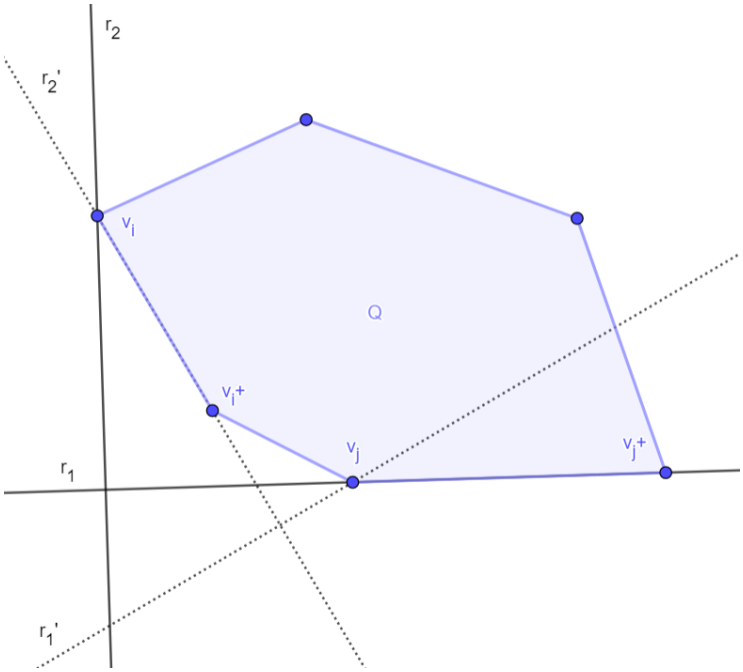
Input: Convex polygon Q with vertices $v_1, \ldots, v_n$.
An angle α and two distinguished vertices of Q, v_L, v_T,
such that the supporting lines from $K(\alpha)_p$ to Q go through v_L, v_T.

$next\_\alpha_L \leftarrow \text{Angle}(v_L v_L^+)$
$next\_\alpha_T \leftarrow \text{Angle}(v_T v_T^+) + \pi/2$
do
  if $next\_\alpha_L < next\_\alpha_T$ **then**
    $\alpha \leftarrow next\_\alpha_L$
    $next\_\alpha_L \leftarrow \text{Angle}(v_L v_L^+)$
    $v_L \leftarrow v_L^+$
  end
  else
    $\alpha \leftarrow next\_\alpha_T$
    $next\_\alpha_T \leftarrow \text{Angle}(v_T v_T^+) + \pi/2$
    $v_T \leftarrow v_T^+$
  end
  $C \leftarrow$ Circumference having $v_L v_T$ as a diameter
  report that ∂L contains the some arc of C
until *some stopping condition is met*;

However, it is possible to find an algorithm that avoids trigonometry, but this will take a little bit of extra work. For this, say that we have some Q, r_1 and r_2 tangent at v_i and v_j, and want to figure out which of the two switches happens first:



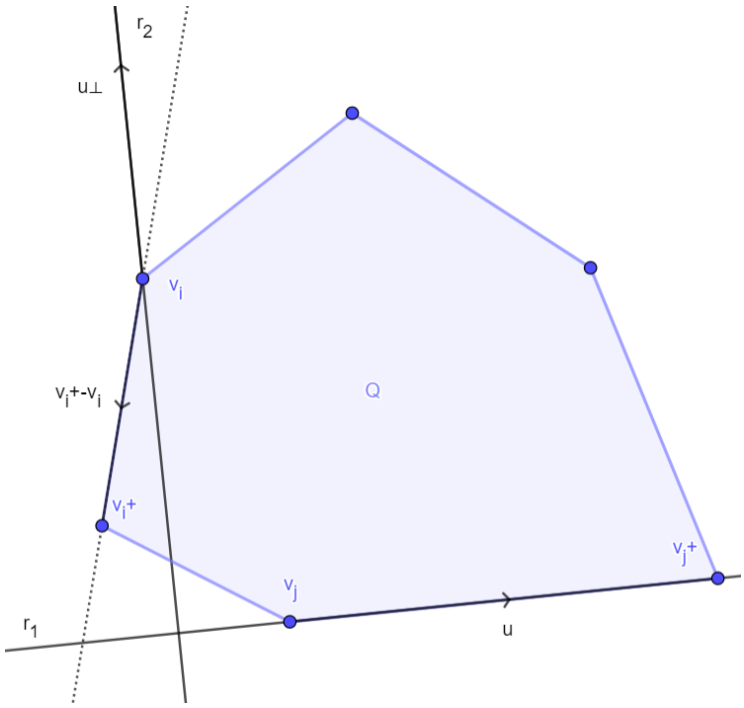
The idea is this: if r_1 switches before r_2, there must be some specific angle at which this switch happens — at this angle, r_1 will be parallel to $v_j v_j^+$, and r_2 will still be tangent to Q at v_i. On the other hand, if r_2 switches earlier, then during this switch r_2 will be parallel to $v_i v_i^+$, and r_1 will remain tangent to Q at v_i. We can try to draw both of these ocurrences:



And we see that the only one that actually works is the one where r_1 switches first, which tells us that the next arc in ∂L is from $C_{i,j+1}$. We just need to distill the decision-making procedure from the drawing above so it can be used in an algorithm.

We need a small preliminary: for any 2-vector (x, y), define $(x, y)^\perp = (-y, x)$. It is easy to check that for any 2-vector v, this satisfies that $v \perp v^\perp$, and $(v, v^\perp)$ is positively oriented.

So let p be the intersection of r_1 and r_2, and $u := v_j^+ - v_j$ — so we have that $r_1 = p + \langle u \rangle$, and moreover $r_2 = p + \langle u^\perp \rangle$. This satisfies that r_1 contains $v_j v_j^+$, and that r_2 is perpendicular to r_1 and contains v_i. The only way that this can go wrong is if r_2 were to intersect the interior of Q. Because r_2 is being rotated counterclockwise around v_i, this will happen if and only if $u^\perp$ is negatively oriented with respect to $v_i^+ - v_i$:



So we obtain that r_1 will switch from v_j to v_j^+ before (or at the same time) r_2 switches from v_i to v_i^+ iff $(v_i^+ - v_i, (v_j^+ - v_j)^\perp)$ is negatively oriented. This yields a much simpler version of the earlier "basic structure" of our algorithm:

Algorithm 2: FIND_L_V2

Input: Convex polygon Q with vertices $v_1, \ldots, v_n$.
Two distinguished vertices of Q, v_L, v_T,
such that the circumference that has $v_L v_T$ as a diameter touches the boundary of L.

do
  if $((v_L^+ - v_L), (v_T^+ - v_T)^\perp)$ *is negatively oriented* **then**
    $v_L \leftarrow v_L^+$
  end
  else
    $v_T \leftarrow v_T^+$
  end
  $C \leftarrow$ Circumference having $v_L v_T$ as a diameter
  report that ∂L contains the some arc of C
until *some stopping condition is met*;

Remark: Note that it could happen that $(v_L^+ - v_L)$ and $(v_T^+ - v_T)$ are perpendicular. This corresponds to having two edges in Q that are perpendicular, in which case v_T and v_R should jump forward at the same time. In the algorithm above, one can see that v_T will jump forward, and then, in the next iteration, v_L will follow — so the algorithm will report that the circle with diameter $v_L v_T^+$ is in ∂L. This is true, but a degenerate case: there is only a point of this circle in ∂L. We do not handle this special case for simplicity, but it would be easy to add an if-clause inside the main loop that incremented both v_L and v_T if $(v_L^+ - v_L)$ and $(v_T^+ - v_T)$ were perpendicular.

It should be noted that, unlike with the prior algorithm, we no longer think about α or $K(\alpha)_p$ at all — we simply obtain the vertices v_i, v_j such that C_{ij} appears in ∂L. To finish this section, we simply need to flesh the above out into an actual algorithm. This requires three things:

Finding an initial point in ∂L

All we need to do is find some particular p that sees Q with a right angle, and two vertices contained in the supporting lines from p to Q. For this, let m_x and m_y be the vertices of Q with smallest x and y coordinates, respectively, and let x_- be the x coordinate of m_x, and y_- the y coordinate of m_y. Then, $p = (x_-, y_-)$ sees Q with a right angle, with the viewcone $p + \langle m_x - p \rangle^+ + \langle m_y - p \rangle^+$ — so we can initialize v_L and v_T to m_y and m_x.

A stopping condition

Note that, in our algorithm, v_L is only ever updated by the line $v_L \leftarrow v_L^+$, and similarly for v_T — so these vertex variables traverse the vertices of Q counterclockwise, only updated to move one vertex forward. Note also that if v_L, v_T ever hold any specific values v_i, v_j, then some (possibly degenerate) arc of C_{ij} is in ∂L, to which $K(\alpha)_p$ belongs for some range of values of α. Because each arc in ∂L is traversed by the algorithm, in counterclockwise order, eventually we will arrive back at C_{ij}, which necessarily means that v_L, v_T will hold the same values of v_i, v_j again. Thus, we can ensure we stop after one full loop by storing the initial values of v_L, v_T, and exiting the main loop when we see those values again.

The last update we need to make to our algorithm is a bit more cumbersome, so we present a preliminar version of the algorithm again:

Algorithm 3: FIND_L_V3

Input: Convex polygon Q with vertices $v_1, \ldots, v_n$.
$v_L \leftarrow \text{argmin}_v(v_y)$
$v_T \leftarrow \text{argmin}_v(v_x)$
initial_$v_L \leftarrow v_L$
initial_$v_T \leftarrow v_T$
do
  if $((v_L^+ - v_L), (v_T^+ - v_T)^\perp)$ *is negatively oriented* **then**
    $v_L \leftarrow v_L^+$
  end
  else
    $v_T \leftarrow v_T^+$
  end
  $C \leftarrow$ Circumference having $v_L v_T$ as a diameter
  report that ∂L contains the some arc of C
until $v_L =$ *initial_*v_L *and* $v_T =$ *initial_*v_T;

Finally, we need to be precise about *what* arc of C_{ij} is in ∂L. Again, we have to make a choice about the format of this — it seems that the most natural option would be to provide the angles that bound the arc of each C_{ij}, but again this would require trigonometry. We opt to provide the extremal points of each arc, and note that one can easily obtain the angles from this information if desired.

Algorithm 4: FIND_L

Input: Convex polygon Q with vertices $v_1, \ldots, v_n$.
$v_L \leftarrow \text{argmin}_v(v_y)$
$v_T \leftarrow \text{argmin}_v(v_x)$
initial_$v_L \leftarrow v_L$
initial_$v_T \leftarrow v_T$

last_$p \leftarrow$ NULL
do
  if $((v_L^+ - v_L), (v_T^+ - v_T)^\perp)$ *is negatively oriented* **then**
    $r_1 \leftarrow$ Line going through $v_L v_L^+$
    $r_2 \leftarrow$ Line perpendicular to r_1 going through v_T
    $v_L \leftarrow v_L^+$
  end
  else
    $r_2 \leftarrow$ Line going through $v_T v_T^+$
    $r_1 \leftarrow$ Line perpendicular to r_2 going through v_L
    $v_T \leftarrow v_T^+$
  end
  $p \leftarrow r_1 \cap r_2$
  if *last_p is not NULL* **then**
    $C \leftarrow$ Circumference having $v_L v_T$ as a diameter
    report that ∂L contains the arc of C between last_p and p
  end
until $v_L =$ *initial_*v_L *and* $v_T =$ *initial_*v_T;

c) Given a fixed squared angle α on some point P and a convex polygon Q, give an algorithm to find the position of Q minimizing the lost area

Setup

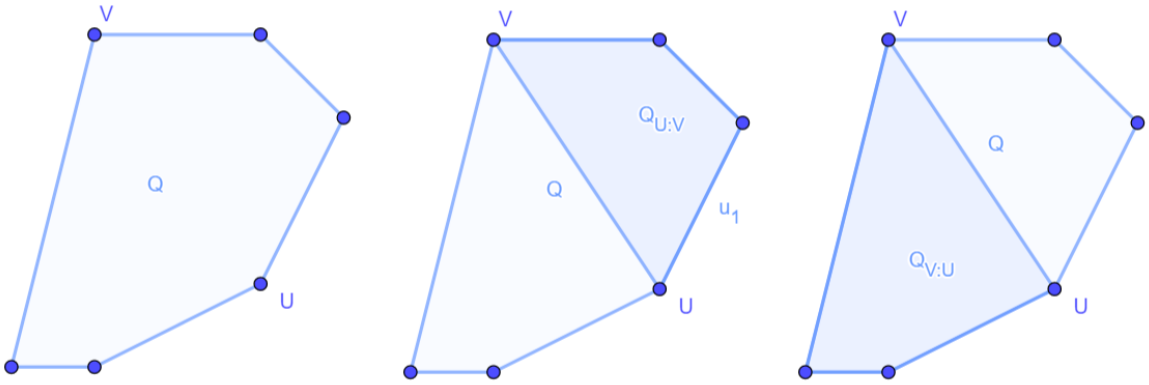
In the previous section we found a linear-time algorithm for computing a description of L. In this section, we will leverage this algorithm to minimize lost area in linear time, and prove that this is asymptotically optimal.

We start by moving the problem to a domain that we are more comfortable with: instead of having a fixed α and finding the minimizing position of Q, we fix any position of Q and find the minimizing position of α. Then, we can build the affine isometry that maps our α to the desired position, and compute the image of Q under that isometry to obtain the solution. Note that this "re-contextualizing" of the problem takes linear time, but since we will later show that the problem takes linear time *in general*, we have not lost any asymptotic performance.

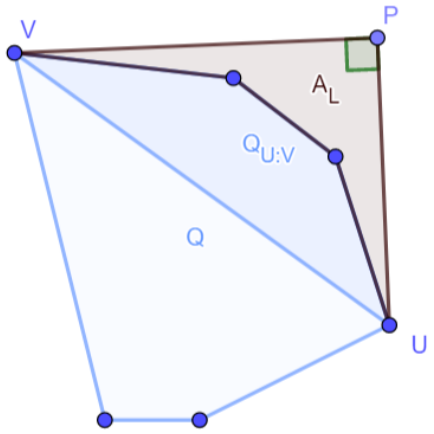
So we want describe an algorithm MIN_LOST_AREA, that takes as input a convex polygon Q and outputs some point p, which can see Q with a right angle, minimizing lost area. To prove the correctness of (and motivate) MIN_LOST_AREA, we will need some preliminary results and definitions.

Polygonal slices

First of all — let v, u be any two distinct vertices from Q. We define the $v - u$ **slice** of Q, denoted $Q_{v:u}$, to be the polygon with vertices $v = v_1, v_2, \ldots, v_k = u$ where $v_1, \ldots, v_k$ are consecutive vertices of Q in clockwise order:



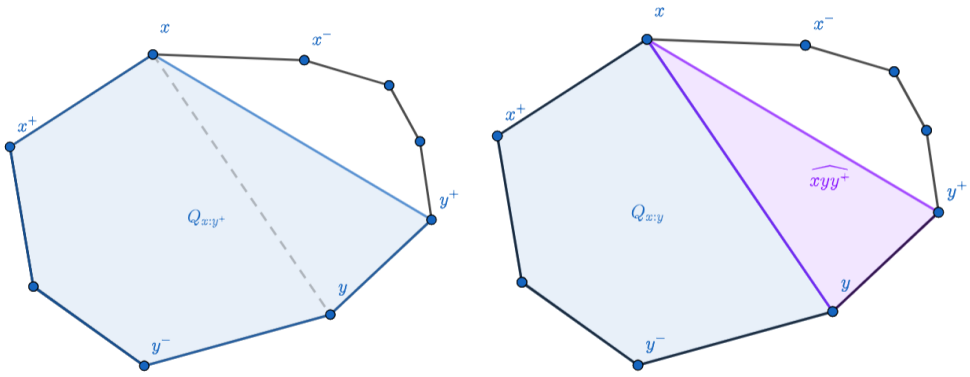
Now, let p be some point seeing Q with a right angle, and $v_1, \ldots, v_k$ be the vertices of Q seen by p, ordered counterclockwise. Note that v_1 and v_k must be in the supporting lines from p to Q. We are interested in the lost area A_L of p with respect to Q, that is, the area bounded by these supporting lines and Q itself. It is easy to see that $|\widehat{v_1 p v_k}| = A_L + |Q_{v_1:v_k}|$:



And thus $A_L = |\widehat{v_1 p v_k}| - |Q_{v_1:v_k}|$. We will be using this expression to compute A_L (for some specific values of p) a number of times, so it is worthwhile to note that, while $|\widehat{v_1 p v_k}|$ can be found in constant time, the naïve way of computing $|Q_{v_1:v_k}|$ will be $O(k)$, which is undesirable. We will remedy this by updating the value of $|Q_{v_1:v_k}|$ "on the fly". To clarify, let x, y be two vertices from Q, and denote by x^- and x^+ the vertices before and after x in counter-clockwise order, and likewise for y. Then:

$$|Q_{x:y^+}| = |Q_{x:y}| + |\widehat{xyy^+}|$$
$$|Q_{x:y^-}| = |Q_{x:y}| - |\widehat{xy^-y}|$$
$$|Q_{x^+:y}| = |Q_{x:y}| - |\widehat{xx^+y}|$$
$$|Q_{x^-:y}| = |Q_{x:y}| + |\widehat{x^-xy}|$$

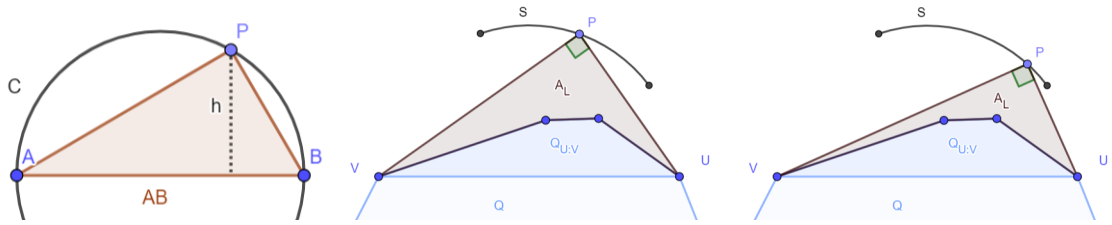
Below is a visual proof of the first statement. The rest are analogous.



This will provide us with the tools to keep track of lost area in constant time — however, we still have an infinite number of points in ∂L that are candidates for minimizing lost area. The second preparation we need before describing MIN_LOST_AREA deals with bringing the number of candidates down to $O(n)$.

Discretization

For this, first consider two points A, B, and let C be the circumference that has AB as a diameter. Let P be some arbitrary point from C, and note that $|\widehat{APB}| = \frac{1}{2} \cdot d(A,B) \cdot h$, where h is the distance from P to the line going through A, B. Note that h is the only part of $|\widehat{APB}|$ that depends on the choice of P, so it is sound to claim that $|\widehat{APB}|$ is monotonous with respect to h.



Now we apply this to our problem: recall that ∂L is composed of segments of circles that have two vertices of Q as a diameter. So let u, v be vertices of Q, and let S be the arc of ∂L that belongs to the circle defined by u, v. Now, let p be some point from S, as per the drawing above. As we've seen, the lost area will be

$$A_L = |\widehat{vpu}| - |Q_{v:u}| = \frac{1}{2} \cdot d(v,u) \cdot h - |Q_{v:u}|.$$

Again, the only term here that depends on p is h, so we know for sure that the points that minimize A_L in S also minimize h — these can only be the extremes of S. We obtain, thus, that the point that minimizes lost area must be one of the points from ∂L where two different arcs meet.

Algorithm

The approach, then, is as follows: we will traverse ∂L in the same way as with FIND_L. At each point of ∂L where different arcs meet, we will evaluate A_L — for this, we will need to keep a variable Q_A updated with the value of $Q_{v_T:v_L}$.

We could use FIND_L as a pre-processing step to find a description of ∂L, but this would be a bit cumbersome, since we have not designed it to keep any explicit information about which values of v_L, v_T correspond to which circumference arcs, and how they are updated between arcs. For this reason, although this might be bad practice in software design, we opt to simply build MIN_LOST_AREA around the same core loop as FIND_L.

Algorithm 5: MIN_LOST_AREA

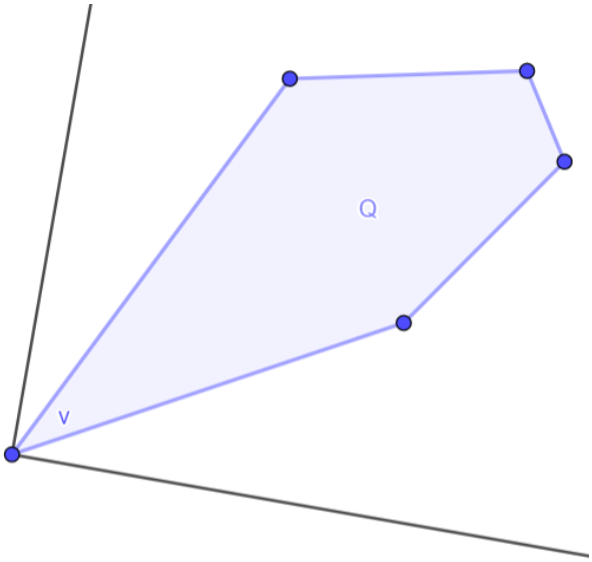
Input: Convex polygon Q with vertices $v_1, \ldots, v_n$.
$v_L \leftarrow \text{argmin}_v(v_y)$
$v_T \leftarrow \text{argmin}_v(v_x)$
initial_$v_L \leftarrow v_L$
initial_$v_T \leftarrow v_T$
$Q_A \leftarrow |Q_{v_T:v_L}|$
minimal_$A_L \leftarrow \infty$
do
  if $((v_L^+ - v_L), (v_T^+ - v_T)^\perp)$ *is negatively oriented* **then**
    $r_1 \leftarrow$ Line going through $v_L v_L^+$
    $r_2 \leftarrow$ Line perpendicular to r_1 going through v_T
    $Q_A \leftarrow Q_A + |\widehat{v_T v_L v_L^+}|$
    $v_L \leftarrow v_L^+$
  end
  else
    $r_2 \leftarrow$ Line going through $v_T v_T^+$
    $r_1 \leftarrow$ Line perpendicular to r_2 going through v_L
    $Q_A \leftarrow Q_A - |\widehat{v_T v_T^+ v_L}|$
    $v_T \leftarrow v_T^+$
  end
  $p \leftarrow r_1 \cap r_2$
  $A_L \leftarrow |\widehat{p v_T v_L}| - Q_A$
  if $A_L <$ *minimal_*A_L **then**
    minimal_$A_L \leftarrow A_L$
    $p^* \leftarrow p$
  end
until $v_L =$ *initial_*v_L *and* $v_T =$ *initial_*v_T;
report that p^* minimizes lost area with minimal_A_L

Other considerations

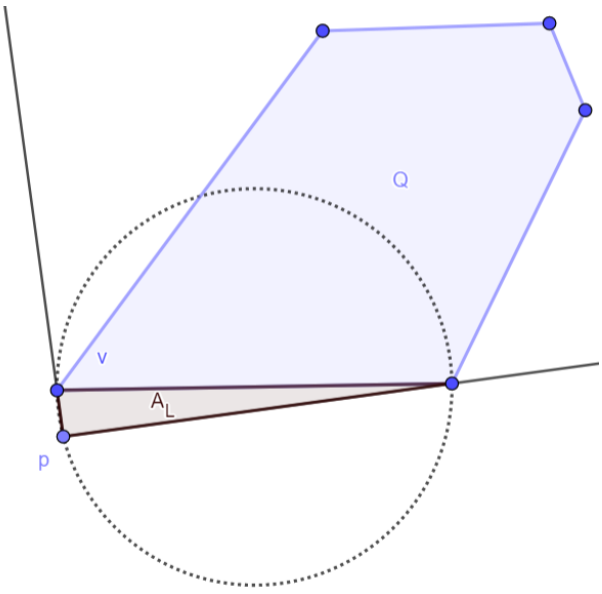
Acute angles

Say that Q has an acute angle located at some vertex v. Then, $v \in \partial L$ — however, this is a bit of a degenerate case, since v can see Q with an angle *smaller* than $\frac{\pi}{2}$, which is generally only the case for p in the interior of L. This does not really present a problem FIND_L, but it will be useful later to note that v (and other vertices with acute angles) is the only vertex where it can happen that $v_T = v_L$ — this is because the difference between Angle($v^- v$) and Angle($v v^+$) is greater than $\frac{\pi}{2}$.

With this in mind, one might realize that during the execution of MIN_LOST_AREA, when v_L and v_R both hold the value of v, we will have $A_L = |\widehat{p v_T v_L}| = Q_A = 0$. That the lost area should be 0 in this case might be disagreeable — consider the illustration in the diagram below:



It might seem that in this instance it would make more sense for the lost area to be undefined, rather than 0. However, if this were the case, then lost area would not have a minimum — the following illustration should show that the presence of an acute angle lets us get arbitrarily small lost area:



Having accepted that lost area must be 0 in acute angles, we note that MIN_LOST_AREA could be modified to exit the main loop directly if it ever came across an acute angle (that is, if it ever ocurred that $A_L = 0$). However, leaving the algorithm as-is does not compromise asymptotic time complexity.

Asymptotics of FIND_L

Space complexity. Other than for the output, it is easy to see that FIND_L only ever needs $O(1)$ memory. The output itself (if stored) would be $O(n)$, but since the data found about ∂L after execution is not used again, FIND_L could easily be implemented as a generator that only needed constant memory (i.e. generated the arcs of ∂L on demand)

Time complexity. Note that the preprocessing before entering the loop is linear-time (particularly the search for the vertices with minimal coordinates). Regarding the loop: all updates within the loop are

constant-time. At each iteration of the loop, either v_L or v_T are updated to v_L^+ or v_R^+, and the algorithm ends when these two variables complete one full loop of Q — meaning the code within the loop will execute $2n$ times, ensuring that the loop is also linear. We conclude that FIND_L has $O(n)$ time complexity.

Optimality. In the prior paragraph we noted that at each iteration, either v_L or v_T are "incremented". Naïvely, one might say that each iteration corresponds to a circular arc in ∂L, and so this in fact proves that the size of the output (in the format we chose) is $O(2n) = O(n)$ — however, it might happen that some iterations of the main loop correspond to "degenerate" arcs, i.e. arcs that only consist of a single point. This can happen if:

- Q contains any acute angle,
- Q contains two edges that are parallel edges *with the same orientation*,
- Q contains two edges that are perpendicular.

This is enough to see that FIND_L has optimal worst-case time complexity. For this, consider any class of arbitrarily large polygons for which the degeneracies above do not occur — for example, regular polygons with an odd number of edges, Q_{2n+1}. It follows that, if $Q = Q_{2n+1}$, then ∂L must have $2n+1$ circle arcs, and so any algorithm describing ∂L will be at least linear.

However, it is not difficult take into account the degeneracies above in such a way that we obtain that our algorithm is also average-case optimal. Regarding acute angles — Q can contain at most three of these, so ultimately this will only add or remove a $O(1)$ amount of arcs from ∂L. It follows that this can be safely ignored.

The second kind of degeneracy is usually excluded in the definition of convex polygons. If we assume that this cannot happen, then each edge can only be perpendicular to two other edges, meaning we will have at most n unordered pairs of perpendicular edges — so the amount of arcs will still be at least $2n - n = n$. This is still $O(n)$, so the algorithm would still be optimal.

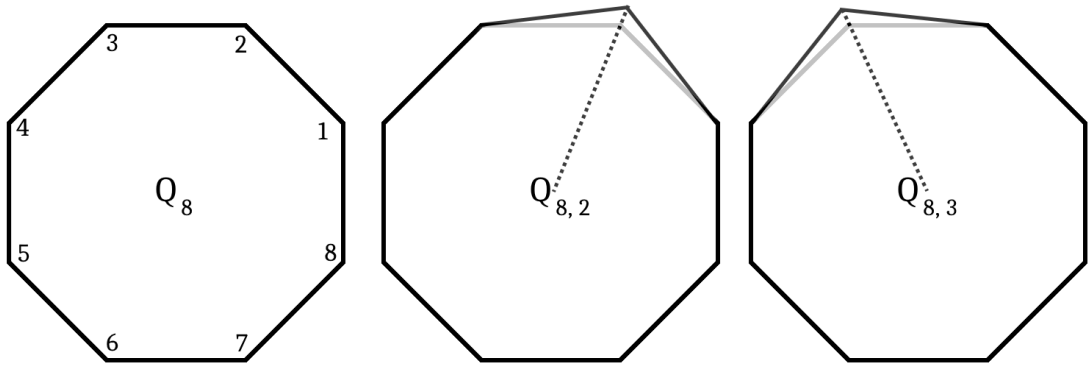
Again, this is assuming that the second degeneracy does not happen. Removing this assumption: note that any polygon (represented by a sequence of vertices) Q with successive parallel edges can be reduced in logarithmic time to a polygon Q' that occupies the same area but without parallel edges. This can be done with multiple binary searches, to find the vertices of Q where the slope changes — if there are k such vertices, this can be done in $O(k \log n)$ time.

Asymptotics of MIN_LOST_AREA

Space complexity. With the same arguments as for the other algorithm, MIN_LOST_AREA only ever needs $O(1)$ space (in particular, it's output is also $O(1)$).

Time complexity. Again with the same argument as before, MIN_LOST_AREA takes linear time.

Optimality. We show that MIN_LOST_AREA has optimal worst-case time complexity. For this, let Q_n be the regular polygon with n vertices. Let v be a vertex of Q_n, and let $Q_{n,v}$ denote Q_n with v slightly raised:



It should be clear[2] that, to minimize the lost area of $Q_{n,v}$ from some external point p, p should see v. Force that $2|n$ and $n \geq 6$, and define v' to be the vertex "antipodal" to v — we also have that p cannot see v', as otherwise its supporting lines would form an angle of at most $\frac{\pi}{3}$, meaning that p is not at ∂L and thus does not minimize lost area.

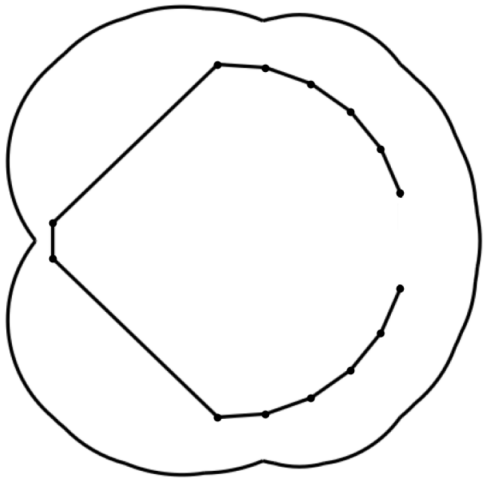
So the idea is as follows: Pair up the vertices of Q_n into antipodal pairs, $\{(v, v')\}_{v \in I}$, such that each vertex only appears once in this list (so $|I| = n/2$). Now consider running some algorithm A to minimize lost area on Q_n.

Claim. If A is correct, it must access at least one element of each (v, v') pair.

Proof. By contradiction: assume A misses some pair (v, v'). Then, for all A knows, it could be looking at $Q_{n,v}$ or $Q_{n,v'}$, or Q_n — these three polygons are equal in all the vertices that A has checked so far. So, if A is correct, it'll spit out a point that minimizes lost area for $Q_{n,v}$ *and* $Q_{n,v'}$ — but such a point does not exist. □

So A must access the data of at least $n/2$ vertices, which will take linear time. This shows that MIN_LOST_AREA has optimal worst-case time complexity.

We note that, unlike with the prior problem, there are instances where an algorithm A could minimize lost area while accessing only a constant amount of vertices. We previously mentioned that this was the case for any Q with an acute angle, but this can also happen for more general polygons such as the one below (where there might be any amount of vertices between the two unconnected vertices, as long as the resulting polygon is convex). The outer figure is the boundary of L of the convex hull of the vertices. This convex hull is contained in the entire polygon, which implies that this L must contain L of the entire polygon. With this, one can show that we can already know that lost area is minimized at the crease in ∂L at the left.



[2] if some point q minimizes lost area without seeing v, then rotate q (with a symmetry of Q_n. This rotated q should have even smaller lost area, contradicting minimality.) in such a way that q can see v.