Computational Geometry—236719

Assignment no. 1

Given: 15/11/2022

Due: 29/11/2022

Question 1

a)

Let S_1, S_2 be two convex sets.

If the intersection is the empty set Φ, in case of disjoint sets intersection, Φ is a convex set by definition.

Otherwise: let x, y by any two points belongs to the intersection of S_1 and S_2: $x, y \in S_1 \cap S_2$

Let $t \in R$ such that: $0 \leq t \leq 1$.

x, y belong to the intersection and therefore: $x, y \in S_1$

By convexity of S_1: $tx + (1-t)y \in S_1$

The same holds for S_2:

x, y belong to the intersection and therefore: $x, y \in S_2$

By convexity of S_2: $tx + (1-t)y \in S_2$

To conclude: $tx + (1-t)y \in S_1$ and also $tx + (1-t)y \in S_2$ therefore: $tx + (1-t)y \in S_1 \cap S_2$.

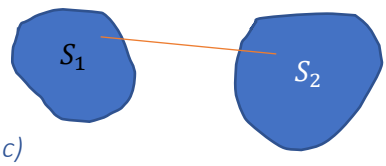
This is true for any x, y in the intersection and any t, hence $S_1 \cap S_2$ is convex.

b)

This is not true.

For example, the union of two **disjoint** convex set is not a star shaped set: there is no point in S_1 such that for any point in S_2 the line segment contained entirely in the union set.

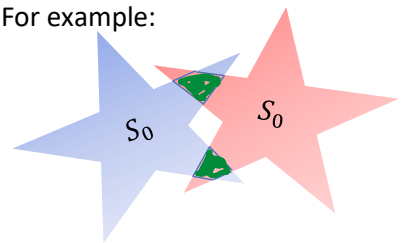
The same holds for any point in S_2.



c)

This is not true.

for example, the intersection of two star-shaped sets can be disconnected, and in particularly not a star shaped. For example:



The green area is the intersection of star-shaped sets S_1, S_2 is disconnected and in particularly not a star-shaped. The is no point in the upper side such that the line to all points in the bottom side are contained in the green area, and vice versa.

d)

This is not true for similar reasons like the previous section.

The intersection of a convex set and a star-shaped set can be disconnected and therefore don't have to be convex. For example:

Clearly, the green area is disconnected and hance not convex.

Question 2

Basic Idea:

Solving it in a similar way to the line-sweep algorithm from the lecture, with a few changes:

1. Segments are circles instead of lines. Each circle is represented by the tuple: $(center \in \mathbb{R}^2, radius \in \mathbb{R})$
2. Events are left/right-most x coordinates of a circle: $x_{center} \pm radius$, or an intersection point.
    a. Insert evert at left-most
    b. Delete event at right-most
    c. Report points at an intersection event.
3. Instead of one insertion to sweep line status we do two insertions: one for the upper half of the circle and one for the bottom half of the circle.
4. Sweep line status is ordered by the upper/lower most y coordinate of the half-circles: $y_{center} \pm radius$
5. Each two circles test-for-intersections can lead to one of the following:
    a. No intersection
    b. one intersection point
    c. two intersection points

Data Structures:

1. Event priority queue: sorted by x coordinate (of the leftmost/rightmost point on each cycle)
2. Sweep line status: stores half-cycles, sorted by y coordinate (of the uppermost/lowermost point on each cycle)

Initialization:

Insert all event to the priority queue according to x coordinates. N circles, each circle has two leftmost/rightmost events, each insertion takes $O(\log(n))$, therefore: $O(nlog(n))$

Algorithm description:

While the event priority queue is not empty, pop event.

- Event of type A: beginning of a circle (leftmost point on it)
    - Slice the circle to two halves horizontally
    - Locate each half position in the sweep line status (ordered by the uppermost/lowermost y coordinate)
    - Insert both halves to the status
    - Test for intersections right to the sweep line for **both halves**, with half-circles adjoint in the status.
    - Can be 0,1 or 2 intersections. Check:

        - Calculate the distance between circle centers: $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
        - If $d < |R_1 - R_2|$ the circle inside the other circle
        - If $d < R_1 + R_2$ the circles intersect
        - If $d = R_1 + R_2$ the circles touch each other
    - Insert intersections to the event queue
    - Time complexity: n events therefore: $O(\log(n))$. For each event, check for intersection in $O(1)$ and insert up to 4 intersections, sum to k intersection points insertions: $O(\log(k))$
- Event of type B: end of a circle
    - Locate both halves in the status
    - Delete both halves
    - Check for intersections 2 times, for each halve deleted, between the elements now become adjoints.
    - Insert intersections to the event queue.
    - Same time complexity as event of type A
- Event of type C: intersection point (which were added before)
    - report point and remove from event queue
    - **circles do no change order right to the intersection, therefore no need to swap the neighbors in the sweep line status**
    - $O\big(klog(n)\big)$ for all intersections
- To conclude: $O(nlogn + klogn)$ time complexity

Question 3

a)

- True. Each edge's twin is the half-edged on the opposite direction, and vice versa.
- True. Both Next & Prev are defined with respect to the same incident face, therefore $Next(e)$ is the next edge while the same face still on the left, and then $Prev(e)$ is the previous edge similarly.
- True. $Prev\big(Twin(e)\big) = Twin\big(Next(e)\big)$
- True. By definition: $Next(e)$ next half-edge on the boundary of $IncidentFace(e)$

b)

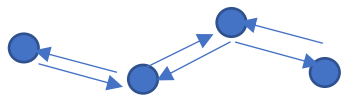
- $$\begin{aligned}
&given: v \\
&V = \Phi \\
&start_e \leftarrow incidentEdge(v) \\
&V = V \cup origin\big(twin(start_e)\big) \\
&e \leftarrow next(start_e) \\
&while\ e \neq start_e \\
&\quad V \leftarrow V \cup origin\big(twin(e)\big) \\
&\quad e \leftarrow\ next(e) \\
&return\ V
\end{aligned}$$
- $$\begin{aligned}
&given: f \\
&E = \Phi \\
&\backslash\backslash \text{ closed face} - \text{non background} \\
&if\ outerComp(f) \neq nil
\end{aligned}$$

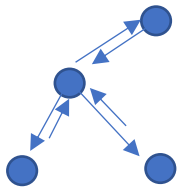
$start_e \leftarrow outerComp(f)$
$E = E \cup start_e$
$e \leftarrow next(start_e)$
$while\ e \neq start_e$
$\quad E = E \cup e$
$\quad e \leftarrow next(e)$
$\backslash\backslash\ open\ face$
$else$
$while\ notEmpty(innerComp(f))$
$\quad start_e \leftarrow pop(innerComp(f))$
$\quad E = E \cup start_e$
$\quad e \leftarrow next(start_e)$
$\quad while\ e \neq start_e$
$\quad\quad E = E \cup e$
$\quad start_e \leftarrow pop(innerComp(f))$
$\quad e \leftarrow next(start_e)$

- How does a boundary vertex is defined? The outer face of the graph, even if connected, is also a face (f1 in the slides)

c)

It can only have one face. if for a half-edge, $twin(e) = next(e)$, it means that there is no 'closing edge' to connect the two or more vertices to form a closed face.
This graph will have s 'snake topology', with one or more heads, each connected to one or more vertices, where each vertex is connected to its predecessor and ancestor only, until the tail of the snake, which is connected to his predecessor only.





Question 4

a)

The proof will work in a similar fashion to what was learned in the lecture:

Theorem: A polygon without *split* and *merge* vertices is *y*-monotone.

while generalized to any-line monotonicity, by defining *split* and *merge* with respect to non-horizontal nor vertical lines.

A polygon with 'no split and merge' dual sentence may be:

- No split: no 3 points such that the angle is greater than π and the first and last points are closer to the line than the middle point.
- No merge: no 3 points such that the angle is greater than π and the first and last points are farther from the line than the middle point.

mathematically, if the line is given as an equation: $ax + by = c$,
for three points $P_1 = (x_1, y_1), P_2 = (x_2, y_2), P_3 = (x_3, y_3)$, P_2 is closer to the line than P_1, P_3 if and only if:

$$ax_2 + by_2 < ax_1 + by_1$$

and:

$$ax_2 + by_2 < ax_3 + by_3$$

and we can break ties by returning True for any point of the three.
So the algorithm will be like (assuming vertices ordered CCW around to polygon):

$given$:
  $P\ polygon\ with\ n\ vertices,$
  $L\ line\ with\ params\ \alpha, \beta$
$for\ i = 1{:}n$
  $v_1 \leftarrow P.V[(i-1)\%n]$
  $v_2 \leftarrow P.V[i]$
  $v_3 \leftarrow P.V[(i+1)\%n]$
  $v_{21} \leftarrow v_1 - v_2$
  $v_{23} \leftarrow v_3 - v_2$
  $if\ \cos^{-1}\left(\frac{v_{21} \cdot v_{23}}{|v_{21}| \cdot |v_{31}|}\right) \geq \pi$
    $if\ \alpha v_2.x + \beta v_2.y < \alpha v_1.x + \beta v_1.y\ and$
      $\alpha v_2.x + \beta v_2.y < \alpha v_3.x + \beta v_3.y$
      $return\ False$
    $if\ \alpha v_2.x + \beta v_2.y > \alpha v_1.x + \beta v_1.y\ and$
      $\alpha v_2.x + \beta v_2.y > \alpha v_3.x + \beta v_3.y$
      $return\ False$

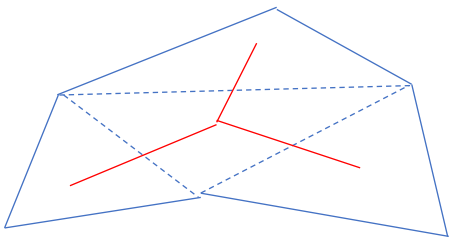
note: in other sources, there is no requirement on the inner angle[1]. But it exists in the lecture's slides, so I added it here as well.

Correctness: as in the lecture

Complexity: $O(n)$: n vertices, for each vertex $O(1)$ operations.

b)

This is not true.
for example:



This polygon is monotonic w.r.t x axis, and its triangulation is drawn in dashed line. The red lines are the dual graph, and as you can see, the center vertex has degree of 3.

Question 5

a)

The proof is similar to the diagonal existence proof.
choose an arbitrary vertex v_2 with neighbors v_1, v_3 on the outer boundary of P and let $d = v_1 v_3$ if d is internal to P, otherwise let $d = v_2 x$ where x is the farthest vertex from $v_1 v_3$.
if d has one endpoint on a hole, then it increases n by 2 and decreases h by 1. If d has both

[1] https://cs.stackexchange.com/questions/1577/how-do-i-test-if-a-polygon-is-monotone-with-respect-to-a-line

endpoints on the outer boundary, it partitioned P to two polygons, each with $n_i < n$ and $h_i \leq h$. then the proof takes an inductive step on h, until no h left.

b)

The number of triangles is: $n + 2h - 2$.

proof:

Let the outer boundary of P have n_0 vertices and let the $i'th$ hole have n_i vertices.

Let us denote the total number of vertices: $n = n_0 + n_1 + \cdots + n_h$

The sum of interior angles of the outer boundary is: $180(n_0 - 2)$ (proof: as there were no holes – $n_0 - 2$ triangles, each with total angle 180).

The sum of exterior angles of each hole is: $180(n_i + 2)$. Therefore:

$$180[(n_0 - 2) + (n_1 - 2) + \cdots + (n_h - 2)] = 180t$$

or:

$$t = n + 2h - 2$$

So, this is the total number of triangles.

c)

following Gabriel Lame theorem:

$$T_n = \sum_{k=2}^{n-1} T_k \cdot T_{n-k+1}$$

Where T_n is the number of triangulations for n-vertices polygon and assuming $T_2 = 1$.

It is actually known as Catalan Numbers…