Computational Geometry - HW1

Shaza Abdallah - 323032474

November 22, 2025

1 Question 1

1.1 (a)

Proof. Let S_1 and S_2 be convex sets, and assume that $S_1 \cap S_2 \neq \emptyset$. Take any two points $p, q \in S_1 \cap S_2$. Then $p, q \in S_1$ and $p, q \in S_2$. Since S_1 is convex, the entire segment pq lies in S_1; similarly, since S_2 is convex, the segment pq lies in S_2. Thus every point of pq lies in both sets, and so $pq \subseteq S_1 \cap S_2$. Therefore, $S_1 \cap S_2$ is convex.

1.2 (b)

Counterexample.

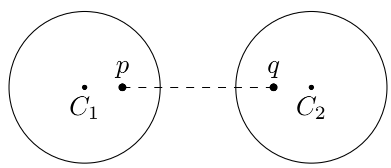


Figure 1: Two disjoint circles C_1 and C_2.

Let C_1 and C_2 be two circles in $\mathbb{R}^2$ whose interiors and boundaries do not intersect, and consider the set

$$S = C_1 \cup C_2.$$

We show that S is not convex.

Pick a point $p \in C_1$ and a point $q \in C_2$, for example as in the figure above. By definition of convexity, if S were convex, then the entire line segment pq would have to be contained in S.

However, since the circles are disjoint and separated in space, the segment pq contains points that lie strictly between the two circles, hence outside both C_1 and C_2. Therefore these points are not in $S = C_1 \cup C_2$.

1.3 (c)

Counterexample.

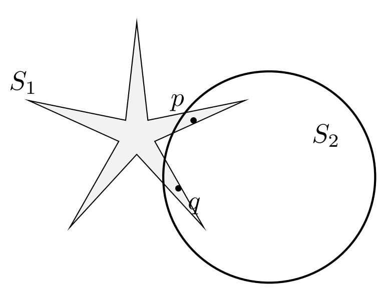


Figure 2: A star-shaped set S_1 and a circle S_2.

The intersection

$$S = S_1 \cap S_2$$

consists of two disjoint connected components, one at the top of the circle and one at the bottom. Assume for contradiction that S is star-shaped with respect to some point $x \in S$. Then p lies in exactly one of the two components. Choose a point q in the other component. Any line segment from p to q must leave S, so in particular $pq \not\subseteq S$, contradicting the definition of a star-shaped set. Therefore $S_1 \cap S_2$ is not star-shaped.

1.4 (d)

Counterexample. Consider the same sets S_1 and S_2 as in part (c): S_1 is the star-shaped set (the star), and S_2 is the circle, which is convex. Their intersection

$$S = S_1 \cap S_2$$

consists of two disjoint connected components.

Pick points p and q in different components of S (as indicated in the figure). The line segment $[p, q]$ necessarily passes through points that lie outside S, so $pq \not\subseteq S$. Hence S is not convex.

2 Question 2

Let S be a set of n circles in the plane, each given by the endpoints of its horizontal diameter. We describe a plane-sweep algorithm that computes all k intersection points in $O((n + k) \log n)$ time.

Events

We use a priority queue sorted by x-coordinate. There are three event types:

1. *Circle Start:* the leftmost point $(x_i - r_i, y_i)$ of circle C_i.
2. *Circle End:* the rightmost point $(x_i + r_i, y_i)$ of C_i.
3. *Circle Intersect:* an intersection point of two circles.

Sweep-line Status

Unlike line segments, a circle does not have a single well-defined vertical order along the sweep line. At a sweep position $x = x_0$, circle C_i intersects the vertical line in two points:

$$y_i \pm \sqrt{r_i^2 - (x_0 - x_i)^2}.$$

Thus the sweep-line status stores *two* entries for each active circle:

$$\text{TopHalf}(C_i), \qquad \text{BottomHalf}(C_i),$$

each ordered by its y-coordinate on the sweep line. The status structure is a balanced BST ordered by these y-values.

Handling Events

1. Circle Start. When encountering the leftmost point of C_i:

1. Insert $\text{TopHalf}(C_i)$ and $\text{BottomHalf}(C_i)$ into the status.
2. For each of them, check for intersections with the immediate predecessor and successor arcs in the BST.
3. For every real intersection point whose x-coordinate is in the future, insert a *Circle Intersect* event.

2. Circle End. When reaching the rightmost point of C_i:

1. Let $a = \text{BottomHalf}(C_i)$ and $b = \text{TopHalf}(C_i)$.
2. Before removing a and b, let p be the predecessor of a and let q be the successor of b.
3. Remove a and b from the status.
4. Check whether p and q intersect; if so, add the corresponding intersection event.

3. Circle Intersect. Suppose arc A and arc B intersect at (x^*, y^*).

1. Output the intersection point.
2. In the status, swap the order of A and B (only the two semicircles swap).
3. After the swap:
   - Check for intersections between A and its new predecessor and successor.
   - Check for intersections between B and its new predecessor and successor.
4. Insert any newly discovered valid intersection events.

Running Time

There are $2n$ start/end events and k intersection events. Each event triggers $O(1)$ neighbor checks and $O(\log n)$ updates in the balanced BST and event queue. Therefore the total running time is

$$O((n+k)\log n),$$

3 Question 3

3.1 (a)

- $Twin(Twin(e)) = e$ (always true)
  Each undirected edge in the DCEL is represented by exactly two half-edges, which are twins of one another and point in opposite directions.
- $Next(Prev(e)) = e$ (always true)
  For any face, its boundary is stored as a circular doubly linked list of half-edges. By definition of a doubly linked list,
  $$Next(Prev(e)) = e$$
- $Twin(Prev(Twin(e))) = Next(e)$ (not always true)

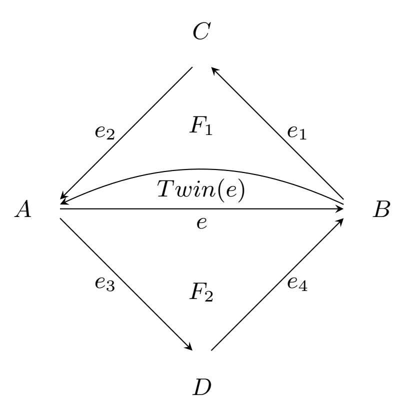


Figure 3: Two triangles sharing edge $\overline{AB}$, showing $e : A \to B$ and its twin $Twin(e) : B \to A$ as distinct half-edges. On face F_2, the boundary order is $D \to B \to A \to D$, so $Prev(Twin(e)) = e_4 : D \to B$, and therefore $Twin(Prev(Twin(e))) = B \to D \neq Next(e) = B \to C$.

- $IncidentFace(e) = IncidentFace(Next(e))$ (always true)
  All half-edges around the boundary of the same face share the same $IncidentFace$ pointer. Since $Next(e)$ walks along the same face,

$$IncidentFace(e) = IncidentFace(Next(e)).$$

3.2 (b)

- List all vertices that are connected by an edge to a given vertex v

Algorithm 1 ListNeighbors(v)

$e \leftarrow IncidentEdge(v)$
$d \leftarrow e$
repeat
  S.add($origin(Twin(d))$)
  $d \leftarrow Next(Twin(d))$
until $d = e$
return S

- List all edges that bound a given face f in a not necessarily connected subdivision

Algorithm 2 ListEdges(f)

$S \leftarrow \emptyset$
$e_{\text{out}} \leftarrow OuterComponent(f)$
if $e_{\text{out}} \neq$ null **then**
  CountCycle(e_{out}, S)
end if
for each e_{in} in $InnerComponents(f)$ **do**
  CountCycle(e_{in}, S)
end for
return S

Algorithm 3 CountCycle(e, S)

$d \leftarrow e$
repeat
  S.add(d)
  $d \leftarrow Next(d)$
until $d = e$

- List all faces that have at least one vertex on the outer boundary of the subdivision

Algorithm 4 ListFacesTouchingOuterBoundary()

```
S ← ∅
for each face f in the subdivision do
    if OuterComponent(f) = nil then
        OuterFace ← f
        break
    end if
end for
BoundaryVertices ← ∅
for each e_0 in InnerComponents(OuterFace) do
    d ← e_0
    repeat
        BoundaryVertices.add(origin(d))
        d ← Next(d)
    until d = e_0
end for
for each vertex v in BoundaryVertices do
    h ← IncidentEdge(v)
    g ← h
    repeat
        f ← IncidentFace(g)
        S.add(f)
        g ← Next(Twin(g))
    until g = h
end for
return S
```

3.3 (d)

The subdivision can have one face. In a DCEL we always have

$$IncidentFace(e) = IncidentFace(Next(e)).$$

Under the given assumption $Twin(e) = Next(e)$ for every half-edge e, we obtain

$$IncidentFace(e) = IncidentFace(Next(e)) = IncidentFace(Twin(e)).$$

Thus a half-edge and its twin always belong to the *same* face for all edges.

4 Question 4

4.1 (a)

Let $P = (p_0, p_1, \ldots, p_{n-1})$ be a simple polygon listed in cyclic order, and let $\ell : y = mx + b$ be the line with respect to which we want to test monotonicity.

Define the projection function

$$h(x, y) = x + my.$$

This value gives the position of a point along the direction of ℓ; points lying on any line perpendicular to ℓ have equal h-value.

Algorithm:

1. for each vertex $p_i = (x_i, y_i)$ of P, compute

$$h_i = h(p_i) = x_i + my_i.$$

2. find the vertex p_s having the minimum projection value and the vertex p_t having the maximum projection value:
$$s = \arg\min_i h_i, \qquad t = \arg\max_i h_i.$$
These serve as the "lowest" and "highest" vertices of the polygon in the direction of ℓ.

3. starting at p_s, follow the polygon boundary forward until reaching p_t. Check that the sequence
$$h_s,\, h_{s+1},\, h_{s+2},\, \ldots,\, h_t$$
is monotonically nondecreasing. If a decrease occurs (i.e. $h_{i+1} < h_i$ for some consecutive pair), then P is not ℓ-monotone.

4. again start from p_s, but now follow the vertices backward (decreasing the index modulo n) until reaching p_t. Confirm that the projection sequence along this second chain is also monotonically nondecreasing. A decrease anywhere implies the polygon is not ℓ-monotone.

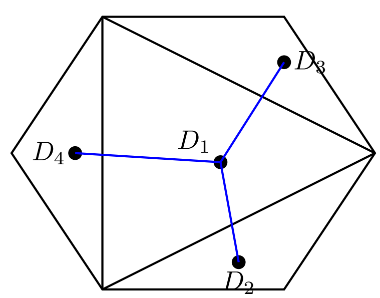
5. if both boundary chains from p_s to p_t have nondecreasing projection values, then the polygon is ℓ-monotone. Otherwise, it is not.

The function $h(x, y) = x + my$ increases exactly in the direction parallel to ℓ. Lines perpendicular to ℓ are precisely those on which h is constant. Thus, checking monotonicity with respect to ℓ amounts to verifying that along each boundary chain between the minimal and maximal projection vertices, the polygon never moves "backwards" in h. A decrease in projection means that some line perpendicular to ℓ intersects the polygon in two disjoint segments. Conversely, if both chains have nondecreasing projections, every perpendicular line intersects P in at most one connected interval, which is exactly the definition of ℓ-monotonicity.

Computing all projection values takes $O(n)$ time, finding the extremal vertices takes $O(n)$ time, and traversing both boundary chains also takes $O(n)$. Thus, the full algorithm runs in $O(n)$

4.2 (b)

counterexample.



This polygon is convex, and therefore monotone with respect to every line. However, in the shown triangulation the central triangle corresponds to a dual vertex D_1 that is adjacent to three other triangles. Thus $\deg(D_1) = 3$, so the dual graph is not a chain.

5 Question 5

5.1 (a)

Proof. Let P have h holes and n vertices in total. We prove the theorem by induction on h primarily, and on n secondarily.

Base case. For $h = 0$, P is a simple polygon with no holes. It was proved in the lecture that every simple polygon can be triangulated. Thus the claim holds for $h = 0$.

Induction hypothesis. Assume the theorem holds for every polygon P_1 with $h_1 < h$ holes, and also holds for every polygon P_2 with $h_2 \leq h$ holes and strictly fewer $n_2 < n$ vertices.

Induction step. Let P be a polygon with h holes and n vertices. Choose an arbitrary convex vertex v_2 on the outer boundary of P, and let v_1 and v_3 be its two neighbours on the boundary.

Consider the segment v_1v_3. If the open segment v_1v_3 lies entirely inside P, then $d = v_1v_3$ is an internal diagonal of P. Otherwise, let $d = v_2x$, where x is the closest vertex to v_2 measured along the line perpendicular to v_1v_3.

We distinguish two cases.

Case 1. The diagonal d has one endpoint on a hole. Cutting along d decreases the number of holes by 1, but increases the number of vertices by 2. Thus the resulting polygon has $(h-1)$ holes, and hence the induction hypothesis on h applies. Therefore it can be triangulated, and this triangulation is also a triangulation of P.

Case 2. Both endpoints of d lie on the outer boundary of P. Then d partitions P into two polygons P_1 and P_2, each with at most h holes and strictly fewer than n vertices. By the induction hypothesis on n, both P_1 and P_2 admit triangulations, which together form a triangulation of P.

□

5.2 (b)

Proof. We prove the statement by induction on the pair (h, n), with induction on h primarily and on n secondarily.

Base case $h = 0$. If P is a simple polygon with no holes, then any triangulation has exactly $n-2$ triangles, as proved in the lecture. Thus the formula holds for $h = 0$:

$$n - 2 + 2h = n - 2.$$

Induction hypothesis. Assume that the statement holds for every polygon P_1 with $h_1 < h$ holes, and for every polygon P_2 with $h_2 = h$ holes and $n_2 < n$ vertices.

Induction step. Let P have h holes and n vertices. Choose a diagonal d obtained by the same construction as in the proof of triangulability: either $d = v_1v_3$ or $d = v_2x$, depending on whether v_1v_3 is internal. We again consider two cases.

Case 1. The diagonal d connects the outer boundary to a hole. Cutting along d produces a polygon P' with $h-1$ holes and $n+2$ vertices. By the induction hypothesis on h, every triangulation of P' contains

$$(n+2) - 2 + 2(h-1) = n - 2 + 2h$$

triangles. Since every triangulation of P corresponds to a triangulation of P', the same number of triangles appears in P.

Case 2. Both endpoints of d lie on the outer boundary of P. Then d partitions P into two polygons P_1 and P_2 with $n_1 + n_2 = n + 2$ and $h_1 + h_2 = h$, and with $n_1, n_2 < n$. By the induction hypothesis on n, their triangulations contain

$$n_1 - 2 + 2h_1 \quad \text{and} \quad n_2 - 2 + 2h_2$$

triangles, respectively. Adding these counts and observing that

$$(n_1 - 2 + 2h_1) + (n_2 - 2 + 2h_2) = n - 2 + 2h$$

gives the desired total number of triangles in P.

□

5.3 (c)

Let T_n denote the number of triangulations of a convex polygon with n vertices. Fix the edge p_1p_n. In every triangulation, this edge must belong to a unique triangle $\triangle p_1p_ip_n$ for some i with $2 \leq i \leq n-1$.

This choice splits the polygon into two smaller convex polygons:

$$\{p_1, p_2, \ldots, p_i\} \qquad \text{and} \qquad \{p_i, p_{i+1}, \ldots, p_n\},$$

having i and $n-i+1$ vertices respectively. Their triangulations are independent, so the number of triangulations with triangle $p_1p_ip_n$ is

$$T_i \cdot T_{n-i+1}.$$

Summing over all possible i gives the recurrence

$$T_n = \sum_{i=2}^{n-1} T_i \, T_{n-i+1},$$

with the base case $T_2 = 1$ (a polygon with two vertices has one trivial "triangulation").