Point Location

- we have seen basic examples:
  - is a given point inside a given \( \text{poly} \) convex polygon \( \Theta(n) \)
    \( \text{star-shaped polygon} \) \( \Omega(\log n) \)

- we can solve those problems faster if the polygon is already known
  different scenarios:
  for one query point
  for \( k \) query points
  \( \leq \) (pre-processing)

2 things to minimize:
pre-processing time
space (data structure)
Pre-processing
1) store all vertices by x-coord.
2) For every vertical slab:
   store a sorted list of segments that cross the slab
   (ordered & not intersecting)

Time: sort + space
Space: $O(n^2)$

Query $\bullet$ in $O(\log n)$
1) Binary search on $x$
2) Binary search in 1 slab
Notice that...

...So its a huge waste of space to store a list of size $O(n)$ in each slab.

(+) $2$

(2)

(polygons are almost identical from one slab to the next, the vertically sorted polygonal segments are)
A WORD ABOUT PERSISTENT DATA STRUCTURES

(just so that you know they exist)

A P.D.S. simply maintains information so that any previous version of the D.S. can be extracted. (without explicitly storing all versions)

It must still support regular operations: ex. insert/delete

For the slab decomposition, think of one binary tree (representing a slab) being modified \( n \) times as it sweeps through polygon

\[ \rightarrow \text{unlike a regular sweep, we care about remembering/retrieving what the tree looked like at any step.} \]
Sarnak & Tarjan: persistent binary search tree for point location via slab decomposition

- $O(\log n)$ query after $n$ operations
- $O(n)$ space .... $O(1)$ per step amortized

This means (?) you could have to waste $O(n)$ time preprocessing between queries, but not many times.

Project ?
our concern is to keep $O(\log n)$ query time but reduce pre-processing time/space to $O(n)$

In fact we will do this for arbitrary

**PLANAR STRAIGHT-LINE GRAPHS**

more general
more interesting
First pre-processing step: triangulate (each region) $O(n)$ time & space

we will actually determine which triangle contains any given query point

even more general
Even more general: create a triangular outer face.

All of this is still $O(n)$ time & space.
What is the average degree of a triangulation?

\[ \frac{1}{n} \sum_{i=1}^{n} d(v_i) = \frac{1}{n} \cdot 2e = \frac{6n-12}{n} \leq 6 \]

\( \Rightarrow \) Every triangulation has a vertex w/ degree \( \leq 5 \)

(but might only have 12)

Can we find many low-degree vertices? → not if "low" = 5. what if "low" = 8?

Say you had \( \frac{n}{2} \) vertices w/ degree \( \geq 9 \)

\[ \sum d(v_i) \geq 9 \cdot \frac{n}{2} \]

\( \sum d(v_i) \geq 3 \cdot \frac{n}{2} \)

\[ \sum + \sum \geq 6n \]

contradiction

always have \( \frac{n}{2} \) w/ deg \( \leq 8 \)
Always has $\geq \frac{n}{2}$ vertices w/ degree $\leq 8$

In fact many of them must be nicely separated into an independent set where no two are neighbors.

Intuitively, a dense set of high degree vertices is similar to what we just analyzed.
Always has $\geq \frac{n}{2}$ vertices w/ degree $\leq 8$

In fact many of them must be nicely separated

\[ \text{independent set} \]

\[ \text{no 2 are neighbors} \]

Why? $\rightarrow$

start w/ $\frac{n}{2}$ of them; pick one, and mark it.
Mark all of its neighbors too (don't care about their deg.)
\[ \Rightarrow \text{now we've marked } \leq 9 \text{ vertices} \]
Repeat: pick any unmarked vertex w/ deg.$\leq 8$, & mark $\times$

We can do this $\frac{n/2}{9}$ times

get $\frac{n}{18}$ independent vertices w. degree $\leq 8$
we find \( \circ \) with degree \( \leq 8 \)

Degree < 8 \( \Rightarrow \) we don't care ...

Degree > 8
we find $\circ$ w/ deg. $\leq$ 8
and then $\circ$ also w/ deg. $\leq$ 8 and not a neighbor

degree $<$ 8 \quad \{ \text{we don't care} \ldots \}

degree $\geq$ 8
Time to find independent set of degree ≤ 8?

- First mark all degree ≥ 9: $O(n)$ ($\exists d(v_i) = O(n)$)
- Place all unmarked vertices in a list
- Each time scan from start of list for first unmarked vertex (& delete marked)
- When marking neighbors can also delete from list

Requires simple graph structure (access to neighbors)
& links to the list.

$O(n)$ overall
Point Location Data Structure
SURROUND GRAPH WITH 3 VERTICES
Find low-degree vertex
RE-TRIANGULATION IS ARBITRARY
STORE THIS
Another Example
HIGH DEGREE VERTICES
LOW DEGREE VERTEX
& NEIGHBORS
End of Round 1:
Independent Set
Remove INDEPENDENT SET
RETRIANGULATE ARBITRARILY
END OF ROUND 2:
INDEPENDENT SET
When we delete a point, we create a hole: remove $O(1)$ triangles.

Retriangulate: create $O(1)$ triangles in the same region.

Between successive stored triangulations, keep links between overlapping triangles: $O(1)$ links per triangle.
Every iteration:

1) get rid of constant fraction of points
   \[ \geq \frac{1}{18} \quad \text{low-degree} \]

2) re-triangulate: \( O(1) \) per hole

For \( k \) remaining points,

- Time: \( [1] \quad O(k) \) scan all points
- \( [2] \quad O(k) \) \# holes

\( \{ \quad O(n) \quad \text{overall} \quad O(n) \quad \text{size of structure} \)
How to use this structure to locate a query point $Q$

1) Look at top-level structure: △ outer triangle: triangulation $T_1$
   Q outside? DONE. Inside? Dig deeper: look at $T_2$

2) $T_2$
   Repeat question: which of these triangles contains $Q$?

3) Dig deeper:
   find what triangles of $T_3$ overlap △ in $T_2$

Repeat: which of these new triangles in $T_3$ contains $Q$?
get 1 triangle & dig deeper
# triangles

Per internal node:

$1 \leq \text{# outgoing edges} \leq 8$

$O(\log n)$ because we remove const. fraction between levels

Search involves traversal: root $\rightarrow$ leaf:

- at every node do triangle test on all children