Recall

Every simple polygon can be triangulated.

Following the proof yields an obvious $O(n^4)$ time algorithm, which can be improved to $O(n^2)$.

Today: practical $O(n \log n)$ time algorithm.

History:
- 1978. First $O(n \log n)$ algorithm. Garey, Johnson, Preparata, Tarjan.
- 1984. Simpler. Fournier and Montuno. — this is what we’ll study
  - . . . $O(n \log \log n)$ . . . $O(n \log^* n)$ . . .
- 1991. $O(n)$ algorithm. Chazelle. But it is too complicated to implement. (Uses polygon-cutting theorem, planar separator theorem, but no fancy data structures.)

There is also an $O(n \log^* n)$ randomized algorithm by Seidel.
Triangulation Algorithm

Assume points have distinct y coordinates (else imagine tipping slightly).

Note: degeneracy and precision are big topics — we may cover some.

**Step 1.** Find a *trapezoidization* of the polygon — from each vertex shoot a horizontal line inside the polygon until it hits the boundary.

Step 1, using plane-sweep, takes $O(n \log n)$.

Other algorithms make step 1 faster.

**Step 2.** From the trapezoidization, compute a triangulation.

Step 2 takes $O(n)$ time.

This divides partitions the polygon into trapezoids (each with horizontal-top & bottom).

Note that a triangle is a degenerate trapezoid.
Step 2. From trapezoids to triangles.

First join trapezoids to form *unimonomone* polygons.

Every trapezoid has a vertex on the bottom and on the top (exactly one, since we assumed distinct y-coordinates).

If these are not joined by an edge, then add a chord.
Step 2. From trapezoids to triangles.

Resulting pieces are *unimonotone* polygons — with vertices in order of y-coordinate.

Proof.

Every trapezoid not cut by a red chord is like this and can only be followed (above/below) by another with vertices on same side both.
Triangulating a unimonotone polygon in linear time.

Any convex vertex (except \( p_1, p_n \)) provides an ear, and cutting off the ear leaves a unimonotone polygon. Note that there is such a convex vertex.

\[
\begin{align*}
v &:= p_3 \\
\text{loop} \\
\text{while prev}(v) \neq p_1 \text{ and prev}(v) \text{ convex} \\
cut off the ear at prev(v) \\
\text{update prev( ), next( ) pointers} \\
\text{EXIT if the polygon is now empty} \\
v &:= \text{next}(v)
\end{align*}
\]

Correctness: \textit{by induction}

RunTime: \( O(n) \)
Step 1. Trapezoidization.

Use Plane-Sweep, a basic technique in planar computational geometry. First used by Bentley and Ottman, 1979 to find intersections of line segments in the plane.

**Plane Sweep** (as a general paradigm)

Sweep a horizontal scan line across the plane, from bottom to top, analyzing the sequence of 1-dimensional cross-sections and the changes to them.

Cross-section = list of edges that cross the scan line (left to right)

The cross section only changes at vertices.
**Plane Sweep Algorithm** (for non crossing segments)

- order vertices by y coordinate (assume distinct)
- initialize cross-section = $\emptyset$ (at $y = -\infty$)
- for each vertex $p$ in order
  - update cross-section at $p$

Possible updates:

1. ![Diagram](image1.png) one edge replaces another
2. ![Diagram](image2.png) two edges deleted
3. ![Diagram](image3.png) two edges added
**Plane Sweep Algorithm** (for non crossing segments)

how to update the cross-section at p

locate p in the current cross section

determine which situation (1, 2, or 3) applies and perform appropriate update

how to locate p in the current cross section

the cross section contains edges ordered by x (we will store these in a balanced search tree)

We need an elementary test:

Does p lie left/right of edge e
Elementary test needed by Plane Sweep:

**Sidedness Test:** Input: 3 points, A, B, P. Is P on/left/right of line through A and B?

\[ P = (x_P, y_P) \]
\[ B = (x_B, y_B) \]
\[ A = (x_A, y_A) \]

How to solve this:

- Compute equation of line \( y = ax + b \) ? - Division by 0
  - Precision issues
- Test signed area of \( \Delta PAB = A \)
  
  \[ 2A = \text{cross product of } (B - A) \text{ and } (P - A) \]
  
  \[ = (x_B - x_A)(y_P - y_A) - (x_P - x_A)(y_B - y_A) \]

Note: Integer arithmetic (if input is integers)
  - Two multiplications
  - \( P \) left of \( AB \) \iff \( 2A > 0 \)
  - \( P \) right of \( AB \) \iff \( 2A < 0 \)
  - \( P \) on \( AB \) \iff \( 2A = 0 \)
Elementary test needed by Plane Sweep:

**Sidedness Test**: Input: 3 points, A, B, P. Is P on/left/right of line through A and B?

\[ P = (P_x, P_y) \]
\[ A = (A_x, A_y) \]
\[ B = (B_x, B_y) \]

Exercise: Use the sidedness test to test if two line segments intersect. Note how this avoids special cases for vertical lines, parallel lines, etc.
Data structure for Plane Sweep

maintain ordered list (the list of edges crossing the scan line) to allow find, insert, delete

use balanced search tree

AVL trees, red/black

$O(\log n)$ per find/insert/delete.

Timing for Plane Sweep

sort $O(n \log n)$

+ $n$ updates at $O(\log n)$ each

$O(n \log n)$
Details of plane-sweep for trapezoidization.

As the cross section is updated, collect information on each trapezoid

- left & right polygon edges, bottom and top coordinates
- bottom and top neighbours (note 0, 1, or 2 of each)
Applications of general plane-sweep

- find all $k$ intersections of $n$ line segments — there may be $\Theta(n^2)$

general plane sweep can solve this in $O(n \log n + k \log n)$ time.
Applications of general plane-sweep

- map overlay

- Boolean operations on polygons

- Is a polygon simple?

Overlaying simply connected planar subdivisions in linear time
U Finke, KH Hinrichs - Proceedings of the eleventh annual symposium …, 1995 - dl.acm.org

Boolean operations union, intersection and difference on two polygons

Is a polygon simple?

O(n log n)
O(n) by Chazelle
General Plane Sweep — segments may cross
(The above plane sweep assumed that segments do not cross.)

The order of segments changes at an intersection point — we must add the intersection point as an “event” (in addition to the endpoints).

How do we find these events? By checking pairs of adjacent edges in cross-sections.
Use a Priority Queue (PQ) to store events.

initialize cross-section = ∅ (at y = −∞)
initialize PQ to contain vertices
while PQ not empty
    get next event from Priority Queue
    update cross-section at the event
    if a new pair of edges becomes consecutive in the cross section, test for intersection and add event

O(n log n + k log n)  k = number of intersections (might be n²)
See [CGAA], or [Zurich Notes] — discusses primitives

improvement to O(n log n + k) (hard)
further improvement to O(n) space
Triangulating a polygonal region.

More general than polygon — *polygonal region* = polygon with holes

Plane Sweep for non-crossing segments still works. $O(n \log n)$
(Note that we did not require connectedness of the boundary.)

But faster algorithms (e.g. Chazelle’s linear time algorithm) do not work.
A lower bound for trianguating a polygonal region.

Asano, Asano, Pinter, 1986.
Idea: the problem is as hard as sorting, so requires Omega($n \log n$).
Must be careful of the model of computation.

Reduce sorting to triangulation.
Given $n$ distinct integers $x_1, x_2, \ldots, x_n$ to sort, construct a polygonal region such that triangulating gives the sorted order.

polygonal region = rectangle with $n$ square holes

Any triangulation must link right side of hole for $x_i$ to left side of next $x_j$.
Follow links to get sorted order.
A lower bound for triangulating a polygonal region.

Note that this requires indirect addressing.

Thus, we need a stronger model of computation than the comparison-based model where sorting has an easy Omega($n \log n$) lower bound.

Model: unit cost RAM (Random Access Machine) with

- indirect addressing
- branching based on comparisons
- arithmetic $+, -, \times$

There is an Omega($n \log n$) lower bound for sorting on this model (Paul and Simon, 1982).

Thus triangulating polygonal region takes $O(n \log n)$ on this model.
Summary

- plane sweep algorithm (basic tool in 2D)

- $O(n \log n)$ triangulation for polygons and polygonal regions

- pay attention to basic steps of geometric algorithms

- pay attention to model of computing for lower bounds

References

- [CGAA] Sections 3.2, 3.3 (slightly different algorithm)

- [Zurich notes] Appendix A

- [O’Rourke] 2.1 - 2.4