Ariel Stolerman

CS623 Winter 2012 Midterm Notes

Fundamentals:

Let p_1, p_2, p_3 be points in E^2:

$$D(p_1,p_2,p_3) = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = \begin{cases} > 0, & left \\ < 0, & right \\ = 0, & collinear \end{cases}$$

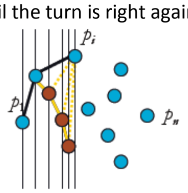
Convex sets and hulls:

- Intersection of convex sets is convex.
- The convex hull of a set of points P is the intersection of all convex sets containing P.
- In E^2 the convex hull is a simple polygon and its vertices are in P.

Representation of a polygon: a CW / CCW chain of vertices ordered on the circumference.

Graham's Scan:

- Partition P to upper and lower sets by the line from the rightmost to the leftmost.
- Sort the 2 sets by their x value.
- Walk each set, and add points that make a right turn with the previous 2.
- If found a left turn, pop previously added points until the turn is right again.



Running time:

- Sorting: $O(n \lg n)$
- Each point is pushed once and popped at most once: $O(n)$
- Total: $O(n \lg n)$

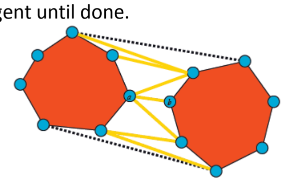
D&C ("Merge-Hull"):

- Sort points by their x value.

Divide: simple $\sim \frac{n}{2}$ division at each phase.

Merge:

- For two convex polygon A, B, we connect them by upper and lower tangents.
- Take a the rightmost point of A, b the leftmost of B, check tangency by orientation in a, b.
- "March" a, b up/down for the upper/lower tangent until done.



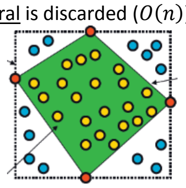
Running time:

- Sorting: $O(n \lg n)$
- Each vertex is searched once on each merge level.
- Total: $O(n \lg n)$

Quick-Hull:

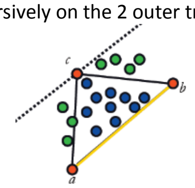
First phase:

- Find min and max x, y-wise to create a boundary box ($O(n)$).
- Cross diagonals between them – every point inside the generated supporting quadrilateral is discarded ($O(n)$).



Second (recursive) phase:

- For each corner triangle with a, b as the 2 red vertices, pick c out of the points in that triangle:
  - Maximize $\angle abc$
  - Maximize the perpendicular from c to ab
- Close the triangle Δabc and eliminate any points in it.
- Run recursively on the 2 outer triangles left.

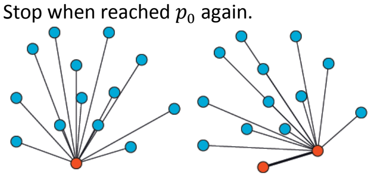


Running time:

- $O(n \lg n)$ expected, $O(n^2)$ worst-case (when all points reside on the boundary).
- $T(n) = T(n_1) + T(n_2) + O(n)$ where n_1, n_2 are the remaining points on each side.

Gift-Wrapping and Jarvis's March:

- Pick the lowest y value point p_0.
- Find the point that minimizes the angle and is a left turn. If more than one is found, take the farthest.
- Stop when reached p_0 again.

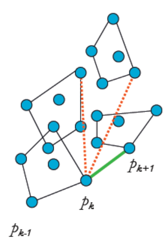


Running time:

$O(hn)$ where h is the number of vertices on the boundary. Worst case: $O(n^2)$.

Chan's algorithm:

- Create $r = \left\lceil \frac{n}{m} \right\rceil$ sets of m points each.
- Run Graham's scan on each one, total: $r \cdot O(m \lg m) = O(n \lg m)$.
- Run Jarvis's march, each time looking for the tangent to all r convex groups. Tangent takes $O(\lg m)$, for r groups, for all h nodes on the boundary: $O(hr \lg m) = O\left(\frac{hn}{m} \lg m\right)$.



How to pick m:

- Run a loop over $t = 1,2, \ldots$ and pick $m = \min\{2^{2^t}, n\}$.
- That squares the previous m on each iteration.
- This process stops at $2^{2^t} \geq h \Rightarrow t = \lceil \lg \lg h \rceil$.
- The total running time: $\sum_{t=1}^{\lg \lg h} n 2^t = n 2^{1+\lg \lg h} = O(n \lg h)$.

Line Segment Intersection:

Sweep-line algorithm:

- Primitive operation: find $s, t \in [0,1]$ that parameterize inner points in the segments p, q. If $p(s) = p(q)$ – they intersect.

Events:

- Sorted list of events be order they meet the sweep line l.
- Endpoint events – the queue is initialized with all.
- Intersection events: added on-the-fly

Sweep line status:

- Balanced tree of the current segments intersecting l, sorted in order of intersection.

Event handling:

At each intersection test, if found – add as new intersection event.

- Left endpoint: insert the new segment, test intersection with its neighbors.
- Right endpoint: delete the segment, test intersection of its previous neighbors (now neighbors themselves).
- Intersection event: swap the segments' position in the status, test intersection with the new neighbors.

Running time:

- The queue has at most $2n + k$ events, k being the number of intersections.
- Each operation: $O(\lg(2n + k)) \underset{k=O(n^2)}{=} O(\lg n^2) = O(\lg n)$
- For every event we do a constant number operations $\Rightarrow$ total: $O\big((n + k) \lg n\big)$

Planar Graphs, Art Gallery Problem:

Planar straight line graph (PSLG):

- Graph in the plain with straight edges that don't intersect.
- Components: vertices, edges, faces.
- Convex subdivision: all faces are convex.

Planar graph:

Euler's Condition:

- $V - E + F = 2$
- If the graph is disconnected and the number of connected components is C: $V - E + F - C = 1$
- $\Rightarrow$
  - $E \leq 3(V - 2)$
  - $F \leq 2(V - 2)$

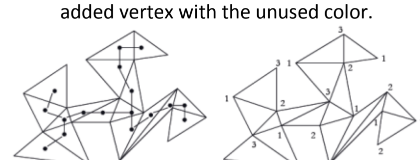
DCEL (doubly connected edge list):

- Vertices: store coordinates and incident edges (that originate from them).
- Edges: points to twin, next, prev, origin vertex, left face.
- Faces: pointer to one of its edges.

The Art Gallery Problem:

$\lfloor n/3 \rfloor$ is guaranteed to suffice:

- Lemma 1: Every n polygon has $n - 3$ diagonals and $n - 2$ triangles:
  - Breaking P with n vertices into two polygons gives $m_1 + m_2 = n + 2$ vertices.
  - Induction: $m_1 - 2 + m_2 - 2 = n - 2$ triangles in P.
  - Similar with diagonals.
- Lemma 2: The triangulation graph of a simple polygon is 3-colorable:
  - Color the dual graph (tree with vertex degree ≤ 3) inductively:
  - Remove all triangles (dual nodes) until you get one, color it.
  - Add the removed ones, coloring the added vertex with the unused color.



  - At least one color appears $\leq \lfloor n/3 \rfloor$ times, place guards on those colors.

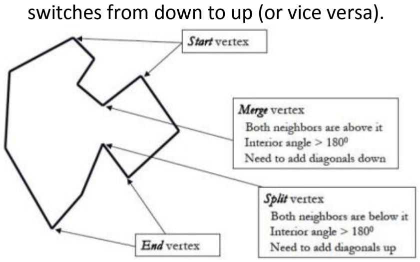
Polygon Triangulation:

Lemma: every simple polygon admits a triangulation with $n-2$ triangles:

- By induction, pick a convex corner qpr and cross a qr. If it is not $qr \subset P$, cross a diagonal pz, z being a vertex of P fartherst from qr inside Δqpr.
- By induction each part has $n_1 - 2, n_2 - 2$ tirangles, thus P has $n_1 + n_2 - 4 = n + 2 - 4 = n - 2$ triangles.

Monotone polygon:

- P is monotone w.r.t. line l if any orthogonal to l intersects P's boundary in ≤ 2 points.
- y-monotone: when we walk from topmost vertex down on each of the chains, it's always a horizontal move down.
- Turn vertices: the direction of the walk switches from down to up (or vice versa).

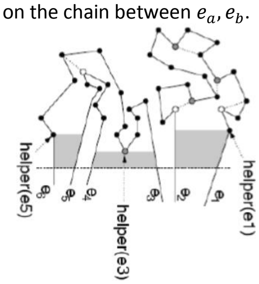


Lemma: a polygon is y-monotone if it has no split/merge vertices.

- Suppose it's not, then an orthogonal intersects P at p, q, r. Go up on the chain from q until it hits r.
- If $p \neq r$, take the topmost vertex on this trail - **split**. Otherwise go from q downwards until some intersection r' and $r' \neq p$ since they're not on the same component.
- The lowest vertex between q, r' is a **merge** vertex.

Phase #1: Split P to monotone pieces:

- There are edges of P, e_a, e_b, on each side of a split vertex v.
- $Helper(e_a)$: maintains the current vertex that is visible by any v between e_a, e_b.
  It is the lowest horizontally visible to the left of e_a on the chain between e_a, e_b.



- On split vertices v, cross $v \to helper(e_a)$

Events:

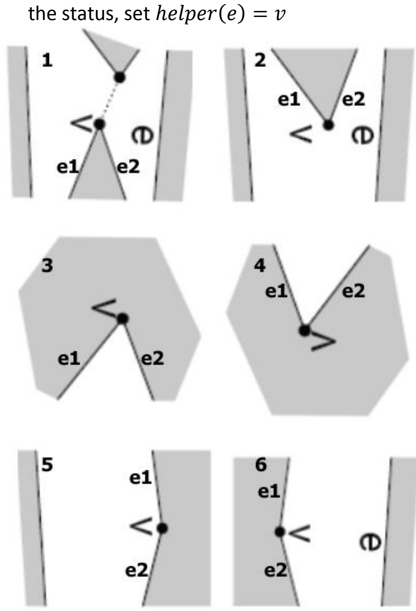
- Sorted list of P's vertices in y-descending order.

Status:

- List of edges that intersect the sweep line from right to left.
- Balanced tree of e_a's: edges that have P's interior to the left of them, keys to $helper$.

Event processing:

1. **Split**: find e, connect v to $helper(e)$, add e_1, e_2 the status, set $helper(e) = helper(e_1) = v$
2. **Merge**: find and delete e_1, e_2, set $helper(e) = v$
3. **Start**: insert e_1, e_2 to the status and set $helper(e_2) = v$
4. **End**: delete e_1, e_2 from the status
5. **Right-chain vertex**: delete e_1 and add e_2 to the status, set $helper(e_2) = v$
6. **Left-chain vertex**: delete e_1 and add e_2 to the status, set $helper(e) = v$



Merge vertices:

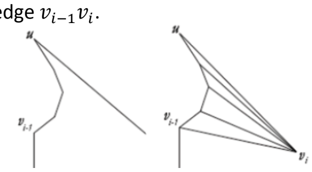
- Either do the same upside-down, or:
- At any helper update, if the old helper was a merge vertex, connect it to the new helper vertex.

Running time:

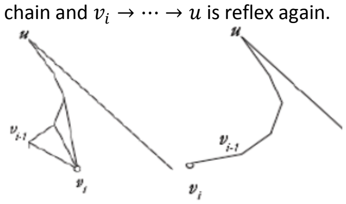
- Each event costs $O(\lg n)$, total of n events.
- Total: $O(n \lg n)$.

Phase #2: Triangulating the monotone pieces:

- Start with the y-sorted vertices of the polygon ($O(n)$ by merging the two chains).
- Hold a stack that accumulates vertices that await for diagonals (a reflex chain), and an indicator if that's on the L or R chain.
- Denote u the current topmost vertex that everything above it is triangulated, v_{i-1} the one just processed and v_i the one about to be processed:
  - $\boldsymbol{v_i}$ **is on the opposite of** $\boldsymbol{v_{i-1}}$: cross diagonals from v_i to all vertices on the reflex chain $v_{i-1} \to u$ (expect u). v_{i-1} becomes u, the reflex chain contains the edge $v_{i-1}v_i$.



  - $\boldsymbol{v_i}$ **is on the same chain as** $\boldsymbol{v_{i-1}}$: walk back on the chain and cross diagonals to visible vertices. May or may not include vertices. At the end v_i is the new endpoint of the chain and $v_i \to \cdots \to u$ is reflex again.



Running time:

- Merging both chains to a sorted one is $O(n)$
- The orientation (visibility) test is constant, as adding a diagonal (in a DCEL).
- There are $n-3$ diagonals $\Rightarrow$ total: $O(n)$