Homework 1

1 Left Turn and Convexity

(a) Derive and verify that, for the given points $A = (x_a, y_a)$, $B = (x_b, y_b)$, and $C = (x_c, y_c)$ (in this order), and determinant

$$D = \begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix},$$

A, B, and C appear in counterclockwise (ccw) order on the boundary of ΔABC if and only if D is positive (i.e. A, B, C form a left turn if $D > 0$).

The determinant can be calculated by:

$$\begin{aligned} D &= x_a \begin{vmatrix} y_b & 1 \\ y_c & 1 \end{vmatrix} - y_a \begin{vmatrix} x_b & 1 \\ x_c & 1 \end{vmatrix} + 1 \begin{vmatrix} x_b & y_b \\ x_c & y_c \end{vmatrix} && (1) \\ &= x_a(y_b - y_c) - y_a(x_b - x_c) + (x_b y_c - y_b x_c) && (2) \\ &= (x_a y_b - x_a y_c) + (x_c y_a - x_b y_a) + (x_b y_c - y_b x_c) && (3) \end{aligned}$$

Notice that the result on line (3) is the result of the cross product of the two vectors $\overrightarrow{AB}$ and $\overrightarrow{BC}$, i.e. $\overrightarrow{AB} \times \overrightarrow{BC}$. I show the determinant method of finding the cross product:

$$\overrightarrow{AB} = \langle x_b - x_a, y_b - y_a, 0 \rangle$$
$$\overrightarrow{BC} = \langle x_c - x_b, y_c - y_b, 0 \rangle$$

$$\begin{aligned} \overrightarrow{AB} \times \overrightarrow{BC} &= \begin{vmatrix} i & j & k \\ x_b - x_a & y_b - y_a & 0 \\ x_c - x_b & y_c - y_b & 0 \end{vmatrix} \\ &= \big[(x_b - x_a)(y_c - y_b) - (y_b - y_a)(x_c - x_b)\big]k \\ &= \big[(x_a y_b - x_a y_c) + (x_c y_a - x_b y_a) + (x_b y_c - y_b x_c)\big]k \end{aligned}$$

By the right-hand rule convention, when the cross product is positive, the direction along the path from $A \to B \to C \to A$ of the boundary

of ΔABC is ccw, where our C is to the "left" of directed line $\overrightarrow{AB}$; when the cross product is negative, the direction along the path from $A \to B \to C \to A$ of the boundary of ΔABC is clockwise.

Reference: [1] [2]

(b) Describe an algorithm to test if polygon P (given by a circularly linked list of its n vertices in order) is convex.

From class, a convex polygon is such that, walking along the boundary of the polygon, all angles "turn" in the same "direction", resulting in all internal angles being less than 180° and only consecutive segments intersecting.

*Let there exist a method that outputs the sign of the determinant of three given coordinate points, and call it DeterminantSign. Assume it runs $O(1)$ time by plugging in the coordinate values in the formula(s) like those mentioned in part (a).

Input: A circularly linked list of the n ordered vertices of polygon P. We will call these vertices in their sequence $p_1, p_2, ..., p_n$, where p_1 is arbitrarily selected as the "first" vertex we examine in the algorithm below, p_2 is the "next" node/vertex of p_1, etc.

Output: **true** or **false** if P is convex.

1. $A \leftarrow p_1$; $B \leftarrow p_2$; $C \leftarrow p_3$
2. `direction` $\leftarrow$ DeterminantSign(A, B, C)
3. **for** $i \leftarrow 2$ **to** n
4. $\quad A \leftarrow B$; $B \leftarrow C$; $C \leftarrow C$.next
5. $\quad$ `turn` $\leftarrow$ DeterminantSign(A, B, C)
6. $\quad$ **if** `turn` $\neq$ `direction` **then** return **false**
7. return **true**

Correctness : The algorithm examines each adjacent angle of the polygon formed by three adjacent points. The direction (in terms of left/right or positive/negative) of the first arbitrary (interior) angle chosen is given by direction (line 2) and formed from 3 vertices. I show by induction that this algorithm works for all polygons of $k \leq n$ vertices, where $3 \leq k \leq n$.

For $k = 3$ vertices (triangle), it is trivially true that P is a convex polygon, and the direction of the k angles match and all are acute. For $k = \{3, ..., i\}$, let all the directions of the angles up to the angle about p_{i-1} (i.e. $\angle p_{i-2}, p_{i-1}, p_i$) match the first, thus far, and therefore P is convex. The next angle formed with the $i \rightarrow \mathsf{next}^{th}$ vertex, given by $\angle p_{i-1}, p_i, p_{i \rightarrow \mathsf{next}}$, will have a turn that either does or doesn't match $\mathsf{direction}$. If it matches, P is convex because all turns are the same. If it doesn't, P is not convex because the angle makes it either a concave polygon or complex polygon.

Complexity : Let accessing the next adjacent node in the linked list be a $O(1)$ time operation. Each line in the algorithm runs $O(1)$ time except for the for loop in lines 3 - 6. The for loop examines the "turn" of only each adjacent triple on the list of vertices. Therefore, the for loop runs $O(n)$ time. We are not concerned with any sorting of the vertices here. Therefore, the total runtime of the algorithm is $O(n)$.

2 Unordered Divide-and-Conquer Convex Hull

(a) For P_1 and P_2 to share a support line, the line must have at least one vertex from each polygon on it, and all the vertices of both polygons must be to one side of the line. If P_1 and P_2 share no vertices, then they are either 1) completely disjointed, 2) one is completely interior to the other, or 3) their edges intersect.

If they are disjointed, then P_1 and P_2 can only have 2 support lines in common, both of which would create the new edges of the convex hull created from $P_1 \cup P_2$, regardless of the values of n_1 and n_2. In other words, each vertex of P_1 has 1 or 0 shared support line with P_2. Figure ...

If one is interior to the other, regardless of the values of n_1 and n_2, then there are 0 support lines, because any support lines the interior polygon has would intersect the exterior polygon, and any support lines the exterior polygon has would never share a vertex from the interior polygon. Figure ...

Finally, the most number of support lines that can occurr at one vertex is 2, because that is when the vertex of one convex polygon is in a position "between" two vertices of the other polygon, and the edges of the polygon that include that 1 vertex intersects the greater polygon at the edges formed by those two vertices; otherwise, it may have 0 shared support lines in the case that one vertex of P_1 is interior to P_2, or it has 1 shared support line in the case that one vertex of P_1 is adjacet to only one vertex of P_2, and only one of the edges formed by that vertex in P_1 intesects only one of the edges formed by that one vertex in P_2. Figure ... This can happen at most $2 \times \texttt{min}(n_1, n_2)$, limited by the number of vertices of the lesser polygon and depending on how the two polygons overlap.

There can be 0 common support lines wherever P_1 and P_2 share an edge, because that would place the vertices of the polygons on opposite sides of the line. Figure ...

Therefore, the number of common support lines of P_1 and P_2 is at most $2 \times \texttt{min}(n_1, n_2)$.

(b) I specify an algorithm that computes the convex hull of convex polygons $P \cup Q$. Since they are already known to be convex, their vertices are already sorted.

Input: The sorted vertices of convex polygons P and Q, ordered respectively by $\{p_1, ..., p_n\}$ and $\{q_1, ..., q_m\}$. Note that $|P| = n$ and $|Q| = m$.

Output: $\mathcal{CH}(P \cup Q)$

MergeConvexHulls(P, Q):

1. $X \leftarrow$ *Merge* P and Q.
2. Find the convex hull of sorted X using any convex hull finding algorithm on a presorted list. I suggest GrahamScan.
   So $Y \leftarrow$ GrahamScan(X).
3. return Y

The runtime is $O(N)$ (for $N = n + m$) because it takes $O(N)$ time to sort the already sorted elements of P and Q into one list X. And then we are running Graham Scan on a presorted list, which itself is then $O(N)$ time.

(c) I specify a divide-and-conquer algorithm that uses MergeConvex-Hulls without presorting to find the convex hull of an arbitrary set of n points in set S.

DC-ConvexHull(S):

1. $m \leftarrow \frac{n}{2}$
2. $L \leftarrow S[0 : m - 1]$ and $R \leftarrow S[m : n - 1]$
3. $L' \leftarrow$ DC-ConvexHull(L)
4. $R' \leftarrow$ DC-ConvexHull(R)
5. $S' \leftarrow$ MergeConvexHulls(L', R')
6. return S'

The runtime is $O(n \log n)$ because it divides S in half to find the convex hulls of the separated set, and takes linear time to "merge" the two convex hulls (convex polygons) that were found, i.e. find the convex hull formed by the two given convex hull.

3 Three

Given a set S of n planar points in general position and in arbitrary order, I construct these polygons with the following algorithms:

(a) A simple monotone polygon R whose vertices are exactly the vertices of S, which contains the line segment given by the min and max x-coord vertices of S.

MonotoneHull(S):

1. $S' \leftarrow$ sort S by x-coordinate. $\Theta(n \log n)$
2. Create the line segment ($y = mx + b$) from the first and last vertices in S' (min and max x-coord vertics of S). Let b be this line's y-intersection, and m the slope.
3. **for** $i \leftarrow 0$ to n $\Theta(n)$
4. $B[i] \leftarrow y_{S'[i]} - mx_{S'[i]}$
   (*the y-insersection of the line given by the point* $S'[i]$ *and the slope* m)
5. $j \leftarrow 1$, $R[0] \leftarrow S'[0]$
6. **for** $i \leftarrow 1$ to n $\Theta(n)$
7. **if** $B[i] > b$
8. **then** $R[j] \leftarrow S'[i]$, $j \leftarrow j + 1$
9. **for** $i \leftarrow n - 1$ downto 0 $\Theta(n)$
10. **if** $B[i] < b$
11. **then** $R[j] \leftarrow S'[i]$, $j \leftarrow j + 1$
12. return R

This algorithm runs $\Theta(n \log n)$ time due to soring the points by the x-coordinates (line 1). We can find the slope and y-intersection of the line segment given by the min and max x-coord points in S' (line 2). Using that m and b, we can create a list of the y-intercepts of each point in S' on a line of slope m. This tells us whether or not a point in S is above or below the line segment. Finally, we compute R by finding the list of points above the line segment in order of the sorted S' (loop 6), and adding to it the points below the line segment (found the same way and discarding any duplicate end points) (loop 9).

(b) A simple starshaped polygon P whose vertices are exactly the points in S for which the min x-coord point in S "sees" every point in P.

StarHull(S):

1.

(c) For a set of points Q such that $\forall q \in Q$, $q \notin S$, describe algorithms to determine if q is interior or exterior to P and R.

RelativeToMino(Q, N):

1.

References

[1] Cross Product of Two Vectors: https://www.cuemath.com/geometry/cross-product/

[2] Math World - Curve Orientation: https://mathworld.wolfram.com/CurveOrientation.html

[3] Jake and Diane's office hours, classmates: Stephanie, Sydney, and MacKenzie