Computational Geometry and Topology: Assignment 1

Ishan Agarwal

January 2017

A Note on Notation

- All angles are in degrees.
- The symbol $\vec{PQ}$ refers to the line segment directed from P to Q.
- As per instructions on Canvas, all points are assumed to be in non-degenerate position unless specifically stated otherwise.

Problem 1(a)

Consider the vector $\vec{PQ}$ and the vector $\vec{QR}$ as vectors in R^3 with 0 z- component. We look at the vector $\vec{PQ} \times \vec{QR}$ (the usual vector cross product).R is to the right of $\vec{PQ}$ as the z component of $\vec{PQ} \times \vec{QR}$ being positive. If it is negative we say R is to the right of $\vec{PQ}$. (Note that it cannot be zero unless R lies on PQ.)

This definition itself provides an obvious test:
Look at the z component of the vector $\vec{PQ} \times \vec{QR}$: $D = (q_x - p_x)(r_y - q_y) - (q_y - p_y)(r_x - q_x)$

If D is positive then R is to the left of PQ while if D is negative R is to the right of PQ.

Problem 1(b)

Suppose at some stage of the Jarvis March algorithm we have last found the point P. Then the next point is Q such that for all points R in the point set, R lies to the left of the directed line segment $\vec{PQ}$ (assuming that the first point we picked in the algorithm was the leftmost point).(for the first point just use the line through the leftmost point that is parallel to the x-axis) Note that this method does still take $O(n^3)$ (or $O(h.n^2)$) time.

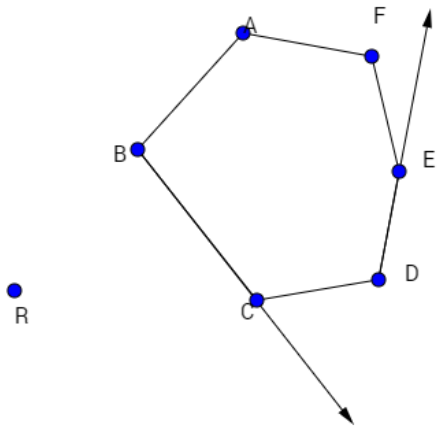
Firstly such a point Q exists and is unique. Say there were two such points Q_1 and Q_2. But then in particular Q_2 is to the left of $P\vec{Q}_1$ but similarly Q_1 is to the left of $P\vec{Q}_2$. It is clear that this is impossible. Existence of such a Q is also clear: say for all Q there was some point R_Q to the left of $\vec{PQ}$. Consider the following procedure: maintain a list of points not yet tried. If Q is such that some point R_Q is to the left of $\vec{PQ}$ then delete Q from the list and try R_Q. This process must end as there are only finitely many points and hence some point exists that fulfils this criteria.

It is easy to prove the above method of finding the next point is correct as we can show that it is equivalent to finding the point that makes the least angle with the last segment, however it is cumbersome to write out the formal proof.

Problem 2(a)

We show how to find the left tangent, the right tangent can be found in an entirely analogus manner. We use a binary search technique. To do so we impose an ordering of the vertices of the convex hull. given two vertices V_1 and V_2 and a point P from which we are trying to find the tangent, we say V_1 is greater than V_2 if V_2 lies to the left of $P\vec{V}_1$. Also an edge $V_i\vec{V}_{i+1}$ is increasing if P lies to the right of $V_i\vec{V}_{i+1}$ else it is decreasing. The left tangent point is clearly the minima with respect to this ordering while the right tangent point is the maxima. Note that as the point s are stored in clockwise order local minima is the same as global minima.(same holds for maxima).

The algorithm is as follows:Consider the input arr[ ]=$V_0, V_1, ...V_n$.



Edge DE is increasing while edge BC is decreasing

- Initialize a=0,b=n
- Set A to be the vector $V_0, \vec{V_1}$.
- (*)$c = \frac{(a+b)}{2}$.
- If V_c is local minima, i.e both edges from C are increasing then output $P\vec{V_c}$ as left tangent and end.
- if not compute if V_c is greater than V_a. If it is we must search between a and c. So set b=c.
- else set a=c
- goto (*)

For this algorithmthe recurrence for time complexity is $T(m) = 2T(\frac{m}{2}) + c$ so this is a O(log(m)) algorithm.

Justification for why above algorithm works: We need only show that the ordering we have described is really an ordering but using test given in q.1(a) it is easy to check transitivity of this relation.(we omit the calculatons here.)It is also easy to see that convexness of the hull implies the existence of a unique maxima and minina under this ordering.

Problem 2(b)

Consider the following algorithm: Consider the vertices, V_i to have been sorted in increasing order of x-co-ordinate and if there is a tie using y- co-ordinate. (this itself takes O(nlog(n)) time.).We store the current hull, in counter clockwise order, in a linked list H_k

We start with the hull being just the leftmost point. $H_0 = V_0$. We show how to compute the hull at the k^{th} stage. We know point V_k lies outside H_{k-1} due to the pre sorting. Thus compute upper and lower tangents from V_k to H_k. This takes at most O(log(n)) time. Now suppose the lower tangent is $\vec{V_kV_l}$ and the upper one is $\vec{V_kV_m}$. Then the original list H_{k-1} must be of the form $V_0, V_1, ...V_l, V_{l+1}, ...V_{m-1}, V_m, ...V_k - 1$. Remove the part in between V_l and V_m, replacing it with just V_k. In a linked list this is a constant time operation. Thus we have $H_k = (V_0, ...V_l, V_k, V_m, ..V_p)$. Each step takes time at most O(log(n) and so total time for this part is also bounded by O(nlog(n)).

Correctness of the algorithm: It suffices to show that at each stage we have the correct convex hull of k points. Suppose it was correct at stage k-1. (it was definitely correct at stage 0). Now at the k^{th} stage we found tangents from the point to be added to the whole hull. Clearly these tangents must be edges as all the points will lie to only one side of each of them. Furthermore no new edges need be added. Now all the edges within the convex hull are precisely those between points between the two points of tangency and these pointts must be removed. Thus the new hull is correctly computed.(this is only a rough justification but it contains all the essential ideas for a full formal proof.)

Problem 3

Lemma 1:
Sorting has time complexity at least nlog(n). [This is a well known result for comparision sorts]

Given a list of numbers $a_1, a_2, a_3....a_n$ consider the tuples $(a_1, a_1^2), (a_2, a_2^2)....(a_n, a_n^2)$ as points in R^2. Suppose we have an algorithm A that outputs, as a linked list, the convex hull of this point set with the leftmost point first.(We may as

wel assume this as it takes only O(n) time to convert the output into the form that has the leftmost point first.) We claim that taking the first co-ordinate of the points in this list from the leftmost to the last point before the list points back to the first element will result in a sorted sequence of the a_i's in ascending order. This would mean that the sorting problem reduces to the convex hull problem with only O(n) overhead. Hence if we can find convex hull in less than O(nlog(n)) time by using A then conside an algorithm B that takes a list of a_i's and inputs $(a_1, a_1^2), (a_2, a_2^2)....(a_n, a_n^2)$ to A and runs A. It can give the sorted output in time only O(n) more than the running time of A. Thus B would have time complexity less tha O(nlog(n)) which is a contradiction (by lemma 1). Hence A must take at least O(nlog(n)) time.

We now show that the process described above would indeed sort the a_i's.let $y = f(x) = x^2$. Note that all the points in the point set lie on this curve. Note that the convex hull algorithm will output a list, with the first point, call it P_0 being the one with the lowest x- co-ordinate and henceforth each new point P_i will be such that all points in the point set lie to the left of the directed line segment $P_i\vec{P}_{i+1}$.

Thus the first point P_0 will definitely have the lowest x- co-ordinate as it is the leftmost. We need to show that at each stage points will be chosen such that all the points are chosen at some stage and they are chosen in increasing order of x- co-ordinate. Thus it suffices to prove that at every stage when the last point chosen is P_i the next point chosen must be Q such that Q has x- co-ordinate greater than P_i and there is no point P_j not already listed with x- co-ordinate between that of Q and P_i. If this is true then all points will be selected and in ascending order of x- co-ordinate, providing of course that two points do not share the same x- co-ordinate. This is however the case as all the points lie on $f(x) = x^2$.

Note that the convex hull algorithm, when the last chosen point is $P(p, p^2)$ has to select a point $Q(q, q^2)$ such that all other points are to the left of $\vec{PQ}$. Suppose some point $R(r, r^2)$ exists with r lying between p and q. We show that Q cannot be chosen because R will not loie to the left of $\vec{PQ}$. $r = tp + (1-t)q$ where t is strictly between 0 and 1. Applying the test in question 1(a) (after simplification): for R to lie to the left $t(q-p)^2(p-q)(1-t)$ should be greater than 0 but p is less than q and $1-t$ is positive so this is

not the case. Thus at each step the point with the nearest next greater x-co-ordinate is chosen leading to the points being output in the convex hull to have as x- co-ordinates the sorted list of numbers.

Problem 4

Lemma 1:
Every vertex in a Voronoi diagram has degree at least 3. The proof of this lemma follows from observing a characterization of the vertices of a Voronoi diagram as precisely those points such that the maximal radius circle that is centred at the point and contains none of the original point set, has on it's boundary 3 or more points from the original point set. Similarly points on the edges have exactly two of the original points on the boundary of the maximal radius circle (which is empty) which is centered at that point.

We state Euler's formula for a planar graph:

$$n_V - n_E + n_F = 2$$

where n_E, n_V and n_F are the number of edges, vertices and faces respectively of the diagram $D^{'}$ defined subsequently. A Voronoi diagram, D contains some "edges" with vertices at one end only so consider the Voronoi diagram with all such edges having a common end point at the other end. Call this $D^{'}$ E,V and F are the number of edges, vertices and faces respectively of the Voronoi diagram D itself.

This is now a planar graph. Note that the number of faces must be exactly n as each point in the point set belongs to exactly one face.(clearly each point must lie in some face as the points are non degenerate so there must be some region, for every point P, where P is the closest point. Furthermore a point cannot lie in two faces as it would have to lie in the intersection of the faces but faces in a Voronoi diagram, due to the fact that they are regions where a particular point is closest, do not intersect.) Thus $F = n$.

Thus:

$$(V + 1) - E + n = 2 \Rightarrow E = n + V - 1$$

Furthermore the total degree is twice the number of edges, however each vertex has degree at least three (Lemma 1) thus the total degree is at least

$3(V+1)$. Thus:

$$2E \geq 3(V+1)$$

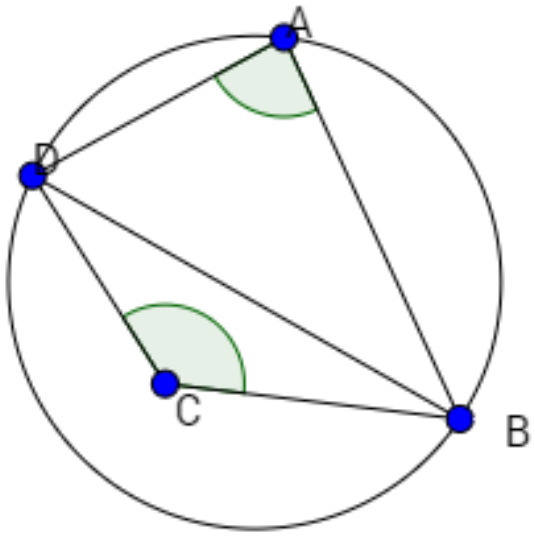
Substituting $E = n+V-1$ we have $2n+2V-2 \geq 3V+3 \Rightarrow V \leq 2n-5$.

Substituting $V = E-n+1$ we have $2E \geq 3E-3n+6 \Rightarrow E \leq 3n-6$

Problem 5

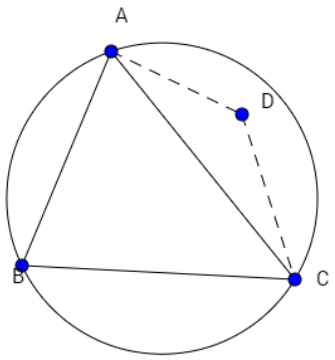
If the triangle empty open circumdisk property holds for every triangle in the triangulation then as every edge is part of some triangle, trivially, the edge empty open circumdisk property holds.

We now prove the other direction. Lemma: If an edge has the edge empty open circumdisk property then say it is a common edge to triangles ABC and ABD. Then the circumcircle of ABC must not contain D.
This is because the sum of the angles DAB and DCB is less than 180 as



angle DCB is less than angle DXB for any X on the circle on the arc between B and D while angle BXD+ angle BAD is itself 180. Hence there is no cicle with BD as a chord, with A and C both outside because if there were the sum of DCB and DAB is less than 180.

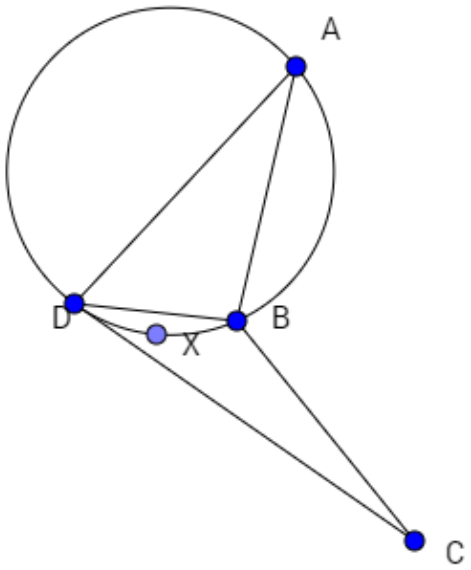
We now prove the result. Consider there is some triangle ABC whose circumcircle contains some other vertex D. Then D must lie in one of the segments corresponding to either AB or BC or CA, as it cannot lie in the triangle ABC as ABC is part of the triangulation. Without loss of generality say it lies (as shown) in the segment corresponding to AB.



But then consider the edge AC. By the lemma D must not lie in the circumcircle of BAC as otherwise, by the lemma, AC will not have the edge open empty circumcircle property. But D lies in the circumcircle of ABC by assumption. CONTRADICTION.

Problem 6

Consider the quadrilateral ABCD which is concave. Say the interior angle ABC is greater than 180 (As the quadrilateral is concave some such angle must exist.) Consider BD. We show that BD must be locally Delauney.



Consider the open circle passing through A, B and D. We must show that this does not contain the point C. Suppose C lies inside the circle ABD. Note that angle ACB would be greater than angle ACX for any point X on the circle ABD in the arc going from B to D. However the sum of opposite angles of a cyclic quadrilateral is 180 (a well known result). thus angle BAD+ angle BCD will be greater than angle BAD+ angle BXD which will be 180. However as ABCD is a quadrilateral the sum of the angles

$$BAD + BCD + ABC + ADC = 360$$

and angle ABC is greater than 180. So BAD + BCD must be less than 180, hence CONTRADICTION.

Assumptions regarding Problem 7

We assume that by trangles on the boundary we mean triangles with edge on the boundary, i.e that edgeis not shared with any triangle. We also assume that the surface is orientable and 2-D which implies that it can be represented(upto homotopy) by a triangulation. We also assume we are given one triangle edge to start counting from.
Further note that we can mark edges with colours, the sign bits of the triangle edge fields can be used to store upto eight different markers as explained in Mucke's thesis chapter. We use three markers m_1,m_2 and m_3 with two different bits.

Lastly we assume that the space is connected. If it has multiple connected components we could apply the algorithm to both connected components successively.

Problem 7(a)

We start with some triangle edge pair. To test is any given triangle edge pair, a, is on the boundary note that this happens if and only if a.fnext = a. Thus we can, in constant time check if an edge is on the boundary. We maintain a list L of triangle edge pairs which at the first step has just the original pair a. At each step do the following:

- for each pair in the list check if that pair is on the boundary . If it is, and it is not already labelled with the marker m_1 increment the count of triangles on the boundary by 1. Now label this pair a, as well as a.enext and $a.enext^2$ by some specific marker, m_1.

- For each pair in the list, a, label a with the marker m_2, add a.enext and a.fnext to the list L, provided they themselves are not labelled m_2 and then delete a. If the list is non empty goto previous step.

At the end of this process the counter will have the total number of triangles with an edge on the boundary. Justification: Firstly note that this algorithm will check every triangle edge pair for lying on the boundary because it performs a breadth first search. At each stage it looks at a set of edges and checks them and then in the next step looks at all the pairs

reachable from them by fnext and enext operations. Then it marks the previous set of edges as visited, by marker m_2 and moves on. Further as every triangle edge pair is part of precisely one edge ring and one triangle ring, it is clear that all triangle edge pairs can be reached by a sequence of these operations from any given pair. Further if one edge is part of a triangle that lies on the boundary Then all edges of that triangle are marked m_1, thus avoiding over counting. Furthermore, each edge is visited at most once and hence the algorithm concludes in time $O(n_e)$ which is $O(n_v + n_f)$.

Problem 7(b)

We use a similar process, i.e we perform breadth first traversal. We start with counters for n_e,n_v and n_f. Finding these is all that is needed to compute the Euler characteristic. To find these we traverse the graph exactly as we did in the first part. . We maintain a list L of triangle edge pairs which at the first step has just the original pair a. At each step do the following:

- for each pair in the list add 1 to n_e. Also if the pair a is not marked with m_1, add 1 to n_f and mark a, a.enext and $a.enext^2$ with m_1.
- for each pair a, in the list L; look at b=a.enext and c=a.fnext. Mark a with m_2. If b is not marked with m_2 add it to the list and do the same for c. Delete a from the list If the list is non -empty goto the previous step.

Similar to the analysis for the first case the use of markers prevents over counting. Also the algorithm runs in O($n_v + n_f$) time as each edge is visited only once. Furthermore each edge does get checked exactly once.

To count the vertices create another list L_1 as above and do the following.

- for each a in the list add 1 to n_v.Mark a with m_2 Now start a new list L_2 mark a , a.fnext and $a.enext^2.sym$. Remove a from L_2 and keep marking v.fnext and $v.enext^2.sym$, for all v in the list, with m_3 as long as these two are not already marked. Keep doing this till all edge pairs reachable from a by the operations a.fnext and $a.enext^2.sym$ have been marked with m_3.

- Now look at the list L_1. For every element a in the list look at a.enext and a.fnext. If these are not marked with m_2 add them to the list and delete a. If the list is non-empty goto step 1.

This algorithm takes time at most $O(n_v + n_f)$ as each edge is checked only once. Furthermore when we add a vertex to the count we mark all edges starting at that vwertex and they are not re-counted.