

Duality, Arrangements

Sudipto Guha

February 2, 2003

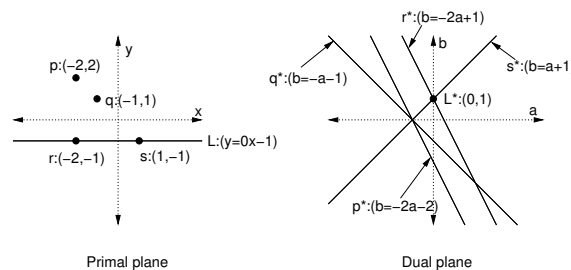
1 The Duality

Consider the canonical equation of a line in the Cartesian coordinate system. The equation of a non-vertical line is typically written as $y = ax - b$. Thus a line is specified by two quantities (a, b) . We can actually view the lines as ordered tuples or *points in some other plane*.

This is the notion of duality. A line $y = ax - b$ is transformed to a point (a, b) in the *Dual plane*.

Consider a point (u, v) in the original or *Primal plane*. The point can be defined as the intersection of lines $\{c(x - u) + v\}$ as c can be arbitrary. In the dual plane the dual of these lines is the set of points $\{(c, cu - v)\}$, all of which are solutions to the equation $Y = uX - v$. Thus the point (u, v) in the primal plane corresponds to the line $Y = uX - v$ in the dual plane.

Given the mapping of points and lines, it is easy to verify that the dual of a dual is the primal plane. For example the point (u, v) in the primal plane gets mapped to line $Y = uX - v$ in the dual and to (u, v) in the dual of the dual. Similarly the line $y = ax - b$ gets mapped to (a, b) in the dual then to (a, b) in dual of the dual. An example map is in Figure 1.



It is easy to verify the following: Given a line $\ell : y = ax - b$ and a point $p = (u, v)$ in the primal plane such that p lies above ℓ (which means $v \geq ua - b$) the dual line $p^* : Y = ux - v$ lies below the dual point $\ell^* = (a, b)$. Since

$$v \geq ua - b \quad \Leftrightarrow \quad b \geq ua - v$$

Thus if a point lies on a line, the incidence is preserved in the dual (the dual line is incident on the dual point). Furthermore as order is preserved – so is betweenness. Consider a line ℓ containing points p, q, r in increasing x values. In the dual the lines p^*, q^*, r^* will pass through the dual point ℓ^* – and their slope will also be in increasing order, since the x -coordinate in the primal is the slope in the dual plane.

These properties of the dual allow us to solve a large number of problems quite easily, by solving the problem in the dual plane.

For example consider the problem: Given n lines not all coincident, prove that there exists a point such that exactly two lines are incident on it. Now consider the statement in the dual plane: we will map lines to points, and coincidences map to collinearity. Thus the statement in the dual is : given n points not all collinear, prove that there exists a line such that exactly two points are incident on it. This is the well known Sylvester’s Theorem and several proofs exist of this fact.

The contrapositive of the statement(s) are also interesting : that if for every pair of lines intersecting at a point, there exists a third line passing through the point; all the lines intersect at a point. The dual statement is: if for every line defined by a pair of points there exists a third point on the line then the lines are collinear.

It may seem that the dual we constructed allowed us prove such dual statement. However, the property of duals we used is (a) order preservation (b) preservation of incidences and (c) *idempotence* i.e that the dual of the dual mapping is the identity mapping. Consider the dual statement of Sylvester’s theorem – it can be proved based on (a), (b), and (c). In fact the proof only uses (b) and (c). Later in the notes we will see an application of all three (angular sort). Any mapping that satisfies the three will yield a dual. Thus depending on the application we may choose one dual over another. But the duals constructed through different maps will be transformations of one another. For example we can also specify every line as $ax + by = 1$ and thus it maps to (a, b) in the dual. Notice that the same equation can be written as $y = (a/b) - (1/b)$ and be mapped to $(a/b, -1/b)$ in our original dual transform.

Notice that the duals have interesting properties since in the transforms based on $ax + by = 1$ the point $(0, 0)$ is excluded from the dual plane. Likewise the lines $x = c$ are ruled out from the primal plane in the definitions of duals we considered. The homework asks you to verify that the transform of $ax + by = 1$ to (a, b) is a proper dual transform.

Consider the following problem:

Question 1 *Given n points in a plane preprocess the pointset such that given a query point q we can construct the angular order of the points with respect to q as fast as possible.*

A $O(n \log n)$ algorithm is trivially possible. The question is: can we do any better. To answer this question, we will look at the dual plane. The points will map to lines and we will be concerned with the geometry of n lines in a plane.

2 Arrangements

Definition 1 An arrangement $A(L)$ is the subdivision of a plane by a finite set L of lines. ■

The lines partition the plane in disjoint convex regions which has no lines passing through them. These regions are defined as faces.

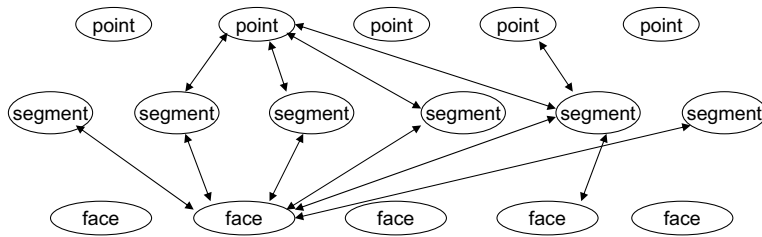
The arrangement comprises of the of the inclusions and incidences between the points, line segments, and faces defined by the n lines. Consider a set of typical questions:

- Given an edge can we find the adjacent faces ?
- Given a face can we find the edges bounding it ? Can we “walk” along the boundary of the face?
- Given a point can we find the faces adjacent to it (possibly through the edges adjacent to the point)?

A relatively simple data structure which captures all the incidences and inclusions and can answer all the questions is the following (Assume no line is vertical.)

1. Corresponding to each point we maintain in order the segments of the lines passing through it. Assume that we have the angular order of the $2n$ line segments exiting the bounding box. This order would correspond to the order in which these segments would be incident on the hypothetical point at infinity.
2. A line segment needs to store the endpoints defining it. The unbounded edges have special symbols marking the unbounded endpoint.
3. For each line segment maintain the previous and the next segment along the line. Once again for an unbounded segment (in direction of increasing x coordinate) the next segment is the unbounded segment in the other direction (in a wrap around fashion).
4. For each line segment maintain the faces above and below it.
5. For each face maintain the edges defining it in (say) order in a doubly linked list. Once again for unbounded edges are assumed to meet far off and thus the order is defined for these faces as well.

The hypothetical chart to remember is:



From now on, we will focus primarily on *simple arrangements* where no more than two lines intersect at a point. It is easy to see that the number of points is $O(n^2)$. Since each point is adjacent to 4 segments, the number of line segments is also $O(n^2)$. However to count the number of faces we have to be a bit more careful. By induction we can prove that the number of line segments is n^2 , and the number of points is $\binom{n}{2}$. Using Euler's formula for the plane (namely that $v - e + f = 1$, where v is the number of points, e is the number of edges, and f is the number of faces in a subdivision), we get the number of faces to be $\binom{n^2-n}{2+1}$ which is also $O(n^2)$.

However the interesting fact is that the total size of the data structure we are constructing is $O(n^2)$. This is because a point or a line segment stores $O(1)$ information. There can be faces which are bounded by non-constant number of edges. But if we sum over all faces the number of edges bounding them, we will effectively sum over all edges twice. Thus the sum of the linked lists associated with the faces is also $O(n^2)$.

To Construct the Arrangement Constructing the inclusion of vertices and segments is relatively easy (brute force – fix a line, intersect others and so on). But there does not seem to be any trivial way to construct the faces. This where the contribution of algorithms for constructing the arrangement come in.

2.1 Plane Sweep

Consider a plane $x = c$ as the value of the parameter c changes. As c increases the plane performs a “sweep” from the left to right. The idea in this section will be to construct the arrangement of n lines

using their hypothetical intersection with this line. Notice that the assumption that none of the lines defining the arrangement being vertical makes the cases easy to handle. Consider the case of $c = -\infty$ the line is

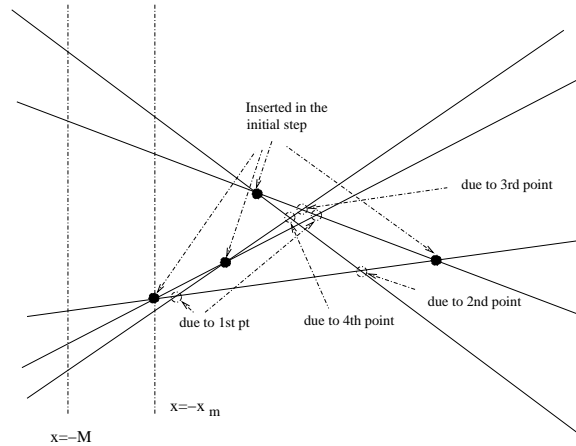


Figure 1: The plane sweep algorithm

$x = -\infty$. The order in which the given set of lines L intersect this line is determined by the slope of the lines in L . We can find this in time $O(n \log n)$. Consider that $c = -M$ for sufficiently large M that there are no intersections points with x -cord $\in (-\infty, -M]$. The order of the lines in L intersecting the line $x = -M$ is the same as the order in which the lines intersect $x = -\infty$.

Notice that the $n + 1$ segments of the line $x = -M$ also belong to $n + 1$ unbounded faces of the arrangement. These faces are unbounded because if we extend the lines towards $-\infty$ none of them will meet. We hypothetically call the faces f_1, f_2, \dots, f_{n+1} in a top to bottom order. Suppose the lines are named $\ell_1, \ell_2, \dots, \ell_n$ – once again in top to bottom order. Thus we know that ℓ_1 is between f_1 and f_2 etc.

As c increases, we will come to the leftmost point of intersection defined by the lines in L . This point of intersection must be defined by adjacent lines ℓ_i, ℓ_{i+1} . Instead of just finding the point with minimum x -coordinate, we will compute all points of intersection of ℓ_i, ℓ_{i+1} and insert them in a priority queue where the priority is defined by a smaller x -coordinate. An inserted point will have pointers to the lines that intersected to create it. The queue is typically termed as the “event queue” – the idea is that the queue contains information about the next interesting event (in our case crossing of lines) in a sweep line algorithm.

From the queue we will extract the point of intersection p_m with the smallest x -coordinate say x_m . Suppose it was created by ℓ_i, ℓ_{i+1} . This means

- Face f_{i+1} has come to an end. This is because the lines defining the intersection are the lines above and below f_{i+1} . The unbounded line segments adjacent to it has p_m has one endpoint and $-\infty$ as the other.
- We switch the order of ℓ_i, ℓ_{i+1} on the sweepline.
- We will be starting a new face as we increase c , so a new face is introduced, say f' , and the order of lines and faces intersecting the line $x = x_m$ is $f_1, \ell_1, \dots, f_i, \ell_{i+1}, f', \ell_i, f_{i+2}, \dots$. Now the next point of intersection we will defined will be defined by the adjacent lines from the order . But we have already inserted most of the possible intersection points with the exception of those defined by

ℓ_{i-1}, ℓ_{i+1} and ℓ_i, ℓ_{i+2} . Thus we compute the intersection points and we insert the points in the priority queue.

- Notice that the line below f_i is now ℓ_{i+1} and likewise the line above f_{i+2} is ℓ_i . Thus update the list of edges bounding f_i and f_{i+2} appropriately.

The plane sweep is shown in Figure 2.1. We repeat the above construction as long as the priority queue is not empty. At every intersection point a face is completed and a new face is opened¹.

The analysis of the algorithm is easy. There are $O(n^2)$ points (since any pair of lines create a unique point) and each insertion/extract-min takes $O(\log n)$ time. The rest of the operations on seeing a point is $O(1)$. Thus the total time is $O(n^2 \log n)$.

2.2 The Ray Shooting Query

We are given a set of lines L and their arrangement $A(L)$; suppose we are asked to find the order in which a query line ℓ_q intersects the lines in L .

This is (almost) the dual problem of Problem 1 that we started the discussion with. Suppose the initial set of points were p_1, p_2, \dots, p_n and their dual lines (which is the set L) be $p_1^*, p_2^*, \dots, p_n^*$. If the query point is p then the lines defined by the pairs of the points (p, p_i) correspond to the points along p^* . But in the dual transform we have $ax - b \rightarrow (a, b)$ which means that the x -coordinate of the points along p^* are the *slopes of the lines defined by (p, p_i)* .

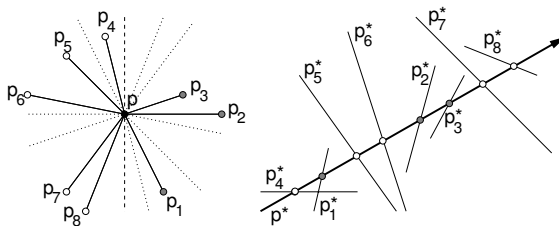


Figure 2: The angular sort and its dual

The angular order is not the slope order. But given the slope order we can create the angular order. Before we proceed let us ponder what we have so far: we have the slope order of all lines defined by the pair of points (p, p_i) . From this information we are interested in constructing the angular order of p_i around p .

The simplest way is to consider all points p_i with x -coordinate greater than p . The slope order amongst these points is exactly the same as the angular order. Thus for this subset we can construct the angular order. Likewise for the set of points with x -coordinates less than p we can construct the angular order. All that remains to be done is to joint the two (in $O(1)$ time) to create the final angular order. Thus from the slope order in $O(n)$ time we can construct the angular order. In what follows we will see how to construct the slope order in $O(n)$ time – by considering the dual problem.

Here is our algorithm:

- Given the line ℓ_q we can find its slope and using the slope ordering of the lines in L we can find the pair of lines ℓ_i, ℓ_{i+1} which have slopes smaller and larger than ℓ_q . Notice that these two lines must be adjacent as $x = -\infty$ and hence corresponds to consecutive lines in our initial ordering.

¹How would you prove this fact? How many faces are adjacent to a point ?

- Thus the line ℓ_q intersects face f_i . So starting from ℓ_{i+1} we start walking along the edges of f_{i+1} till we intersect ℓ_q . This gives us the first point of intersection. Let this edge be e ; at this edge the line ℓ_q leaves and enters the face on the other side of e . Since edges have pointers to faces, we know which face ℓ_q enters.
- We perform the same walk along the edges of this new face f' till we intersect ℓ_q again. This gives us the second point of intersection.
- We repeat the above till we are done. Figure 2.2 depicts the walk.

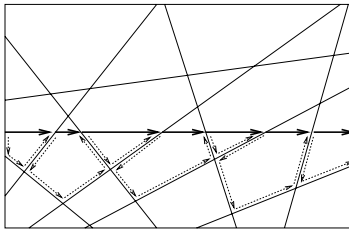


Figure 3: The walk along the arrangement

Running time: The algorithm is clearly correct – we can find all the points of intersection in order along ℓ_q . The running time of the algorithm is however

$$\sum_{f: \ell_q \text{ intersects } f} |f|$$

where $|f|$ denotes the number of edges bounding f . This brings us to an interesting theorem, the Zone Theorem. The Zone of a line is defined as the faces intersecting the line.

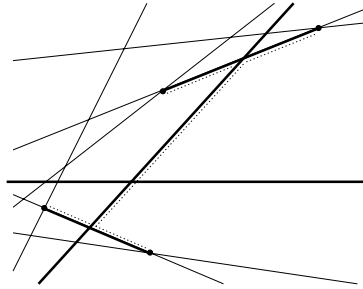
Theorem 2 $\sum_{f \in \text{Zone}(\ell_q)} |f| = O(n)$

Proof: Define an edge e to be a left bounding edge if there exists a face $f \in \text{Zone}(\ell_q)$ such that e bounds f and is on the left of f (or e is horizontal and below f). We will show that the total number of left bounding edges is $3n$ (where n is the number of lines). Likewise we will show that the number of right bounding edges (defined analogously) is also $3n$. Putting them together we get the statement of the theorem.

We will prove by induction on n . The base case is $n = 1$ and we have two faces. The single line is a left bounding edge to one of them and the claim is true.

Consider the case where we have n lines. Consider the *rightmost* point where ℓ_q intersects a line of the arrangement, say ℓ_1 . Now consider the arrangement *without the line* ℓ_1 . By induction the number of left bounding edges is at most $3(n - 1)$. Let us consider the number of left bounding edges added by the introduction of ℓ_1 .

The worst case is shown in Figure 2.2. The line ℓ_1 can cut at most two left bounding edges – above and below the line ℓ_q . When ℓ_1 intersects the left bounding edges it subdivides the edge and therefore creates extra edges. Thus ℓ_1 creates at most two extra left bounding edges by subdivision. ℓ_1 itself can also become a left bounding edge. Thus the maximum addition to the number of left bounding edges is 3. Hence proved. ■



Thus we have a $O(n)$ algorithm for Problem 1.

2.3 A Incremental Algorithm for Arrangement

Actually we can use the Zone Theorem to give a faster construction of arrangements. As we perform the walk along the Zone of ℓ_q , we can actually update the faces – as if ℓ_q were a line in the arrangement! By our previous analysis, this takes $O(i)$ time if we have i lines defining the arrangement.

Thus we have a simple algorithm. We insert one line at a time in our arrangement. Inserting line ℓ_i takes time $O(i)$. Summing over i we get an $O(n^2)$ algorithm for constructing the arrangement of n lines.

2.4 Points to Ponder

It is a curious point that we used a “query” algorithm to design a data structure. This is in line with construction of balanced binary search trees – that a query takes $O(\log n)$ time, and by repeated insertion we end up with a $O(n \log n)$ algorithm. The same is true for the point location algorithms that we will see.