COMPUTATIONAL GEOMETRY LECTURE NOTES

Scribes : H. Wang,H. Wong, edited by D.Lubinsky and D.Heller

May 16, 2004

In this lecture, we look at the transformation which maps points in the plane to lines: $(a, b) \mapsto y = ax + b$. We'll discuss it's properties and some problems to which it can be applied.

DUALITY

Consider the transformation T in R^2: $T : (a, b) \mapsto y = ax + b$

Given a point (a, b) in the plane, T maps it to a line $y = ax + b$.
Under this transformation,

1. $T : l : y = cx + d \mapsto (-c, d)$

   Consider any two points $P = (p, cp + d)$ and $Q = (q, cq + d)$ on line $l : y = cx + d$. Their image under transformation T are $l_P : y = px + (cp + d)$ and $l_Q : y = qx + (cq + d)$ respectively. Both l_P and l_Q pass through the point $(-c, d)$. So, T maps line $l : y = cx + d$ to a pencil of lines passing through point $(-c, d)$ and we take $(-c, d)$ as the image of l. Notice that the slope of the line is preserved as the negation of x-coordinate of the image point.

2. $T : x = m \mapsto \infty_m$
   $T : \infty_m \mapsto x = -m$

   For a vertical line $l : x = m, T$ maps it to a pencil of parallel lines of slope m. We assume that these parallel lines intersect at a point ∞_m at Infinity. This point is called an improper point. For every direction,

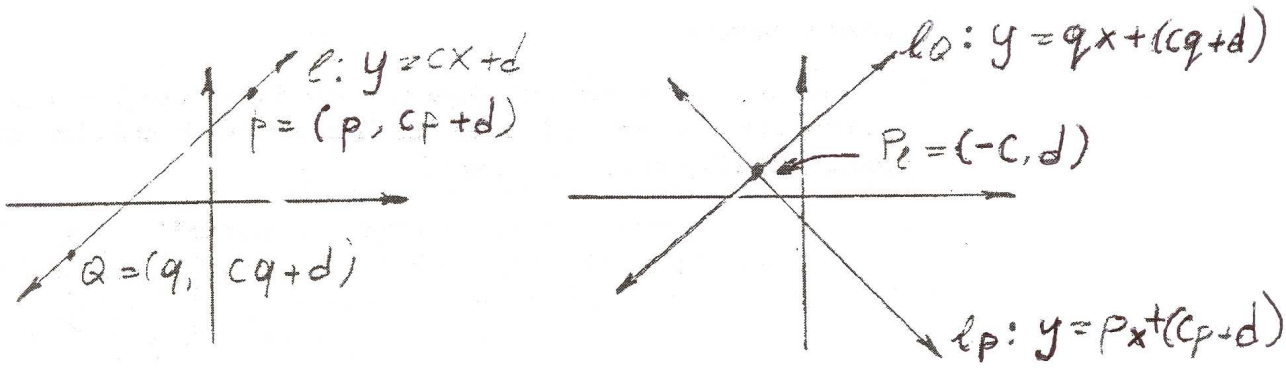


Figure 1: The Duality Transformation

let its slope be m, there is an improper point ∞_m associated with it. These improper points form a line called improper line.

Conversely, each line $y = mx+a$ which passes through ∞_m gets mapped to a point $(-m, a)$ which lies on the line $x = -m$. Consequently, the image of the improper point ∞_m under T is the vertical line $x = -m$.

3. The image of a line segment $\overline{AB}$ is the double wedge defined by the image line l_A, l_B of the two endpoints. The double wedge may not contain a vertical line since a line segment does not pass through a point at infinity.

4. The image of a ray $\overrightarrow{AB}$ with slope m is the double wedge that has vertical line $x = -m$ and image line l_A as its two boundaries and contains the line image l_B of point B.

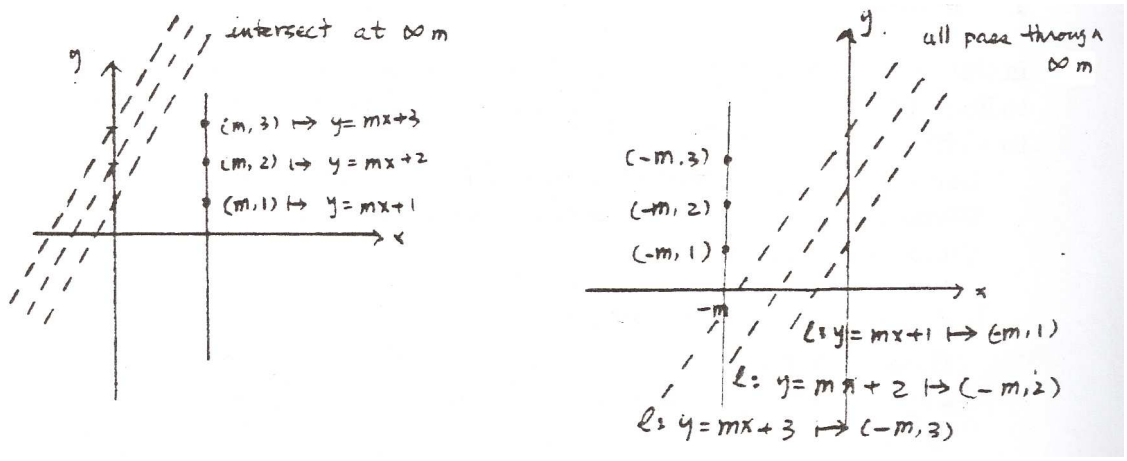


Figure 2: improper points

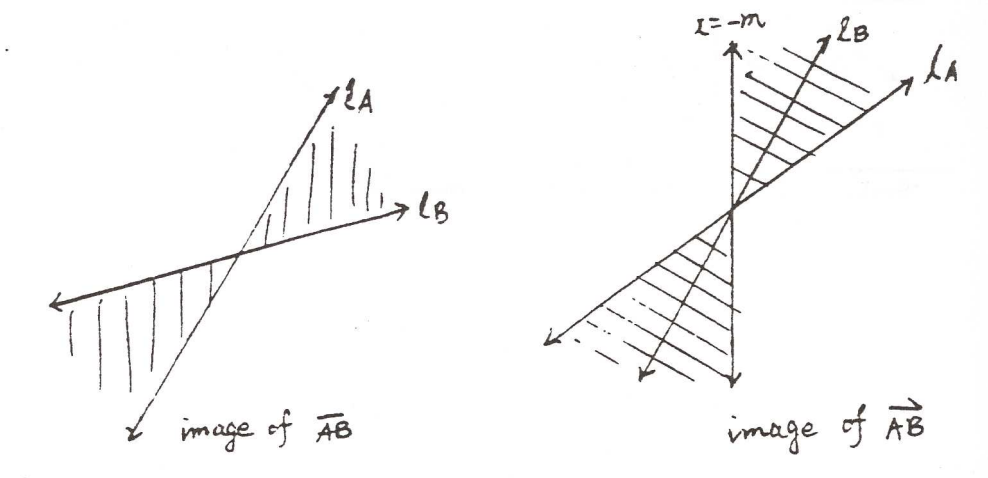


Figure 3: Double Wedges

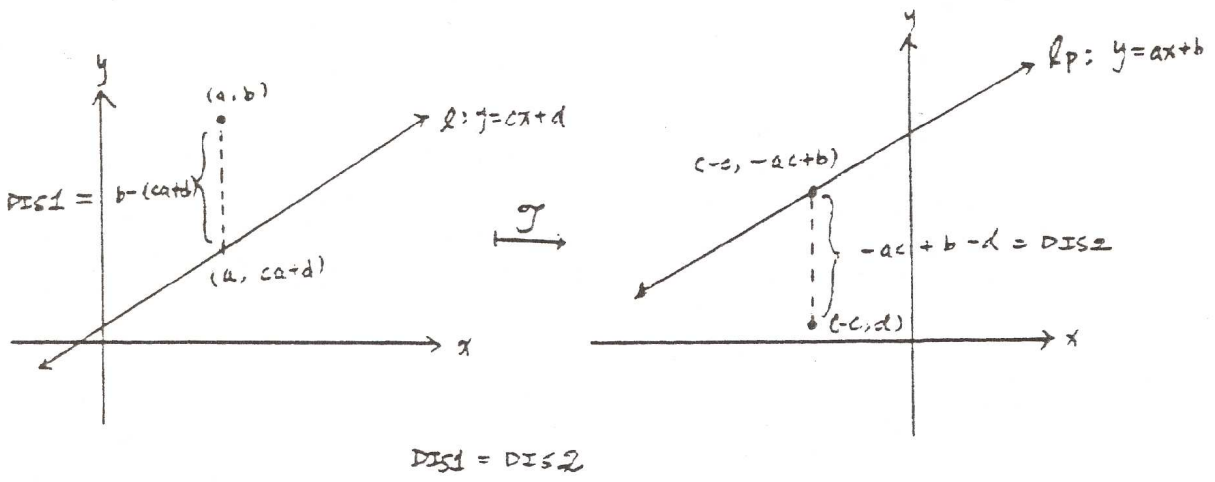


Figure 4: Vertical Distance is Preserved

From now on, we'll use the following notation:

- l_P denotes the line image of the point P under T.
- P_l denotes the point image of line l.
- (l_A, l_B) denotes the double wedge to which segment $\overline{AB}$ gets mapped.
- $(x = -m, l_A)$ denotes the double wedge image of the ray with slope m and starts at point A.

Lemma 1 *The transformation T preserves slope, vertical distance, and the above/below relation.*

Proof: For a line $l : y = ax + b$, its image under T is a point $(-a, b)$. It is clear that the slope a of l is preserved as the negation of x-coordinate of the image point.

The vertical distance of a point $P = (a, b)$ and a line $l : y = cx + d$ equals to $b - (ca + d)$. In the dual world, image of P becomes $l_P : y = ax + b$, and line l becomes point $(-c, d)$. The vertical distance between the two is $(-ac + b) - d$. The two distances are equal. So, the vertical distance and the above/below relation under T are preserved.

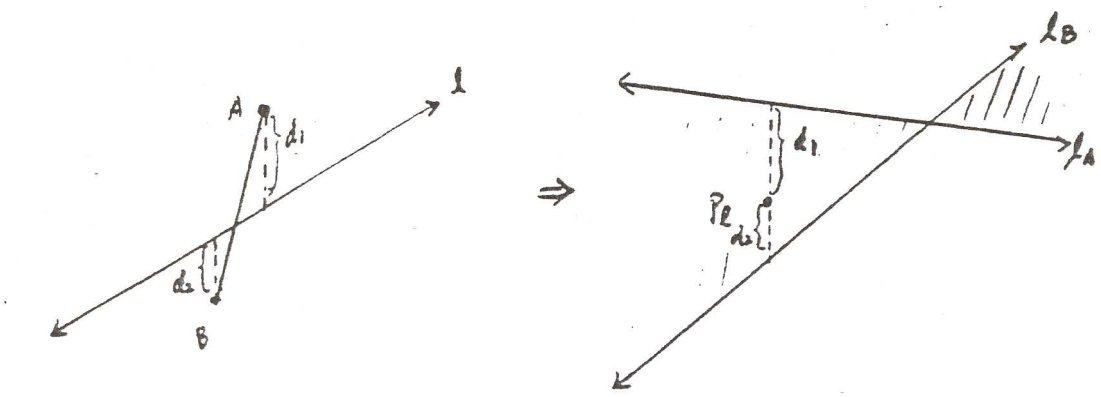


Figure 5: Intersection of a line and a segment

Application 1: Intersections of Lines, Segments, and Rays

1. Intersection of a line and a segment:
   Consider the problem of checking whether a line l intersects with a segment $\overline{AB}$. If $\overline{AB}$ intersects with l, then it must be that A lies above l and B lies below l, or vise versa. Since T preserves the above/below relation between the two objects, The point image P_l of line l must lies between the line images l_A and l_B of endpoints A and B. So, l intersects $\overline{AB}$ if and only if P_l lies in the double wedge (l_A, l_B).

2. Intersection of two line segments:
   Two line segments $\overline{AB}$ and $\overline{CD}$ intersect if and only if $\overline{AB}$ intersect with $\overline{CD}$ and $\overline{AB}$ intersect with $\overline{CD}$. This is equivalent to say: $\overline{AB}$ intersects $\overline{CD}$ if and only if the intersection of l_C and l_D lies within the double wedge (l_A, l_B) and the intersection of l_A and l_B lies within the double wedge (l_C, l_D).

3. Checking intersection between a ray and a line, a ray and a line segment, and two rays can be solved similarly.

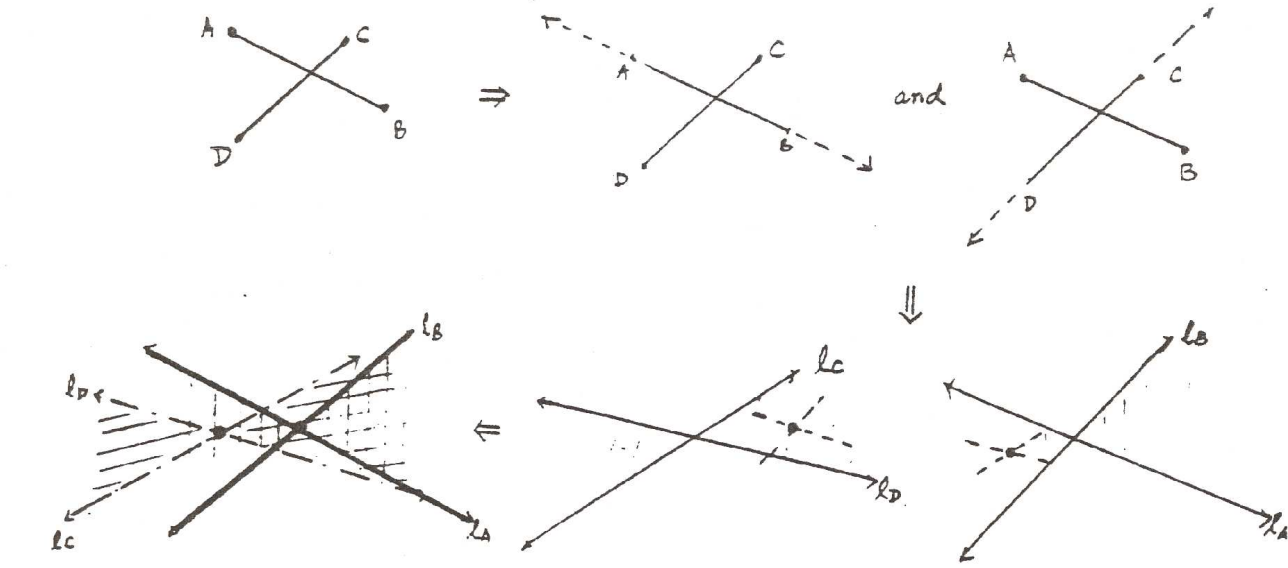
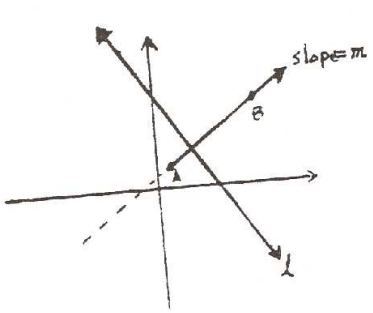
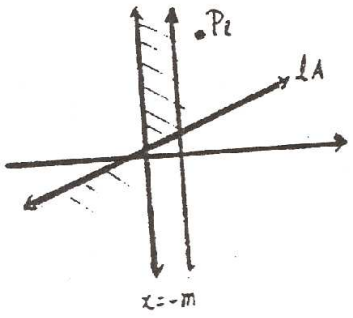
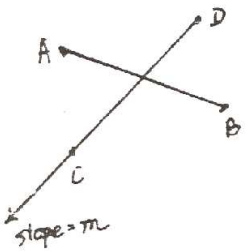


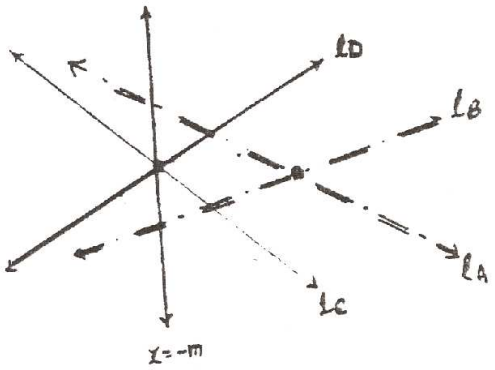
Figure 6: Intersection of two line segments



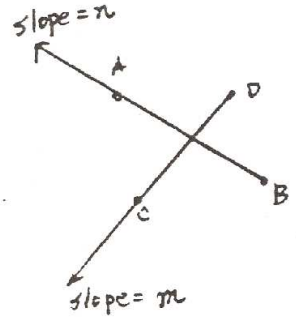


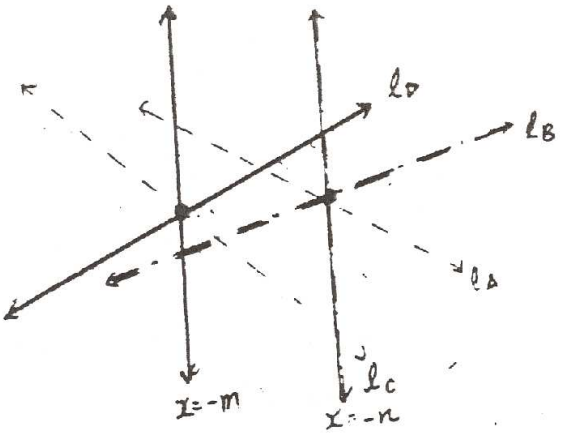
① Intersection of a ray and a line.





② intersection of a ray and a line segment.





③ intersection of two rays.

Application 2: Intersection of Lower Half-planes

Given: n half-planes $H_i = \{(x.y)|y < a_i x + b_i\}$
Find : their intersection

Not all of the boundary lines of these half-planes are on the boundary of the intersection. So, we must first remove those half-planes whose boundary line is redundant. Then, sort the remaining half-planes by the slopes of their boundary lines. Then, we can get the boundary of their intersection by computing the intersection points of consecutive boundary lines in the sorted list.

To remove the redundant lines, we first notice that a boundary line $l_k : y = a_k x + b_k$ is redundant if and only if there are two boundary lines $l_i : y = a_i x + b_i$ and $l_j : y = a_j x + b_j$ such that $a_i < a_k < a_j$ and l_k lies above the intersection point of l_i and l_j.

Because of the property that T preserves slope and above/below relation, we can determine the redundancy of the boundary lines by considering the dual image points of all boundary lines: $B_H = \{P_{l_i}, 1 \leq i \leq n\}$. The redundancy condition becomes: a point P_{l_k} is redundant if and only if there are two points P_{l_i} and P_{l_k} such that the x-coordinate of P_{l_k} is in between that of P_{l_i} and P_{l_j} and P_{l_k} lies above $\overline{P_{l_i} P_{l_j}}$. The only points that do not lie above any segment are those points on the lower convex hull. Therefore, the redundant boundaries are those whose image under T is not on the lower hull of the image point set B_H.

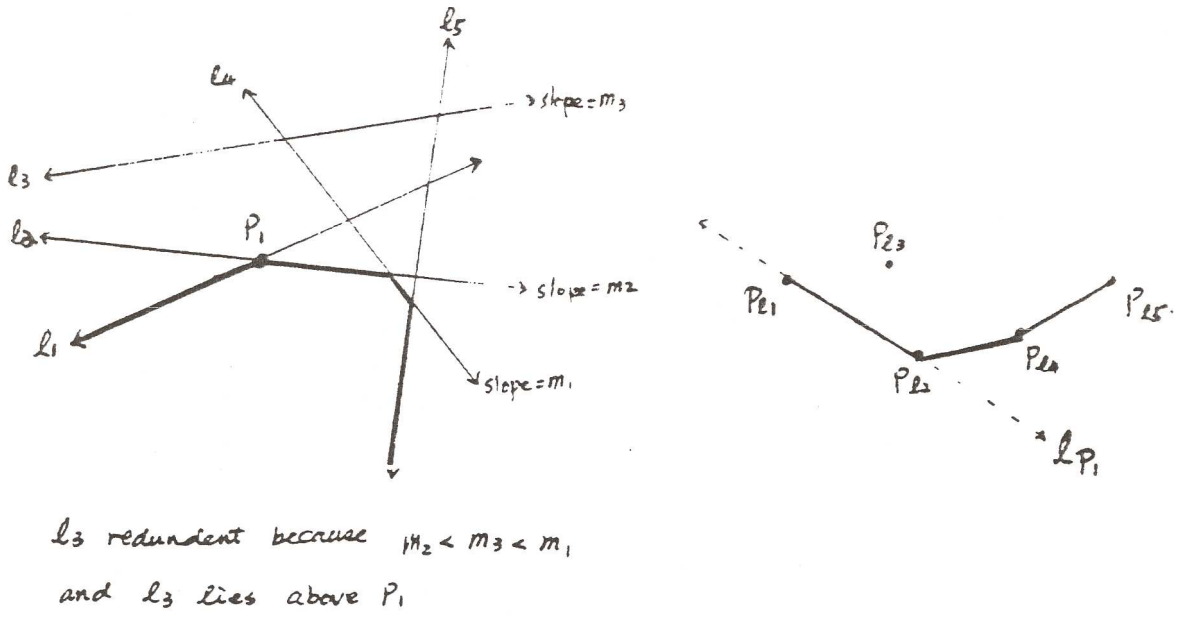


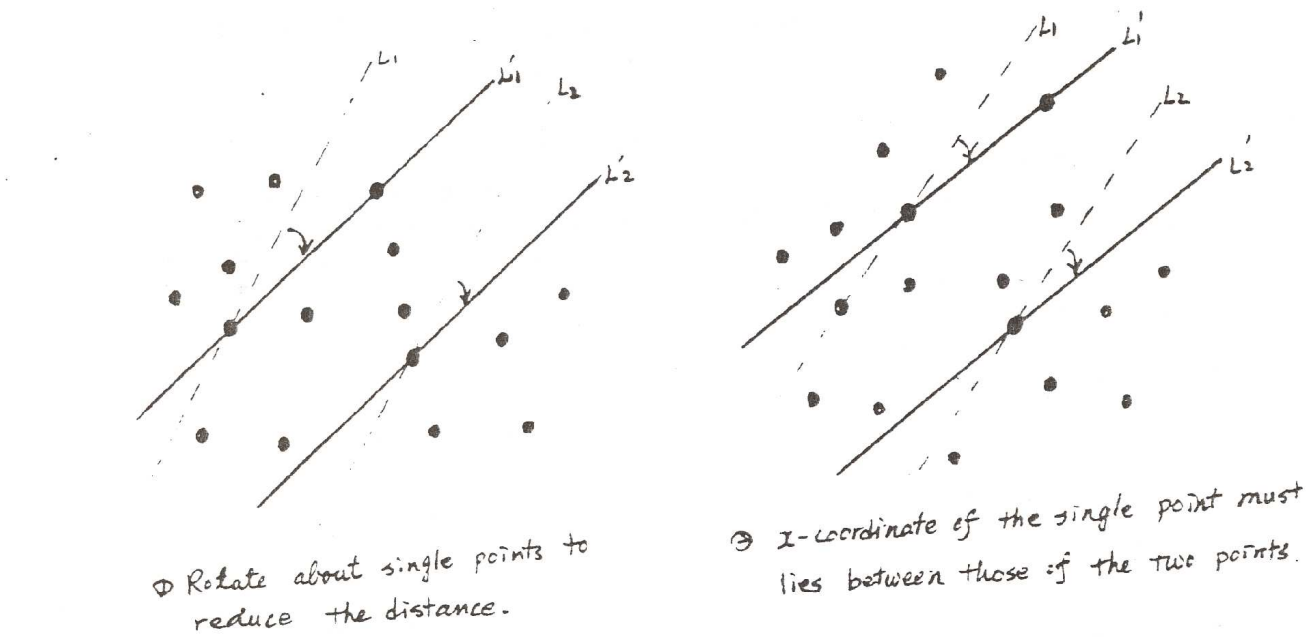
Figure 7: Intersection of Half-Planes

Application 3: Least Median of Squares Regression

Consider the following problem in statistics: find the "best" fit of a line to a set of data points. The prevailing method of doing this is the *least sum of squares* method, which minimizes the sum of the squares of the y-distance between the fitted line and the data points. (The y-distance is called the *residual* of the point.) The problem with this method is that the choice of line may be perturbed by a single very bad data point. We'd like to find a fitting line which is less sensitive to noise.

The least median of squares regression finds a line such that the *median* of the squares of the residuals is as small as possible. Another way of defining this problem is to find two parallel lines L_1 and L_2 at minimum vertical distance from each other, with half of the data points contained in the slab they define. The regression line shall be the median axis of the strip.

L_1 and L_2 must each contain at least one point, or they could be moved closer to each other (and were thus not at minimum distance). Furthermore,



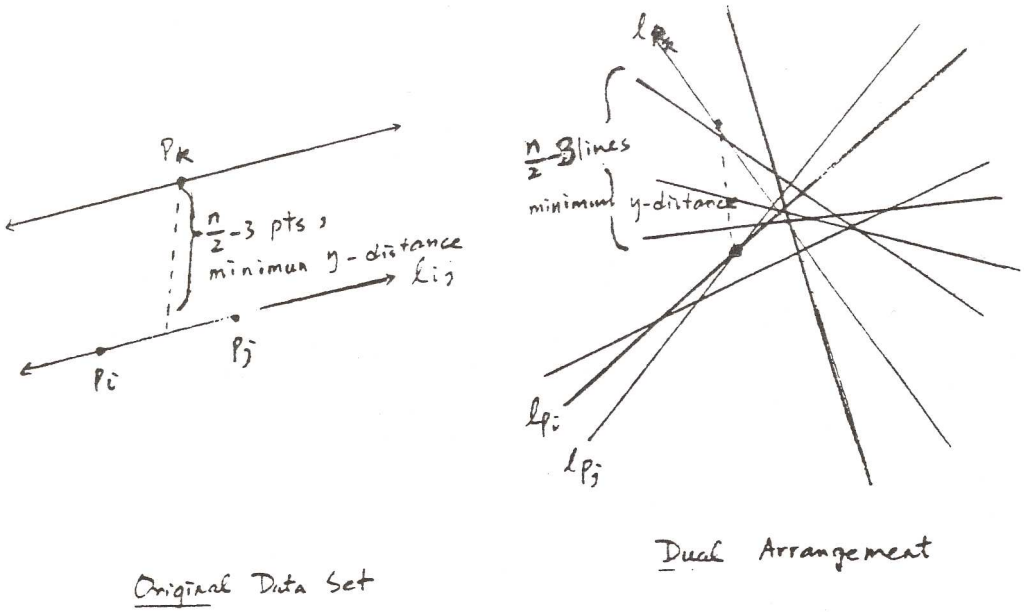
one of the pair of lines must contain *two* points, or the lines could be rotated about their single points to reduce the distance between them. If L_1 is the line containing the single point, the x-coordinate of the single point must lies between those of L_2's points; if this is not the case, the lines could be rotated closer together without losing any points between them.

Knowing this, we can construct a naive algorithm to find L_1 and L_2 by inspecting every triple of points. Each time we do this, we must determine whether there are $\lfloor \frac{n+1}{2} - 3 \rfloor$ remaining points lie in the slab defined by the two lines. The total time complexity is thus $\mathcal{O}(n^4)$.

A refinement of the naive algorithm is to determine, for each pair of points (p, q) a line l parallel to $\overline{[q}$ and passing through some other point r such that precisely $\lceil \frac{n+1}{2} \rceil$ points lie in the closed slab defined by $\overline{pq}$ and l. The time complexity for this is $\mathcal{O}(n^3)$.

If we map the set of points using T, the above problem would become: find the intersection point of two lines l_p, l_q such that the y-distance between this point and the line l_r, which is $\lfloor \frac{n+1}{2} - 3 \rfloor$ above or below, is the shortest among all such distances.

Initially, a "best distance" between lines is set to ∞. A vertical line is then swept through the arrangement formed by the lines dual to the set of



points, to find a line and an intersection point of two other lines such that the closed vertical segment from the point to the line intersects $\lfloor \frac{n+1}{2} \rfloor$ lines and is as short as possible.

Pointers are maintained between each line intersection point and the line which is $\lfloor \frac{n-3}{2} \rfloor$ lines above and below it. As we sweep over every intersection of the duality lines, we can determine in constant time, using these pointers, whether the distance to the line $\lfloor \frac{n-3}{2} \rfloor$ lines above is less than the best distance till now. If so, the new distance becomes the best distance. The complexity of this algorithm using T is $\mathcal{O}(n^2 \log n)$.

Conclusion

From the discussion above, we can conclude that the transformation T is very useful when dealing with problems between lines and points which can be determined by slope and above/below relation between them. If we consider the image of a triangle $\triangle ABC$, the corresponding image under T will become a non-convex and unbounded region which makes it more difficult to deal with. Therefore, T is a BAD transformation for problems involving triangles and polygons.

1 Topological sweep

Let H be a set of n lines in the plane. Assume that no three lines intersect at a point and that none of the lines is vertical. Each line intersects $n-1$ other lines and thus is divided into n edges. The regions, edges and vertices partition the plane into a subdivision known as arrangement. If we use a vertical sweep line, we need to sort n^2 intersection points. Whether it is possible to sort the n^2 intersection points determined by n lines in $o(n^2 \log n)$ is still an open problem. In topological sweep we compromise the straightness of the sweepline to acheive better time and space complexities than vertical line sweep. The idea of topological sweep is to use a curved line (*topological line*) with some special properties to simulate a vertical line. Using a topological line to sweep the arrangement, we need only $O(n^2)$ time and $O(n)$ space.

A topological line (cut) is a monotonic line in y-direction which intersects every other line exactly once. It is specified by a sequence of edges $(c_1, c_2, , , , , c_n)$, each contains an intersection point of the cut with a different line in the arrangement. Notice that a vertical sweep line runs from $-\infty$ to ∞ in the y-direction and intersects each line in the arrangement exactly once. A cut has the same properties by definition.

The sweep will be implemented by starting with leftmost cut which includes all semi-infinite edges and pushing it to the right till it becomes the rightmost cut, in a series of elementary steps.

An elementary step is performed when the topological line sweeps past a vertex of the arrangement. To keep the sweep line a topological line, we can only past a vertex which is the intersection point of 2 consecutive edges in the current cut. (Otherwise, it will intersect some line more than once.) Do we always have such a vertex during the process of sweeping? That is, will

the topological sweep get stuck?

Lemma 2 *There always exist two consecutive edges of the cut with a common right endpoint, unless we are considering the rightmost cut.*

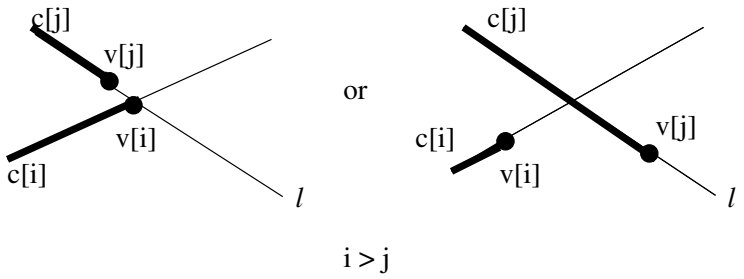


Figure 8: The right endpoint of an edge of the cut.

Proof: Assume that there are vertices still unprocessed but there is no pair of cut edges c_k, c_{k+1} which share a common right endpoint. Let c_i be the cut edge with the leftmost right endpoint. Let c_j be the edge of the cut on l which cuts c_i at its right endpoint v_i. c_js right endpoint (v_j) is either to the right or to the left of c_is right endpoint (v_i). See figure 8.

If v_j is to the right of v_i, then the topological line cuts l more than once. So, this can not be the case.

If v_j is to the left of v_i, then v_j is the leftmost right endpoint. Contradiction.

Therefore $v_i = v_j$. And the leftmost right endpoint is always an elementary step.

2 Data Structure

E[1:n] is the array of line equations:
$E[i] = (a_i, b_i)$ if the i^{th} line l_i of arrangement H is $y = a_i x + b_i$.

HTU[1:n] is an array representing the upper horizon tree.
HTU[i] is a pair (λ_i, φ_i) of indices indicating the lines that delimit the segment of l_i in upper horizon tree to the left and to the right respectively.

HTL[1:n] represents the lower horizon tree and is defined similarly.

I is a set of integers, represented as a stack. If i is in I, then c_i and c_{i+1} share a common right endpoint.

M[1:n] is an array holding the current sequence of indices that from the lines m_1, m_2, , , , , m_n of the cut.

N[1:n] is a list of pairs of indices indicating the lines delimiting each edge of the cut.

The *upper horizon tree* of a cut is constructed by starting with the cut-

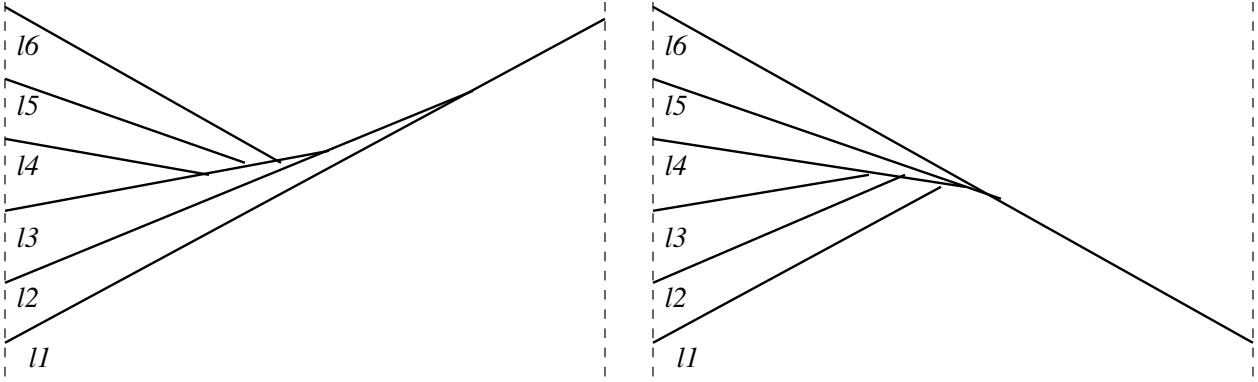


Figure 9: Upper and Lower Horizon Trees

edges and extending them to the right. When two edges come together at an intersection point, only the one of higher slope continues to the right. If the segment of l_i in HTU is the leftmost on l_i, then $\lambda_i = -1$, if it is the rightmost then $\rho_i = 0$. See figure 9.

The *Lower horizon tree* is constructed similarly. The difference is that when two edges intersect, only the one of lower slope continues to the right.

Given HTU and HTL, the right endpoint of the edge on l_i is identified by the closer of HTU[i] and HTL[i].

3 The algorithm

Initialization:

1. Sort the lines of the arrangement by slope.
2. Find the leftmost and the rightmost intersection point of the lines. Let the two points be (x_l, y_l) and (x_r, y_r).
3. Create vertical lines $x = x_l - 1, x = x_r + 1$ as left boundary and right boundary. Determine the intersection points of lines $l_1, , , l_n$ with the boundaries.
4. Create upper horizon tree:
   Insert $l_1, , , l_n$ in order to make a "hammock":
   Assume $l_1, , , l_k$ have been inserted. These lines form an *upper bay* as shown in figure 10. To insert l_{k+1}, begin at its endpoint on the left boundary. Walk in counter clockwise order around the bay till we find the intersection point of l_{k+1} with an edge.
5. Create lower horizon tree similarly by starting the travers at endpoints on the right boundary.

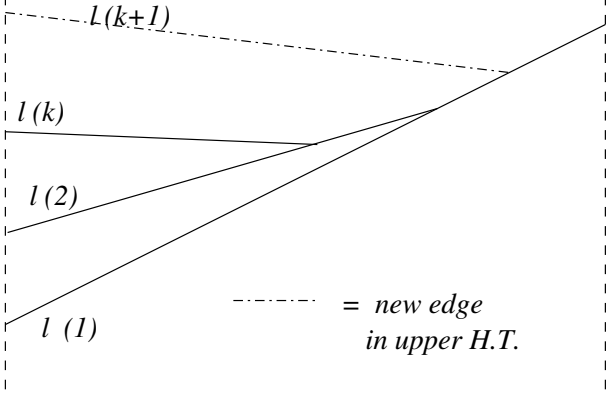


Figure 10: Creating a hammock.

6. Initialize N: Let HTU[i] = (-1, r) and HTL[i] = (-1, s). If l_r intersects l_i to the left of the intersection point of l_s and l_i then the right delimiting line of e_i is r. Otherwise, the right delimiting line of e_i is s.

7. Initialize I by scanning N.

Step 1 takes $O(n \log n)$ time. Step 2 could be done in linear time since the leftmost and rightmost intersection points must be the intersection points of two consective lines in the sorted order. Step 3,6,7 take linear time.

Now lets look at step 4. When inserting line l_i, each edge on the bay will be traversed once. There are $O(n)$ edges on the bay and there are n lines to be inserted. So we have $O(n^2)$. Could we find a tighter bound? Notice that our lines are inserted in decreasing order of slopes. Edges on the bay that fail to intersect with line l_i will not appear on the new bay formed after we insert line l_i. Each line can only fail once, and therefore be traversed once. So, HTU can be created in $O(n)$ time.

By similar argument, step 5 also takes $O(n)$. Hence, the initialization can be done in $O(n)$ after sorting of lines.

Elementary Step

After initialization, a series of elementary steps has to be executed. Notice that when passing a valid vertex (stored in I), only the two cut edges involved get changes, all other edges remain the same.

While I$\neq \Lambda$

1. Pop i from I

2. Swap M[i], M[$i+1$] /*lines are going to cross, after the elementary step*/

3. N[i].$\lambda \leftarrow$ N[$i+1$].ρ
   N[$i+1$].$\lambda \leftarrow$ N[i].ρ /*the point of elementary step becomes the left endpoint of the two new cut edges */

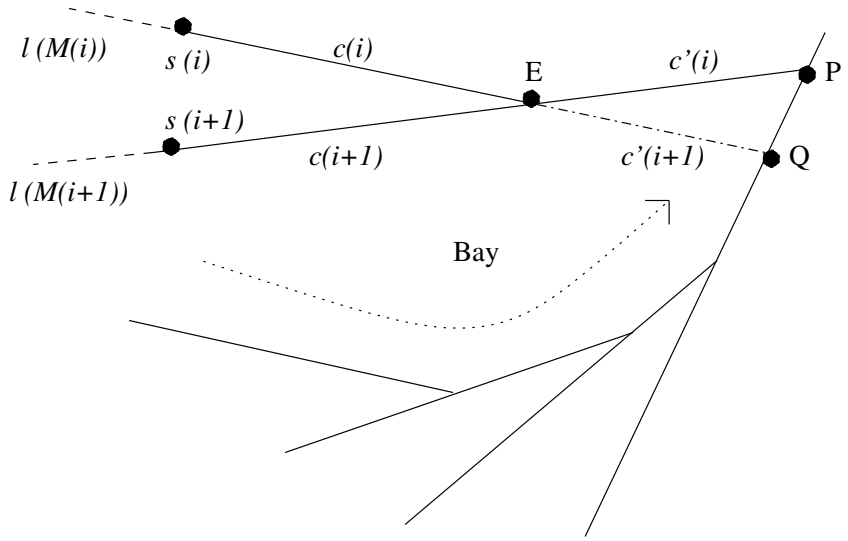


Figure 11: Updating the Horizon Tree

4. Update HTU, HTL.

5. N[i].$\rho \leftarrow$ closer HTU[M(i)].ρ,HTL[M(i)].ρ
   N[$i+1$].$\rho \leftarrow$ closer HTU[M($i+1$)].ρ,HTL[M($i+1$)].ρ
   /* find the new right endpoints */

6. If N[$i+1$].ρ = M[$i+2$] then push $i+1$ into I.
   If N[i].ρ = M[$i-1$] then push $i-1$ into I.
   /* push valid vertices formed after sweeping into I if there is any */

The steps 1,2,3,5,6 require just O(1) time. Lets take a look at step 5.

Update HTU

Assume that an elementary step is done after passing point E, which is the common right endpoint of edge c_i and c_{i+1}. Let s_i and s_{i+1} be the left endpoints of c_i and c_{i+1} respectively. At first, HTU[M[i]] contains the segment $\overline{s_iE}$ and HTU[M[$i+1$]] conatins the segment $\overline{s_{i+1}P}$. After sweeping over E, HTU[M[i]] should contain $\overline{EP}$ and HTU[M[i+1]] should contain $\overline{EQ}$. All other entries of HTU remain the same. Updating HTU[M[i]] takes only constant time since we already have E and P. To update HTU[M[$i+1$]], we need to traverse the bay till line $l_{M[i]}$ hits the bay. See figure 11.

To see the time complexity, consider the following charging system: For each edge traversed, we charge a unit cost to the node x corresponding to the elementary step. If some where later, an elementary step at node z makes the edge that charges x invisible from x, then we'll transfer the one unit charge to node z.

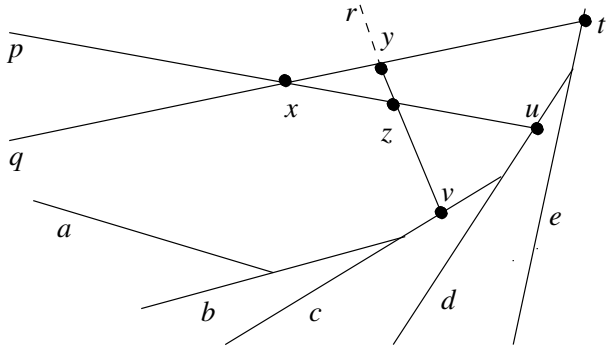


Figure 12: The elementary step at z creates edge zv, making x invisible from d.

For example, consider the arrangement in figure 12. First there is an elementary step at x, which modified the corresponding HTU entries from $\overline{qt}$ and $\overline{px}$ to $\overline{xt}$ and $\overline{xu}$ respectively. So, each of edge a, b, c and d charge one unit to x. Next elementary step occurs at point y, which changes the involving HTU entries from $\overline{xt}$ and $\overline{ry}$ to $\overline{yt}$ and $\overline{yz}$. Finally, the elementary step at point z changes HTU entries from $\overline{xu}$, $\overline{yz}$ to $\overline{zu}$ and $\overline{zv}$. The newly created edge $\overline{z}v$ makes edge d invisible from node x, the charge due to traversing d is transferred from node x to node z.

Lemma 3 *At the end of the algorithm each vertex is charged at most once for every edge on an incident region for the HTU (HTL) computation.*

Proof: A vertex v is charged only during the excecution of the elementary step at v. It gets one charge for every edge traversed, each of which is currently visible from it. It also gets one charge for each edge of the same region that is seperated by the current edge from its old vertex. If sometime later, an elementary step makes the edge invisible from v, then this edge will not be in the same region with v and the charge of the edge will be transferred to some other vertex. So, at the end of the algorithm, only the edges that is in the incident region of v will charge v and each of them can charge v at most once.

Lemma 4 $\sum_{R \in l} \mid R \mid = O(n)$.

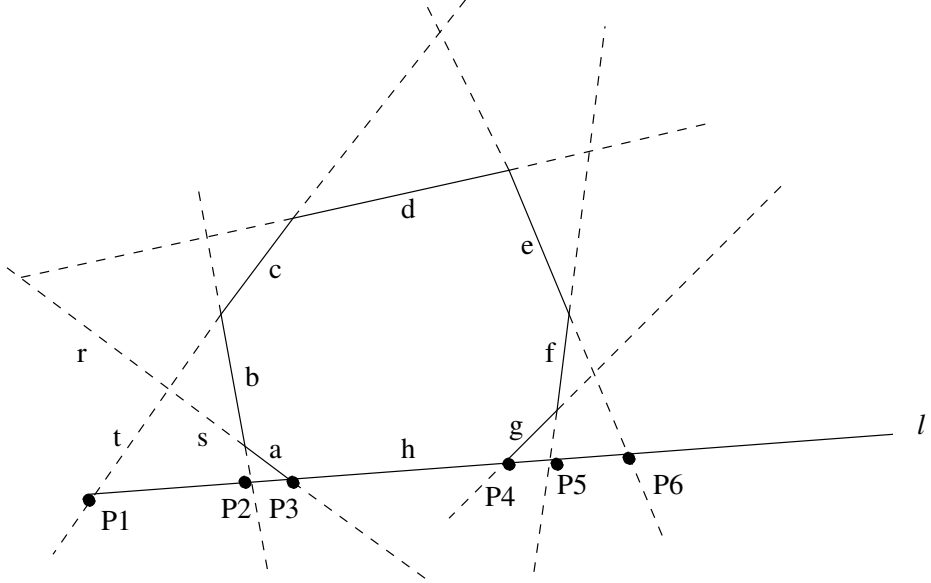


Figure 13: Edge Charges

Proof: The idea of this proof is to charge each edge of the faces that touch l to vertices on l and bound the number of charges of each vertex. Since the number of vertices on l is $O(n)$, we have $\sum_{R \in l} \mid R \mid = O(c * n) = O(n)$.

Let $l(e_i)$ denotes the line that contains edge e_i. Consider the following charging system:

For each face F adjacent to line l and above l
  For each edge e_i arranged in counter clockwise order except the ones touching l
    If $l(e_i)$ intersects l to the left of F
    then charge one cost to the intersection point of $l(e_{i-1})$ with l
    else charge one cost to the intersection point of $l(e_{i+1})$ with l

For example, in figure 13, P_1 gets 2 charges, one from edge d, one from edge r. P_2 gets 2 charges from c and s. P_3 gets one charge from b, etc.

Similarly, for each face below l, we charge the vertices on l in the similar way. For each vertex of l, the number of charges it gets is less than or equal to 4 - 2 from edges of the face above l, and 2 from those below l. Therefore, there are $O(n)$ edges that get charges.

Now, let's see the edges that don't get any charge: For each vertex v on l, there are two edges that have v as its right or left endpoint and don't get any charge (for example, edges a and g in the figure). Also, the edges that lie on l do not get any charge either. There are $O(2n + n) = O(n)$ such edges

altogether.

So, $\sum_{R \in l} \mid R \mid = O(4n + 2n + n) = O(n)$.

Lemma 5 $\sum_{R \in H} \mid R \mid^2 = O(n^2)$.

Proof: $\sum_{l \in H} \sum_{R \in l} \mid R \mid = O(n^2)$ and
$O(n^2) = \sum_{R \in H} \mid R \mid^2$ from 4.

Theorem 1 *The total cost of updating HTU (or HTL) through all the elementary steps is $O(n^2)$.*

Proof: From lemma 3 we know that an edge e charges a vertex only if they are in the same region and may do so only once. Consider region R in the arrangement. Each of its vertices can get at most $\mid R \mid$ charges, and there are $\mid R \mid$ vertices in R. So, there are at most $\mid R \mid^2$ charges associate with R. Summing this over all regions we get $\sum_{R \in H} \mid R \mid^2 = O(n^2)$ by lemma 5.

Hence all the topological sweep can be carried out in O(n^2) time. As for the space requirement, all the data structures maintained (6 arrays) are of linear size. So, space requirement is $O(n)$.

Reference

Duality:

[1] Dobkin D.P. and Souvaine D.L. *Computational Geometry - a User's Guide* Chapter 2 'Advances in Robotics 1: Algorithmic and Geometric Aspects of Robotics' Schwartz J. and Yap C. editors. Lawrence Erlbaum Associates, Hillside, NJ,1987.

[2] Edelsbrunner H. and Souvaine D.L. *Computing Least¡ Median of Squares Regression Lines and Guided Topological Sweep* JASA March 1989.

[3] H. Edelsbrunner, L.J. Guibas, Topologically Sweeping an Arrangement.

[4] Class Notes of Computational Geometry, Spring 1989.

Topological Sweep:

1. H. Edelsbrunner and L.J. Guibas, Topologically Sweeping an Arrangement. *J. of Computer and System Sciences*, 38:165-194, 1989

2. D. Dobkin and D. L. Souvaine, "Computational Geometry - A User's Guide", Chapter 2 of *Algorithmic and Geometric Aspects of Robotics*, J.T.Schwartz and C.K.Yap.

3. Class Notes of Computational Geometry, Spring 1990.