# Basic Algorithms and Combinatorics in Computational Geometry*

*Leonidas Guibas*

Graphics Laboratory
Computer Science Department
Stanford University
Stanford, CA 94305
guibas@cs.stanford.edu

## 1 Introduction

Computational geometry is, in its broadest sense, the study of geometric problems from a computational point of view. At the core of the field is a set of techniques for the design and analysis of geometric algorithms. These algorithms often operate on, and are guided by, a set of data structures that are ubiquitous in geometric computing. These include *arrangements*, *Voronoi diagrams*, and *Delaunay triangulations*. It is the purpose of this paper to present a tutorial introduction to these basic geometric data structures.

The material for this paper is assembled from lectures that the author has given in his computational geometry courses at the Massachusetts Institute of Technology and at Stanford University over the past four years. The lectures were scribed by students and then revised by the author. A list of the students who contributed their notes appears at the end of the paper. Additional material on the data structures presented here can be found in the standard texts *Computational Geometry, An Introduction*, by F. P. Preparata and M. I. Shamos (Springer-Verlag, 1985), and *Algorithms in Combinatorial Geometry* by H. Edelsbrunner (Springer-Verlag, 1987), as well as in the additional references at the end of the paper.

Let us end this brief introduction by noting that the excitement of computational geometry is due to a combination of factors: deep connections with classical mathematics and theoretical computer science on the one hand, and many ties with applications on the other. Indeed, the origins of the discipline clearly lie in geometric questions that arose in areas such as computer graphics and solid modeling, computer-aided design, robotics, computer vision, etc. Not only have these more applied areas been a source of problems and inspiration for computational geometry but, conversely, several techniques from computational geometry have been found useful in practice as well. The level of mathematical

*This work by Leonidas Guibas has been supported by grants from Digital Equipment, Toshiba, and Mitsubishi Corporations.

sophistication needed for work in the field has risen sharply in the last four to five years. Nevertheless, many of the new algorithms are simple and practical to implement—it is only their analysis that requires advanced mathematical tools.

Because of the tutorial nature of this write-up, few references are given in the main body of the text. The papers on which this exposition is based are listed in the bibliography section at the end.

## 2 Arrangements of Lines in the Plane

### Computation Model

We need to define the model of computation we will use to analyze our algorithms. Our computing machine will be assumed to have an unbounded random-access memory (Although it's unbounded, we will frequently worry about how much space we are actually using.) that is separated, for convenience, into an Integer Bank $I$ and a Real Bank $R$. We can read or write any location by indexing into the memory, denoted $I[j]$ or $R[j]$, where $j$ is an integer. Note that indirection is allowed, as in $R[I[I[j]]]$. We will assume we can perform all standard arithmetic operations (addition, multiplication, modulo, etc.) in constant time. Operations on real numbers (and the real number memory) are infinite precision.

At this point, we may start worrying about this model, because the assumptions for arithmetic operations, infinite precision reals, and possibly huge integers allow all sorts of strange and completely impractical algorithms. (For example, you could encode a huge pre-computed table as a large integer or an infinite-precision real.) To address these concerns, we add two caveats to the model, which will prevent any of this weirdness: (1) integers cannot be too big, *i.e.* if the input is size $n$, there must be some polynomial $P(n)$ that bounds the legal values of integers, and (2) the operations we allow in constant time must be algebraic, so a square root is fine, for example, but an arctangent isn't.

While this model of computation is a theoretically reasonable one, there are still some practical problems. The basic problem is the infinite-precision numbers. On a real computer, the floating-point representation introduces round-off error, so the laws of real arithmetic don't quite hold for floating-point numbers. In many applications, this problem is insignificant. In computational geometry, however, we are blending combinatorics and numerics, so a small round-off error may throw an algorithm completely off. (For example, consider an algorithm that branches on the sign of some value. A small perturbation that changes a $10^{-9}$ to a $-10^{-9}$ may cause a completely different computation to occur.) If we try to use integers or rationals instead, we no longer have round-off error, but we lose the closure property of real numbers; the intersection of a circle and a line defined on integer coordinates, for example, can be not only non-integral, but non-rational as well. Dealing with these issues is currently an active area of research.

### Duality

We continue with a brief introduction to some geometric dualities between points and lines. We denote the dual of a point $P$ by $D(P)$ and the dual of a line $l$ by $D(l)$.

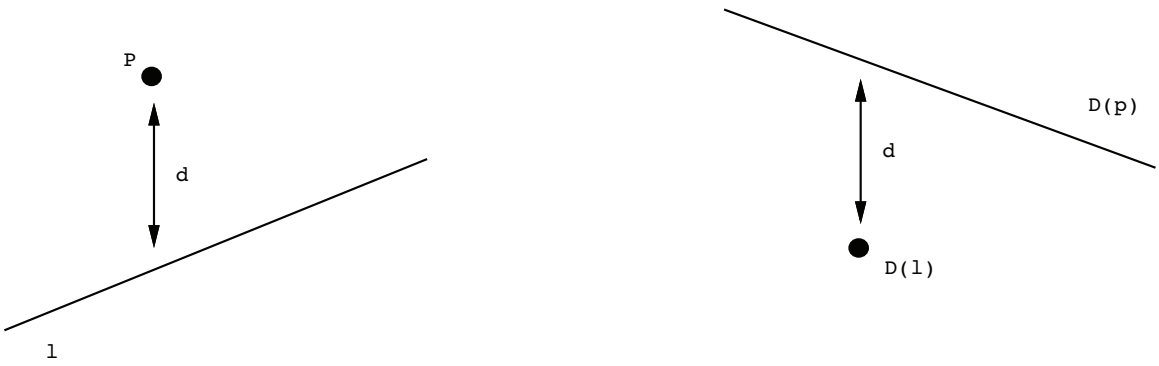


Figure 1: The duality $(a, b) \mapsto y = ax + b$ preserves aboveness.

The first duality we consider maps a point $(a, b)$ to the line $y = ax + b$; and the line $y = \lambda x + \mu$, to the point $(-\lambda, \mu)$. This duality maintains incidence between points and lines, *i.e.* point $P$ is on line $l$ if and only if point $D(l)$ is on line $D(P)$. Let's check this claim: Point $(a, b)$ is on line $y = \lambda x + \mu$ if and only if $b = \lambda a + \mu$. Thus, $b - \lambda a - \mu$ tells us how far the point is above the line. In the dual case, we check point $(-\lambda, \mu)$ against line $y = ax + b$ and find that the distance is $\mu + \lambda a - b$, which is exactly the negative of the distance in the primal. This duality preserves not only incidence, but the relationship (and distance) of aboveness. (See Figure 1.) Another nice property of this duality is that the $x$ coordinate of a point corresponds to the slope of the dual line. Unfortunately, this duality doesn't allow vertical lines.

Let's consider a second duality (which does handle vertical lines). Map a point $(a, b)$ to the line $ax + by + 1 = 0$, and a line $\lambda x + \mu y + 1 = 0$ to the point $(\lambda, \mu)$. This duality also has a nice geometric interpretation. The point $(a, b)$ maps to a line with $x$-intercept $-1/a$ and $y$-intercept $-1/b$. (See Figure 2.) Therefore, the line from the origin to $(a, b)$ is perpendicular to the dual line. Furthermore, since triangle $M$ is similar to triangle $N$, we have $\frac{m}{a} = \frac{1/a}{n}$, which shows that the distance from a point to the origin is the reciprocal of the distance from the dual line to the origin.

A little thought shows that, although the second duality can handle vertical lines, it doesn't handle lines through the origin, which the first duality could. A natural question is "Is there a duality that can handle any line?" The answer turns out to be no, unless you go to a projective plane with a line at infinity. However, there are plenty more dualities to choose from. (We direct any interested reader to the works of H.S.M. Coxeter.) Why should we bother with all these dualities? Many times we have problems dealing with a collection of points (or lines). Sometimes, working in the dual turns out to be much easier than the original problem. We'll see an example of this later.

## Arrangements

With the background material out of the way, we start on our first topic in computational geometry: line arrangements on the plane. Intuitively, the problem is, given $n$ straight lines on the plane, determine how they partition the plane and how the pieces fit together. (See Figure 3.) Formally, an arrangement consists of cells: 0-dimensional cells called "vertices," 1-dimensional cells called "edges," and 2-dimensional cells called "regions" or "faces."

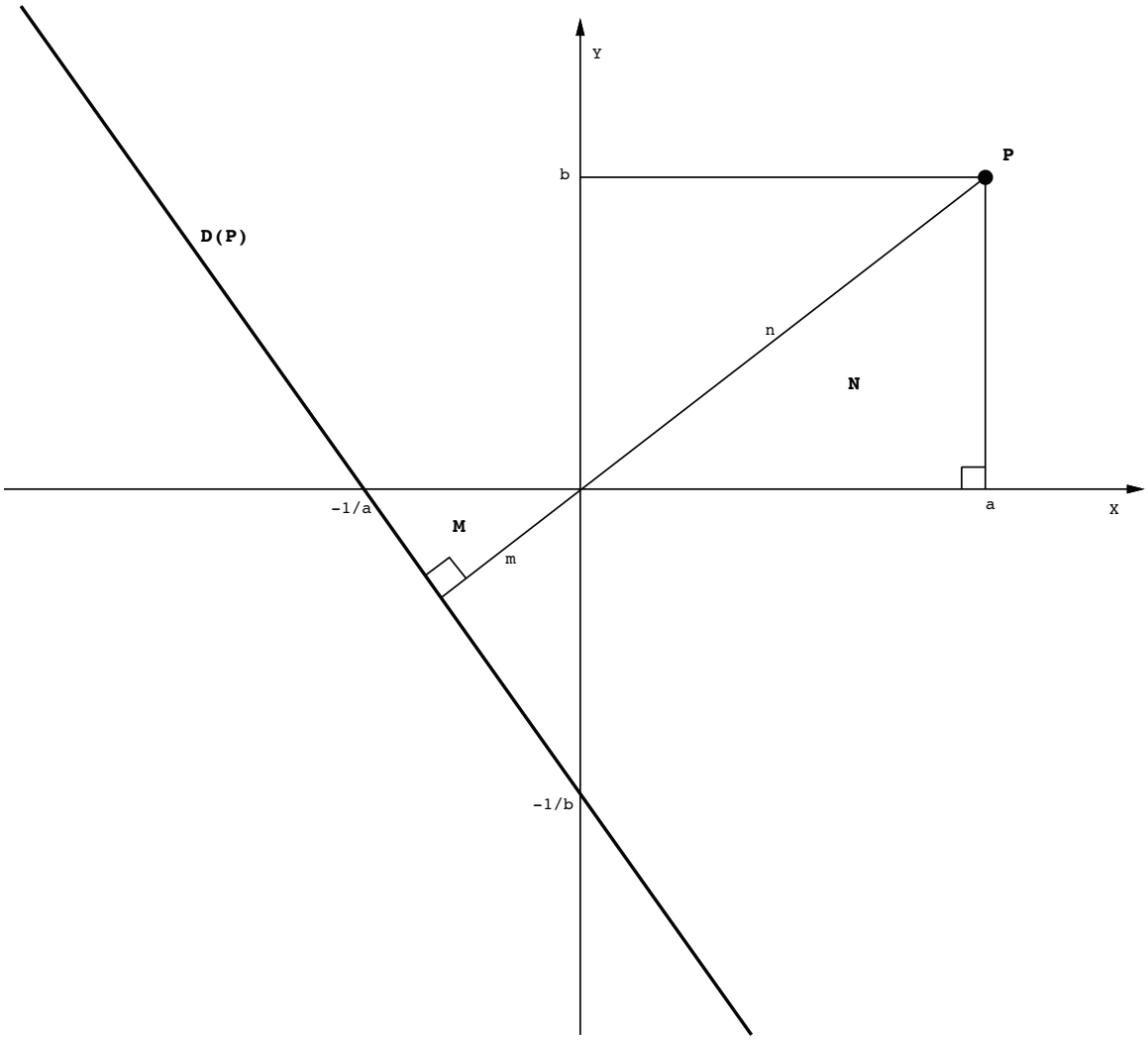


Figure 2: The duality $(a, b) \mapsto ax + by + 1 = 0$ inverts distance to the origin.

Many problems in computational geometry can be viewed as arrangement problems.

Suppose we have $n$ lines. How many vertices are there in the arrangement? Well, every two lines will intersect to produce a vertex, so the number is just $\binom{n}{2}$. At this point, the reader may object, "What if some lines are parallel or concurrent?" In our discussion, we will consider things to be in "general position." In this case, we assume that no 2 lines are parallel and no 3 lines are concurrent. In some other cases, we will also assume no vertical or horizontal lines. (In real life, we can perturb the input slightly to put things in general position.) General position yields combinatorially larger results than the specialized degenerate cases, so it makes sense to do our analysis this way.

How many edges are there? Each line gets chopped into $n$ edges by the other $n - 1$ lines, so there are $n^2$ total edges. How many faces? Each vertex is the bottom vertex of a face. This gives $\binom{n}{2}$ faces. In addition, there are $n + 1$ faces at the bottom of the arrangement that don't have bottom vertices. Therefore, the total number of faces is just $\binom{n}{2} + n + 1$.

Here is a problem to think about. In the above arrangement, vertices and edges are

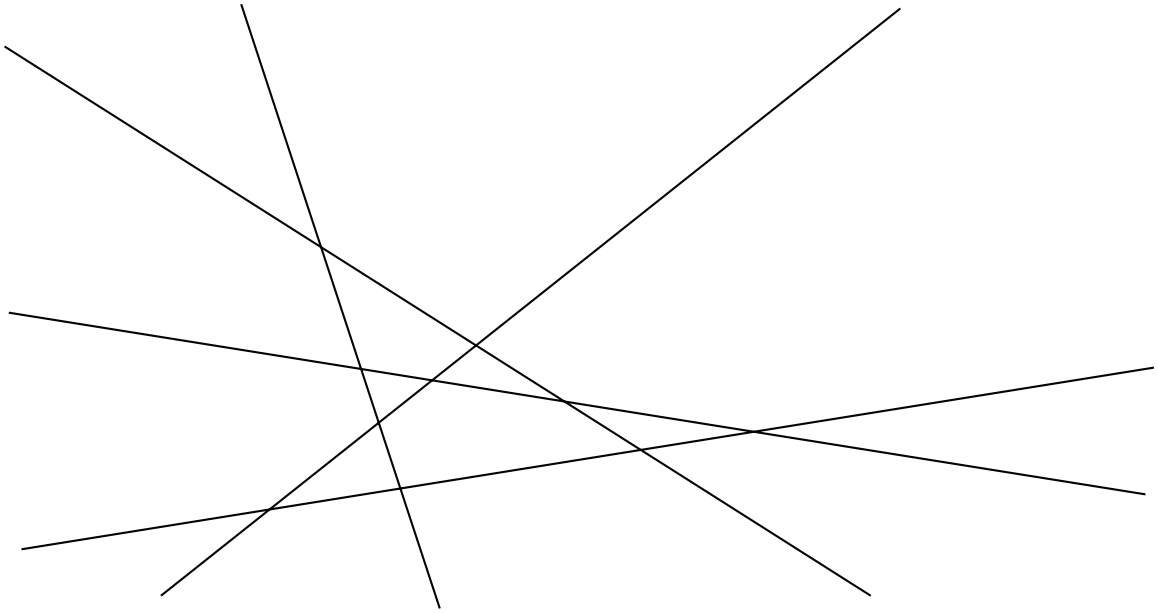

Figure 3: An Arrangement of Lines on the Plane

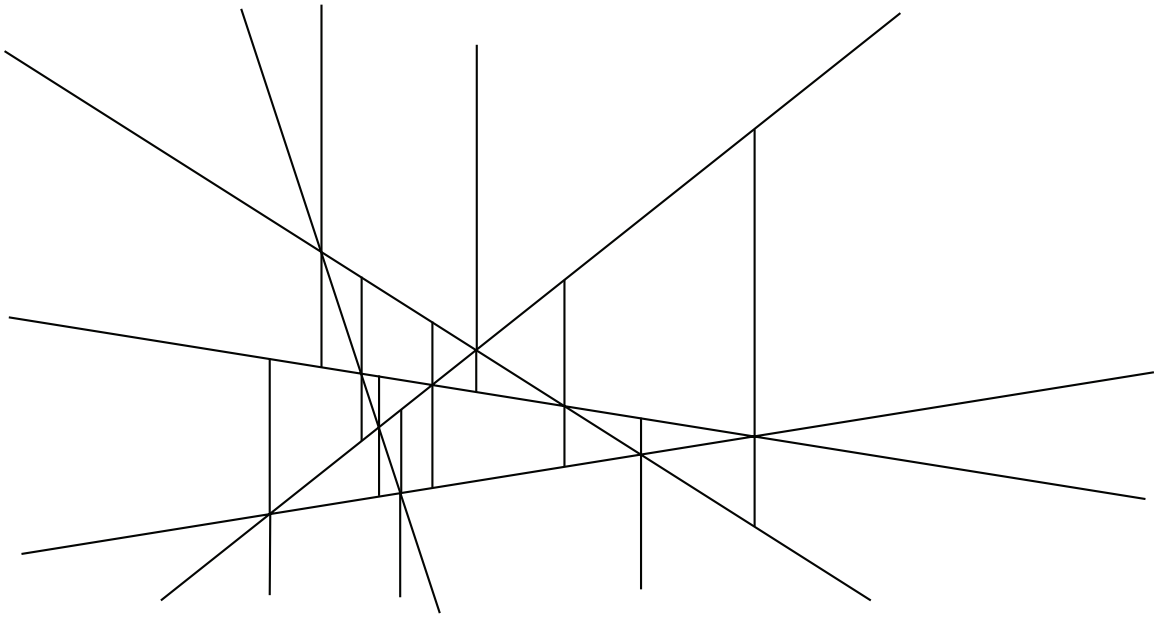

Figure 4: Extending "threads" from each vertex triangulates an arrangement.

nice, simple objects, but the faces can get messy. (You can keep track of a vertex as just a pair of coordinates and an edge as just a pair of vertices, but you may need up to $n$ edges to define a face.) In many applications, we want to have faces of bounded complexity. This modification is called a "triangulated arrangement." One way to produce a triangulated arrangement is to take a normal arrangement and extend a "thread" up and down from each vertex until it hits a line. (See Figure 4.) Now, each face is a trapezoid, which only needs 4 edges to describe. (One might be tempted to call what we just did "trapezoidalization" as opposed to "triangulation." In general, we'll use "triangulation" to refer to any

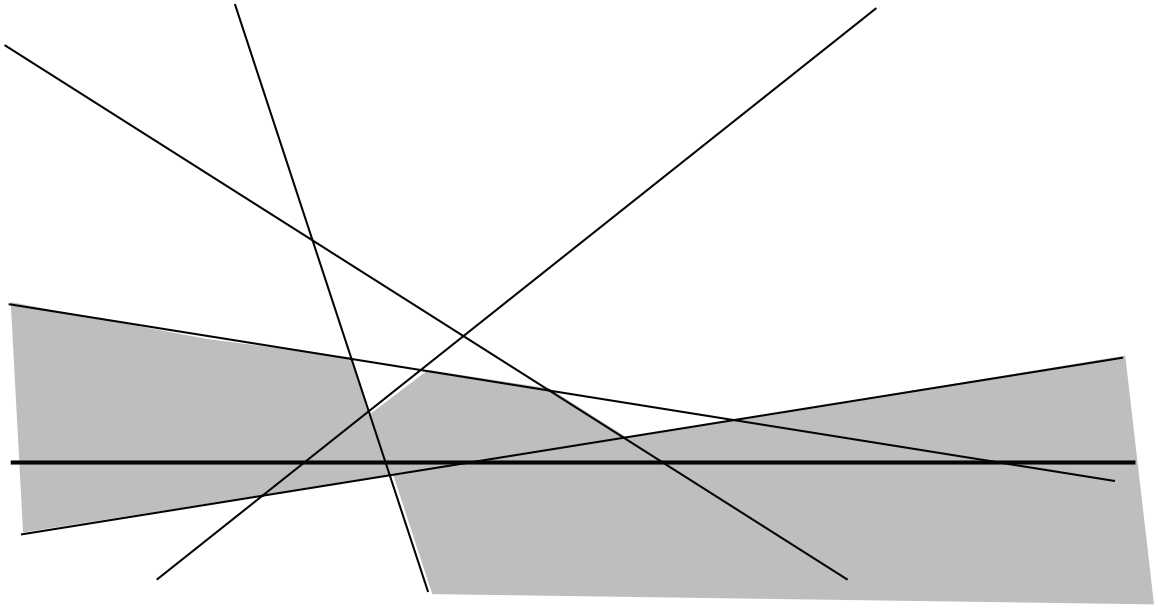

Figure 5: The zone of the darker line consists of the marked faces.

scheme to reduce the faces to finite complexity.) The question is, "How many trapezoids do you get?"

## More Counting

The correct solution is $3\binom{n}{2} + n + 1$. We arrive at this result by noting that each thread breaks an existing face into two and that each vertex produces two threads, giving $2\binom{n}{2} + \binom{n}{2} + n + 1$.

Returning to the original (non-triangulated) problem, we note that a face can have as few as 2 and as many as $n$ edges. How many edges does an *average* face have? Each edge counts toward both faces that it touches, so we have twice the number of edges divided by the number of faces or $2n^2/(\binom{n}{2} + n + 1)$, which is approximately 4.

Now, suppose we add a new line to the arrangement of $n$ lines. What faces of the original arrangement do we touch? These faces are called the "zone" or the "horizon" of the new line. (See Figure 5.) Let's do some more counting. How many faces are there in a zone? Since we can intersect each of the original lines exactly once, and each time we do this, we touch exactly one new face, there are $n + 1$ faces in the zone of a line.

A harder problem is counting the number of edges (or vertices) in a zone. We can get a quick upper bound by noting that each face can have at most $n$ edges, and there are $n + 1$ faces in the zone, yielding an $O(n^2)$ upper bound. Since the average number of edges per face is 4 and the number of faces is $n + 1$, we might wishfully hope to get a linear bound. This argument, however, isn't logically valid since the average we computed was over *all* faces, so a given line might just run into the bad ones. A careful argument, though, does give us a linear bound. This argument is the Zone Theorem.

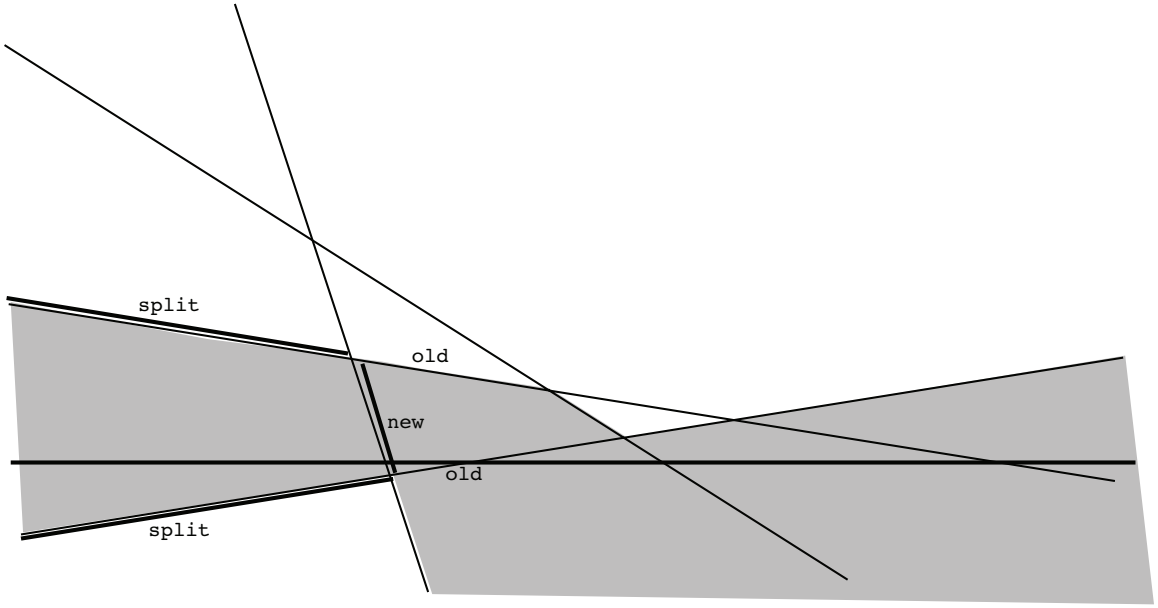


Figure 6: Introducing the dashed line creates the 3 additional marked edges.

## The Zone Theorem

We're actually only dealing with the Zone Theorem (also known as the Horizon Theorem) for lines on the plane. The Zone Theorem states:

**Theorem 1.** *The zone of a straight line intersecting an arrangement of $n$ straight lines in the plane has at most $6n$ edges. The same bound holds for the number of vertices as well.*

**Proof:** Without loss of generality, we can assume the new line is horizontal. (If not, we can always rotate our head slightly until it is.) Let's only count the right edges of the faces of the zone. (Remember we're in general position, so every edge of a face is either a right edge or a left edge.) If we can bound the number of right edges by $3n$, we can then stand on our heads and apply the same argument to get the left edges. Combining the two counts will give us the desired $6n$ bound.

Suppose we introduce the $n$ lines of the arrangement one-at-a-time, ordered from right to left by the intersection point of the line with the zone line. Each time we introduce an additional line, we can create at most 3 new right edges, since the new line introduces one new right edge and splits two previous ones. (See Figure 6.) Above and below the intersection points, the already-added lines shield the new line from having any further effect on the zone. Thus, we have the $3n$ bound on right edges. **QED**

(We mention that applying this argument even more carefully gives a bound of $6n - 2$, and that an even more precise argument gives a slightly lower bound, about $5.5n$.)

So, we have the Zone Theorem. Who cares? Well, we can use the Zone Theorem to create an efficient incremental method of computing arrangements. Suppose we have an arrangement (like Figure 3) and we want to add an additional line (like the dark one in Figure 5), we can compute the new arrangement by putting our left hand on the wall of

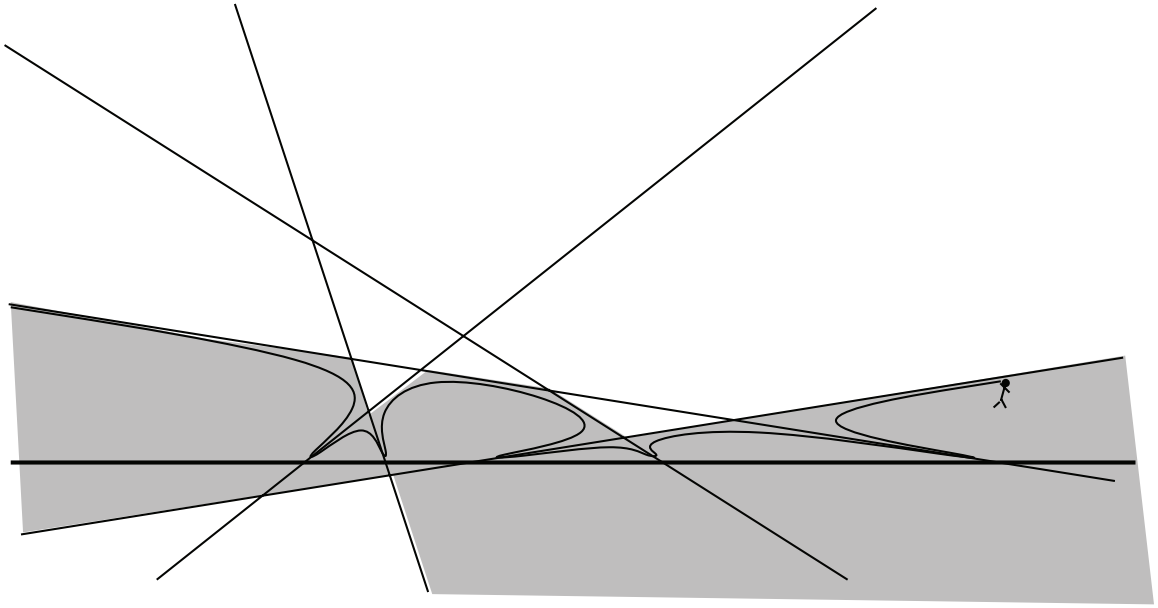

Figure 7: Walking around the zone follows the dashed line.

the zone and walking around the zone, marking off faces whenever we cross the new line. (See Figure 7.) (This is called a "labyrinth" walk, since you can get out of certain kinds of mazes this way.) By the Zone Theorem, the cost of this walk is linear. Thus, we can add a line to an arrangement in linear time, yielding an $O(n^2)$ incremental algorithm to construct an arrangement, which is optimal up to a constant factor. This example nicely illustrates an application of combinatorial geometry (the Zone Theorem) to computational geometry (the incremental method).

### Some Applications

Now let's look at some applications of arrangements. In vision, we are frequently given a bunch of points on the plane and asked if there are a bunch on a line. (In robotics, this same problem arises in three dimensions, because we'll be looking for solid surfaces based on sensor input. Also, collinear points screw up certain algorithms, so it's important to be able to find them.)

Obviously, we can look at every set of 3 points and check if they're collinear. This process requires time $\binom{n}{3} = O(n^3)$. The only known lower bound for this problem is $\Omega(n \log n)$. By taking the dual of the points and looking for three concurrent lines, we can solve this problem in time $O(n^2)$. This result is the best known for the last 15 years and is an open area for research.

Let's generalize this problem. Given a bunch of points, what is the smallest area triangle defined by three of them? We claim that again, by duality, we can get an $O(n^2)$ algorithm as opposed to the $\Theta(n^3)$ obvious one. Fix any two points. The smallest triangle is given by the closest point to the line determined by those two points. By similar triangles, we can use the distance in the vertical direction, rather than the true distance. Recall that our first duality preserves vertical distance. So, in the primal we take any two points,

consider the line they determine, and find the nearest point to that line. In the dual, we take any two lines, consider the point they determine (the intersection vertex), and find the nearest line to that point. But the problem in the dual is exactly the problem of finding the shortest "thread" in a triangulated arrangement! We can dualize the points in linear time. Constructing the arrangement takes an additional $O(n^2)$ time. So we need to be able to find the shortest thread in $O(n^2)$ time as well in order to achieve the desired $O(n^2)$ overall bound.

To find the shortest thread, one possibility is to merge the vertices of the upper and lower edges of each face, walking along both at the same time, to compute the length of all the threads through a face. Since the total number of edges is quadratic, this method clearly gives us the desired bound.

Another possibility is to walk around each face, computing the threads for each one as we go. At first glance, this method appears to be quartic, since there are $O(n^2)$ faces, each face has $O(n)$ edges, and each step of the walk will require checking the $O(n)$ other edges of the current face. A more careful analysis, however, shows that the cost is equal to

$$\begin{aligned}\sum_{(\text{faces } f)} \sum_{(\text{edges of } f)} \sum_{(\text{other edges of } f)} 1 &= \sum_{\text{lines } l} \sum \text{edges of faces incident on } l \\ &= \sum_{\text{lines } l} \text{Zone}(l) \\ &= O(n^2)\end{aligned}$$

by interchanging the order of summation and applying the trusty Zone Theorem.

Let us note that the $O(n^2)$ method we just described require $\Omega(n^2)$ space. Is there a more efficient methond space-wise? The answer is yes, by using a new paradigm called "Line Sweeping," in which we sweep a vertical line across the arrangement, only keeping track of the zone of this sweep line. This gives us an algorithm with $O(n^2 \log n)$ time and $O(n)$ space directly. Further refinements bring the bounds to $O(n^2)$ time and $O(n)$ space, as we will see.

# 3 Topological Sweeps for Computing Arrangements

### Topological vs. Linear Sweeps

There is a very simple algorithm for computing the arrangement of $n$ lines in the $x$-$y$-plane, based on the "line-sweeping" paradigm. For lack of space, we do not present the details here. Intuitively, the algorithm takes a "vertical" (perpendicular to the $x$-axis) line in the plane and "sweeps" it along the $x$-axis. As the sweep proceeds, changes in the order in which the different lines crossed the sweep line are noted, and thus the complete arrangement is determined. Unfortunately, the algorithm implicitly "sorts" the $n^2$ intersection points according to their $x$-coordinates, and consequently, its running time was $\mathcal{O}(n^2 \lg n)$. In addition, the algorithm makes use of balanced-tree data structures, and consequently is not easy to implement.

We now consider a variation of the sweep algorithm which replaces the sweep line with a "topological" wavefront. The algorithm has $\mathcal{O}(n^2)$ running time, yet still requires

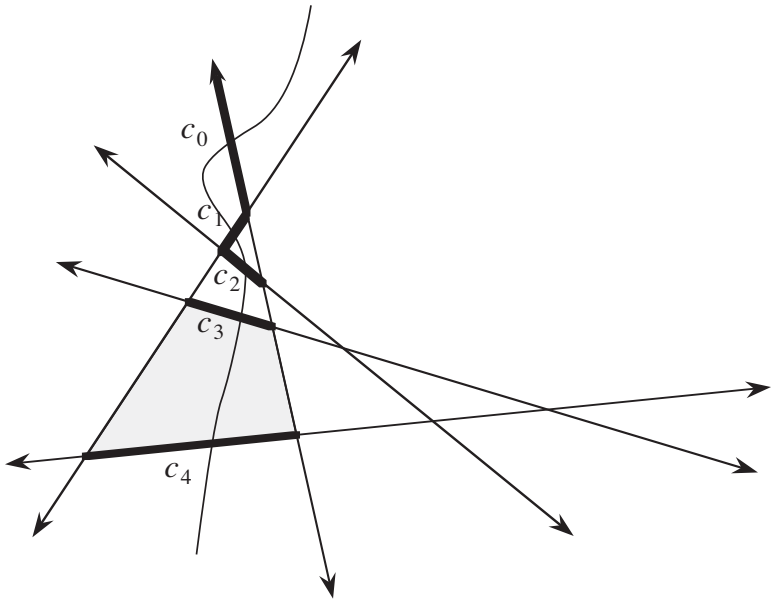


Figure 8: The cut $c_0, c_1, c_2, c_3, c_4$ defines the indicated topological line. The shaded region indicates the convex polygonal region defined by $c_3$ and $c_4$.

only $\mathcal{O}(n)$ working space. The algorithm is also easier to implement, since only array data structures are needed. Throughout the lecture, we assume that the set of $n$ lines is *nondegenerate*, i.e., no three lines cross at a single point, and no lines are vertical (perpendicular to the $x$-axis).

## Topological Lines

The definition of the desired wavefront makes use of the "above" ordering described in the previous lecture. Specifically, a *topological line* is defined by an ordered sequence of edges $c_0, c_1, \ldots, c_{n-1}$ such that $c_0$ borders the top (as defined by the "above" ordering) region, $c_{n-1}$ borders the bottom region, and each adjacent pair of edges $(c_i, c_{i+1})$ is such that $c_i$ and $c_{i+1}$ lie on a single convex polygonal region in the plane, and $c_i$ is before $c_{i+1}$ in the "above" ordering. Figure 8 shows a topological line for a simple set of 5 lines. The shaded region indicates the convex polygonal region shared by edges $c_3$ and $c_4$. Formally, the sequence of edged defining a topological line is called a *cut*.

Observe that each of the $n$ lines in the plane must have exactly one segment in any cut. If two segments of a single line were in a single cut, then it would be possible to show that a cycle exists in the "above" ordering. Consequently, a line in the plane can have at most a single edge in any cut, since it was shown previously that such a cycle cannot exist. In addition, each line must have at least one segment in every cut, since a cut essentially defines a path from the top region to the bottom region (crossing over each of the segments in the cut) and every line separates the top and bottom regions.

A cut $C$ is said to be to the *left* of another cut $C'$, if for every line $l$ in the plane, the segment of $l$ in $C$ is to the left of the segment of $l$ in $C'$. Using this ordering, it is possible

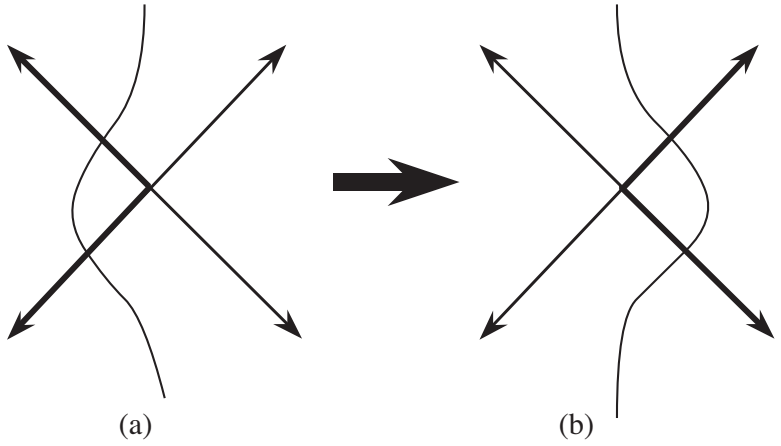


Figure 9: An elementary step removes two edges from a cut, and replaces them with two new edges, as shown.

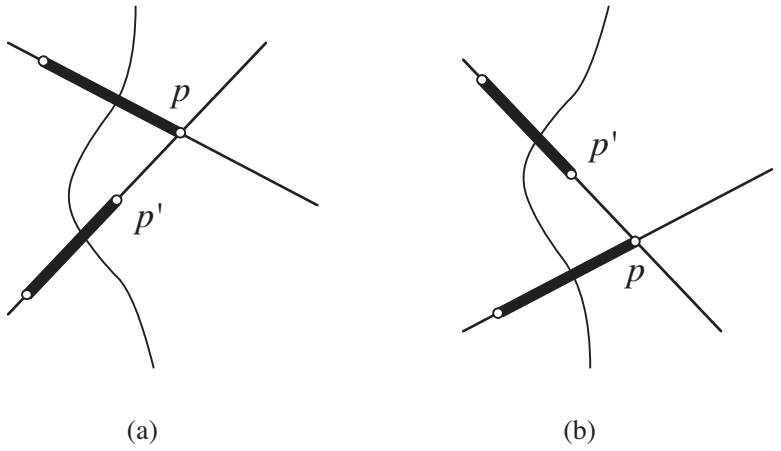


Figure 10:

to define a leftmost cut and a rightmost cut.

## Topological Sweeps

The topological sweep algorithm essentially computes a sequence of cuts $C_0, C_1, \ldots, C_k$, where $C_0$ is the leftmost cut, $C_k$ is the rightmost cut, and each adjacent pair of cuts $(C_i, C_{i+1})$ is such that $C_i$ is to the left of $C_{i+1}$. Each $C_{i+1}$ is computed from $C_i$ with a simple transformation. Specifically, if there exists in $C_i$ a pair of adjacent edges that share a right endpoint, as shown in Figure 9(a), then the cut $C_{i+1}$ is obtained by locally replacing the adjacent pair of edges in $C_i$ with the pair of edges indicated in Figure 9(b). This replacement is called an *elementary step*. Observe that applying an elementary step to a cut $C_i$ is guaranteed to result in a cut $C_{i+1}$ which is to the left of $C_i$.

Computing the sequence of cuts $C_0, C_1, \ldots, C_k$ can clearly be done, as long as it can be

shown that each cut $C_i$ in the sequence contains at least one point where an elementary step can be applied. Fortunately, it is possible to show that the leftmost right endpoint of all the edges in any cut $C_i$ must be such a point. Let the point $p$ be the leftmost right endpoint of the edges in cut $C_i$, and assume for the purposes of contradiction, that $p$ is the right endpoint of only a single edge $c$ in $C_i$, *i.e.*, an elementary step cannot be applied to $p$. The fact that $p$ is an endpoint implies that the line containing $c$ must cross some other line $l$ at point $p$. Now, consider the point $p'$ that is the right endpoint of the segment $c'$ of $l$ that appears in $C_i$ (we argued earlier that each line must have a segment in any cut). If the segment $c'$ appears after the segment $c$ in the cut $C_i$, then it possible to show that $p'$ must be the left of $p$, which is clearly a contradiction. In particular, the spatial relationship between $p$ and $p'$ must be similar to that shown in Figure 10(a), or else it can be easily argued that the "above" ordering contains a cycle, which previously was shown to be impossible. Similarly, if the segment $c'$ appears *before* the segment $c$ in the cut $C_i$, then the spatial relationship between $p$ and $p'$ must be similar to that shown in Figure 10(b), and once again, the contradictory conclusion that $p'$ is to the left of $p$ can be drawn. Thus, since $c'$ must either appear before or after $c$ in $C_i$, $p$ must be right endpoint of two segments in $C_i$, and consequently is a point where an elementary step can be applied. The preceding argument is not valid for the rightmost cut of the arrangement, but this is of no consequence, since the rightmost cut is always the last in the sequence $C_0, C_1, \ldots, C_k$.

Since it is always possible to find a pair of edges that share a right endpoint, and given the fact that each line in the plane must have exactly one segment in each cut, it becomes apparent that each cut essentially specifies a permutation of the lines in the plane (the leftmost cut corresponds to the permutation that lists all lines in inverse-slope order), while each elementary step corresponds to a transposition of two adjacent lines in a particular permutation. In the algorithm using the "vertical" sweep line, the order of the transpositions was determined by the $x$-coordinate ordering of all the intersections of the $n$ lines. By switching to a topological sweep line, it becomes possible to perform the transposition on any suitable pair of edges, while still obtaining the desired arrangement. By eliminating the need to compute the $x$-coordinate ordering, it is possible to reduce the amount of time needed to compute the arrangement.

### Horizon Trees

At this point, the overall structure of the topological sweep algorithm is easily stated. The algorithm begins with the leftmost cut, and maintains a "bag" of points that elementary steps can be applied to. The algorithm then proceeds to pull a point from the bag, perform an elementary step, check for new points that belong in the bag, and add any new points that are discovered. What remains to be described, are methods for generating the leftmost cut, and finding new points that belong in the bag. Observe, that the need to identify new points for the bag is particularly troublesome, since a naive search would result in a "elementary step" that took $\mathcal{O}(n)$ time to perform, and an algorithms that was $\mathcal{O}(n^3)$ overall. Fortunately, both these tasks can be performed using a complementary pair of data structures: the *upper* and *lower horizon trees*. By making use of horizon trees, the amortized time required for each elementary step is constant.

The *upper (lower) horizon tree* of a cut $c_0, c_1, \ldots, c_{n-1}$ is constructed by extending the edges in the cut to the right, and "killing" edges as they intersect. In particular, whenever

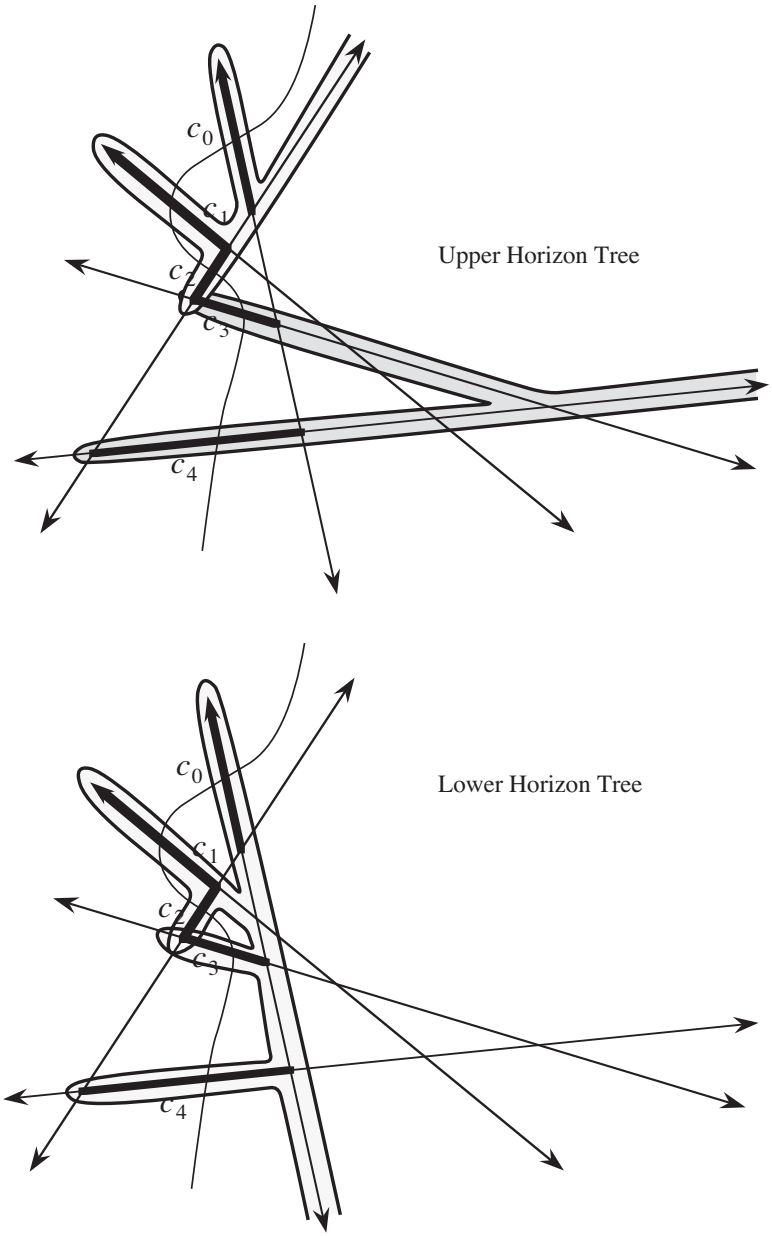


Figure 11: Upper and lower horizon trees for the indicated topological line.

two extended edges intersect, the edge of lower (higher) slope no longer continues to be extended, while edge of higher (lower) slope continues on to the right. Formally, if the edges $c_0, c_1, \ldots, c_{n-1}$ are part of the lines $l_0, l_1, \ldots, l_{n-1}$, then a point $p$ on line $l_i$ is part of the upper horizon tree, if

- $p$ is above all lines $l_j$, where $j > i$, and
- $p$ is below all lines $l_k$, where $k < i$ and $l_k$ is of greater slope than $l_i$.

Similarly, a point $p$ on line $l_i$ is part of the lower horizon tree, if

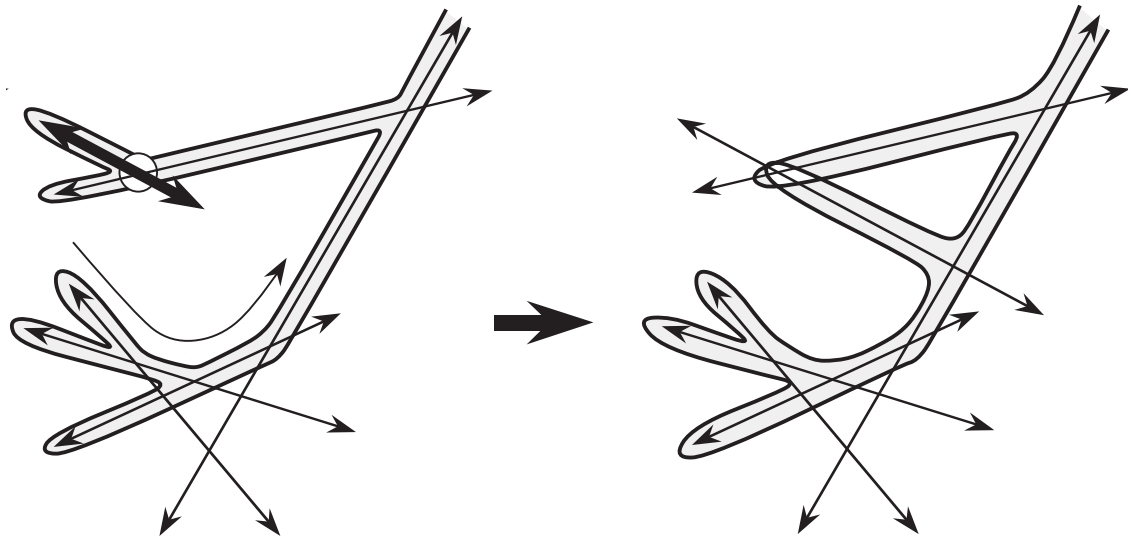

Figure 12: The basic update procedure for the upper horizon tree. Performing and elementary step on the circled point requires extending the darkened line, and searching along the "bay" to find where the extended line intersects the current tree. The final upper horizon tree is shown to the right.

- $p$ is below all lines $l_j$, where $j < i$, and
- $p$ is above all lines $l_k$, where $k > i$ and $l_k$ is of lesser slope than $l_i$.

Figure 11 shows the upper and lower horizon trees for a simple cut. Observe, that the upper horizon tree is not a tree, but rather a forest of two trees. This "difficulty" can be remedied by adding a "vertical" line at $+\infty$, which collects the forest into a single tree. In addition, observe that (as is true in general) the intersection of the upper and lower horizon trees for the cut is the cut itself. Consequently, the cut can be extracted from the upper and lower horizon trees, by simply taking for each line in the plane the shorter of the two edges that exist for the line in the upper and lower trees.

Intuitively, updating the horizon trees after an elementary step is straightforward. Consider the upper horizon tree shown in Figure 12. The circled point is where the elementary step is to be performed. Updating the upper horizon tree consists of removing the edges that approach the point from the left, extending the darkened line to the right until it intersects another "branch" of the current horizon tree, and adding the resulting segment to the new horizon tree. The most difficult part of the update is the search for the point where the extended line intersects the current horizon tree. Fortunately, this search can be accomplished by taking the edge, in the current tree, corresponding to the next edge in the current cut, and then searching back toward the root of the tree for the edge that intersects the extended line.

Computing the upper horizon tree for the leftmost cut is accomplished in a similar fashion. Simply presort the lines in reverse slope order, and incrementally construct the initial horizon tree by "inserting" the lines, in sorted order, with the update routine just described. Since newly inserted lines act as "shields" which prevent some parts of the tree

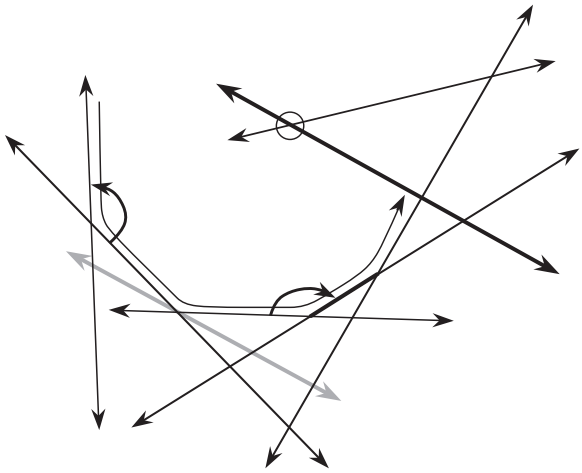

Figure 13: Basic charging scheme for bay traversals. The circled point indicates the site of the elementary step being performed, while the broken line indicates the parallel support line that separates the bay into edges whose slopes are greater/less than the slope of the line being extended.

from ever being examined during subsequent insertions, the total time to construct the initial upper horizon tree is $O(n)$ overall. The construction of the initial lower horizon tree is similar.

All that remains to be shown, is that the search for new intersection points can be accomplished in amortized constant time. Essentially, what needs to be shown is that the time to perform all such searches is $O(n^2)$ total. This can be accomplished by arguing that the time to perform all the searches associated with the "extension" of a particular line $l$ is $O(n)$. Consider the search associated with the update of the upper horizon tree after a particular elementary step where line $l$ is being extended. The search traverses a sequence of edges, which defines a "bay" of the current horizon tree. The time required for each "hop" of the traversal, is charged to individual lines in the following fashion. If the hop is from a edge $c$ to a edge $c'$, and the slope of $c$ is greater than the slop of line $l$, then the time to perform the hop is charged to the line containing $c'$. Alternately, if the slope of $c$ is less than the slope of $l$, then the time for the hop is charged to the line containing $c$. Since the slopes of segments must increase monotonically with the traversal, this charging scheme is guaranteed to account for all the time needed to perform the traversal, except possibly for the time needed for the first and last hops. Since, however, this accounts for at most a constant amount of error per traversal, it will be of no consequence. This charging scheme is depicted graphically in Figure 13.

The key to the argument, is to show that each line in the plane is charged at most once during all the traversals that are associated with the elementary steps on points of a particular line $l$. Consider a line $l'$ that has been charged for a hop to it from another line $l''$. If $p$ is the point where $l'$ and $l''$ intersect, then clearly, no segment of $l'$ that begins to the left of $p$ can be traversed during any search associated with a previous extension of $l$. To see this, simply note that the part of $l''$ to the left of $p$ effectively "shields" all points on $l'$ to the left of $p$ from any previous searches. This fact is depicted graphically in Figure 14. In addition, no segment of $l'$ that begins to the right of $p$ can be traversed during any

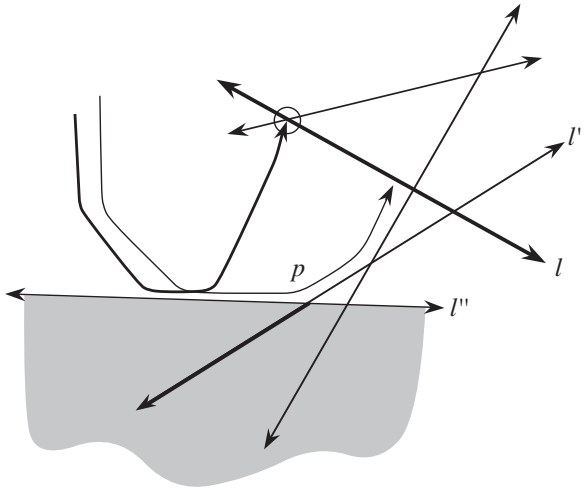


Figure 14: The fact that line $l''$ charges line $l'$ in one bay traversal implies that $l''$ shields the portion of $l'$ (darkened) to the left of point $p$ from being touched by any previous bay traversal.

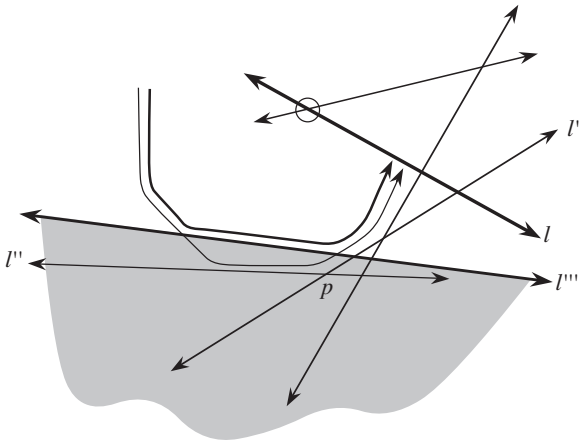


Figure 15: The fact that some previous traversal of a bay traversed an edge of $l'$ that began to the right of point $p$ implies the existence of a line like $l'''$. which shields $p$, and thus, implies a very different form (darkened) for the current traversal.

search associated with a previous extension of $l$. If such a segment were traversed in some previous search, then there would have to exist a line $l'''$ that intersected $l'$ at a point below $l$, as shown in Figure 15. Observe, however, that such a line $l'''$ would "shield" the point $p$, thus contradicting the premise that $l'$ was charged by $l''$. Similar arguments demonstrate that a line charged for a hop from it can also never have been charged during any previous search associated with an extension of line $l$. Consequently, since the above arguments hold even if the charge to line $l'$ is the last associated with all the extensions of line $l$, each line in the plane can be charged at most once. This in turn implies that the time need to perform all the searches is $O(n^2)$ total, and thus, that computing the arrangement of the $n$ lines can be done in $O(n^2)$ time with a topological sweep.

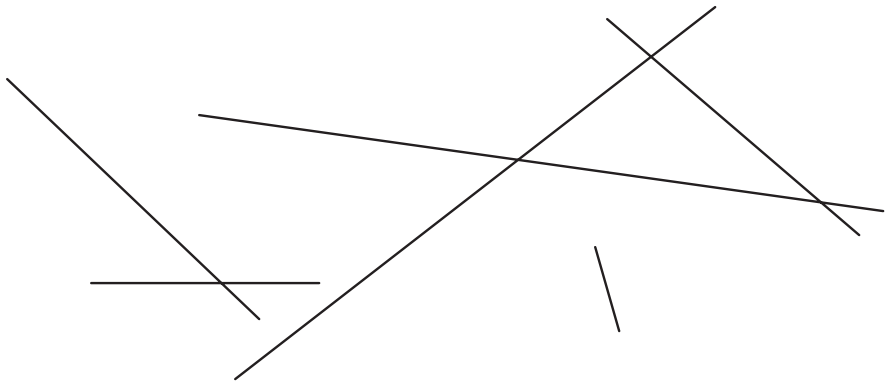

Figure 16: An arrangement with $n = 6$, $k = 4$, $c = 3$.

# 4 Davenport-Scchinzel Sequences

### Arrangements of Line Segments

When dealing with arrangements of lines, we needed only a single parameter (the number of lines) to calculate the exact number of vertices, edges, and faces. This is clearly not good enough for line segments, since an arrangement of $n$ segments can have anywhere from $n$ to $n^2$ edges. Sometimes in order to analyze a geometric structure, we will need to add extra parameters relating to the output. In this case, we are interested in three parameters:

- $n$, the number of line segments
- $k$, the number of intersections between segments, where $0 \leq k \leq \binom{n}{2}$
- $c$, the number of connected components in the arrangement

Using these parameters, we can determine the exact number of vertices, edges, and faces in the arrangement. As before, we will assume that there are no degeneracies (i.e. no three segments ever coincide, and segments do not intersect at their endpoints). Under these conditions there is one vertex for each endpoint, plus one for each intersection. For edges there are $n$ initially, and each intersection adds two more (one for each of the segments that it splits). Hence we have

$$v = 2n + k$$
$$e = n + 2k.$$

To count faces we make use of Euler's Theorem (one of many) which says that $v + f = e + c + 1$ for any planar graph. (The constant at the end depends on the topological properties of the space the graph is embedded into.) Applying this to arrangements of line segments, we get

$$f = k - n + c + 1.$$

The arrangement of figure 16 has 16 vertices, 14 edges, and 2 faces (one of which is the outer face).

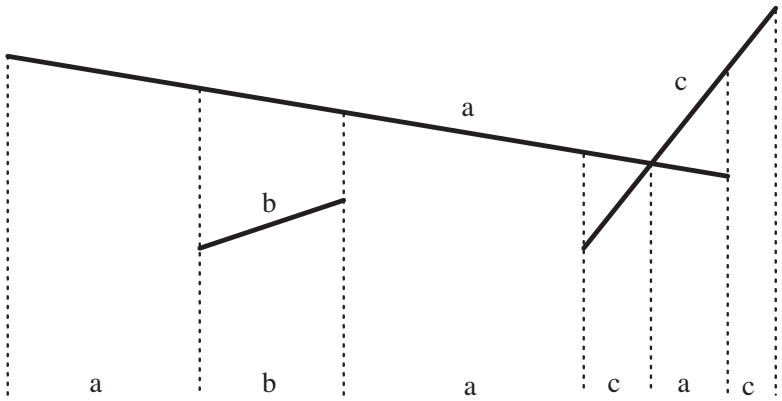


Figure 17: The lower envelope of three line segments.

## Lower Envelopes

We are interested in calculating upper bounds on the complexity of single faces and zones, just as we did for arrangements of lines. A useful concept in calculating these bounds is the *lower envelope* of an arrangement, which can be defined as the portion of the arrangement visible from $z = -\infty$ (see figure 17). The complexity of the lower envelope is simply the number of segment portions which are visible.

Consider looking at the arrangement from below at $z = -\infty$. If no segments intersect, then the lower envelope can change only when we see an endpoint, giving an $O(n)$ upper bound on the complexity. Similarly, in the general case an upper bound is the number of endpoints and intersections that we can see. So far the only upper bound that we have proven is the trivial bound $2n + k = O(n^2)$. It turns out that we can do much better:

**Theorem 2.** *The lower envelope complexity of an arrangement of $n$ line segments is $O(n\alpha(n))$.*

where $\alpha(n)$ is the extremely slow-growing inverse of the Ackermann function (a constant for all practical purposes). A separate construction, not described here, actually achieves this upper bound. Thus the lower envelope complexity of an arrangement of $n$ line segments is $\Theta(n\alpha(n))$. The same technique we will use below shows that this is also the worst-case complexity of a single face.

## Davenport–Schinzel Sequences

Given an alphabet $\Sigma = \{\mathsf{a},\mathsf{b},\mathsf{c},\dots\}$ of $n$ symbols, a *sequence* $\omega$ is a finite string of symbols from $\Sigma$ (i.e. $\omega \in \Sigma^*$). A *subsequence* of $\omega$ is a sequence obtained by deleting some symbols from $\omega$.[1] A *substring* of $\omega$ is a sequence obtained by taking a consecutive group (possibly empty) of symbols from $\omega$.[2]

A Davenport–Schinzel sequence is a sequence where no two symbols are allowed to alternate more than a specific number of times. More precisely, for $s \geq 1$ we say a sequence $\omega$ is an $(n, s)$ Davenport–Schinzel sequence (hereafter an $(n, s)$-sequence) iff:

[1] For example, warm and ten are subsequences of watermelon.

[2] Like in the ad, usa is a substring of wausau.

- For every $\mathsf{a} \in \Sigma$, $\mathsf{aa}$ is not a substring of $\omega$. In other words, every two adjacent symbols are distinct.
- For every pair of distinct symbols $\mathsf{a}, \mathsf{b} \in \Sigma$, the sequence of $s+2$ alternating $\mathsf{a}$'s and $\mathsf{b}$'s is not a subsequence of $\omega$. Thus $\mathsf{aba}$ is forbidden if $s = 1$, $\mathsf{abab}$ is forbidden if $s = 2$, $\mathsf{ababa}$ is forbidden if $s = 3$, and so on.

We are interested in estimating $\lambda_s(n)$, the maximum length of any $(n, s)$-sequence. As an initial exercise, it is easy to check that this function is bounded, i.e. $\lambda_s(n) < \infty$ for all $s$ and $n$.

When $s = 1$ no symbol may appear twice, thus $\lambda_1(n) = n$. A slightly trickier argument shows that $\lambda_2(n) = 2n - 1$, achieved by $\omega = \mathsf{ab}\cdots\mathsf{yzy}\cdots\mathsf{ba}$ or $\mathsf{abacad}\cdots\mathsf{aza}$. The first difficult case is $s = 3$, where $\lambda_3(n) = O(n\alpha(n))$. We present below a "simple" proof of this result due to Peter Shor.

How does this relate to lower envelopes? Two line segments can intersect in at most one place. It is easy to check that while the subsequence $\mathsf{abab}$ can occur in the lower envelope, the subsequence $\mathsf{ababa}$ cannot (it requires two intersections). Thus the sequence of segments in the lower envelope is an $(n, 3)$-sequence (recall figure 17). The proof of Theorem 2 reduces to proving the bound on $\lambda_3(n)$ mentioned above.

Lower envelopes and Davenport–Schinzel sequences arise naturally in many geometric problems. Often we will be dealing with geometric primitives where there is some small constant bound on the number of intersections between any pair (eg. line segments, circles, polynomials). The lower envelope of a collection of such primitives will be a Davenport–Schinzel sequence of some small order. Fortunately, $\lambda_s(n)$ is "almost" linear for every $s$, ie. it is the product of $n$ and some very slow-growing function. For example, $\lambda_4(n) = \Theta(n2^{\alpha(n)})$.

### The Ackermann Function

Here we review the Ackermann function (which grows faster than any primitive recursive function). For each $k \geq 1$, we define a function $A_k$ on the positive integers:

$$A_1(n) = 2n \tag{1}$$
$$A_{k+1}(n) = A_k^{(n)}(1) \tag{2}$$

where $f^{(n)}$ denotes the function $f$ iterated $n$ times. For example

$$\begin{aligned} A_2(n) &= 2^n, \\ A_3(n) &= 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} \text{a tower of } n \text{ 2's}, \\ A_4(n) &= \underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} \Big\} \cdots \Big\} 2^{2} \Big\} 2 \Big\}}_{n \text{ towers}} 1, \end{aligned}$$

and so on (becoming increasingly difficult to typeset). Define the Ackermann function by diagonalization:

$$A(n) = A_n(n). \tag{3}$$

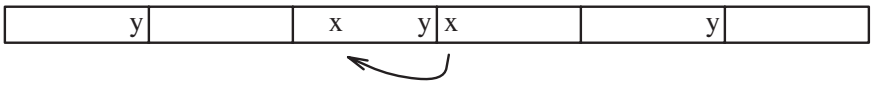


Figure 18: A symbol appearing last in three different chains.

Define the inverse functions in the natural way:

$$\alpha_k(n) = \min\{j \geq 1 : A_k(j) \geq n\}, \quad (4)$$

$$\alpha(n) = \min\{j \geq 1 : A(j) \geq n\}. \quad (5)$$

For example $\alpha_1(n) = \lceil n/2 \rceil$, $\alpha_2(n) = \lceil \lg n \rceil$ (lg denotes the base 2 logarithm), and $\alpha_3(n) = \lg^* n$ (the number of times we need to apply lg to $n$ to get at or below 1). Since $A(3) = 16$ and $A(4) = A_3(65536)$ is much larger than any realistic number, for practical purposes we can assume $\alpha(n) \leq 4$. From the definitions we note the following easy consequences.

**Claim 3.** *For all positive integers $n$ and $k$:*

*(i)* $A_k(n) > n$, *and* $A_k(n)$ *is strictly increasing in* $n$;

*(ii)* $A_k(1) = 2$, $A_k(2) = 4$, $A_k(3) > k$;

*(iii)* $\alpha_{\alpha(n)}(n) \leq \alpha(n)$;

*(iv) if* $n \geq 3$, $\alpha(n) \leq \alpha_{\alpha(n)-1}(n) + 1$;

*(v)* $\alpha_{\alpha(n)+1}(n) \leq 4$;

*(vi)* $\alpha_{k+1}(n) = \min\{l \geq 1 : \alpha_k^{(l)}(n) = 1\}$.

## An Upper Bound Recurrence

Hereafter we fix $s = 3$, and our goal is to bound $\lambda_3(n)$. Given a $(n, 3)$-sequence $\omega$, we want to partition it up somehow into a sequence of *chains*, which are substrings of distinct symbols (hence any chain has length at most $n$). Let $m$ denote the number of chains in our partition.

**Claim 4.** *Any $(n, 3)$-sequence $\omega$ has a partition into $m \leq 2n - 1$ chains.*

**Proof:** Given $\omega$, we construct our chain partition greedily from left to right, starting a new chain only when we come to a symbol already in the current chain. We now show this partition has at most $2n - 1$ chains.

Consider the last symbol of each chain. Suppose a symbol y appears last in three different chains. Then the symbol x immediately after the middle y begins a chain. Since we built the chains greedily, x must also occur in the chain containing the middle y, giving us the forbidden subsequence yxyxy (figure 18).

Thus every symbol appears at most twice as the last in a chain, so there are at most $2n$ chains. To improve this to $2n - 1$, note that the first chain has at least two symbols.

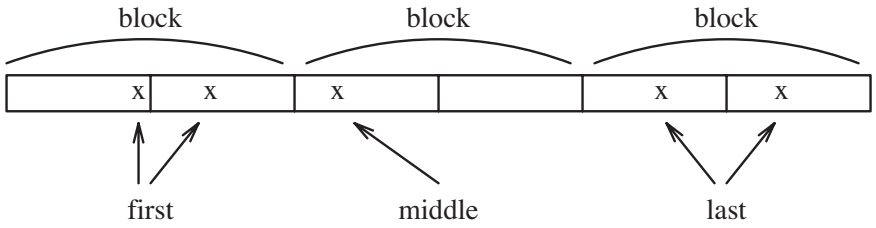


Figure 19: First, middle, and last appearances of x.

Consider the second-last symbol of the first chain, and the same argument shows that it cannot occur last in two other chains. □

We have already shown $\lambda_3(n) < 2n^2$. Our refined analysis of $\lambda_3(n)$ will recursively need to keep track of $m$, so we define $\psi(m,n)$ to be the maximum length of a $(n,3)$-sequence with a partition into at most $m$ chains. Thus $\lambda_3(n) = \psi(2n-1,n)$. We note the trivial bound $\psi(m,n) \leq mn$, which we will use as a base case when $m = 2$.

**Theorem 5.** *Given $n$, $m$, and $2 \leq b < m$, there exists a partition $n = n_1 + \cdots + n_b + n^*$ such that*

$$\psi(m,n) \leq \psi(b-2,n^*) + \sum_{i=1}^{b} \psi(\lceil m/b \rceil, n_i) + 4m + 4n^*.$$

**Proof:** Let $\omega$ be a maximum length $m$-chain $(n,3)$-sequence. We begin by cutting $\omega$ into *$b$ blocks*, each consisting of $c = \lceil m/b \rceil$ consecutive chains each (the last block may have less than $c$ chains). We call a symbol *internal* if it appears in only one block, otherwise we call it *external*. Let $n_i$ denote the number of internal symbols in the $i$-th block, and let $n^*$ denote the number of external symbols; this gives us our partition of $n$.

Consider the subsequence of internal symbols in the $i$-th block. This subsequence would be an $(n_i,3)$-sequence except that it may have repetitions. Since repetitions may only occur at chain boundaries, we may delete at most one symbol at each boundary within the block to remove all repetitions. After these deletions, the remaining symbols are an $(n_i,3)$-sequence, and hence there are at most $\psi(c,n_i)$ internal symbols left in the block. Summing over all blocks, at most $m$ internal symbols are deleted, and at most $\sum_{i=1}^{b} \psi(c,n_i)$ internal symbols remain in the $(n_i,3)$-subsequences.

Now we need to count the external symbols. We divide the appearances of the external symbols into three kinds: *middle*, *first*, and *last* appearances. An occurrence of x is a middle appearance if x occurs in both an earlier and a later block. An occurrence of x is a first appearance if x does not appear in any earlier block; since x is external, x must appear in a later block. Last appearances are defined symmetrically (figure 19).

Consider the subsequence of middle appearances of external symbols. First we need to fix any repetitions; as before, we need delete at most one symbol per chain boundary, for at most $m$ deletions. Call the resulting sequence $\omega^*$.

**Claim 6.** *No symbol of $\omega^*$ appears twice in the same block.*

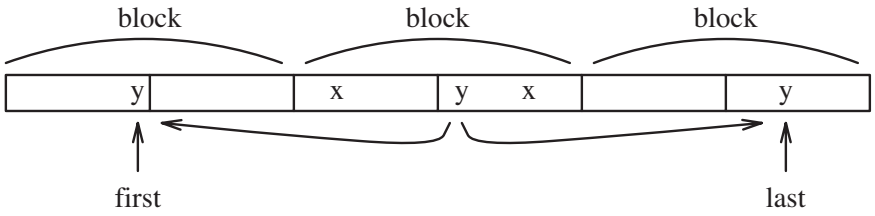


Figure 20: A symbol of $\omega^*$ appearing twice in the same block.

**Proof:** If x appeared twice in the same block of $\omega^*$, then some other external symbol y must appear between these two occurences in $\omega^*$. Since this y is a middle appearance in $\omega$, y also appears in earlier and later blocks of $\omega$, giving us the forbidden subsequence yxyxy in $\omega$ (figure 20). □

Thus the $b$ blocks of $\omega$ have become chains for $\omega^*$. Since no symbol of $\omega^*$ appears in the first or last block, $\omega^*$ has a decomposition into at most $b-2$ chains. Hence $\omega^*$ has length at most $\psi(b-2, n^*)$.

Now all we have left to count are the first and last appearances of external symbols. We argue a bound on first appearances; last appearances are handled symmetrically. Consider the subsequence of all first appearances of external symbols. As before, eliminate all repetitions using at most $m$ deletions; let $\omega^{\text{FIRST}}$ be the resulting sequence.

**Claim 7.** *The sequence $\omega^{FIRST}$ is an $(n^*, 2)$-sequence.*

**Proof:** Suppose xyxy appears as a subsequence of $\omega^{\text{FIRST}}$ for some distinct external symbols x and y. Then these four appearances must all occur in the same block of $\omega$ (since all first appearances of a symbol occur in the same block). Since the x's are first appearances, there must be another appearance of x in a later block, giving us the forbidden subsequence xyxyx in $\omega$. □

Thus $\omega^{\text{FIRST}}$ has length at most $\lambda_2(n^*) < 2n^*$, so there are at most $2n^* + m$ first appearances of external symbols altogether. Likewise for last appearances. We have accounted for all the symbols of $\omega$, so adding all the terms proves the recurrence. □

## Analyzing the Recurrence

Given the diagonal definition of the Ackermann function, we should expect some kind of iterated argument to prove a bound including $\alpha(n)$ as a factor. In effect we prove a sequence of bounds, each bound depending on the previous, which converge to our desired bound. We prove a series of bounds for $k \geq 2$ of the form

$$\psi(m,n) \leq \psi_k(m,n) \stackrel{\text{def}}{=} c_k \alpha_k(m) \cdot m + d_k n \tag{6}$$

where $c_k$ and $d_k$ are sequences of constants to be determined below.

**Claim 8.** *Lets fix $k \geq 2$. Suppose we know $\psi(m,n) \leq \psi_k(m,n)$. Then $\psi(m,n) \leq \psi_{k+1}(m,n)$, where $c_{k+1} = c_k + 4$ and $d_{k+1} = d_k + 6$.*

**Proof:** Consider the recurrence of theorem 5 if we choose $b = \lceil m/\alpha_k(m) \rceil$, so each block has at most $\lceil m/b \rceil \leq \alpha_k(m)$ chains. Then

$$\psi(m,n) \quad \leq \quad \psi(\lfloor m/\alpha_k(m) \rfloor, n^*) + \sum_{i=1}^{b} \psi(\alpha_k(m), n_i) + 4m + 4n^*.$$

To analyze this equation (when $m > 2$), we apply our bound $\psi_k$ to the first term and recurse on the $b$ summed terms. The recursion stops when we reach $m = 2$, at which point we apply the naive bound $\psi(2,n') \leq 2n'$.

We view the recursion as a tree; non-leaf nodes of this tree partitions its sequence, symbols, and chains among its children. At a given node $v$ of the recursion tree, let $m_v$ denote the number of chains at that node, and (when $v$ is not a leaf) let $n_v^*$ denote the number of external symbols found when we recurse at $v$.

At the root of the tree (level 0) we have all $m$ chains and all $n$ symbols. At each node $v$ in the next level of the tree (level 1) we have $m_v = \alpha_k(m)$ chains and $n_i$ symbols (where $i$ here indexes the child). At each node $v$ in level 2 we have $m_v = \alpha_k^{(2)}(m)$ chains, and so on, down to level $l-1$ (the leaves of the recursion) where we have $m_v = \alpha_k^{(l-1)}(m) = 2$ chains per leaf. By claim 3 (parts (ii) and (vi)), we know that $l = \alpha_{k+1}(m)$.

To bound $\psi(m,n)$, we need to sum up the sequence lengths at all the leaves plus the nonrecursive terms ($\psi(\lfloor m_v/\alpha_k(m_v) \rfloor$, $4n_v^*$, and $4m_v$) at all the interior nodes $v$ of the tree. We consider these interior terms first:

$4n_v^*$: Since a symbol becomes external at most once in the entire tree, we have $\sum_v n_v^* \leq n$. Thus summing $4n_v^*$ over all $v$ gives us at most $4n$.

$4m_v$: Since the chains at one level $i$ are all disjoint, we know $\sum_{\text{level } i} m_v \leq m$. Summing $4m_v$ gives us at most $4m$ per level, for an overall total of at most $4\alpha_{k+1}(m) \cdot m$.

$\psi(\lfloor m_v/\alpha_k(m_v) \rfloor, n_v^*)$:

This is where we apply our given $\psi_k$ bound. We estimate

$$\begin{aligned}
\psi(\lfloor m_v/\alpha_k(m_v) \rfloor, n_v^*) &\leq \psi_k(\lfloor m_v/\alpha_k(m_v) \rfloor, n_v^*) \\
&= c_k\ \alpha_k(\lfloor m_v/\alpha_k(m_v) \rfloor) \cdot \lfloor m_v/\alpha_k(m_v) \rfloor + d_k n_v^* \\
&\leq c_k\ \alpha_k(m_v) \cdot \lfloor m_v/\alpha_k(m_v) \rfloor + d_k n_v^* \\
&\leq c_k m_v + d_k n_v^*.
\end{aligned}$$

To sum this over all interior nodes $v$, we reapply the estimates of the last two cases, getting the bound $c_k \alpha_{k+1}(m) \cdot m + d_k n$.

Now the leaf terms are easy to count up: since different leaves count disjoint sets of symbols and have at most $m_v = 2$ chains each, the leaf terms altogether contribute at most $2n$ at the bottom of the unfolded recurrence.

Adding everything up gives $\psi(m,n) \leq 4n + 4\alpha_{k+1}(m) \cdot m + c_k\alpha_{k+1}(m) \cdot m + d_k n + 2n$ $= c_{k+1}\alpha_{k+1}(m) \cdot m + d_{k+1}n = \psi_{k+1}(m,n)$. □

We are almost done, we just need to show how to get started with values for $c_2$ and $d_2$, and also how to choose a good value for $k$ to balance $n$.

**Claim 9.** *$\psi(m,n) \leq \psi_2(m,n) = c_2 m\lceil \lg m \rceil + d_2 n$, where $c_2 = 2$ and $d_2 = 6$.*

**Proof:** We follow a simplified version of the previous argument. We take $b = 2$ on each step. Since we divide the sequence into only two blocks, there are no middle appearances of external symbols (i.e. $\omega^*$ is empty), so we have the simpler recurrence

$$\psi(m,n) \quad \leq \quad \sum_{i=1}^{2} \psi(\lceil m/2 \rceil, n_i) + 2m + 4n^*,$$

where the $4m$ in the general theorem may be reduced to $2m$ in this special case. Our recurrence tree has leaves at level $l - 1$ where $l = \alpha_2(m) = \lceil \lg m \rceil$. Now summing terms as before (again getting $2n$ from the leaves), we get the claimed bound. □

**Corollary 10.** *For every $k \geq 2$, $\psi(m,n) \leq (4k-6)\alpha_k(m) \cdot m + (6k-6)n$.*

**Theorem 11.** *$\lambda_3(n) = O(n \cdot \alpha(n))$.*

**Proof:** Setting $m = 2n$, we have $\lambda_3(n) = \psi(m,n) = O(k(\alpha_k(m) \cdot m + n)) = O(k\ \alpha_k(n) \cdot n)$ for all $k \geq 2$. It now suffices to take $k$ just large enough so that $\alpha_k(n) = O(1)$. By part (v) of claim 3, it suffices to take $k = \alpha(n) + 1$. □

# 5 Arrangements in Higher Dimensions

## Overview

Until now, we have looked at arrangements of lines in the plane. We shall now take a short digression and consider higher dimensional versions of the problem. After some definitions, this lecture will be dedicated to generalizing the zone theorem to higher dimensions.

### Definitions

**Definition 12.** *$E^d$ is the $d$-dimensional Euclidean space.*

**Definition 13.** *A* hyperplane *in $E^d$ is a $d-1$ dimensional linear subspace.*

**Definition 14.** *A $k$-*flat *is a $k$-dimensional linear subspace. An intersection of $d-k$ hyperplanes forms a $k$-flat.*

**Definition 15.** *The* codimension *of a $k$-dimensional subspace of $E^d$ is $d-k$.*

**Counting Objects in the Arrangement**

Recall that an arrangement of lines in the plane creates the following structures: objects of dimension 0 (vertices), of dimension 1 (edges), and of dimension 2 (faces). A similar phenomenon takes place in an arrangement of $n$ hyperplanes in $E^d$, only now structures of all dimensions between 0 and $d$, collectively called *faces*, are created. We will carry over a lot of the terminology from 2 dimensions: 0-dimensional objects (points) will be called *vertices*; 1-dimensional objects are *edges*; $(d-1)$-dimensional objects are called *facets*; $d$ dimensional objects are called *cells*. The terms *facet* and *cell* make a great deal of sense if we consider the case $d = 3$, i.e. the arrangement is a collections of planes in 3-dimensional space. As in the 2-dimensional case, each facet is part of the boundary of a cell; and in general each face of dimension $i$ is a boundary of a face of dimension $i + 1$.

We begin as we did in the 2-dimensional case, by studying the number of vertices formed. Assuming no degeneracies, each group of $d$ hyperplanes intersects to define a single vertex. Thus the number of vertices is just the number of ways to choose $d$ hyperplanes from the arrangement, i.e. $\binom{n}{d}$. Counting higher dimensional faces requires more work.

**Lemma 16.** *The number of cells in an arrangement of $n$ hyperplanes in $E^d$ is*

$$\binom{n}{d} + \binom{n}{d-1} + \cdots + \binom{n}{1} + \binom{n}{0}$$

**Proof:** Recall the 2-dimensional proof. We let each vertex correspond to the face of which it was the bottom vertex, which gave $\binom{n}{2}$ faces, and we then had to count the faces with no bottom vertex, namely those which were unbounded below. To do this, we drew a horizontal line below all the vertices. It clearly passed through $n + 1$ faces, since it intersected each line in the arrangement exactly once, and entered a new face at each intersection. This gave the result we needed. To generalize to the $d$ dimensional case, do the same thing. Each cell bounded from below corresponds to a vertex, namely the lowest vertex in that cell (where "lowest" is measured according to the last dimension). It remains to count the unbounded cells. If we draw a hyperplane $H$ perpendicular to the last dimension and below all the vertices, our $n$ hyperplanes in $E^d$ each intersect $H$ in a $(d-2)$-dimensional space. Thus these $n$ hyperplanes impose an arrangement of $n$ $(d-2)$-dimensional hyperplanes in $H$. Each cell in this arrangement corresponds to one of the unbounded cells in the original arrangement. The result then follows by induction. □

**Lemma 17.** *The number of faces of codimension $k$ in an arrangement of hyperplanes in $E^d$ is,*

$$\binom{n}{k} \sum_{0 \le j \le d-k} \binom{n-k}{j}.$$

**Proof:** Each such face "lives in" some $(d-k)$-flat. The total number of such faces is thus equal to the total number of $(d-k)$-flats, namely $\binom{n}{k}$, times the number of faces living in a single flat. This can be determined using the previous theorem, because for each such $(d-k)$-flat, which is an intersection of $k$ hyperplanes, the remaining $n-k$ hyperplanes impose an arrangement in that flat, namely an arrangement of $n-k$ hyperplanes in $d-k$

dimensions. Each cell (from the perspective of this new arrangement) is a $(d-k)$ dimensional face of the original arrangement. □

This number of faces can be written more cleanly as

$$\sum_{j=k}^{d} \binom{n}{j}\binom{j}{k}.$$

## The Zone Theorem in Higher Dimensions

We now come to the main topic of this lecture, namely the zone theorem in higher dimensions.

**Theorem 18 (The Zone Theorem).** *The zone of a hyperplane in an arrangement of $n$ hyperplanes in $E^d$ has complexity $\Theta(n^{d-1})$, counting all faces of all dimensions.*

**Proof:** Let $H$ be the original set of $n$ hyperplanes, and $A(H)$ the arrangement defined by them. We consider some new hyperplane $b$ and want to study the cells which are in the zone of $b$ in $H$, written $zone(b, H)$. Let

$$\begin{aligned} z_k(b,H) &= \sum_{c \in zone(b,H)} (\text{the number of faces of codimension } k \text{ in } c) \\ z_k(n,d) &= \max_{b,H} z_k(b,H). \end{aligned}$$

Our goal is to show that $z_k(n,d) = O(n^{d-1})$ (showing the corresponding lower bound is left as an exercise). We will do this by first proving and then analyzing the following recursion, which we claim holds for every $d > 1$, $0 \le k \le d$, and $n > k$:

$$z_k(n,d) \le \frac{n}{n-k}(z_k(n,d-1) + z_k(n-1,d-1)). \tag{7}$$

A caveat: by counting over all cells individually, we are actually overcounting the zone complexity, since, *e.g.* a vertex appears in many cells, and so we count it many times. To identify each counting of a given face, define a *border* of codimension $k$ to be a pair $(f, C)$ where $f$ is a $k$-codimensional face and $C$ is a cell that has $f$ as a boundary. Since we will count the total number of borders, our result is actually *stronger* than just counting the total complexity. But the bound is shown to hold even with this overcounting. This is fortunate, since often we need to overcount for applications in precisely this way.

We will prove Equation 7 by induction on the number of hyperplanes in the arrangement. Let $H$ be some arrangement, and let $b$ be the new hyperplane. For each hyperplane $h \in H$, we will consider the effect of removing $h$. Let $H/h$ denote the arrangement induced by $H$ in the hyperplane $h$, *i.e.* the $(d-1)$-dimensional arrangement of hyperplanes $\{j \cap h \mid j \in H\}$. Consider the quantity

$$z_k(b, H - \{h\}) + z_k(b \cap h, H/h)$$

We claim that this counts all the borders $(f, C)$ of codimension $k$ in $zone(b, H)$ with $f \not\subseteq h$. To see this, consider what happens when we remove $h$ from the arrangement. Since $f$ is not

in $h$, $f$ remains part of the zone, although it may now merge with some other faces from which it was previously separated by $h$. Consider some cell $C$ in $b$'s zone with $h$ removed. If $h$ does not cut $C$, then every border in $C$ in the original arrangement is counted by the first term in the above equation. The same holds true if $h$ does cut $C$, but only one of the two resulting cells is in the zone of $b$. This is true since for any border which is cut by $h$, one of the two resulting borders is a border for the cell not in $b$'s zone. The only case which can cause trouble is if both of the cells formed by $h$ splitting $C$ are in the zone of $b$. But for this to happen $h$ must hit $b$ within $C$. This being the case, the second term counts what happens: each border such that both halves are seen by $b$ forms (by intersection with $h$) a $k$-codimensional border in the zone of $b$ in the arrangement of $H/h$.

Now apply this argument to each of the $n$ planes in $h$. Each time we remove a hyperplane, we count all the borders not in that hyperplane using the above equation. Thus each border is counted once each time we remove a hyperplane which does not contain it. Since the border is in a $(d-k)$-flat, which is an intersection of $k$ hyperplanes, this happens $n-k$ times. Thus

$$\begin{aligned}(n-k)z_k(b,H) &= \sum_{h\in H}(z_k(b,H-\{h\})+z_k(b\cap h,H/h))\\ &\leq \sum_{h\in H}(z_k(n-1,d)+z_k(n-1,d-1))\\ &= n(z_k(n-1,d)+z_k(n-1,d-1)).\end{aligned}$$

Since the proof holds for any choice of arrangement, it must hold for the arrangement which gives the maximum count. Equation 7 then follows.

It remains to analyze the inequality in Equation 7. This is straightforward. If we let

$$w_k(n,d)=\frac{(n-k)!}{n!}z_k(n,d)=O(\frac{z_k(n,d)}{n^k})$$

Then our recurrence becomes

$$w_k(n,d)\leq w_k(n-1,d)+w_k(n-1,d-1). \tag{8}$$

We will now prove the result by induction on $n$ and $d$. We have already proved the two dimensional zone theorem, which tells us that for $d=2$, $z_k(n,2)=O(n)$. This gives the base case for the induction. Now assume we have proved the result for $d-1$, and prove it for $d$. Assume first that $k<d-1$. Unwinding the recurrence gives

$$w_k(n,d)=\sum_{j=1}^{n}w_k(j,d-1)+w_k(d,d).$$

Note that $w_k(d,d)$ is just a constant (though it depends on $d$). By our induction hypothesis, $w_k(n,d)=O(n^{d-1-k})$. We can thus rewrite the equation as

$$\begin{aligned}w_k(n,d) &= \sum_{j=1}^{n}O(j^{d-2-k})+O(1)\\ &= O(n^{d-1-k}).\end{aligned}$$

and the result for $z_k$ follows. What remains are the cases $k=d$ or $k=d-1$. Unfortunately, the above recurrence isn't powerful enough—solving it for these values of $k$ overestimates the result by a factor of $\log n$. We thus turn to a different tool—Euler's Theorem.

**Definition 19.** *For a cell $c$, let $g_k(c)$ be the number of facets of codimension $k$ bounding $c$.*

**Theorem 20 (Euler's Theorem).**

$$\sum_{i=0}^{d}(-1)^{d-k}g_k(c) = \begin{cases} 0 & \text{if } c \text{ is unbounded} \\ 1 & \text{otherwise.} \end{cases}$$

If we apply this theorem to our arrangement, and sum over all cells in the zone, we get

$$\sum_{k=0}^{d}(-1)^{d-k}z_k(b,H) \geq 0.$$

If we now move all the $z_k$ whose values we have already bounded to the right side of the equation, we get

$$\begin{aligned} z_{d-1}(b,H) - z_d(b,H) &\leq \sum_{k=0}^{d-2}(-1)^{d-k}z_k(b,H) \\ &= O(n^{d-1}). \end{aligned}$$

Note that $z_{d-1}$ is the number of edges and $z_d$ the number of vertices of the arrangement. We therefore claim that

$$dz_d(b,H) \leq 2z_{d-1}(b,H)$$

To see this, observe that a given vertex is at the end of at least $d$ edges: since a vertex is the intersection of $d$ hyperplanes, the intersection of any $d-1$ of these hyperplanes forms an edge with the vertex as an endpoint. On the other hand, every edge can have at most two vertices as endpoints. Thus the left hand side is an undercount of the number of edge-endpoint pairs, since each vertex is in at least $d$ such pairs. On the other hand, the right hand side is an overcount of the number of such pairs, since each edge is in at most two of them.

Combining these two equations gives

$$\begin{aligned} (1-2/d)z_{d-1}(b,H) &\leq z_{d-1}(b,H) - z_d(b,H) \\ &= O(n^{d-1}). \end{aligned}$$

Therefore $z_{d-1}(b,H) = O(n^{d-1})$, and similarly $z_d$.

□

# 6 Voronoi Diagrams and Delaunay Triangulations

This section deals with two methods of solving proximity problems: Voronoi diagrams and Delaunay Triangulation. These fundamental diagrams are duals of one another. We will begin by presenting some preliminary facts about these diagrams.

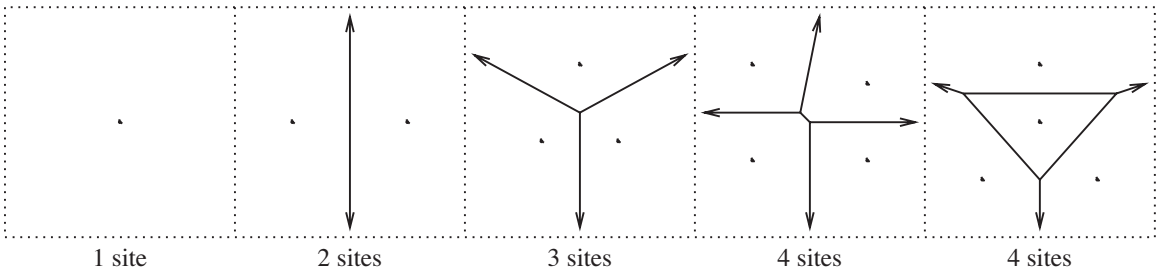


Figure 21: Some simple Voronoi diagrams for $n \leq 4$ sites.

## Voronoi Diagrams

We are given *n sites* (points) in the plane; for any given point, we wish to know which site (or sites) is closest. This problem has been referred to as the "post-office problem" in Knuth, Volume 1. It generates a subdivision of the plane, with each region corresponding to the locus of all points closest to a particular site. This partitioning of the plane is called the Voronoi diagram.

It is worth considering a few simple cases (see figure 21). If there is only one site, then the entire plane is one region. If there are two distinct sites $A$ and $B$, then there are three distinct regions: the points equidistant from the two sites (the perpendicular bisector of the segment between the sites), the locus of points closest to $A$, and the locus of points closest to $B$. If there are three sites (not collinear), we have three unbounded regions divided by three rays from a central vertex (point); this central vertex is equidistant from all three sites (i.e., it is the center of the circle through the sites), and the rays lie on the bisectors of the segments joining the sites. If there are four sites, there are two topologically distinct (and nondegenerate[3]) cases; if the four sites are in convex position, then all four regions are unbounded. If one of the four sites is surrounded by the other three, then there is one bounded face (a triangle) corresponding to the surrounded site.

What distinguishes these two cases? Note that sites on the convex hull of the sites give rise to infinite regions, while those sites within the convex hull correspond to finite regions.

**Claim 21.** *A site $x$ has an unbounded region if and only if $x$ lies on the convex hull of the sites.*

**Proof:** If $x$ is a site on the convex hull, then we may take a supporting line $l$ through $x$ such that all the sites lie on the same side of $l$. Then every point on a ray normal to $l$ going away from the hull has $x$ as its nearest point; hence, the Voronoi region of $x$ is unbounded since it contains this ray. (Note this argument even applies in the degenerate case when there are three collinear sites on the hull).

Conversely, if site $x$ has an unbounded region, then since the region is convex, it must contain an infinite ray emanating from $x$. Taking the line $l$ through $x$ normal to this ray, we see that every other site $y$ must lie on the other side of $l$ (otherwise by going out sufficiently far on the ray, we would find a point closer to $y$ than to $x$). □

[3]Loosely speaking, the arrangement of sites is nondegenerate if no sufficiently small perturbation of the sites changes the topology of the diagram. For Voronoi and Delaunay diagrams non-degeneracy means that no two sites are identical, no three sites are collinear, and no four sites are co-circular.

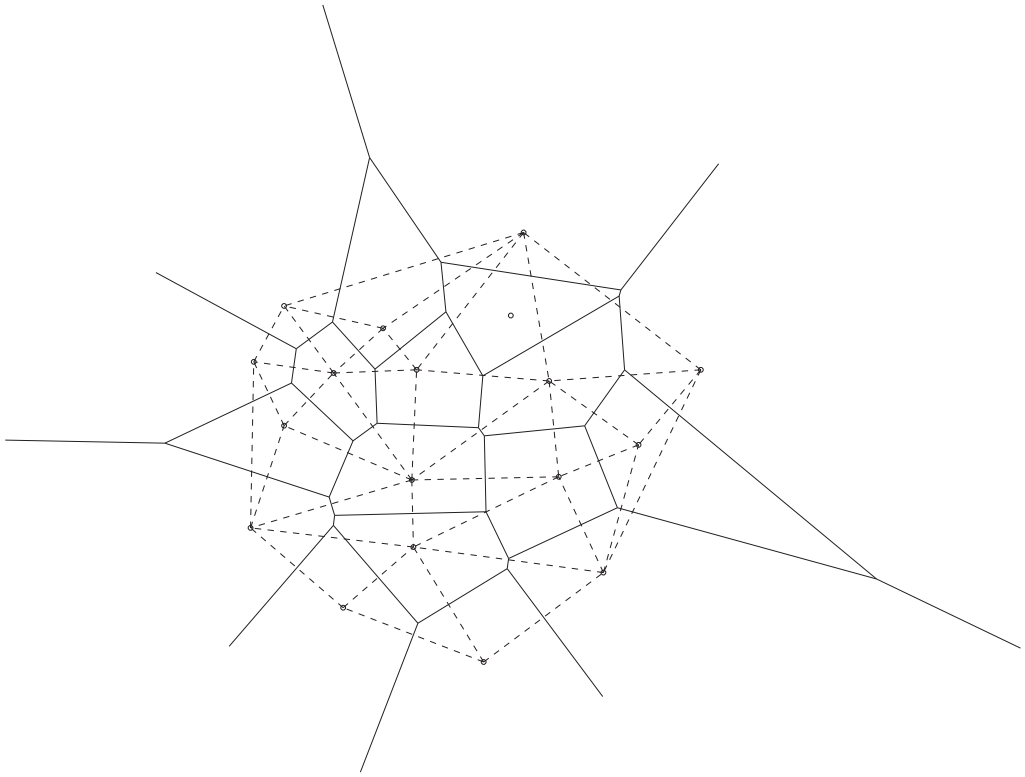

Figure 22: The Voronoi diagram (solid) and the Delaunay diagram (dashed).

**Claim 22.** *All the Voronoi regions are convex.*

**Proof:** Fix a site $x$. For every other site $y$, the Voronoi region of $x$ is contained in the half-plane of points that are at least as close to $x$ as they are to $y$. Thus, the Voronoi region of $x$ is exactly the intersection of all such half-planes for different sites $y$. Since each half-plane is convex and the intersection of convex regions is also convex, the Voronoi region of $x$ is convex. Note this argument also shows that the boundary of the Voronoi region of $x$ is polygonal (i.e., it is bounded by line segments or rays). □

Note that each vertex or edge of the Voronoi diagram depends on only a bounded number of the sites: a vertex is defined by its three nearest sites, and an edge is defined by four sites – the two sites on whose bisector it lies, and two other sites delimiting its extent. This "finite dependency" property will arise in many other settings in geometric algorithms. Thus, if we compute the coordinates of one of the vertices of the diagram, those coordinates are low degree rational functions in the coordinates of the three sites that define it. But still it would be useful to compute a diagram with no "new" real numbers, and this motivates the Delaunay diagram.

## Delaunay Triangulations

The Delaunay triangulation is the *graph theoretic dual, topological dual,* or the *combinatorial theoretic dual* of the Voronoi diagram (see figure 22): the vertices of the Delaunay triangulation are the sites (corresponding to the regions of the Voronoi diagram), and we connect

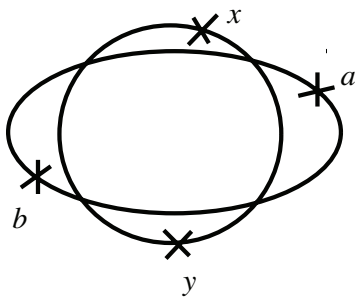


Figure 23: Two circles cannot cross at 4 points.

two sites with an edge in the Delaunay triangulation iff their Voronoi regions share an edge (thus, the two diagrams have the same number of edges, although the Delaunay triangulation has no unbounded edges). Thus, a vertex in the Voronoi diagram corresponds to a Delaunay triangle, a Voronoi edge to a Delaunay edge, and a Voronoi face to a Delaunay vertex (site). Note, however, that dual edges of the two graphs do not necessarily intersect. Furthermore, we may recover the Voronoi diagram by reversing the process. Note there are no 'new' real numbers needed to describe the Delaunay triangulation.

We now prove various simple properties of the Delaunay triangulation. In particular, we present the notion that site-free circles through two or three sites are "witnesses to Delaunay-hood" for edges or triangles, respectively.

**Claim 23.** *For every edge $ab$ in the Delaunay triangulation, there is a circle, called a* witness, *through sites $a$ and $b$ that contains no other site. The converse is also true.*

**Proof:** Since $a$ and $b$ are connected, their Voronoi regions must share an edge. Take a point $p$ in the interior of this edge, then $a$ and $b$ are the same distance $r$ from $p$, and all other sites are further away. Hence, the circle of radius $r$ about $p$ suffices.

Conversely, the existence of a circle through $a$ and $b$ with all other sites strictly outside implies that the center of the circle is on a boundary edge between the Voronoi regions of $a$ and $b$. □

**Claim 24.** *The Delaunay triangulation is planar.*

**Proof:** It suffices to prove that no two edges may cross. Suppose edges $ab$ and $xy$ appear in the Delaunay triangulation, and these edges cross. Then there is a circle through $a$ and $b$ not containing $x$ or $y$, and likewise there is a circle through $x$ and $y$ not containing $a$ or $b$. But then these two distinct circles must intersect at four points, which is impossible. See figure 23. □

Various other properties may also be argued. If $abc$ is a triangle in the Delaunay triangulation, then no other site lies inside its circumcircle, and conversely.

We now turn to counting problems. We will assume non-degeneracy (i.e., no four points are co-circular, no three points are collinear). We will make precise claims about the number of edges and triangles in the triangulation.

Let $n$ be the number of sites, $t$ the number of triangles, and $e$ the number of edges. From Euler's Theorem for connected planar graphs, we have

$$n - e + t + 1 = 2$$

or

$$n - e + t = 1$$

where $t + 1$ is the number of faces (each triangle is a face and so is the single unbounded face). We need one more parameter to describe the arrangement. Since each edge belongs to two faces, we have $3t + k = 2e$, where $k$ is the number of sites on the convex hull. Solving, we have

$$\begin{aligned} e &= 3(n-1) - k, \\ t &= 2(n-1) - k. \end{aligned}$$

By duality, the Voronoi diagram will have $e$ edges (bounded or unbounded) and $t$ vertices. Hence, these graphs have only linear complexity in the size of the input, which will be very important to the computational complexity of various planar proximity problems. For example, this linear complexity implies that the corresponding data structures can be stored in $O(n)$ space and that the arrangement can be computed in $O(n \log n)$ time. However, this only holds for two dimensions. For the $d$-dimensional case, we have $O(n^{\lfloor \frac{d+1}{2} \rfloor})$ complexity.

The Delaunay triangulation is somewhat more intuitive than the Voronoi diagrams, so we will tend to use the Delaunay triangulation in this write-up. Another comment: Since the triangulation starts with numerical data and ends up with combinatorial output (e.g. a graph), complications may arise due to round-off errors, etc.

Delaunay triangulations are used in many applications (e.g., finite element analysis, interpolation). This triangulation is especially useful because it is easy to compute and it provides a "nice" triangulation (i.e., the smallest angle in the triangulation is the largest possible).

Suppose we have $n$ sites, and someone claims a given triangulation is a Delaunay triangulation. How much work do we have to do to verify the claim? If for each site and each triangle we verify that the site is not in the circumcircle of the triangle, then this is clearly sufficient. Unfortunately, this is an $O(n^2)$ time algorithm.

Delaunay proved a remarkable theorem, that it suffices to only check adjacent triangles, so that the verification may be performed in linear time.

**Theorem 25.** *Given a triangulation of $n$ sites such that for every pair of adjacent triangles $abc$ and $bcd$, $a$ is not in the circumcircle of $bcd$, then that triangulation is the Delaunay triangulation.*

**Proof:** First, we define the "power" of a point with respect to a circle. Given a point $x$ and a circle $C$, draw a line $l$ through $x$ that intersects $C$ at two points $a$ and $b$. Then the power of the point is $xa \cdot xb$ (here $xa$ denotes the distance from $x$ to $a$). It turns out that this quantity is independent of the line chosen. Furthermore, given a suitable choice of signs

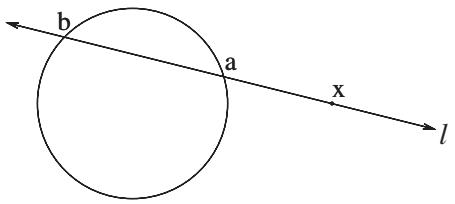


Figure 24: The power of point $x$ with respect to the circle is $ax \cdot bx$, which is independent of the choice of line $l$.

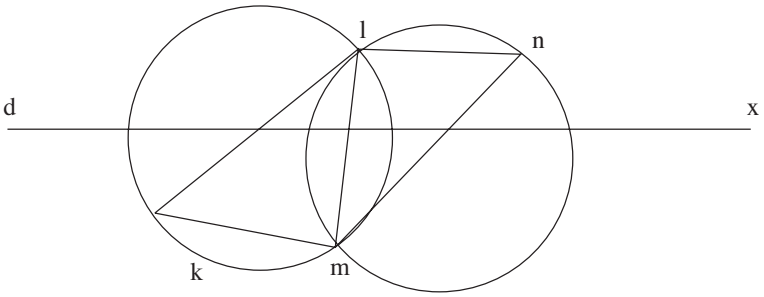


Figure 25: The power of $x$ with respect to $lmn$ is less than the power of $x$ with respect to $klm$.

for distances along the line, the sign of the power determines whether $x$ is inside, outside, or on the circle (the usual convention makes the power positive iff $x$ is outside the circle).

Take a triangle $abc$ of the given triangulation and a site $x$ that lies on the opposite side of (say) line $bc$. If the neighboring triangle to $abc$ along $bc$ is $xbc$, then we are done by assumption.

Otherwise, consider the line connecting site $x$ to a point $d$ in the interior of triangle $abc$. As we move along $xd$ from $x$ to $d$, we pass through a sequence of triangles, starting with a triangle having $x$ as a vertex (call it $vwx$) and ending with triangle $abc$.

Now consider the sequence of circumcircles of these triangles. We start with the circumcircle of $vwx$ where the power of $x$ is 0 (since $x$ is on the circle). We claim that the power of $x$ can only increase as we move to circumcircles of triangles further away from $x$. This claim implies that the power of $x$ with respect to the circumcircle of $abc$ must be positive, so that $x$ is not in the circumcircle of $abc$, which is what we wanted to prove.

We now argue our claim informally. Given adjacent triangles $lmn$ and $klm$ on the line from $x$ to $d$ (see figure 25), consider their circumcircles. These circumcircles are both members of the family of circles through $l$ and $m$, and we may continuously "push" the circumcircle of $lmn$ away from site $n$ until it reaches site $k$. During this push operation, both intersection points of the circle with $xd$ move further away from $x$, hence the power of $x$ increases. □

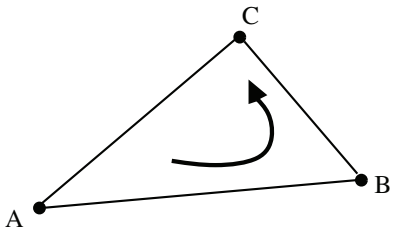


Figure 26: Three points in counterclockwise order

# 7 Geometric Primitives and A Delaunay Triangulation Algorithm

In the first part of this lecture, we are going to introduce two geometric primitives which enable us to extract some fundamental combinatorial properties of points. In the second part, we will introduce an algorithm based on the divide and conquer idea for the Delaunay triangulation.

## Geometric Primitives

Computational geometry problems are characterized by relations among geometric entities (vertices, edges, ...) and many of these relations can be described quantitatively by real numbers. But many geometric algorithms are combinatorial by nature. So we need to have a conversion scheme. Geometric primitives are used to extract combinatorial information from the input real numbers; the geometric primitives will map the real numbers to bits (hence these primitives may also be called predicates). The nice thing about using geometric primitives is that the complexity of these primitives is constant.

More precisely, these geometric primitives will depend on the *signs* of certain determinants. Thus the primitives will actually have three values: one of $\{+, -, 0\}$. These correspond to *true*, *false*, and *degenerate*. If we assume our input points are in general position, then 0 (the degenerate case) will never occur, and it is convenient to specify our algorithms under this assumption. In principle we should specify robust algorithms which also correctly handle the degenerate cases, but we avoid this for brevity of exposition.

### The *CCW* Primitive

Our first geometric primitive is *CCW*. Given three points $A$, $B$ and $C$, *CCW* tells us whether these three points are in counterclockwise order around their common circle by computing the sign of the determinant

$$\begin{vmatrix} x_A & y_A & 1 \\ x_B & y_B & 1 \\ x_C & y_C & 1 \end{vmatrix}$$

If the sign is positive then $A, B, C$ are in counterclockwise order; if the sign is negative then $A, B, C$ are clockwise in order; if it is zero, then $A, B, C$ are collinear (see figure 26).

Note that to determine the sign of a determinant is as expensive as to calculate its value.

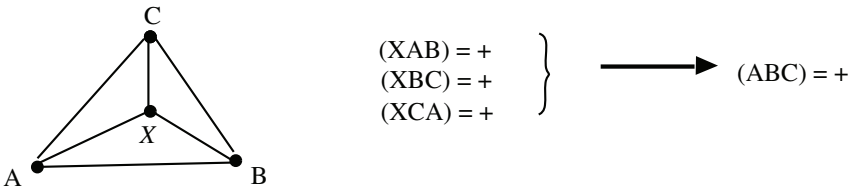


Figure 27: Dependence of the sign bits

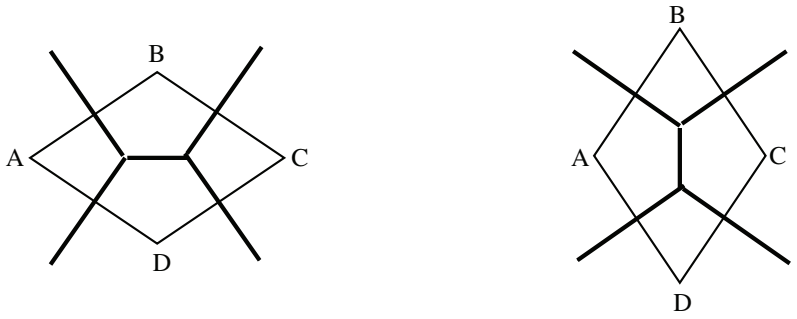


Figure 28: Point sets with the same *CCW* bits but different Voronoi Diagrams

An interesting question is, given $n$ points and the $\binom{n}{3}$ sign bits corresponding to the $\binom{n}{3}$ different triangles, how many of the bits are independent? It turns out that only $\Theta(n \log n)$ of them are independent. For example, figure 27 shows that if $CCW(X, A, B)$, $CCW(X, B, C)$, and $CCW(X, C, A)$ are all positive, then $CCW(A, B, C)$ must be positive as well.

We may apply the *CCW* primitive to the convex hull problem. Given $n$ points, the convex hull problem is to find all the extreme point in order around the hull. A point $x$ is not an extreme point of the convex hull if and only if if $x$ is in the interior of triangle formed by three other points, i.e. iff we can find $A, B, C$ with their *CCW* sign bits as in figure 27. Similarly we can recover the order of points around the hull using *CCW*, hence we can construct the convex hull using only *CCW*.

But the *CCW* primitive is not sufficient for computing the Voronoi Diagram or the Delaunay Triangulation. In some arrangements each of the $\binom{n}{3}$ triangles have the same *CCW* bits but the $n$ points have non-isomorphic Voronoi Diagrams; see figure 28 for an example (with $n = 4$). Our next primitive will be sufficient for constructing the Voronoi Diagram and the Delaunay Triangulation.

**The *InCircle* Primitive**

Given 4 points $A, B, C, D$, the primitive $InCircle(A, B, C, D)$ is true if and only if $D$ is in the left face of the oriented circle through $A, B, C$. The left face is the interior of the circle if $CCW(A, B, C)$, otherwise the left face is the exterior of the circle. Thus if we take $A, B, C$ in counterclockwise order, *InCircle* tells us whether $D$ is in their circle (see figure 29).

Note that there is a degenerate situation if $D$ is on the circle, or if all four points are collinear. Curiously the situation is not degenerate when $A, B, C$ are collinear and $D$ is off

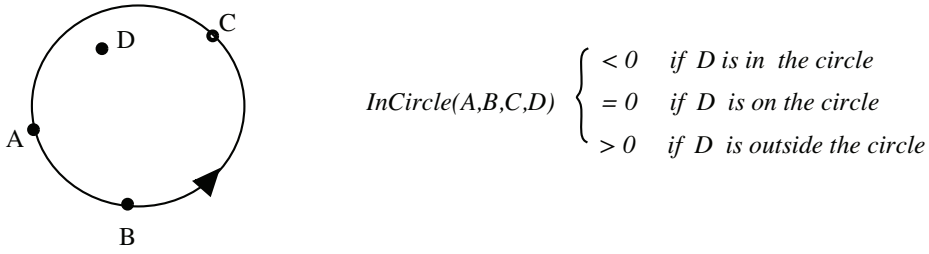


Figure 29: The *InCircle* primitive

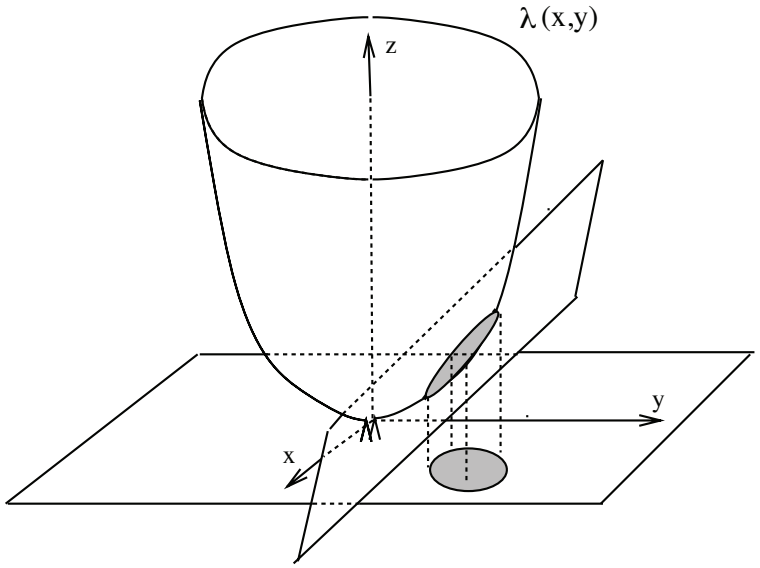


Figure 30: The projection of cocircular points on the parabola

their line, since there is still a definable left face of the line.

**Theorem 26.** *The value of $InCircle(A, B, C, D)$ is given by the sign of the determinant*

$$det = \begin{vmatrix} x_A & y_A & x_A^2 + y_A^2 & 1 \\ x_B & y_B & x_B^2 + y_B^2 & 1 \\ x_C & y_C & x_C^2 + y_C^2 & 1 \\ x_D & y_D & x_D^2 + y_D^2 & 1 \end{vmatrix}$$

Before proving the theorem, we prove the following lemma.

**Lemma 27.** *$InCircle(A, B, C, D)$ is degenerate if and only if the value of the above determinant is 0.*

**Proof:** We project a point in the plane to a point on the hyperbolic paraboloid $z = x^2 + y^2$ by mapping (see figure 30):

$$\lambda : (x, y) \longmapsto (x, y, x^2 + y^2)$$

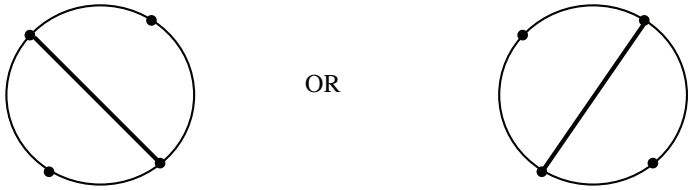


Figure 31: Two possible Delaunay edges for four cocircular points

Points $A, B, C$ and $D$ are mapped to points $A', B', C'$ and $D'$ on the parabolic surface. We claim that points $ABCD$ are cocircular if and only if points $A'B'C'D'$ are coplanar.

Suppose first that the points are cocircular. Let $(p, q)$ be the center of circle $ABCD$ and r be radius; we have the circle equation:

$$(x-p)^2 + (y-q)^2 - r^2 = 0$$

that is

$$(-2p)x + (-2q)y + (1)(x^2+y^2) + (p^2+q^2-r^2)(1) = 0$$

The equation holds for the coordinate pairs $(x_A, y_A)$, $(x_B, y_B)$, $(x_C, y_C)$, and $(x_D, y_D)$. Therefore there is a linear dependence on the columns of the determinant *det* and its value must be zero. (Similarly, if the four points are collinear, then there is a linear dependence among the first, second, and fourth columns of *det*.)

Conversely, suppose the determinant is zero, i.e., the four columns are linearly dependent. If there is a linear dependency not involving the third column, then the four points are collinear. Otherwise, the third column may be written as a linear combination of the other three columns, i.e. there are constants $a$, $b$, and $c$ such that $A, B, C, D$ all lie on the curve

$$x^2 + y^2 = ax + by + c$$

By completing the squares in this equation, we see that it is in fact a circle centered at $(-a/2, -b/2)$, hence the points are cocircular. □

To complete the proof the theorem, interpret the sign of the determinant as the sign of the volume of tetrahedron $A'B'C'D'$. Since the parabolic surface is convex, the interior of the circle $ABC$ projects to points below the plane $A'B'C'$ and the exterior of the circle projects to points above the plane; hence as we move $D$, the sign of the determinant changes when we cross the circle. Now we only need to check the sign in one specific situation to finish the proof (omitted). □

When we apply the *InCircle* test in our Delaunay Triangulation algorithms, we may face a degenerate situation if some four points are cocircular (see figure 31). For simplicity when we present our algorithms we will assume that the points are in general position: no three points are collinear and no four points are cocircular.

## A Divide and Conquer Algorithm

In what follows,we present an algorithm to compute the Delaunay triangulation (and Voronoi diagram) of n sites in the plane, and analyze its complexity. Suppose we have

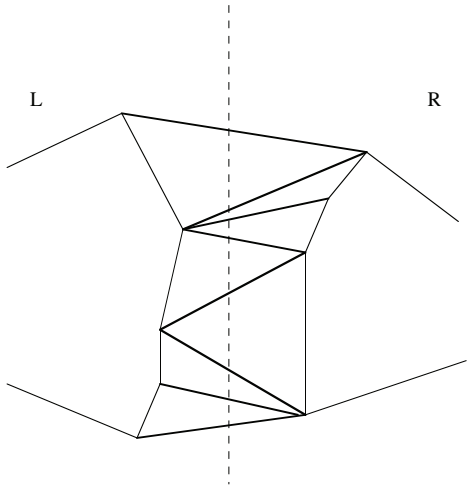


Figure 32: The structure of the cross edges

$n$ sites in the plane $S_1, \ldots, S_n$. Sort them by $x$–coordinate (in case of equal $x$ sort by $y$) and then (as divide and conquer algorithms typically work) divide them in two equal halves, the left one (denoted by $L$) and the right one ($R$), compute recursively the Delaunay diagrams of $L$ and $R$, and then recombine. The recursion stops when $n = 2$ or $n = 3$ since in these cases the diagram is easy (also our quad-edge structure cannot represent a single point!). As usual the merge step needs to be analyzed in more detail. During this step we have to delete some edges and add some others. Clearly we will delete some $L$–$L$ (or $R$–$R$) edges, and we will add only $L$–$R$ (called sometime *cross*) edges. Indeed no new $L$–$L$ (or $R$–$R$) edge can be added, since such edges would have already been Delaunay.

We can think of the cross edges ordered in ascending $y$–order and the algorithm that we present will produce them in this order. The first cross edge will be the lower common tangent of the convex hull of $L$ and the convex hull of $R$; similarly the last cross edge will be their upper common tangent. Thus we can begin finding the lower common tangent. To find it we use this algorithm:

1. Start from the edge $e$ connecting the rightmost site of $L$ with the leftmost site of $R$.
2. While $e$ is not a lower tangent to $R$, move its right endpoint counterclockwise around the hull of $R$.
3. While $e$ is not a lower tangent to $L$, move its left endpoint clockwise around the hull of $L$.
4. If $e$ is not yet the common lower tangent then go to step 2.

This algorithm ends with the lower common tangent of $L$ and $R$, and takes only linear time (since it passes each hull edge at most once).

**Lemma 28.** *Any two consecutive cross edges share a common vertex.*

**Proof:** Since the Delaunay diagram is a triangulation and the cross edges are exactly the Delaunay edges crossing a vertical line, two consecutive cross edges must belong to a triangular face and thus share a common vertex. □

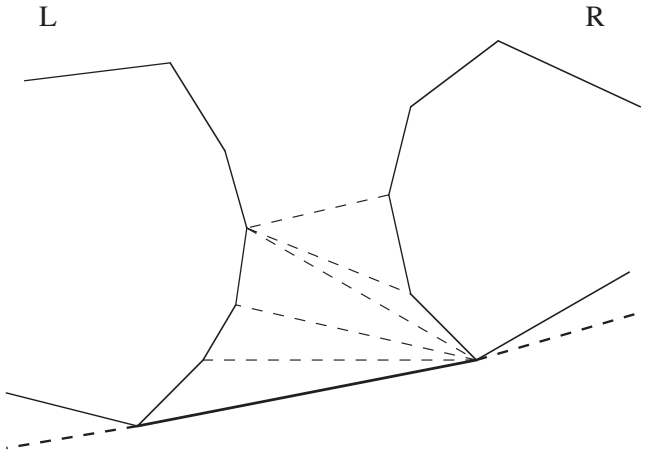


Figure 33: Finding the lower common tangent

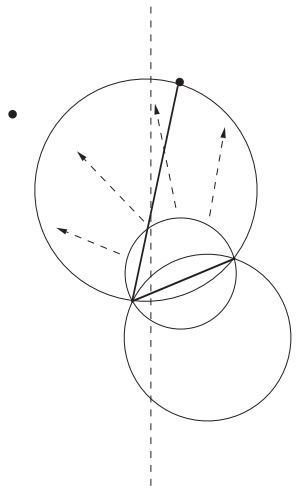

Figure 34: The rising bubble

A consequence of this lemma is that if we start from a known cross edge (let us call it *basel*), its successor will be an edge going from one of the vertices of *basel* to a neighbor of the other vertex lying above *basel*. We can think of a circular bubble that has *basel* as a chord and rises through that cord until it meets a new site; the edge connecting this site to the vertex of *basel* in the opposite half will be the next cross edge. This process continues until our bubble becomes the upper common tangent of L and R (see figure 34).

To find this edge, the following two lemmas are useful. Here $Us(AB)$ denotes the upper halfplane determined by $AB$, $Ls(AB)$ the lower halfplane, $Hs(AB)$ either of upper or lower and $Cir(XYZ)$ denotes the interior of the circle through the three points $X, Y, Z$.

**Lemma 29.** *Let AB be an edge. For any point M and N, we always have* $Cir(ABM) \cap Hs(AB) \supseteq Cir(ABN) \cap Hs(AB)$ *or* $Cir(ABM) \cap Hs(AB) \subseteq Cir(ABN) \cap Hs(AB)$

**Proof:** The proof is trivial and is omitted here. □

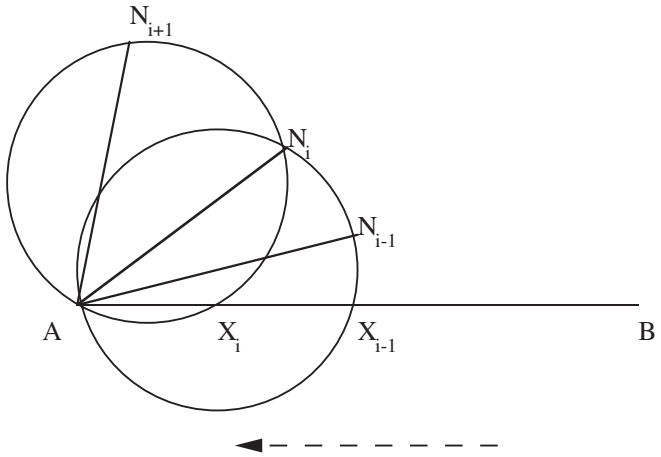

Figure 35: Proof of lemma 30

**Lemma 30.** *Let $AB$ be the basel edge and let $N_i$ $i = 1, \ldots, k$, be the $L$-neighbors of $A$ in $Hs(AB)$ in counterclockwise order; then the succession $\Gamma_i = Cir(ABN_i) \cap Hs(AB)$ is unimodal in the sense that there is some $j$ such that for $1 \leq i < j$ we have $\Gamma_i \supseteq \Gamma_{i+1}$ while for $j \leq i < k$ we have $\Gamma_i \subseteq \Gamma_{i+1}$.*

**Proof:** For each circle $AN_iN_{i+1}$ let $X_i$ be its intersection with line $AB$. We claim the sequence of points $X_i$ move monotonically toward $A$. Consider 3 consecutive neighbors $N_{i-1}, N_i, N_{i+1}$ of $A$. The point $N_{i+1}$ is not inside the circle $AN_{i-1}N_i$, for otherwise $AN_i$ could not have been a Delaunay edge in $L$. Thus we have to push circle $AN_{i-1}N_i$ through chord $AN_i$ to reach point $N_{i+1}$, and the intersection with $AB$ moves toward $A$, proving our claim.

The lemma is now proved by observing that while $X_i$ is to the right of $B$, $B$ is inside the circle $AN_iN_{i+1}$ or equivalently $N_{i+1}$ is inside the circumcircle $ABN_i$, so $\Gamma_i \supseteq \Gamma_{i+1}$. After $X_i$ moves to the left of B, a similar argument shows that $N_{i+1}$ is outside the circle $ABN_i$, so $\Gamma_i \subseteq \Gamma_{i+1}$. □

We can prove an analogous lemma for the $R$-neighbors of $B$, and in this way we can find two candidates (one for each side) to be the next cross edge. A final test between the two candidates will give us the next cross edge. The following lemma formally proves this result.

**Lemma 31.** *Suppose AB is the previous Delaunay edge. AM and BN are two new candidates. Assume BN is the one to be picked (i.e., N is inside Cir(ABM). Then we claim that BN is the new Delaunay edge.*

**Proof:** First we show that no vertices in $Ls(AB)$ is inside of $Cir(ABN)$. Assume ABC was the last Delaunay triangle. Since $Cir(ABC)$ does not contain N, $Cir(ABC) \cap Ls(AB) \supseteq Ls(ABN) \cap Ls(AB)$. ABC implies that no vertices are in $Cir(ABC) \cap Ls(AB)$. So no vertices are in $Ls(ABN) \cap Ls(AB)$. So no vertices in $Ls(AB)$ are inside of $Cir(ABN)$. Now we claim that no vertices in $Us(AB)$ are in $Cir(ABN)$. Since M is outside of $Cir(ABN)$,

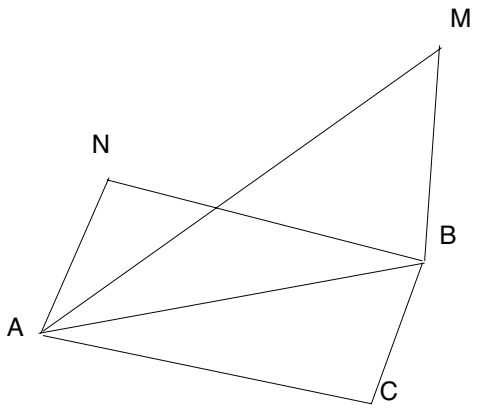


Figure 36: Proof of lemma 31

$Cir(ABM) \cap Us(AB) \supseteq Ls(ABN) \cap Us(AB)$. But we know that all vertices in $L \cap Us(AB)$ are not inside of $Us(AB) \cap Cir(ABN)$ and all vertices in $R \cap Us(AB)$ are not inside of $Cir(ABM) \cap Us(AB)$; this implies that all vertices in $Us(AB)$ are not in $Ls(ABN) \cap Us(AB)$. This completes the proof. □

Now we are going to analyze the time complexity for the algorithm. Suppose $A$'s candidate is $N_j$. The unimodularity proved in the lemma allows us to find $N_j$ in $j$ comparisons, by searching until the first "local minimum" going counterclockwise around $A$. Furthermore the edges $AN_i$ for $i < j$ can be immediately deleted (even if $B$'s candidate won over $N_j$) because we know that these edges cannot be part of the merged Delaunay diagram ($B$ prevents them).

Thus the cost of searching through the candidate vertices can be charged to the deleted edges (all other costs are clearly linear). Since the old triangulations had a linear number of edges, this gives us linear time overall to find all the cross edges, completing the merge. Since the first part of the merge (finding the lower common tangent) was also linear time, the entire merge process takes $O(n)$ time, and the time complexity of the algorithm is $O(n \log n)$ (Note that the initial sorting step needs to be done only once).

# 8 Randomized Incremental Algorithm for Delaunay Triangulations

In the previous lecture we discussed a divide-and-conquer algorithm to find the Delaunay triangulation of a set of sites. In this lecture we present a *randomized* incremental algorithm for the same problem. We will first discuss the basic incremental algorithm (without introducing randomization). We will then *randomize* the algorithm to improve its expected running time.

## A Basic Incremental Algorithm

The idea in our incremental method is to add the sites in some sequence to the current set and *update* the Delaunay triangulation of the set of sites to reflect the addition of the new

site. For convenience in describing the algorithm, we add three hypothetical sites at the very beginning; they are chosen such that they form a very large triangle that contains all the other sites. If we make this triangle large enough, it will clearly not affect the Delaunay triangulation of the original sites.

We now define our incremental Delaunay triangulation algorithm, starting with the triangle on these three sites, adding the other sites in some order, and updating the Delaunay triangulation after each addition.

Suppose that we add a new site $S$. Assuming non-degeneracies, it will be inside a triangle. The idea is to locate the triangle that the new site falls into, link the new site to the three vertices of the enclosing triangle, and reconsider the Delaunayhood of the edges of the enclosing triangle; if such an edge fails the circle test, the edge is flipped, and the test-and-flip procedure continues for edges further out.

Let $XYZ$ be the old Delaunay triangle containing $S$, with circumcircle $C$. Then the new edges $SX$, $SY$, and $SZ$ are Delaunay: the circle passing through $S$ and tangent to $C$ at $X$ is a site-free witness for the Delaunayhood of the edge $SX$ (we can argue analogously for $SY$ and $SZ$).

The edges $XY$, $XZ$, $YZ$ are *suspect* since we do not know if they pass the *InCircle* test with respect to $S$ and the triangle on their other side. We have to check these suspect edges; if an edge fails the *InCircle* test then it will be swapped, creating a new Delaunay edge emanating from $S$ and creating two new suspect edges that must now be tested. We call this the "wave of suspicion" spreading from the newly inserted site $S$ (illustrated in Fig 37).

We will now prove a worst case upper-bound on the running time of the algorithm over all possible locations of the sites and all possible sequences of insertions of the sites.

**Lemma 32.** *When a new site $S$ is inserted, no edge is tested more than once.*

**Proof:** The wave of suspect edges moves away from $S$ to the edges on the convex hull (which are always Delaunay), and all the new edges (forming a star around $S$) are never suspect. □

When the wave stops all the edges pass the *InCircle* test, so the triangulation is again Delaunay. The previous lemma tells us that each insertion needs at most $O(n)$ time, so the total time required by the algorithm is $O(n^2)$.

It is possible to construct example input sequences which achieve this quadratic running time (we leave that as an exercise). We will now see that this algorithm has much better $O(n \lg n)$ expected running time when the input sites are inserted in a *random* order.

## Randomizing the Incremental Method

We saw that the obvious sequential incremental method may create $O(n^2)$ Delaunay triangles during the running of the algorithm. To try to achieve $O(n \log n)$ time we need to randomize. The algorithm is very easy. Given a set of sites, we simply choose a random ordering of the sites and add them one at a time, updating the Delaunay triangulation as

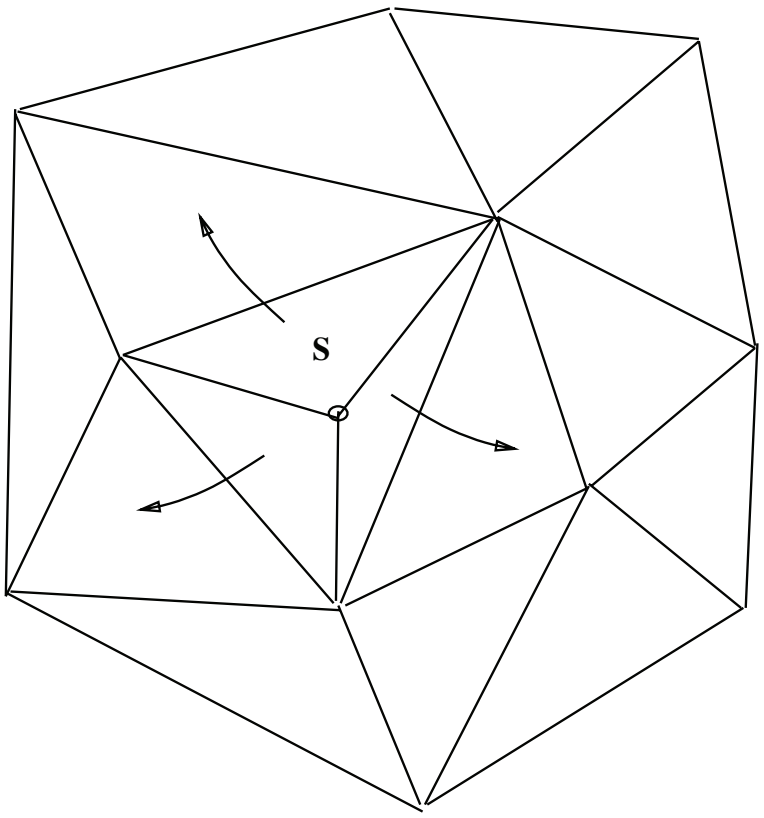


Figure 37: The wave of suspect edges

already described. Note that we have not yet described a way of locating the triangle in which the new site lies (we will do this in the next section).

We observe that the total amount of work performed during the algorithm is related to the number of Delaunay triangles that ever arise during the running of the algorithm. We will prove that the expected number of such triangles is $\Theta(n)$.

**Theorem 33.** *If $\tau$ is the expected number of Delaunay triangles that ever arise during the running of the algorithm, then $\tau = O(n)$. In fact, $\tau \leq 6n - 15H_n + 10.5$, where $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$.*

**Proof:** For a triangle $\triangle$ defined by three sites, let $w(\triangle)$ denote the *scope* of the triangle: the number of sites inside the circumcircle of $\triangle$. Let $T_j$ be the number of triangles of scope $j$ (so in particular $T_0$ is the number of triangles in the final Delaunay triangulation). Then

$$\tau = \sum_{\triangle} Pr[\text{at some point of the algorithm, } \triangle \text{ appears as Delaunay}]$$

If $w(\triangle) = j$, then $\triangle$ appears as Delaunay sometime during the algorithm iff the three sites defining $\triangle$ are chosen before any of the $j$ sites inside its circumcircle. Thus

$$Pr[\triangle \text{ appears as Delaunay}] = \frac{3!j!}{(j+3)!} = \frac{6}{(j+1)(j+2)(j+3)}$$

and

$$\tau = \sum_{j=0}^{n-3} \frac{6}{(j+1)(j+2)(j+3)} T_j$$

Now, suppose we choose $r$ of the sites at random and look at the Delaunay triangulation of this $r$-set. The expected number of triangles in this triangulation is $2r - h_r - 2$, where $h_r$ is the expected number of sites (from among the $r$-set) on the convex hull of the $r$-set.

$$\begin{aligned} 2r - h_r - 2 &= \sum_{\triangle} Pr[\triangle \text{ is in Delaunay triangulation of } r\text{-set}] \\ &= \sum_{j=0}^{n-r} \frac{\binom{n-j-3}{r-3}}{\binom{n}{r}} T_j \end{aligned}$$

Let $T_{\leq k} = \sum_{i=1}^{k} T_i$.

**Lemma 34.** $T_{\leq k} = O(n(k+1)^2)$.

**Proof:**

$$\begin{aligned} 2r &\geq 2r - h_r - 2 \\ &= \sum_{j=0}^{k} \frac{\binom{n-j-3}{r-3}}{\binom{n}{r}} T_j \quad \geq \quad \sum_{j=0}^{k} \frac{\binom{n-k-3}{r-3}}{\binom{n}{r}} T_j \\ &= \frac{\binom{n-k-3}{r-3}}{\binom{n}{r}} \sum_{j=0}^{k} T_j \quad = \quad \frac{\binom{n-k-3}{r-3}}{\binom{n}{r}} T_{\leq k}, \end{aligned}$$

therefore

$$T_{\leq k} \quad \leq \quad \frac{2r\ n!}{r!\ (n-r)!} \cdot \frac{(r-3)!\ (n-k-r)!}{(n-k-3)!}$$

This is true for all $r$. It turns out that the $r$ value that makes this formula as small as possible is $\frac{2n}{k+2}$. Thus, $T_{\leq k} = O(n(k+1)^2)$. □

To finish the proof of the theorem we recall the technique often used to simplify integrals, namely integration by parts. Here we do a summation by parts.

$$\begin{aligned} \tau &= \sum_{j=0}^{n-3} \frac{6}{(j+1)(j+2)(j+3)} T_j \\ &= T_0 + \sum_{j=1}^{n-3} \frac{6}{(j+1)(j+2)(j+3)} (T_{\leq j} - T_{\leq j-1}) \\ &= T_0 + 18 \sum_{j \geq 1} \frac{T_{\leq j}}{(j+1)(j+2)(j+3)(j+4)} \\ &= T_0 + \sum_{j \geq 1} \frac{O(n(j+1)^2)}{(j+1)(j+2)(j+3)(j+4)} \end{aligned}$$

$$
\begin{aligned}
&= T_0 + O(\sum_{j \geq 1} \frac{n}{j^2}) \\
&= O(n)
\end{aligned}
$$

□

### Efficient Randomized Point-Location

While insertion of a particular site can take $O(n)$ time in the worst case, we showed that if sites are inserted in random order, the expected overall running time of the update part of the incremental algorithm for constructing the Delaunay triangulation is $O(n)$. We are yet to describe a crucial part of our randomized incremental algorithm: namely, locating the triangle in which the new site falls.

In order to obtain an overall efficient algorithm for the randomized incremental construction of the Delaunay triangulation, we need an efficient algorithm for locating the triangle that contains a new site. We will give an algorithm for this point-location problem that contributes an overall randomized time complexity of $O(n \log n)$, resulting in an overall randomized time complexity of $O(n \log n)$. This bound is optimal, since sorting is $O(n)$ reducible to computing the Delaunay triangulation, and sorting is known to have a lower bound of $\Omega(n \log n)$ in the randomized time complexity model.

The trick to achieving an efficient algorithm for the point-location problem is to keep a history of the construction of the triangulation. We can think of this history as consisting of a stack of paper triangles glued on top of one another.

Given a history of the construction, the algorithm for finding the triangle in the current triangulation containing a query point $P$ is as follows. Assume we know that the triangle $T$ contains point $P$ at some step in the construction of the triangulation. If $T$ is present in the final triangulation, we are done. Otherwise, $T$ will be split in one of two ways: either a new point of the triangulation is added into the triangle $T$, in which case $T$ will be split into three children, or one of the edges of $T$ gets flipped, in which case $T$ will be split into two children (see Figure 38). In both cases, it takes only constant time to determine which "child" of $T$ contains the query point $P$.

In order to prove that the overall running time of the point-location part of the algorithm for the incremental construction of the Delaunay triangulation is $O(n \log n)$ on the average, we need to obtain a bound on the number of triangles examined during each of the point-locations. In the "stack of paper triangles metaphor", we need to estimate the average thickness of the stack of paper triangles that are glued on top of one another at each site.

**Lemma 35.**

$$
\sum_{i=1}^{n} \mathcal{E}(\#\text{ of triangles crossed during the } i\text{-th insertion}) = O(n \log n)
$$

Note that this lemma does *not* establish that any specific point-location query can be carried out in expected time $O(\log n)$ (why?).

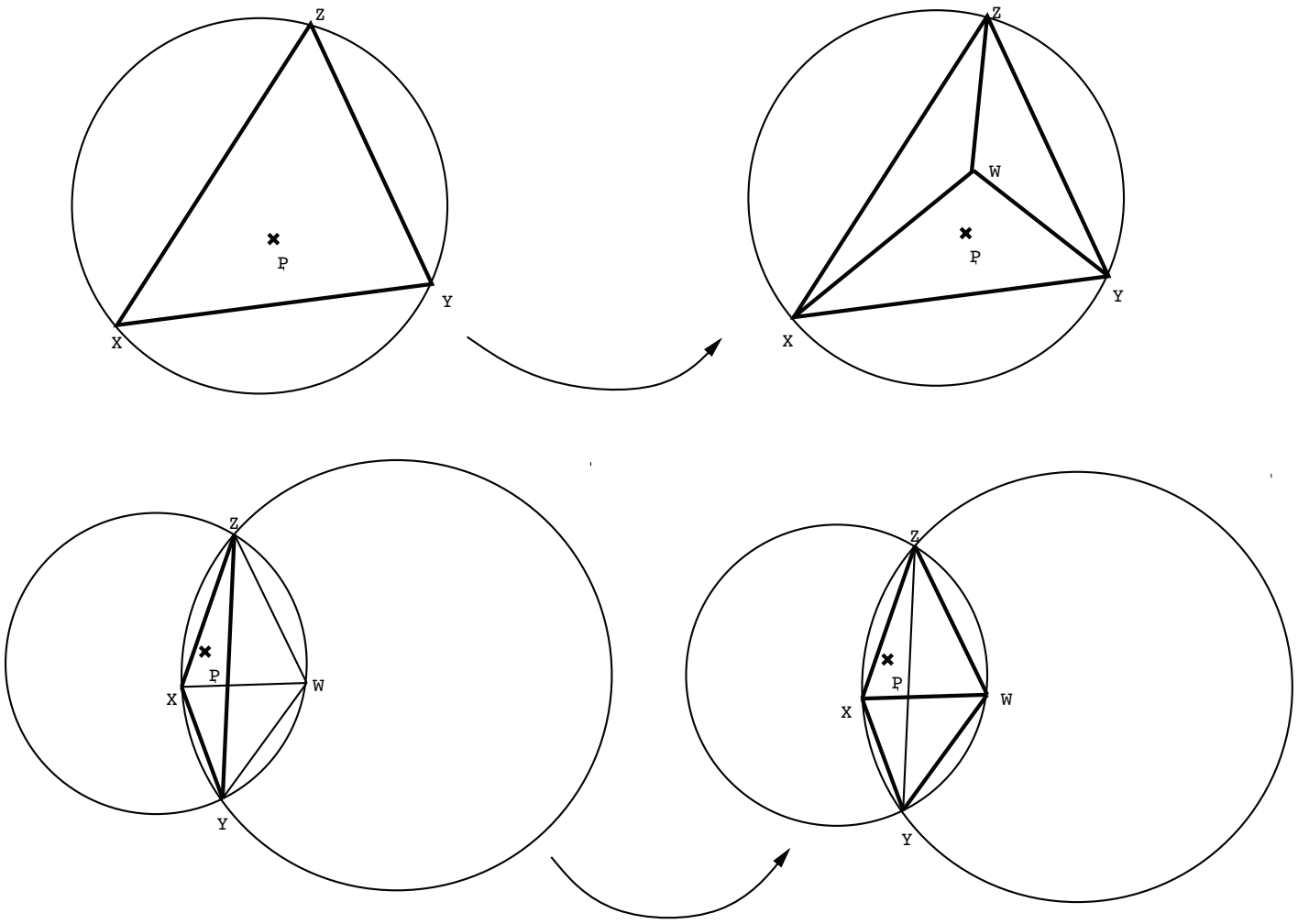


Figure 38: The different ways in which a Delaunay triangle can get split during the incremental construction of the Delaunay triangulation.

**Proof:** We will carry out an amortized analysis. We will relate the number of triangles crossed during the search to the number $T_j$ of triangles of scope $j$ (remember that the scope of a triangle is the number of sites contained in the circumcircle of that triangle).

Assume that we are trying to locate the triangle containing point $P$ and that we have already localized point $P$ in the triangle $XYZ$. If $XYZ$ is part of the final Delaunay triangulation, we can ignore it in the count (there are only $O(n)$ such triangles). (Please refer to Figure 38 for a rough sketch of the geometry of the situation.)

Consider now the point at which triangle $XYZ$ gets destroyed by the insertion of a site $W$ during the incremental construction of the Delaunay triangulation. There are two possible ways in which this can happen. Either $W$ gets inserted into the interior of the triangle $XYZ$, or the triangle $XYZ$ gets destroyed because one of the edges of the triangle flips during the construction. In the first case, we know that $XYZ$ must have been Delaunay before the insertion of the site $W$ into its interior, and $P$ is obviously contained in its circumcircle, so we simply charge the scope of $XYZ$ with it.

In the second case, let $XW$ be the edge that replaces $YZ$. Now, depending on which way the "wave of suspect edges" has arrived at the edge $YZ$, either $XYZ$ or $WYZ$ was Delaunay before the flip. If $XYZ$ was Delaunay, we can again simply charge $P$ to its scope. If $WYZ$ was Delaunay before the flip, the evidence against its Delaunayhood must have come from site $X$. But if $X$ is contained in the circumcircle of $WYZ$, so is $P$, and we can charge $P$ against the scope of the Delaunay triangle $WYZ$.

Therefore, we can charge every crossing to an event of the form that $P$ is contained in the circumcircle of a triangle that was Delaunay at some point during the construction. And, if $P$ charges the circumcircle of some triangle, it is easy to see that it can do so only once.

Therefore, we obtain:

$$\sum_{i=1}^{n} \mathcal{E}(\#\text{ of triangles crossed during the } i^{\text{th}} \text{ insertion})$$

$$= \mathcal{E}(\sum \text{scopes of all Delaunay triangles that ever arose})$$

$$= \mathcal{E}(\sum_{j=0}^{n-3} \frac{j\, T_j}{(j+1)(j+2)(j+3)})$$

This last expression is easily seen to be $O(n \log n)$—we use the same technique we used for obtaining the $O(n)$ bound on the number of triangles formed during the construction. Roughly, the above expression is of the form $\sum^n \frac{T_j}{j^2}$; if we consider the number $T_{\leq j}$ of triangles of scope less than or equal to $j$, this sum can be rewritten to be roughly of the form $\sum^n \frac{T_{\leq j}}{j^3}$. Using our previous results for $T_{\leq j}$, this expression takes on the form $\sum^n \frac{nj^2}{j^3} = n \sum^n \frac{1}{j} = O(n \log n)$. □

With this lemma, and the above observation about the cost of locating subtriangles, we can therefore state the following theorem.

**Theorem 36.** *The expected overall cost of point-location for the randomized incremental construction of the Delaunay triangulation is $O(n \log n)$.*

## Convex Hulls and Delaunay Triangulations

In the previous lecture we introduced a correspondence between the sites of 2-space to points on a paraboloid in 3-dimensions such that the mapping satisfies some special properties. Recall that we project a point in the plane to a point on the hyperbolic paraboloid $z = x^2 + y^2$ by mapping (see figure 39):

$$\lambda : (x, y) \longmapsto (x, y, x^2 + y^2)$$

Points $A, B, C$ and $D$ are mapped to points $A', B', C'$ and $D'$ on the parabolic surface. We proved that points $ABCD$ are cocircular if and only if points $A'B'C'D'$ are coplanar.

This implies that every circle in the plane maps to a planar curve on the paraboloid (the intersection of some plane with the paraboloid).

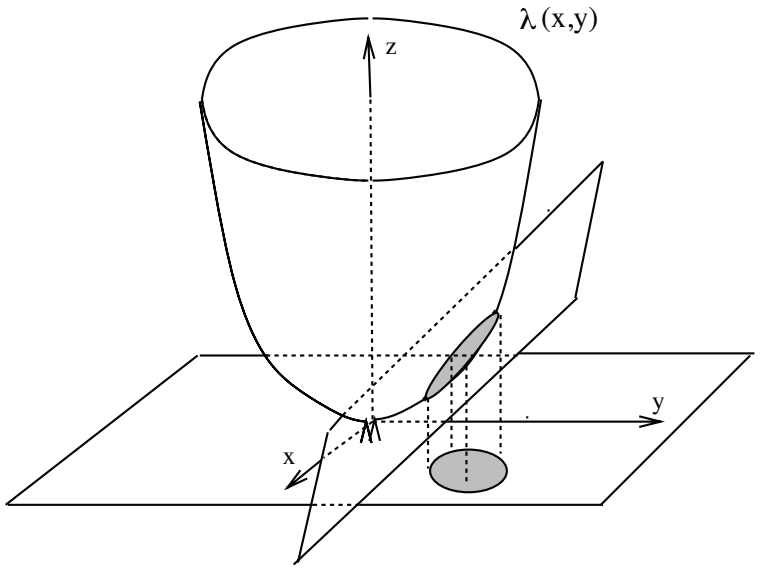


Figure 39: The projection of cocircular points on the parabola

Our construction has another nice property. Suppose we are given a triangle formed by three sites in the plane, which maps to three points on the paraboloid. From our construction, it easily follows that a fourth site lies inside the circle formed by the three sites in the plane iff the mapped point lies below the plane formed by the three points in 3-space. This means that Delaunay triangles in the plane (whose circumcircles are empty) map to triangles in 3-space which don't have any point *below* them. In other words, a Delaunay triangle maps to a lower convex hull triangle of the mapped set of points in 3-space.

Hence, the problem of finding the Delaunay Triangulation of a set of points reduces to that of finding the lower convex hull of a set of points on a paraboloid in three dimensions. It turns out that this correspondence extends to arbitrary dimensions. That is, the problem of finding the Delaunay diagram of $n$ points in $d$ dimensions corresponds to the problem of finding the convex hull of $n$ points in $d+1$ dimensions. We do not prove this claim here, although it is very straightforward.

## Applications

Computing the Voronoi diagram and Delaunay triangulation is useful not only for its own sake, but also is an important part of many other efficient geometric algorithms.

For example, the edge connecting the closest pair of sites in a collection of sites is a Delaunay edge. Finding the closest pair is therefore reducible to finding the minimum length Delaunay edge.

Finding all nearest neighbors is similarly easily accomplished starting with the Delaunay triangulation, because all edges connecting a site and its nearest neighbor must be Delaunay.

As a final example consider the computation of the Euclidean Minimum Spanning Tree

(EMST). The EMST must be a subset of the Delaunay triangulation (considered as a graph), and can be obtained from it by application of a minimum spanning tree (MST) graph algorithm. In fact, all the known EMST algorithms with good complexity compute the Delaunay triangulation first.

**Acknowledgements:** The author wishes to thank the following students whose scribed notes were used in preparing this writeup: Zhenyu Li (Harvard), Michelangelo Grigni, Mark Day, Rosario Gennaro, Dina Kravets, Thomas Breuel, Songmin Kim, Alex Ishii, Mark Smith (MIT), Alan Hu, Eric Rose, Eric Veach, David Karger, Dan Yang, G.D. Ramkumar (Stanford), and Ken Stanley who scribed a lecture of Raimund Seidel (Berkeley).

# 9 Bibliography

# References

[ASS90] Pankaj Agarwal, Micha Sharir, and Peter Shor. Sharp upper and lower bounds on the length of general Davenport–Schinzel sequences. *Journal of Combinatorial Theory Series A*, 52:228–274, 1990.

Shows $\lambda_4(n) = \Theta(n \cdot 2^{\alpha(n)})$, as well as near-tight bounds for higher $s$: $\lambda_s(n) = n2^{(\alpha(n))^{\lfloor (s-2)/2 \rfloor + o(1)}}$ where the $o(1)$ goes to zero with large $n$ (but not $s$).

[CGL85] Bernard Chazelle, Leonidas Guibas, and D. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

As an application of geometric duality, this paper gives an optimal $O(\log n + k)$ algorithm for the problem of half-planar queries in the plane. Here $n$ is the total number of points, and $k$ the size of the desired answer. Unfortunately the methods of this paper seem to work only in two dimensions.

[Dav71] H. Davenport. A combinatorial problem connected with differential equations II. *Acta Arithmetica*, 17:363–372, 1971.

Shows $(8 - o(1))n \leq \lambda_3(n) \leq O(n \log n / \log\log n)$.

[Del34] B. Delaunay. Sur la sphère vide. *Izvestia Akademii Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(6):793–800, October 1934. English translation manuscript by M. Dillencourt.

[DS65] H. Davenport and A. Schinzel. A combinatorial problem connected with differential equations. *American Journal of Mathematics*, 87:684–694, 1965.

Source of the Davenport–Schinzel sequence problem. Solves the $s \leq 2$ cases, shows $\lambda_s(n) \leq (s+1)n(n-1)+1$, $5n - O(1) \leq \lambda_3(n) \leq 2n(1 + \ln n)$, and $\lambda_s(n) = n2^{O(\sqrt{s \log s \log n})}$.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, 1987.

An advanced topical reference work.

[EG89] Herbert Edelsbrunner and Leonidas Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38:165–194, 1989.

This paper shows how a "topological line" can be used to sweep over a line arrangement in $O(n^2)$ time with $O(n)$ working storage.

[GKS90] Leonidas Guibas, Donald Knuth, and Micha Sharir. Randomized incremental construction of delaunay and voronoi diagrams. In M. S. Paterson, editor, *Automata, Languages and Programming: 17$^{th}$ International Colloquium*, volume 443 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990. To appear in Algorithmica.

This paper shows that incremental techniques work very well for the construction of Voronoi/Delaunay diagrams, if one first randomizes the sequence of insertions of the defining points. The paper also shows how the history of the data-structure thus constructed can be kept around so as to aid the location of future points in the structure.

[GS85] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi diagrams. *ACM Transactions on Graphics*, 4(2):74–123, April 1985.

This paper is really about the *quad-edge* data structure for representing subdivisions of two-dimensional manifolds. The authors give a mathematically rigorous development of the data structure and prove its "topological completeness." As an afterthought, they also show how to use it to implement efficient Voronoi/Delaunay algorithms in the plane using only a few geometric and topological primitives.

[GS87] Leonidas Guibas and Jorge Stolfi. Ruler, compass, and computer: the design and analysis of geometric algorithms. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, pages 111–165. Springer-Verlag, 1987.

This is a survey paper trying to illustrate each of the major techniques in Computational Geometry as of 1987 with a worked out example.

[HS86] S. Hart and Micha Sharir. Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6:151–177, 1986.

Shows $\lambda_3(n) = \Theta(n\alpha(n))$.

[Meh84] Kurt Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1984.

One of the first texts on computational geometry, the third of three volumes by Mehlhorn.

[PS84] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry*. Texts and Monographs in Computer Science. Springer-Verlag, 1984.

A good elementary introduction.

[Sha87] Micha Sharir. Almost linear upper bounds on the length of general Davenport–Schinzel sequences. *Combinatorica*, 7:131–143, 1987.

Shows $\lambda_s(n) = O(n\alpha(n)^{O(\alpha(n)^{s-3})})$.

[Sha88] Micha Sharir. Improved lower bounds on the length of Davenport–Schinzel sequences. *Combinatorica*, 8:117–124, 1988.

Shows $\lambda_{2s+1}(n) = \Omega(n\alpha^s(n))$, by an inductive construction based on that of [HS86].

[Sze74] Endre Szemerédi. On a problem of Davenport and Schinzel. *Acta Arithmetica*, 25:213–224, 1974.

Shows $\lambda_s(n) \leq C_s n \log^* n$, where $C_s$ depends only on $s$. Difficult.

[vL90] Jan van Leeuwen, editor. *Handbook of Theoretical Computer Science*. Elsevier and MIT Press, 1990.

[Yao90] F. Frances Yao. Computational geometry. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, chapter 7, pages 343–389. Elsevier and MIT Press, 1990.

An excellent recent survey of the field, including an extensive bibliography.