Computational Geometry

Prof. Sandeep Sen

Department of Computer Science and Engineering

Indian Institute of Technology, Delhi

Module No. # 07

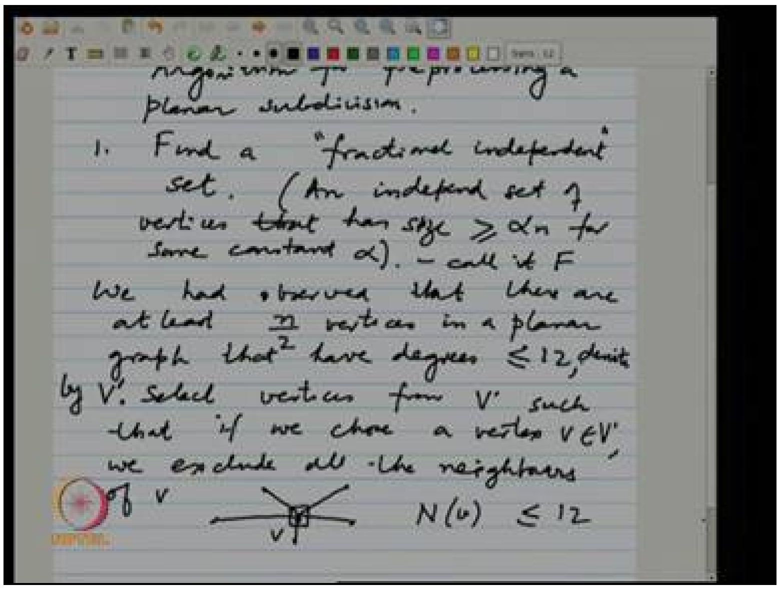
Planer Point Location

Lecture No. # 02

Point Location and Triangulation (Contd.)

We will continue today's lecture with point location algorithm that we started with last time. To recap - the idea that we were pursuing is that you start with a planar sub-division, and we identified the following problem that how do you find out a large independent set. Let me just write the algorithm again.

(Refer Slide Time: 01:13)



Algorithm for preprocessing a planar subdivision.

1. Find a "fractional independent" set. (An independent set of vertices ~~that~~ has size $\geq \alpha n$ for some constant α). - call it F

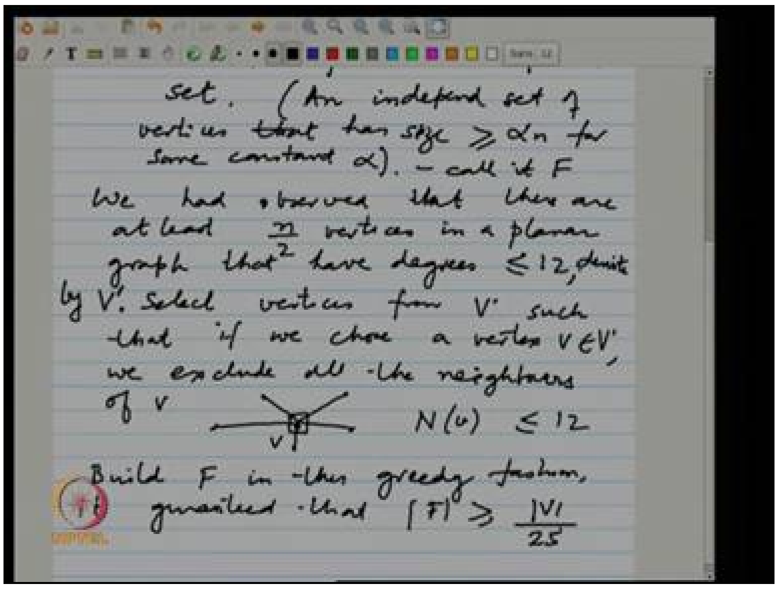
We had observed that there are at least $\frac{n}{2}$ vertices in a planar graph that have degrees ≤ 12, denote by V'. Select vertices from V' such that if we chose a vertex $v \in V'$, we exclude all the neighbours of v

Algorithm for preprocessing a planar subdivision; one - find a fractional independent set; this word fractional independent set means that an independent set of vertices that has size greater than equal to, let us say alpha n for some constant alpha. How do we find this thing? We noticed that we had observed that we had observed that there are at least n over 2 vertices in a planar graph; that have degrees less than or equal to 12 - that followed from the Euler's formula, right?

We have a fairly large number of vertices which are bounded degree - bounded by this number 12; what we do is, we just find (( )) select vertices from find a fractional dependent set call it F - I am calling this as F, I am denoting this as F; the set of vertices that (( )).

You want to construct this set F and we had observed that there are that least n over 2 vertices in a planar graph that have degrees less than equal to 12 - denoted by say - sorry denote by now, let us say, V prime; select vertices from V prime. Such that if we choose a vertex V in V prime we exclude all the neighbors; we have some vertex V, we have a number of neighbors of this vertex, but we know that there are no more than 12 neighbors.

(Refer Slide Time: 04:15)



set. (An independ set of vertices ~~that~~ has size $\geq \alpha n$ for some constant α). - call it F

We had observed that there are at least $\frac{n}{2}$ vertices in a planar graph that have degrees ≤ 12, denote by V'. Select vertices from V' such that if we chose a vertex $v \in V'$, we exclude all the neighbours of v

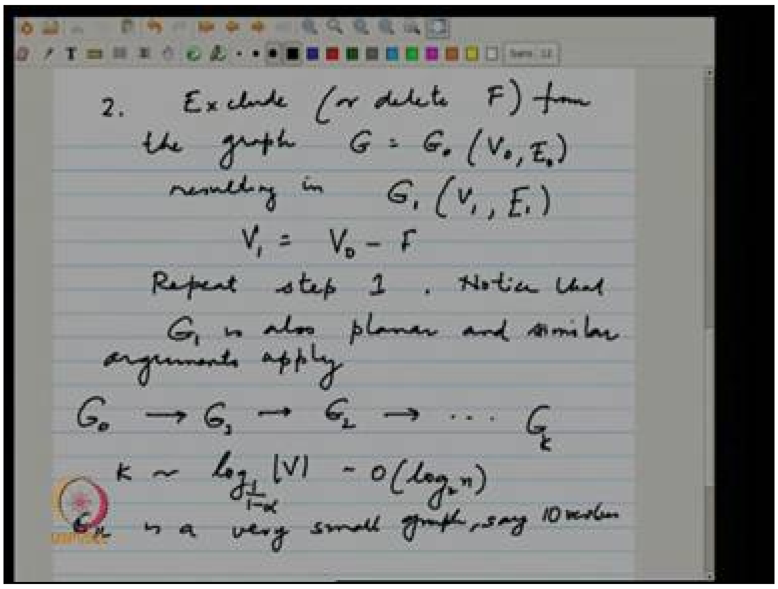
$N(v) \leq 12$

Build F in this greedy fashion, guaranteed that $|F| \geq \frac{|V|}{25}$

No more than 12 neighbors - the neighborhood of V is less than 12 - and if we choose V, if we select V in the fractional independent set - if we choose a vertex V to V prime - we exclude all the neighbors of v which means that you know any other neighbor cannot be in the independent set, by definition - that is what we do; if we just do this in a greedy fashion - you start with an arbitrary vertex which has degree less than 12 and continue this process keep adding this to F. If you build F in this greedy fashion, it is guaranteed that size of F will be greater than equal to… If we choose a vertex we are certainly…we are excluding at most 11 12 vertices, and we are working on half the number of vertices anyway.

This means about… Just to be safe I am just saying that this is, let us say V over 25; alpha is 1 over 25 for sure - some fraction; it is actually a very small fraction and that will show up in the final constants; if you can get a better bound by somehow with a more tight argument may be this 25 can improve to twenty or something like that, but you cannot improve it too much, because that is a fairly naive and simple approach of computing the fractional independent set.

(Refer Slide Time: 07:17)



2. Exclude (or delete F) from the graph $G = G_0(V_0, E_0)$ resulting in $G_1(V_1, E_1)$

$V_1 = V_0 - F$

Repeat step 1. Notice that G_1 is also planar and similar arguments apply

$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \cdots G_k$

$k \sim \log_{\frac{1}{1-\alpha}} |V| - O(\log_2 n)$

G_k is a very small graph, say 10 vertices

The next step is… So, after having done this we… Step 2 is exclude or delete F from the graph; initially, let us call the graph G equal to G 0, let us say, V 0, F 0; exclude these vertices from the graph resulting in a modified graph - call it G 1 of V 1 F 1, where V 1 is basically the initial set of vertices minus (( )) not not sorry sorry we should call V 0 - V 0 V 1, V 0 minus F. We have thrown away those vertices; exclude this and then repeat step one that is, we again find a fractional independent set; notice that G 1 is also a planar graph - you are only throwing away vertices and edges.

So, if a graph was planar initially it must remain planar also planar and similar arguments apply; that is, we can again pick a fractional set using the same argument and the same procedure and you keep repeating this. What is happening? We started with some graph G 0; then we threw away some vertices resulting in graph G 1; then we threw away some more vertices resulting in graph G 2 and we continue how long can we continue this process?

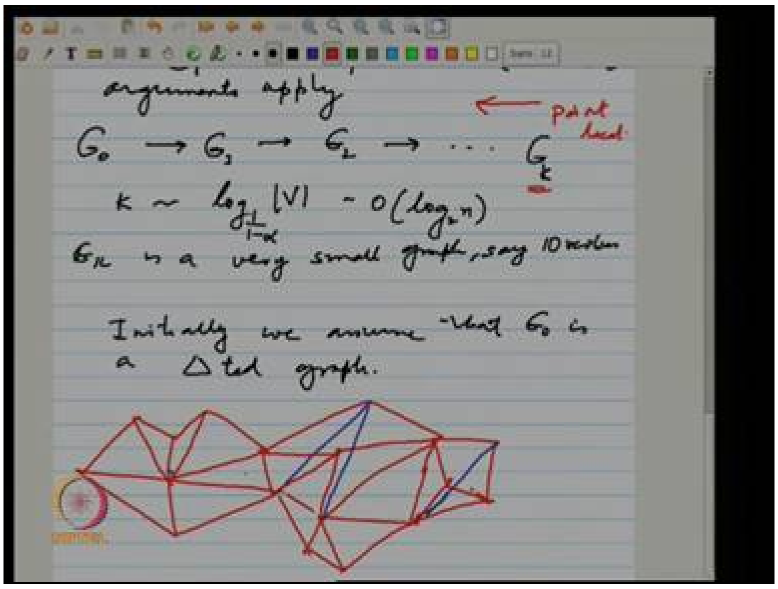
Every time we are throwing in, throwing away a constant fraction of the vertices, right? So, how many times can we repeat this process? Roughly, logarithmic - log to the base of…it is a very small number; we are only throwing away point one by twenty fifth fraction that is guaranteed to be thrown away; it will be log to the base of… Let us call it some G of k, where k is roughly log to the base - no, not log to the base (( )) it has to be very small number.

Alpha fraction here.

One over one minus alpha - whatever that number is; this is a very small number, because alpha is a point nine; alpha is point…whatever…point nine or something so it will be - no no sorry 0.4 or something; but still it is… Which is order, we can still use big O notation saying there is an order log N; in logarithmic number of phases, we are done with this process; eventually, the graph becomes small enough or essentially… graph, let us say, G k is a very small sized graph.

We can stop, let us say, when there are ten vertices or some such thing - you can say ten vertices, you can stop it at that point; because once your problem size becomes constant you can apply any kind of brute force method to solve the problem - we do not go any further. G naught to G 1 to G 2 all the way to G k - these are graphs that we obtain by eliminating some vertices from the previous graph; this is clearly some kind of these are sub graphs of the previous graph; but what is it that we are trying to solve? We are trying to solve - build a data structure for the point location problem.

(Refer Slide Time: 07:17)


arguments apply

$$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \cdots G_k$$

← point head

$k \sim \log_{\frac{1}{1-\alpha}} |V| - O(\log_2 n)$

G_k is a very small graph, say 10 vertices

Initially we assume that G_0 is a Δted graph.

Initially, we assume that G naught is a triangulated graph; if G naught is not triangulated we will triangulate it - now, we do not know at this point how to triangulate quickly; but you know that all planar graphs are can be triangulated - planar sub divisions; because each sub division is a simple polygon and a simple polygon can be triangulated - that is something that we kind of proved of proved in the first couple of lectures, but we did not quite go through the analysis carefully and did not seem like a very efficient algorithm, but certainly it can be triangulated.

Each sub division, which is a simple polygon can be triangulated; we can assume that either G naught was triangulated or if necessary we will triangulate it; when we throw away some vertices. What happens is suppose there was some triangulated graph and we eliminated, let us say, this vertex; if we eliminate a vertex we are not going to eliminate any of the neighbors that is the definition of the independent set; we could have eliminated this and we could have eliminated where this looks like a higher degree so we are not going to eliminate that, but something else may be… Let me (( )) diagram it will be the…

Suppose we eliminate - we had eliminated this, once we eliminate this, this vertex is gone; now once that vertex disappears I would like to again re-triangulate the graph; how do you re-triangulate? We are going to you can re-triangulate in many ways and at this point actually it does not matter what algorithm we are using to re-triangulate, because the vertex that we threw out has what kind of degree? Degree less than 12…some kind of constant degree; even if we use the most inefficient algorithm for triangulation it does not matter because we are dealing with a simple polygon of size at most 12. So, we can re triangulate using any method that you want practically, if we take some constant time.

Suppose, this was re-triangulation like this, you can probably see the significance as to why we did not choose neighboring vertices; because that made this re-triangulation process much easier; if two vertices - neighboring vertices - where to disappear then I could not invoke such a simple triangulation - re-triangulation - mechanism, right? Once a vertex is eliminated we have to deal only with a simple polygonal side at most well; this would not have been the case if we had 2 adjacent vertices to deal with.

Now, what was the original triangulation? We had a vertex somewhere here, right? Let me throw it back.

But whereas, the independent set has about alpha and multiple sign …

Yeah…

Each of the triangulation will become a constant (( ))

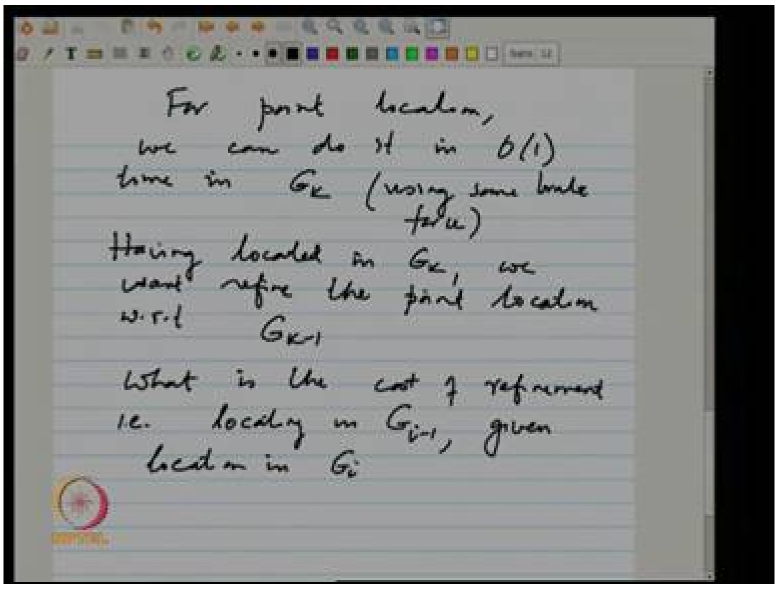
Right…

But alpha times N of the...

Each triangle – sorry each triangle is constant amount of work, therefore the total work that will be doing is linear in the number of vertices; yes, in iteration between; I will do the analysis, but for a single vertex that disappears we are only spending constant amount of time to re-triangulate; If the constant fraction vertices are disappearing then we are spending linear amount of time - linear in the number of vertices - that is important; the blue ones - the blue triangles are the new triangles red ones are the old triangles.

How does the actual point location work? The point location actually works in the reverse direction; when we are when we only have a, suppose we have inductively consider the last one - Gk, this graph only has a constant number of vertices; I can use whatever algorithm that I want to do my point location here; let us say, I have done my point location, when doing the point location I am going to actually look at this entire hierarchy in the reverse fashion for the point location.

(Refer Slide Time: 18:06)



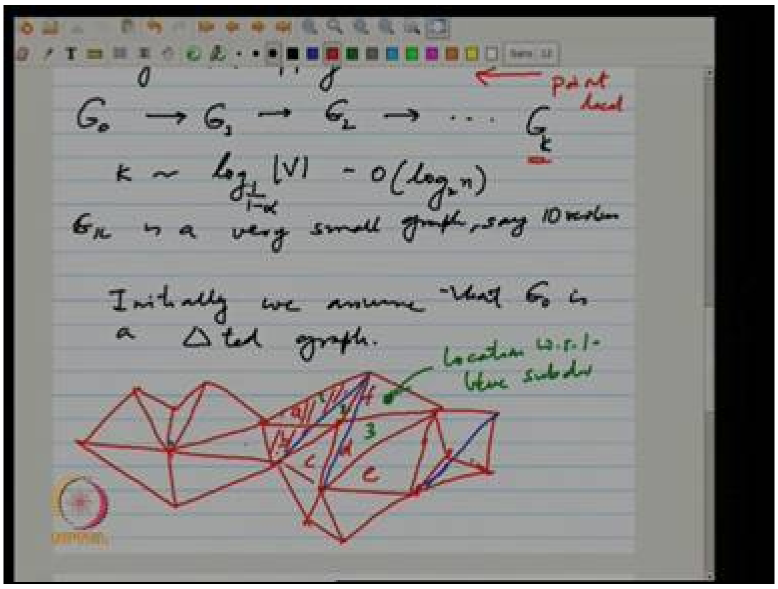
For point location, we can do it in $O(1)$ time in G_k (using some brute force)

Having located in G_k, we want refine the point location w.r.t G_{k-1}

What is the cost of refinement i.e. locating in G_{i-1}, given location in G_i

My first step is for point location, we can do it in, let us say, some constant time in G sub k using whatever algorithm using some brute force; having located in G k we want to refine the point location with respect to G k minus one. Then having located the point in the sub division of G k minus one we want to proceed backwards and do the point location - refine the point location - in G k minus 2 and eventually we are going to do the point location all the way to G naught which was a original planar sub division, and then we will be done; what is the cost of refinement? Cost of refinement i.e., locating in G i minus one given location in G i right.

(Refer Slide Time: 19:39)



$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \cdots G_k$ ← Point loc

$k \sim \log_{\frac{1}{1-\alpha}} |V| = O(\log_2 n)$

G_k is a very small graph, say 10 vertices

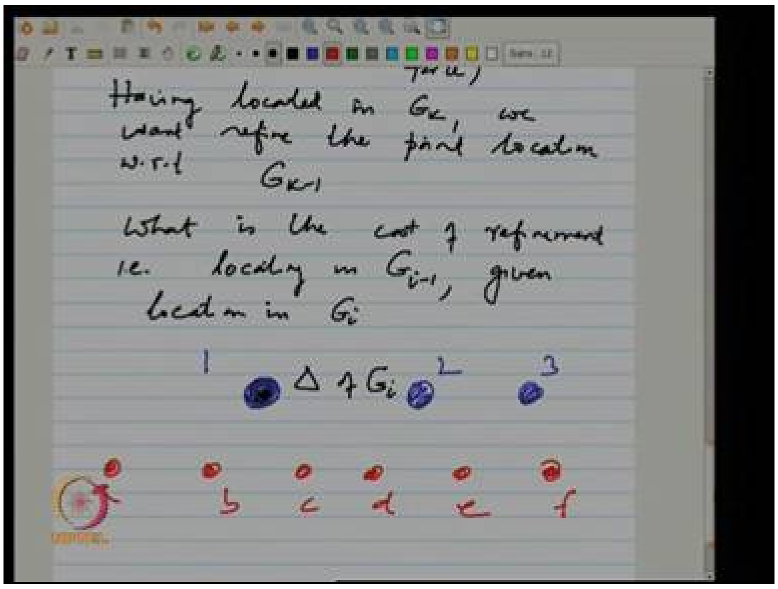
Initially we assume that G_0 is a △ted graph.

Suppose, I have this point - this is my point - and I know that the point is located the blue ones basically means that it is the next sub division that I have obtained by throwing away vertices; when I am doing the point location I am doing it in the reverse direction - that is, I know the point with respect to location with respect to blue - with respect to the blue triangle, blue lines, blue sub division. Let us say; I know that, there are these 1, 2, 3 triangles with respect to the blue sub division; and I now want to somehow know what is the point location when we consider the red - the triangulation defined by the red edges; how do we do that? Yes, tree structure, but what is the easy way to justify whether it can be done easily.

By means one triangle (( )) we find triangulation what are the triangle that is intersecting the

Exactly, yes; we consider these 1, 2, 3 triangles - these are the triangles of the blue triangulation; then, I can also number the triangles of the red triangulation - let me call it use - a b c d e f; if you - sorry sorry - what is it? I have missed one edge of the red triangle; you see, there are these a, b, c, d, e, f in the red triangulation, and there are these 1, 2, 3 triangles in the blue triangulation; all we need to know is how these triangles interact with each other right. Let us say triangle one intersects triangle a of the red triangulation and triangle b of the red triangulation; I am just talking about this particular portion - this is a blue triangle; this blue triangle intersects a and b.

(Refer Slide Time: 22:16)

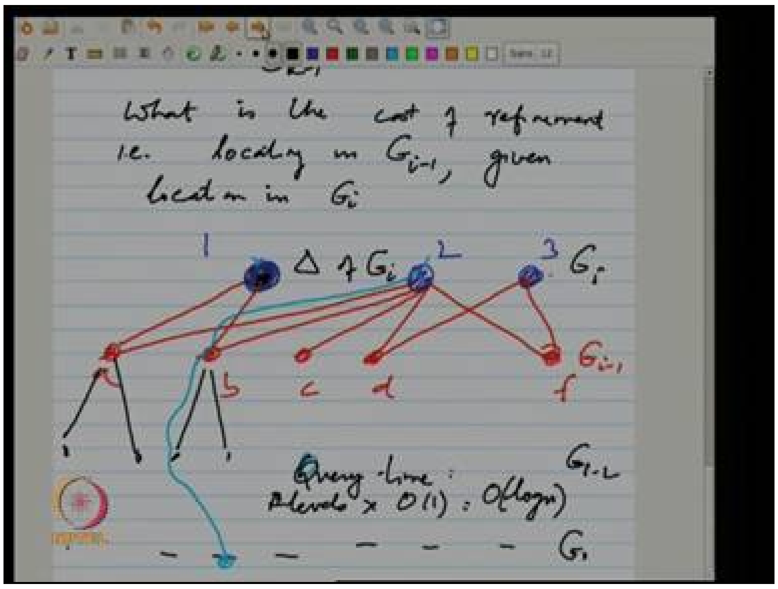


Having located in G_k, we want refine the point location w.r.t G_{k-1}

What is the cost of refinement i.e. locating in G_{i-1}, given location in G_i

If I can somehow store this information, that here is, let us say, a triangle of G i - this is a triangle of G I - this is blue, so let us use blue; this is my - sorry - this is my…it is a triangle of the blue triangulation and there are a number of triangles in red triangulation; here my 1, 2, 3 the three of these triangles - I am not drawing the entire picture, only for that particular sub division. And there are these a b c d e f; so 2, 3, 4, 5, 6, a b c d e f - these are the these nodes or whatever represent the six triangles of the red triangulation of that particular sub division; what we noticed is that triangle 1 - the blue triangle - intersects a and b, so I will just draw an edge between 1 and a and b; likewise, 2 intersects a b c d - all of them, all of them except e.

(Refer Slide Time: 22:16)


What is the cost of refinement i.e. locating in G_{i-1}, given location in G_i

2 intersects b c d f and 3 intersects f and d, no, e is not a part of that - e was never a part of that; I should probably have never drawn e actually in the first place – anyway, does not matter - d and f, right. This is spurious triangle - it is not a triangle that has come from that particular sub division; we have this kind of a structure and if we know that the point appears in the triangle 3 of G I, the point is in triangle 3 of G i - the only possibilities are d and f.

I will just again using brute force I can check whether it intersects d or f; in fact, whatever the case may be, none of these nodes in level, I will intersect more than… not intersect. I mean we do not have more than, what is the degree of each node basically?

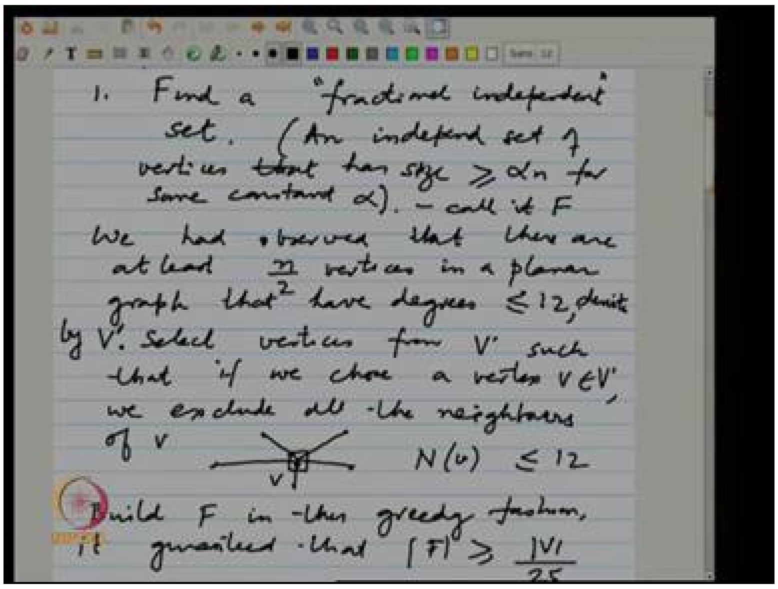
12.

12 at most, right; because you know there can be at most…that is the good thing about the degree bound, because I when we throw out a vertex - this vertex - we created a maximum of - whatever was the degree of vertex; if this sense the degree was 12, we cannot have a situation where a blue node will have more than 12 neighbors in the level of G i minus one - this is true; when you complete this graph, basically or this is actually a layered graph structure so this corresponds to G I, then G i minus one and then likewise there will be some triangles corresponding to G i minus two, etcetera - eventually all the way up to G naught; a point location will just proceed starting from somewhere along this layered graph till we hit a triangle in the original planar sub division.

Every time we descend by one level, what is that kind of extra operations of work that we have to do? We have to only check with respect to… If we are here, we have to check all the connected neighbors of that node and there are at most 12 neighbors of that node; each of these polygons is a triangle, so if you can find - figure out, well, whether it is a triangle or rectangle it is constant number of sides.

In constant time, we can determine whether the point is inside the triangle or outside the triangle; we are spending constant time to refine our search from level G i to G i minus one and we already noticed that the total number of levels is order log n; that immediately gives us a query time of order log n - point location query time is order log n; the order log n. We will do that, but construction time it will be counted separately. We parameterized three things: query time, preprocessing time and the space; the query is number of levels times order one and since the number of levels is order log n, the whole thing is order log n.

(Refer Slide Time: 28:14)



1. Find a "fractional independent" set. (An independent set of vertices ~~that~~ has size $\geq \alpha n$ for some constant α). - call it F

We had observed that there are at least $\frac{n}{2}$ vertices in a planar graph that have degrees ≤ 12, denote by V'. Select vertices from V' such that if we chose a vertex $v \in V'$, we exclude all the neighbours of v

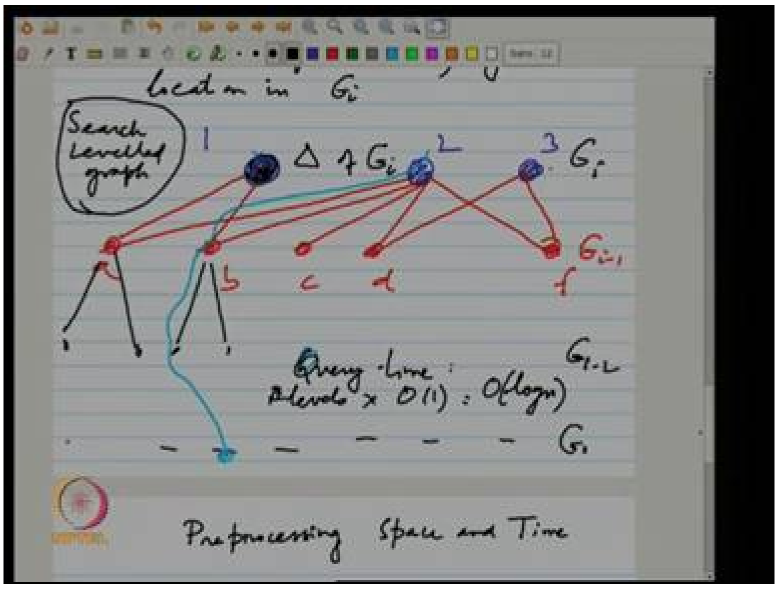
$N(v) \leq 12$

Build F in this greedy fashion, it guaranteed that $|F| \geq \frac{|V|}{25}$

Next, let us look at what is the preprocessing time and space; what are we doing? We are… What is the algorithm? All you, if you do not count the cost for the initial triangulation - we leave that aside and we will take it up when we actually look at triangulation - the problem of triangulation. Suppose, initially the graph was triangulated, after that what is the kind of work that we are doing? We are essentially identifying a fractional independent set; how are we doing it? We are finding out, I am going to every vertex and basically just see whether it has a degree less than 12 or not - linear time.

Then, when I find the fractional independent set, I do it greedily; again, I am dealing with only the vertices of degree less than equal to 12; I can run this entire greedy thing in time proportional to the number of vertices; at every level. And then of course, I have to also make sure that I construct this search well, this is not quite a tree it is a level graph.

(Refer Slide Time: 29:08)



We should not call it a search tree, it is a search leveled graph; the edges go between consecutive levels only; so, we also have, we must also construct this search graph when we are doing the preprocessing, but then again, when you take out a vertex it is only creating a sub division of size 12. The new triangles and the old triangles they are all bounded by constant; again, even if you use brute force - just to find out which a nodes interacts with which nodes in the previous level - that is again a constant amount of time; because you are always dealing with some constant, that is why we choose this constant degree vertices.

Just to recap, see in this red triangulation - what are you doing? We are creating some new triangles and there are some old triangles; we are finding out the interaction between these sets of two triangles - these two sets of triangles; but there are constant number of triangles that we are dealing with - the number of red triangles is no more than 12, the number of blue triangles is no more than 12, therefore we are done.

Sir (( )) a triangle can (( )) multiple levels right and. So, on and why do you are saying that the edges should be unlimited (( )) there can be from that 2 levels (( )).

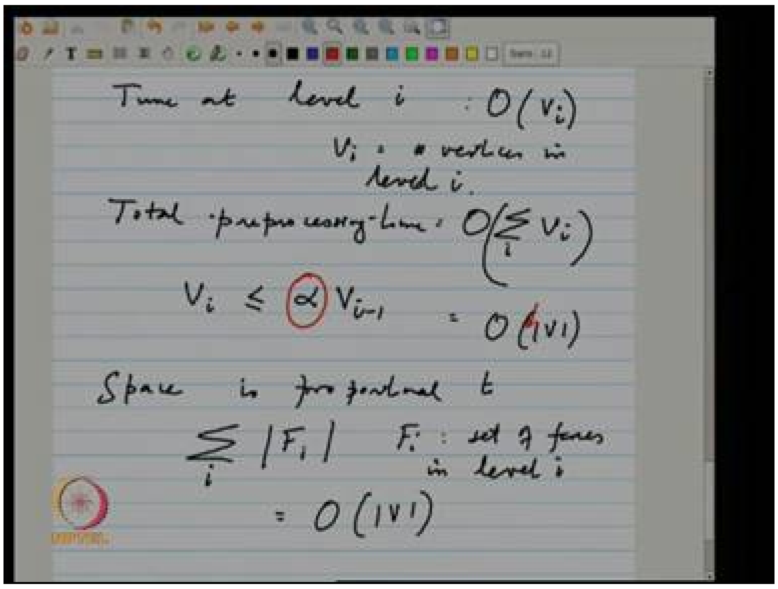
Yes, what you are saying is correct, but conceptually we can view all the triangles that, let us say, survive to actually move up.

Sir, we can duplicate.

Yes, we can duplicate.

What you are saying is correct; some triangles can survive, but you know just to simplify conceptually we consider a triangle that has survived, you now make a copy of that triangle to the next level; we do not lose anything asymptotically; what the force when you are fairly implementing it you may not want to do that; I will be with you so about point. Constructing this search graph as you go on eliminating the vertices will also take us time proportional to the number of vertices in that particular level.

(Refer Slide Time: 31:30)



Time at level i : $O(V_i)$

V_i : # vertices in level i.

Total preprocessing time : $O(\sum_i V_i)$

$V_i \leq \alpha V_{i-1}$ $= O(|V|)$

Space is proportional to

$\sum_i |F_i|$ F_i : set of faces in level i

$= O(|V|)$

Time at level i is basically order of V i - V i is the number of vertices in level i; because they are planar graphs the number of edges is proportional to the number of vertices and so on and so forth; therefore, total time - preprocessing time, we know that V i less than or equal to alpha times…

No, but even if you duplicate vertices what is the total number of triangles - number of phases? Number of phases is always proportional to the number of vertices in a planar graph; your total cost is always going to be proportional to the number of vertices; it will be multiplied by some constant; yes, this way, I can… In fact this actually simplifies the analysis, otherwise we will be worried about what is duplicated and what is not duplicated. At every level, the vertices are a constant fractional less in the previous level the total work that we do is proportional to the number of vertices, because edges and phases are all proportional to the number of vertices.

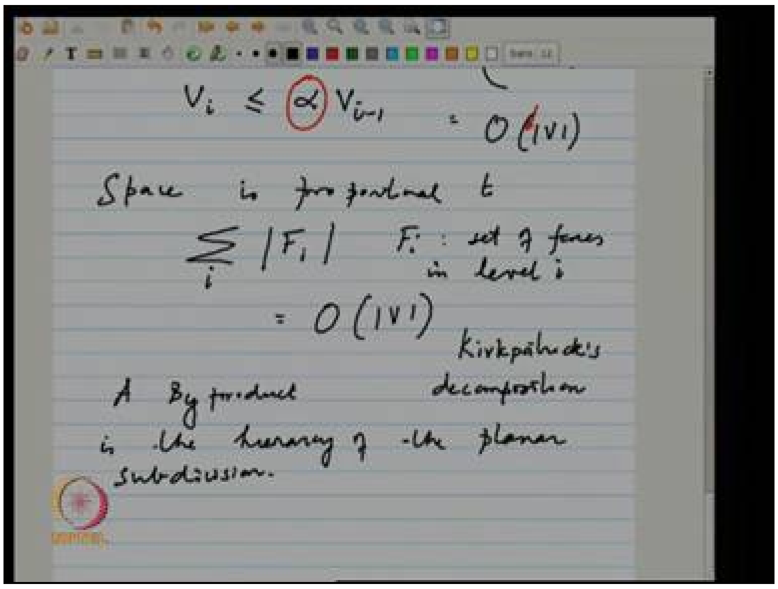
Everything basically becomes proportional to the number of vertices and since this is a constant fraction - this is just geometric series - which is basically nothing but order of… How much is the space? Each vertex of the search graph has constant degree - we do not need to count those edges; again, pre-proportional to the number of vertices in each level and again the number of vertices in each level correspond to the number of phases in each level.

Again, by these previous arguments, space is proportional to summation of number of phases in each level; number of phases in each level is again proportional to number of vertices, they will be also decreased by constant factor so, overall again it is going to be order V. The search graph is the structure, that is the data structure essentially - that is the data structure; this is the size of the data structure - this data structure is also bounded by something that is linear.

Of course, this alpha - the smaller it is the better; in the sense that, everything is hidden inside this constant subsumed by this big O notation; so, if alpha is close to one the constant here is going to be larger - it is 1 over 1 minus alpha; if alpha is smaller then the constant is also going to be smaller; if you can get a larger reduction, the constant, and the constant that will show up here is very large - alpha is like - we noticed here is very close to one point nine point nine (( )) so this constant could be like 25, 30 or something like that, which is not such a nice thing; when one has to implement this, one should use…

As I said, we do not need to actually duplicate the nodes etcetera; even this bound that we obtain - that is, one over 25 is kind of a loose bound - it can be tightened; moreover, when we actually, if you were to actually implement this in practice - which actually, by the way, people do not - the reason why this algorithm is very significant is not simply because it is it is elegant and conceptually simple, but it also leads to… it gives you a method where we have a…When we have a planar sub division or planar structure, it gives you a hierarchy of the planar structure; this is not only used from a point location, but in some other problems as well.

(Refer Slide Time: 31:30)



$$V_i \leq \boxed{\alpha} V_{i-1} \qquad = O(|V|)$$

Space is proportional to

$$\sum_i |F_i| \qquad F_i : \text{set of faces in level } i$$

$$= O(|V|)$$

Kirkpatrick's decomposition

A By product is the hierarchy of the planar subdivision.

One byproduct of this algorithm by the way, this goes under the name of Kirkpatrick's algorithm; a byproduct is the hierarchy of the planar sub division; it basically starts, if you go bottom-up and look at the G k that is the most coarse subdivision; then you get a slightly more defined sub division and so on. It basically gives you, .if you did not want, let us say in many applications - in graphics, etcetera - you may not want the details of the surface in certain applications, but you may listen and the closer you get to a surface the more details kind of show up; this kind of hierarchy is used all the time for visualization.

Especially, if you are in a moving body or moving towards a scene - as you come closer to a scene, you know you see more and more details; one way that it can be implemented basically is that you have a triangulation of the surface, and if you can store a hierarchy of this triangulations that gives you this effect that as you move closer you see more and more details of it - that is one application; another application is that - sorry - we already observed that three-dimensional polytope also have the structure of the planar graph and that is the reason why the three-dimensional polytope also has a number of phases, number of vertices and number of corner points all proportional to each other; if I pose the problem that we posed in two dimensions - that is, I give you a convex polytope and I give you a plane. And I ask you, which is the corner point of the polytope nearest to this plane? In two-dimensions we could solve this problem or at least I gave it as an exercise to solve this problem in log n time using some kind of binary search - some variation of binary search; if I give you a convex polygon and a line, you report that either the line intersects or find the point in the convex - sorry - the corner in the convex polygon which is closest to the line.

Now, the analogous problem in the three-dimension is I give you a convex polytope in three-dimension and a parametric equation of a plane and again you tell me the same thing - does this intersect or whatever - find me the nearest corner point of the polytope to this plane; that problem can be very nicely solved using this hierarchy of sub divisions; and it can actually be solved in logarithmic time; even in three-dimensions you can do it and it is because of this kind of a subdivision.

So, the Kirkpatrick's decomposition has many other applications, but the reason why it is studied is not because it is important for just two-dimensional planar point location, but it has other kinds of application; it is a very nice - it is a very conceptually simple way of generating a hierarchy of the planar sub division structure; As I said, to the contrary, because these high constants - people do not use this in practice for planar point location; what is used for planar point location is something that I will take up next and that will go hand in hand with what I postponed just a moment ago, that is, triangulation - how do you triangulate a simple polygon? In fact, we will take up a more general problem - if you do not have any questions we will move forward; any questions with regards to this?

We have done two triangulation right polypore.

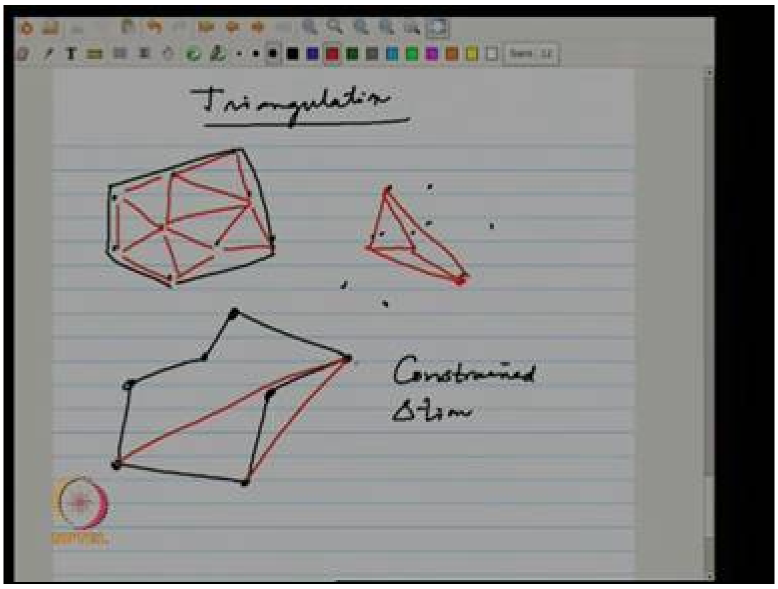
Here there is two triangulation right (( ))

No, you will triangulate. So, see convex polytope, well you generate what is called called tetra hadrons.

Yes, triangulation of higher dimensions basically is the simplex - the simplicial decomposition of that; the algorithm will be very similar to this in the sense that you know are going to generate this convex polytope, which is the original convex polytope, can now basically, throw away vertices it becomes a simpler polytope throw away more vertices becomes a simpler polytope.

Then, when you when you do this - try to find the nearest vertex - you can there is some neighborhood property that, you can you look at the simplest or let us say the bottom most in the hierarchy is a convex polytope - let us say, it has a constant number of vertices. Then, you want to…You can find by brute force the closest corner point; now you throw in some more vertices that we have eliminated in the structure then you can kind of prove that whatever is the nearest point in the present hierarchy as you throw more points in, the nearest point will be one of the neighbors of these points - that follows from the Kirkpatrick's decomposition; it is a very clever algorithm. And not only that you can also determine the closest neighbors of two convex polytopes using this method - not just a convex polytope and a plane, but I give you two convex polytopes - find me the closest pair of vertices of that; that can also be done using this method.

(Refer Slide Time: 42:32)



Let us talk little bit about triangulation; triangulation of simple polygons is what seems like a very important problem, but it is not just triangulation of simple of polygons - triangulation can occur in other contexts also; I can give you just a set of points and ask you to add edges between pairs of points such that each face becomes a triangle.

Can you always triangulate a set of points - is it always possible? We talked about triangulating a simple polygon here I am just taking about a set of points - I mean, these are points in the plane, yes; what is a quick justification of that?

Sir, maybe I can take the convex hull of the set of points and then triangulate it.

Yes sure; I should actually define it properly - what you are saying is correct, that when you are talking about triangulation of a given set of points you can well, it is not necessary, but you can think about it like. You can you construct the convex hull of the set of points, but there points inside and it is not enough to just consider the convex hull; but you can say that we can limit ourselves to triangulating this boundary - I will not bother about that this outer phase I can, sort of, ignore at this point.

Let us say that we are only interested in triangulating all the points inside the convex hull.

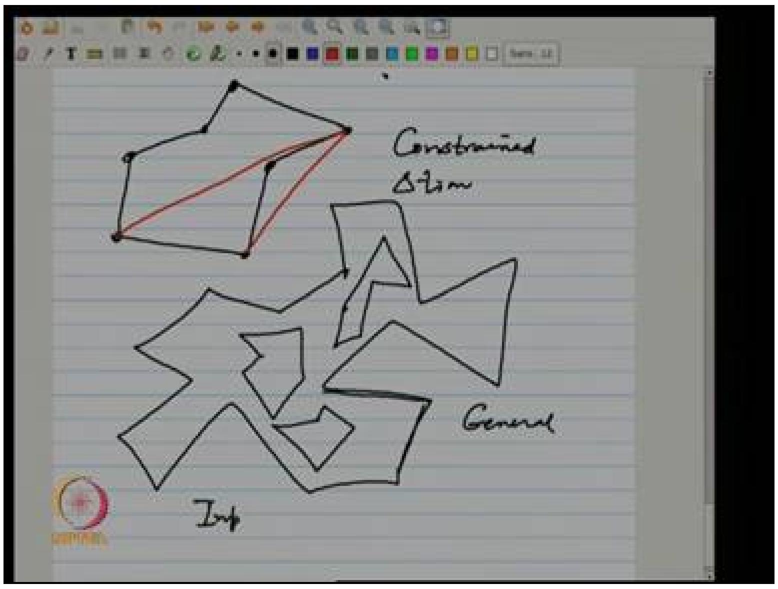
(( ))

What you are saying is the following: right now, I have a set of points; I take any 3 points - suppose, I take this point, this point, and this point, I triangulate.

I triangulate, then suppose there is a point inside (( )) there is more than one point inside; I add this - there can be more points inside; I can keep doing that and that is essentially proof that everything can be triangulated - good; but there are…one thing to notice is that triangulation is not unique - there can be many ways that you can triangulate a set of points and for certain applications, certain kind of triangulations become important; one very important kind of triangulation that we will learn very soon is called the (( )) triangulation.

But, there other kinds of triangulation that where these are all mostly used for surface modeling purposes - that is very important; we can triangulate any given set of points. Actually, the problem of triangulating a simple polygon is harder - why? Because we have to also, we are conscious that we should not cross any edges of the simple polygon; it is called a constrained triangulation; if you are just given the points of these…suppose we are only given these points, that is not the same as a triangulating a simple polygon through this. Because now I have to make sure that when I draw this triangle we should not intersect the simple polygon itself; this is called a constrained triangulation and in fact, it can be even more general where it needs not to be a simple polygon, but a polygon with holes inside.

(Refer Slide Time: 42:32)



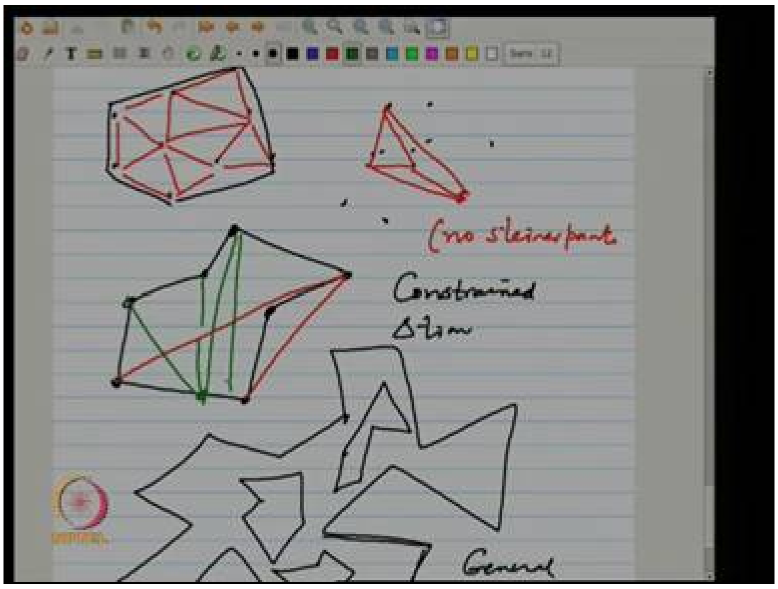
We can have a situation where you can say that this is the outer simple polygon - something - but I could also have some structure inside; again, when I triangulate I must make sure that the edges that I introduce for triangulation do not intersect the original edges. We will take up this more in general case for triangulation, and we will treat this given - whatever this polygon with holes as essentially a set of non intersecting line segments.

Sir, you said there can be many triangulations, is each triangulation the same size or I mean they are (( )). You mean in terms of number of phases? All these triangulations have fixed number of vertices. We will now look at the problem of this one - we check the more general case; input is, let us say we can think about the input as essentially a set of line segments which question?

Sir, where is the conjugating of polygons? Are we introducing points in the interior?

No, we are not introducing any Steiner points - that is the question, right? We are not allowed to introduce Steiner points.

(Refer Slide Time: 48:27)



Sir corner points will be (( ))

Yes, only the corner points can be used; maybe, I should say no Steiner points. I guess what the question is, is this - you know are we allowed to triangulate like this - here we are basically introducing these Steiner points; we are not allowed to use Steiner points; we need to…when we talk about triangulation in the context of simple polygon or computational geometry we are not using any Steiner points.

But the inverted convex polygon (( )) polygon might has at least exactly N minus 2 triangles right and…

Exactly.

But I do not know is it true also called general polygons also

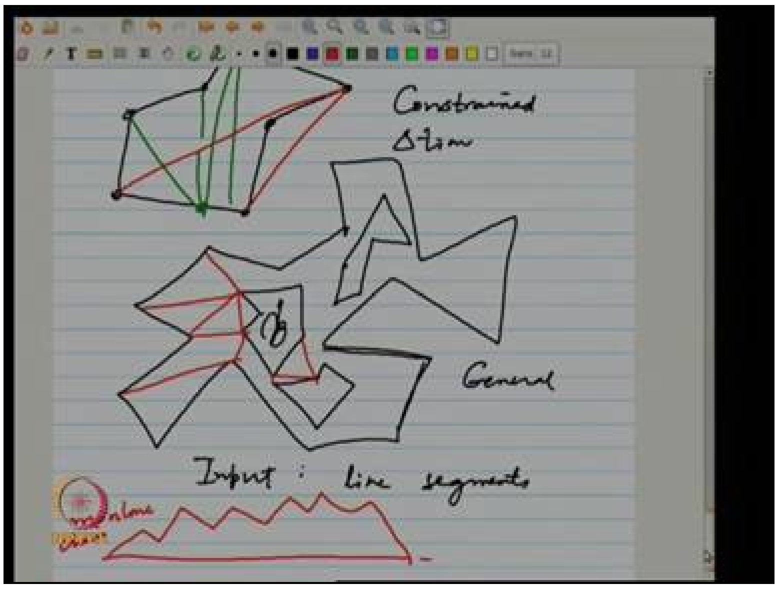
General polygons as in…

Kind of holes and (( ))

(( )) should be able to prove that.

There are no extra points (( ))

(Refer Slide Time: 49:37)



Well, why do not you compute that? See, you can triangulate the interior of these things right and you just look at the entire triangles and then subtract the number of interior triangles; everything is fixed, right? So, number of triangles inside will be fixed and the outer number of triangles will be fixed - so, the total number of triangles should be fixed.

Input is given in terms of line segments and we have to… We would like to introduce edges - some kind of edges to triangulate this whole thing. We will take up this problem tomorrow and we will actually establish a nice reduction between the ray shooting problem and the triangulation problem.

But, before that I will just leave you with this - just to think about it; suppose, my triangle was very simple - in the sense that it looked like, it is like this; we have one edge and we have a monotone edge - a monotone a polygonal chain - on this edge; suppose, we are to deal only with this situation. I will leave you with this to think about whether this particular case can be handled easily - are you with me? There is only one base edge and there is a monotone polygonal chain.

Of course, simple polygons being anything but simple; crooked, very curvaceous simple polygons you cannot deal with, but this is a very nice structure. Can you deal with this situation very easily? Something for you to think about; stop it here.