Introduction to Computer Graphics
Dr. Prem Kalra
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 29
Hidden Surface Elimination (Contd...)

We will continue on hidden surface elimination. So, if you remember there are basically two categories of hidden surface elimination methods. One is image base or image space method and another one is the object space method. Basically there are two approaches: image space and object space. Image space is where we consider the visibility through the pixel as you did in the case of ray tracing. From a pixel you try to establish the visibility of the scene for that particular pixel.

If there are N number of objects and P number of pixels in the computational effort is of the order NP. So here you have the input as the geometry of the scene ad the output is the image of these pixels. In object space it is basically through objects themselves, we try to establish which objects are hidden or part of the same object is hidden from the other part of the object so it is basically through primitive or the object. So the precision of the computation here is of the object as apposed to pixel in the case of image space.

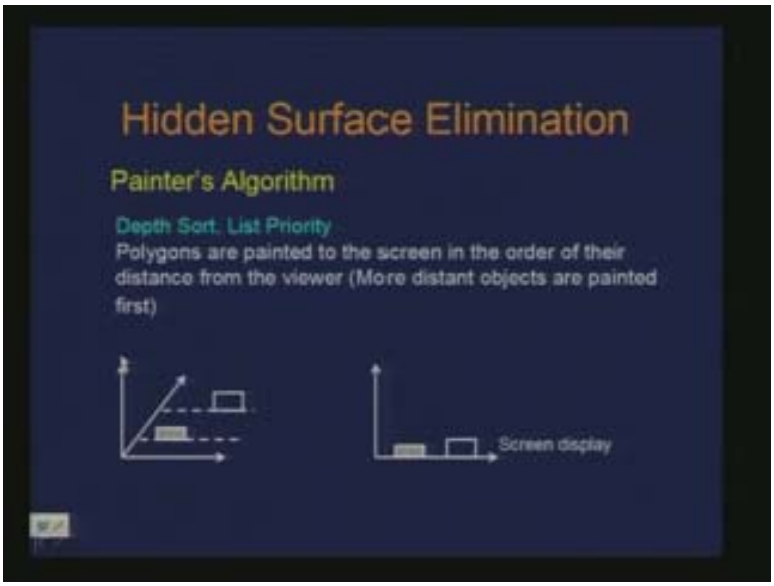
(Refer Slide Time: 03:20)

So, if you look at the input and output in this case, here the input is the geometry of the scene and output is also the geometry of the scene and then you perform the projection and rasterization. These are to broad class of methods of hidden surface elimination. We looked at the method of floating horizon which was primarily the hidden line elimination. And then we looked at the z buffer method which is very popular.

Let us look at some more methods of hidden surface elimination. We were discussing about the natural way of dealing with hidden surface elimination in the sense like a painted house so what you do is while you paint anything you draw or paint something which is at the farthest and then you try to something which is in front of that and then you try to paint further and so on. So you start from the back then you come to the front while painting. So you paint the sky first then mountains and then the house etc so that is the kind of order you maintain.

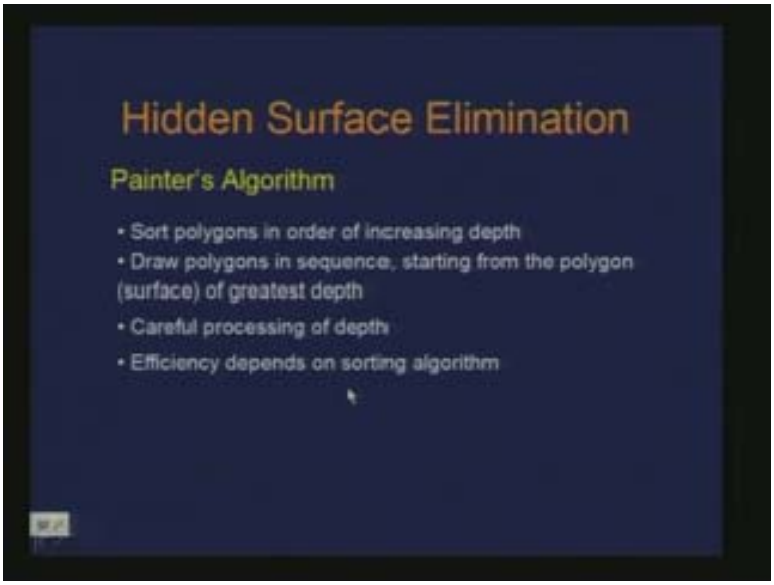
So similar thing you can observe for devising a hidden surface elimination algorithm where you consider the order from back to front. Now we are going to look at the Painter's Algorithm which inherently involves a sort of depth. So you are going to make a sort in depth and then decide that you will first paint the object which is at the extreme back and so on. Or in another words you establish some sort of a priority, there is a list where you give priority to the objects which is in terms of the depth. So the way it works is the polygons are painted on the screen in the order of the distance from the viewer and that is what involves a sort which you need on depth so more distant objects are painted first.

(Refer Slide Time: 06:34)



Hence, if you look at the example here these are the objects we have so you are going to plot this e first and then this if the viewer is in this direction facing from here.

(Refer Slide Time: 07:30)

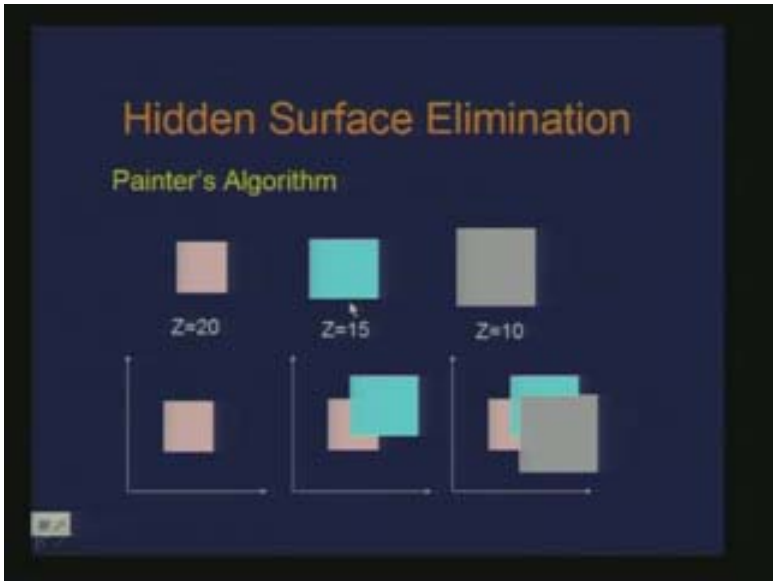


While we are talking about the shortened depth, the processing of the depth actually may not be trivial in all the cases.

Features of Painter's Algorithm: It involves sorting of polygons in order of increasing depth and that in turn decides the efficiency of the computational effort involved in this algorithm. And then you draw polygons in sequence starting from the polygon which is at the greatest depth. While we are dealing with establishing this order depth we have to process the depth that means we have to find out the depth and established as this is the farthest object or the greatest depth for the object and the efficiency actually depends on the sorting algorithm you use.

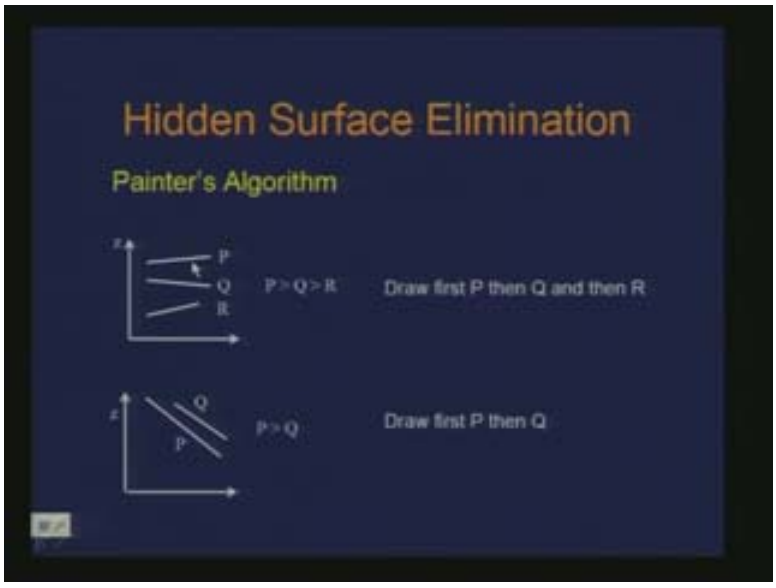
Now if we consider a simple example where the depth is captured by the value Z so Z is the depth of the object and we have 3 objects here where the depth of the each point in this polygon for instance is the same as 20, it is a constant Z value.

(Refer Slide Time: 09:04)



Similarly, here Z is equal to 15 and here Z is equal to 10. So, when we do the display of these objects so first I am going to draw this as this is the farthest in Z then I am going to draw this therefore it is going to occlude part of the object here and then I will draw this. So this is the final display I will get. As long as I could figure out that these are distant in depth all these 3 objects I could just paint them in that order without any ambiguity. So if I have a polygon p so this is a polygon in 3D just take a projection on some (z, x) (z, y) plane.

(Refer Slide Time: 17:41)



So this is a polygon P, this is the polygon Q and this is a polygon R. So depending on the order of the depth I obtain I set a priority saying P greater than Q greater than R and that gives me the order of its display. So first we draw P then Q and then R. First of all when we are talking about determining depth what is it that determines depth. This is a polygon so from where do I measure this depth? It could be the maximum Z value, it could be minimum Z value or it could be the Z value of the centroid or anything so there has to be a measure of depth.

A situation like this in fact has an advantage because as far as Z is concerned there is no overlap so the Z minimum of this polygon P is greater than Z maximum of **K** so there is a clearance between these two that allows me to draw this in a straight forward way. So I can establish in some sense that this whole polygon is behind this polygon just by looking at Z min Z max. If I am able to establish this or in other words consider a small bounding box on Z and these bounding boxes of polygons do not overlap then in that case just the order of the distance of these bounding boxes decides the sequence of display.

If we are talking about the Z max as an indication of depth that is what decides the priority which is true in this case P greater than Q greater R just by looking at the Z max of the polygon. So if I try to apply the same thing here what happens? So Z max of P is greater than Z max of Q which gives me the priority as P greater than Q which tells me to draw P first then Q what happens? This is not the right way to draw P first and then Q but I have do to something more.

One thing I can find out is the extends are overlapping. Therefore I tried to split these polygons and treat those splits etc which is quite costly. It may be necessary to do that in certain situations but this situation does not really demand that. Therefore you will have to fragment these polygons, fragmentation is a costly process.

First of all you have to find fragments and that itself is costly. If I want to split this polygon into the appropriate polygons such that this has a clearance from Q it is costly. We have to find all those boundaries which is going to be clear, we have to split that, find the necessary edges which make this split possible. So processing for splitting is in additional cost. Once you have this split done then of course it is a simple ordering. But one thing which we observe here is that the entire Q is on one side of P. The entire Q is actually behind P. If I define this as the front of P on one side of P then this is behind P so the entire queue is actually behind P so you should be able to find that out.

That is, if I figure out that Q is actually behind P then I am going to draw Q first and then P so it is not just finding out the measure of Z max, Z min or Z of the centroid which is an indicator of the order of depth. This is just a projection, I am going to project on this. This is just an indication of what is happening to the Z values so these are the polygons, these are the edges. So, if I try to plot a polygon like this on this paper then this is going to meet an edge there. Here this priority was set looking at Z max of the polygon so P has a Z max here, Q has a Z max here and I have a Z max here so P was greater than Q was greater than R here. The Z max of P is this one, Z max Q is here so P is greater than Q. And if I just consider the Z max as a measure and indication of depth and decide the

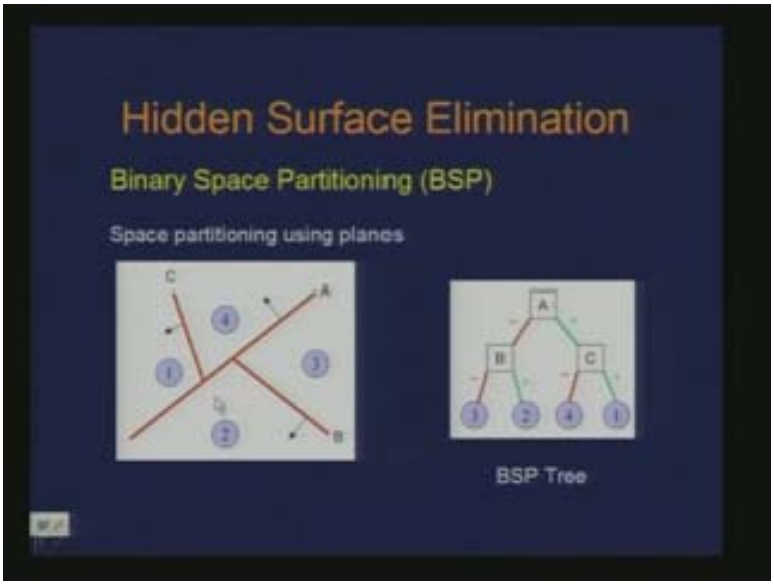
sequence of display then in this situation it is fine but in this situation it is not. The situation is still not so easy. What happens once you get things like this? Here there is a cyclic overlap. Part of the polygon is behind and part of the polygon is in front and so on and so forth. This differently requires some splitting.

(Refer Slide Time: 19:12)

Though the Painter's Algorithm looks conceptually very simple […….] 18:25 so these situations have to be dealt with. Now we move onto another method which is actually a way to define this order of back to front. This is actually based on what is called as binary space partitioning BSP so the idea here is that you want to partition this space this is a space partitioning method where the space is occupying object or clusters like 1 2 3 4 so you want to separate these objects or clusters.

One of the ways you can do is that you define a separating plane like A which basically divides or partitions this space in two parts where you have 1 and 4 on one side of this partitioning plane and 2 and 3 on the other side of the partitioning plane. So you define a separating plane which divided this space into two partitions. Then within that you again define separating planes. So you define the separating plane B which in turn partitions the space where you have the object 2 and the object 3 and similarly here.

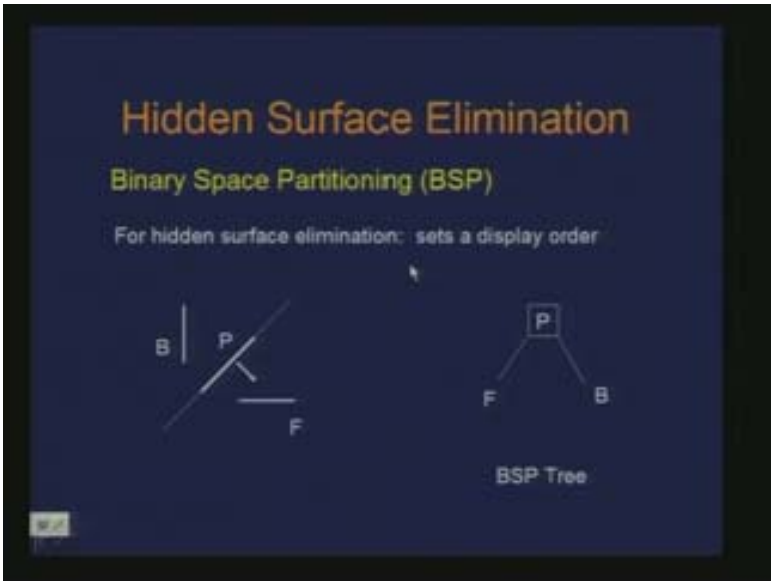
(Refer Slide Time: 22:57)



So using these planes you are able to partition the space such that these 1 2 3 4 are separated. You can see this as a tree where the root node is the first separating plane which was A and then the subsequent node here you have is another separating plane B, here another separating plane C so the two parts are just the two sides of the separating plane and the minus and plus is same that one is behind and the other one is in front of the plane. And then you have at the leaf notes these clusters or the objects you want to partition. This is fine as far as the partitioning of the space is concerned but how does it help in the context of hidden surface elimination? What it does is actually it helps you setting the order.

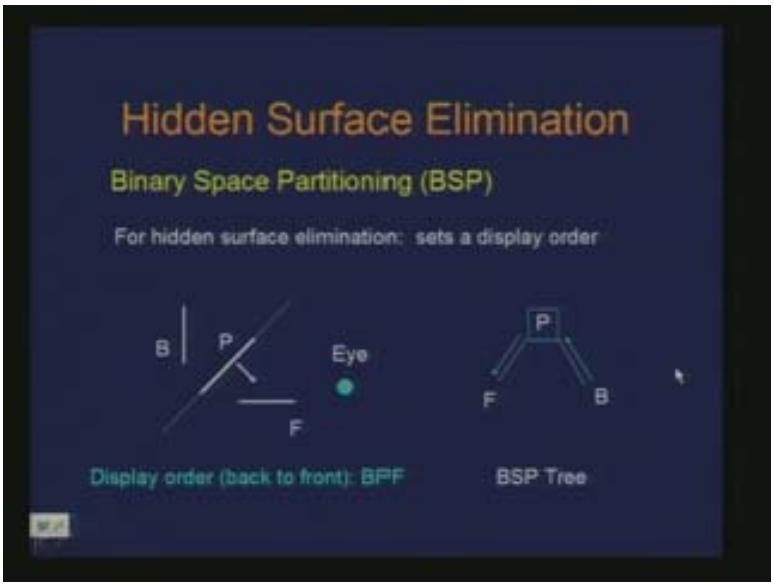
How does it do it?
For example, if I have these 3 planes, now consider one of the plane or the polygon as the separating plane the way we have looked at for partition. If this is my first plane P and I have a notion of what is in front and what is behind with respect to that plane. There is a normal defined for this plane which can decide the two sides of the plane. Now, again I form this BSP tree where the root is given by the polygon P and then I have this polygon F which I call it which is in front of the polygon. There is another polygon B which I call as the plane which is behind the polygon P.

(Refer Slide Time: 24:46)



Now if I have this scenario I actually have the way to display these. Now I define the viewer because the viewer has to come into the play where I am talking about display and only then the display are gets built. But BSP tree gives me a data structure to give me the display order instantaneously the moment I define where the viewer is, that is the whole capture.

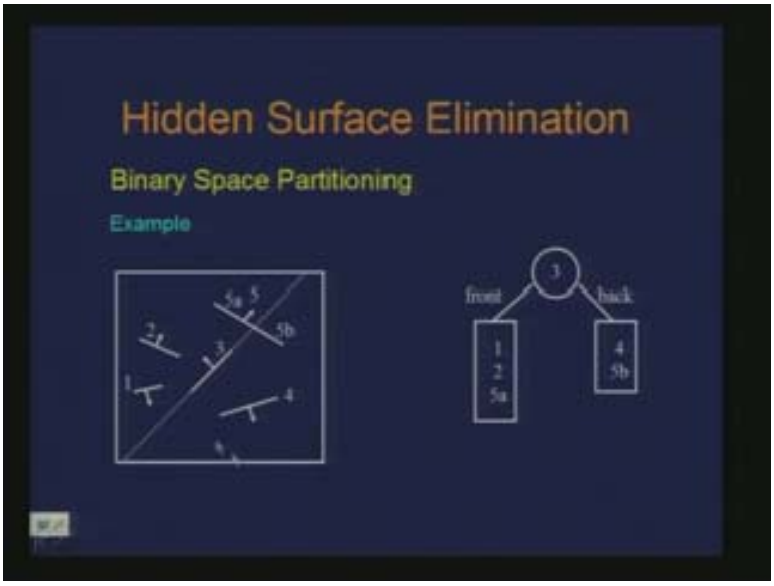
(Refer Slide Time: 27:15)



So let us say I define the viewer or the eye here which is in front of the polygon P. I had already formed this BSP tree. Basically we are interested in establishing the back to front order which is required in Painter's Algorithm, so the moment I do this with having

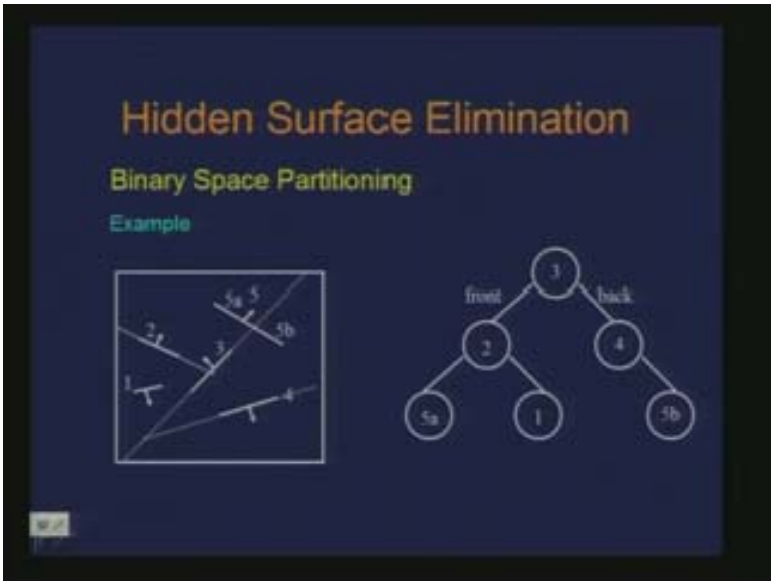
defined eye there now this back to front basically means some sort of a traversal of this tree so here if do this D first I display D first then F that gives me the display order and that is nothing but a traversal of this tree. Thus, if I have the definition of the eye in front of the root plane here then the display order is behind the root front or back root front.

(Refer Slide Time: 30:58)



Here is an example where these are the various polygons 1 2 3 4 5. If I had placed this eye behind the root polygon P then the display order would have been, of course the depth. If I put the eye here then you could have this plane so this is bpf and what would be this? This would be fpb. Therefore if I had placed this viewer here then it would be fpb here, it is bpf and it will be fpb here. It is because again I want to see what the back most is and then what is in the front. This is because I still want to maintain the back to front order. So, if I consider an example of this kind where I have 5 planes 1 2 3 4 5 in the scene and I choose plane 3 as the root plane to start with. Now I know how to form the BSP tree so I just put whatever is in front on this side and whatever is on back on this side.

(Refer Slide Time: 35:15)



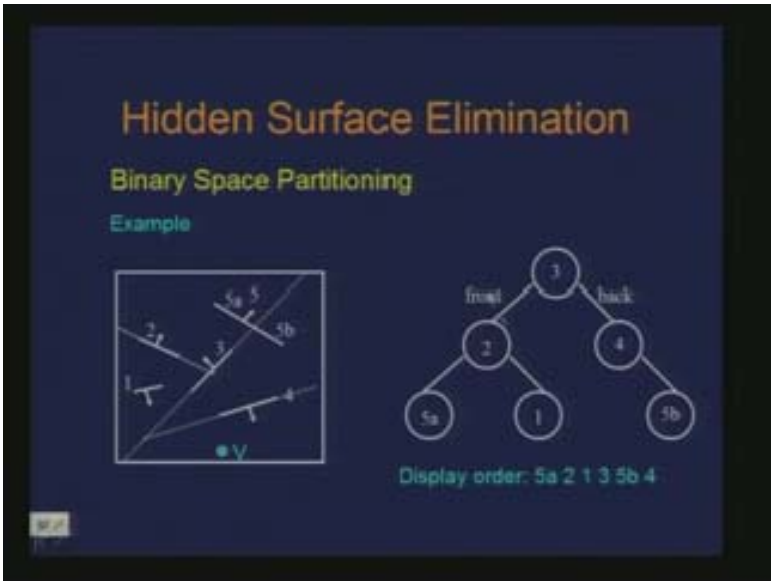
So, with respect to plane 3 I notice one two are in front and in fact part of 5 has to be split. So I split into 5a and 5b and it is the 5a which is in front. Now on the back I have 4 and 5b. This process of defining the plane as a root has to continue till I exhaust all the planes. Earlier I had taken only 3 planes but here I have to build this tree till I exhaust all the planes. So I again define two as the root of this sub tree and then I figure out that 5a is in front and one is behind, the same thing has to be done here now.

I choose 4 as the root plane and I figure out that 5b is behind that is where 5b comes. This is the complete BSP tree now. Now for this if I define V here which is the viewer then what is the display order? Now this has to be done in a recursive form. We have to do the traversing in a recursive way. This is when I have the viewer in front of or actually behind the polygon 3. So if I have the viewer behind the polygon 3 then it is front, root and back.

The normals are defined for the polygon. So whatever we have done so far up to this point is with respect to polygons so there is no viewer which is there in this. But later on it helps you with just a matter of checking the viewer with respect to the polygon whether it is front or behind and that decides the order of the display. Therefore as a base case it is just the back to front. If you just had 3 planes, with respect to P you have F and D, if it is the viewer which is behind the root plane then you have the order front root back and vice versa.

Now this case when we define the viewer here which is behind polygon 3 the root here the order is front root back. Now this has to be done again in a recursive fashion so when you come here again you see what happens with respect to this polygon to the viewer and decide what order you would get. So again it is 2 with respect to behind. So you have the order front root back and then of course come 3 and 4. That is the way you get the display order.

(Refer Slide Time: 39:21)



So here I had actually chosen polygon 3 as the first root of the tree which was an arbitrary choice. So what happens if I choose something else? For example, if I choose 5 as the root things are different the entire tree is different. And again if I try to define viewer there it gives you a display order as this. So how should I decide this root? Arbitrarily decided I first took 3 as the root and then I took 5 as the root so would there be any consideration? What is the difference between the two? When I had chosen 3 as the root and when I had chosen 5 as the root is there any difference?

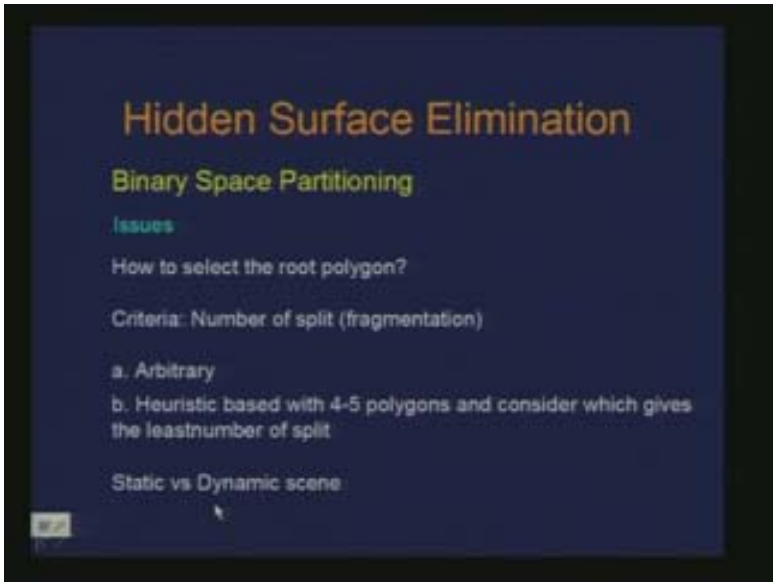
In one case you might have a nice looking tree something kind of balanced and in the other case it is the cube tree so the traversal you would have the depth of the tree may be different. In the first case we had to split polygon 5 when we had to consider 3 as this 5a and 5b so we were splitting polygon. Obviously splitting is not good so there has to be some check on that. So this is one of the issues that how do you decide the root of the tree. Only the normal of the plane decides what is in the front of the polygon and what is behind this polygon.

The only question is that if you have the viewer also aligned to one of the polygon then you are in trouble then you have to do something else and again you change the root of the tree. So the issue here is how to select the root polygon. Therefore clearly one of the criteria is the number of splits of the fragmentation which is being cost. So, one choice could be that we do an arbitrary selection, take one of the polygons in root.

The other could be that you have a heuristic and all it does is it actually takes some sort of a local processing. That means you pick one polygon and figure out what is the kind of fragmentation to a certain level in the tree so you do not form an entire thing and you pick another polygon and so on. So you just do that with three or four or five polygons and pick the one which gives you the least fragmentation so there is no global optimum guaranteed in this case. It is just a heuristic to possibly select a good candidate. What

have we considered about the scene so far? We considered that the scene is static, objects are not moving and the moment object starts moving things are a havoc.

(Refer Slide Time: 42:50)

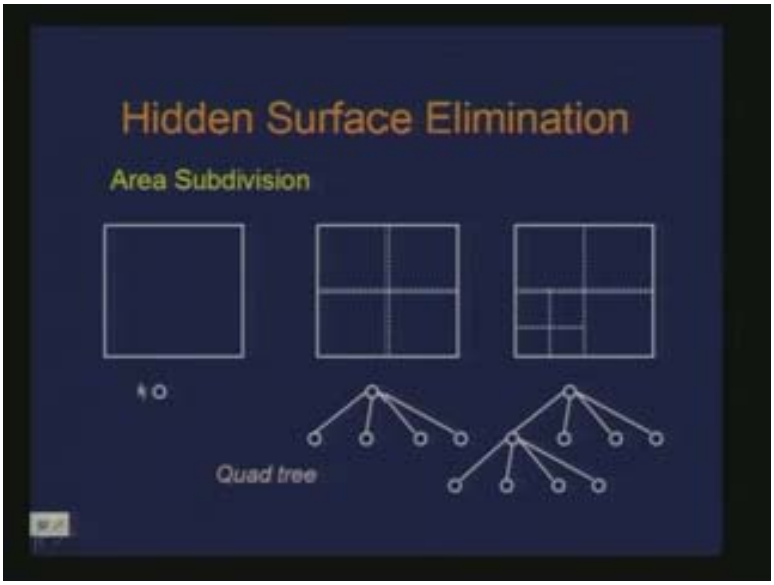


One of the strength of the method is that you just have to build the BSP tree once and then I can keep moving the viewer anywhere I want. So this handles very nicely the movement of the viewer and there is no problem. I can keep moving in the viewer and just do the ordering by traversing the tree.

But I just have to construct the tree once and that is because of the fact that the scene is static. The moment the scene is dynamic things are going to be hard. Some people have worked on dynamic insertions and dynamic deletions in the tree but it is not as straight forward as it should be. This was another way of doing hidden surface elimination where we are basically establishing the order of display. And you have assumed the scene composed of polygons because each of the polygons is actually acting like a separating plane.

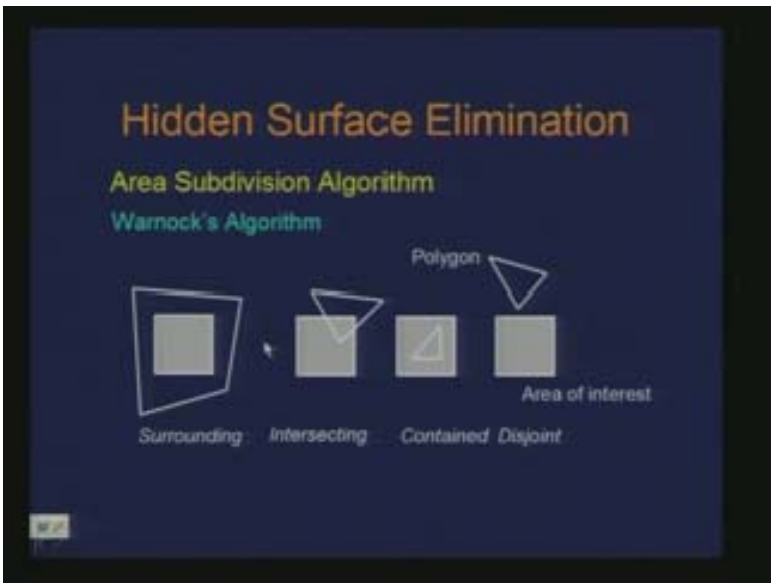
The other approach is using area subdivision. In area subdivision what is happening is that there is an area here which you equally subdivide in case your criteria of subdivision demands so. So, there is a criteria of subdivision which you evaluate on and then when your criteria demands to subdivide you subdivide equally into four parts and then you do that evaluation in each of these subdivided part and subdivide if need be.

(Refer Slide Time: 45:03)



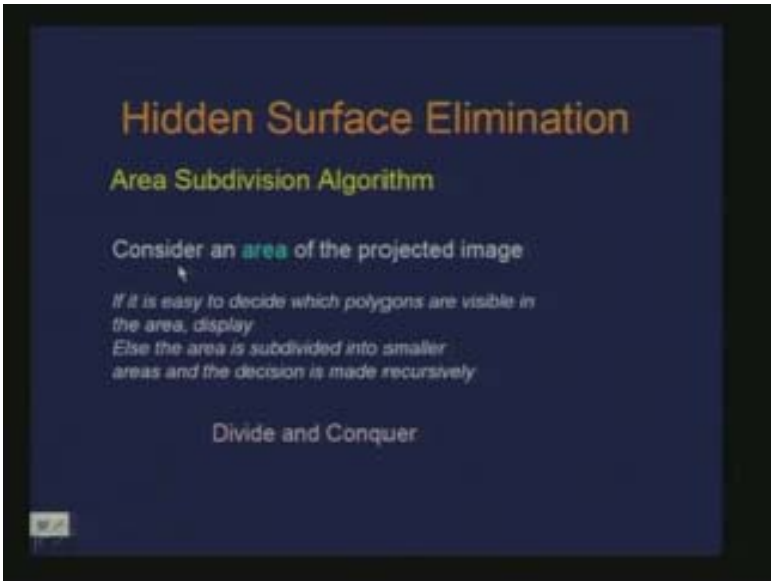
You can also think this to be a data structure of the kind tree which is quad tree as you have 4 nodes in every subdivision level so every node can be subdivided into 4 nodes so basically you have a quad tree construction. Now the question is how do we use this area subdivision for hidden surface elimination?

(Refer Slide Time: 45:16)



So the way we do is you consider an area of the projected image and if we observe that it is easy to decide which polygons are visible in this area you display it.

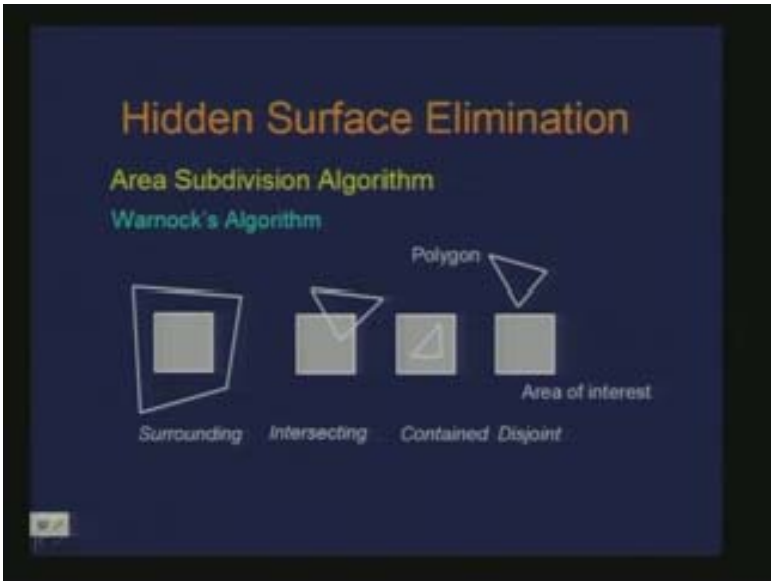
(Refer Slide Time: 46:49)



You are basically considering an area, you are starting from an area which is to be displayed it is different from the other approaches so you are starting from the area to be displayed and then if you figure out that it is easy to decide which polygons are visible in the area you display those polygons else the area is subdivided into smaller areas and the decision is made recursive. It is basically a divide and conquer kind of a method. So you do an evaluation of something which is saying that it is easy to decide whether to display or not and once you do that you display otherwise you subdivide. The Warnock's Algorithm is actually based on this concept of subdivision.

What is done is there are four cases which are made. So if you define this to be your area of interest which is shown in grey here and these are the polygons this is a polygon, this is another polygon inside this polygon and this is also a polygon. So what we are trying to observe is the relationship of the polygon with respect to the area to be displayed and that is what we are deciding as four different cases. So in this case the polygon is surrounding the area so we call that case to be a surrounding polygon.

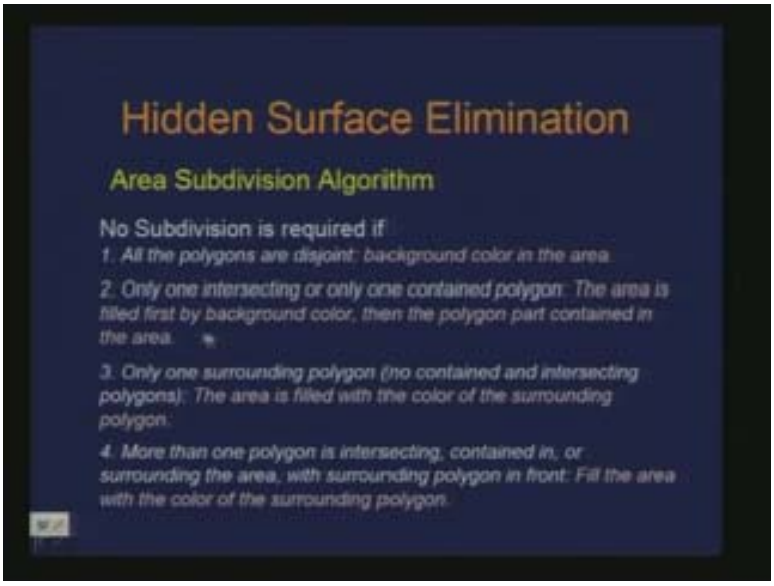
(Refer Slide Time: 51:17)



The idea is that these cases can decide easily what to do when we say it is easy to decide, that is what the cases are trying to do. If there was only one surrounding polygon with respect to the area to be displayed then what action would you do? You will display area with the color of the surrounding polygon. If you have only one surrounding polygon to the area you know what to do. It is a matter of establishing this case relationship area with the polygon to be able to decide easily. Similarly, if the case is when the area of interest is disjoint with respect to the polygon that means the polygon is there and there is no overlap between the polygon and the area of interest we call that case to be a disjoint case.

If there is only one polygon which is disjoint with respect to the area display what do you do?
You just paint with the background color so it is an easy case to resolve. Now there is another case where the polygon is completely contained inside the area. So once again if there is only one polygon which is contained inside the area what would you do? You would paint or display the polygon in that area. This is again an easy case to resolve. And if you have an intersecting polygon this is the polygon intersecting with the area so you have this case of intersecting polygon. So here depending on how you implement your method you may keep subdividing or you may just figure out the area which is intercepted by this area if there is only one polygon and you display this. Therefore in some sense you do a clipping. Now let us see how these four cases are useful for developing the algorithm and what they really mean.

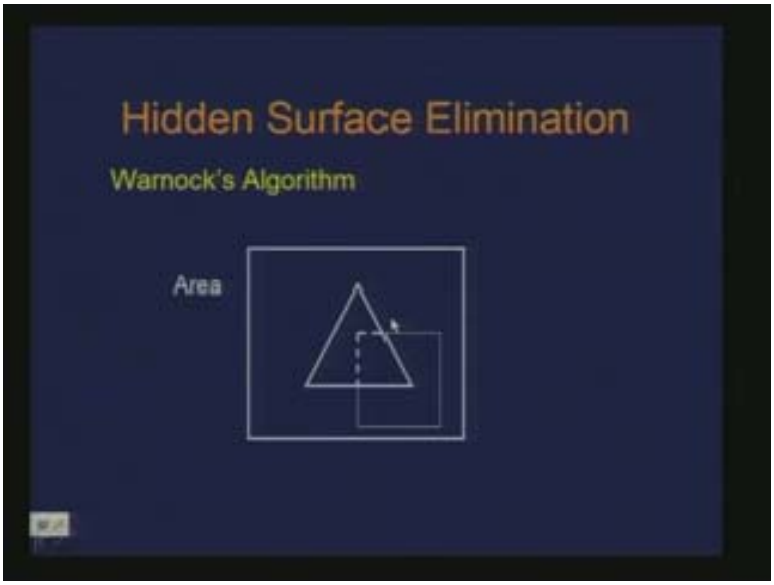
(Refer Slide Time: 54:03)



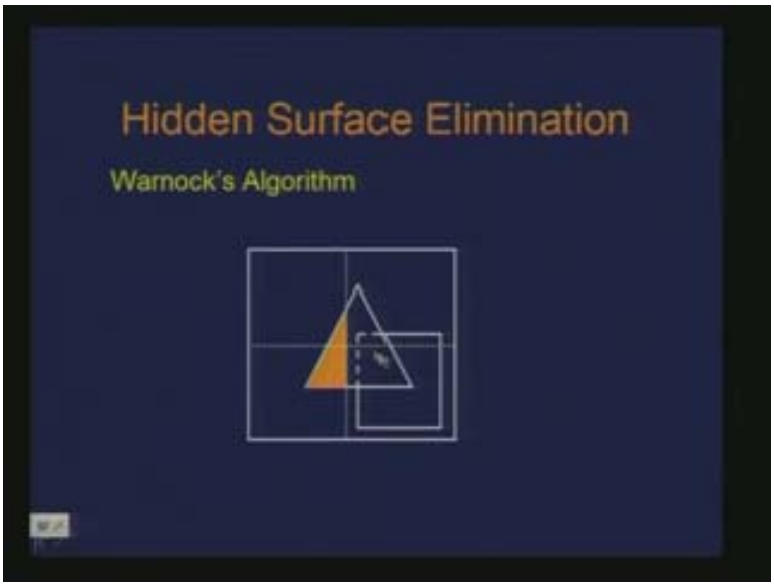
Basically we are saying that there is no subdivision required if one of these situations is there. These situations are nothing but the cases which we define as simple cases as we meant that if none of the four conditions happen then you subdivide. That means you have not reached a simple case. First all the polygons are disjoint so all you need to do is color area with the background of this. In the other case we have only one intersecting or only one contained polygon then the area is first filled by background color and then the polygon part contained in the area. In the third case if there is only one surrounding polygon and if there are no contained and intersecting polygons then the area is filled with the color of the surrounding polygon.

The fourth one is also a simple case, if there is more than one polygon intersecting contained in or surrounding the area but the surrounding polygon is in front then you fill that area with the color of the surrounding polygon. So you do not require subdividing if any one of these situations occur otherwise you subdivide. Example: Here you have an area to start with so actually when you are displaying the screen you can consider the entire screen to be the area to start with, whatever display area you are using. This is the area to start with and these are the polygons that are inside. So what we do if this is the situation? Does it require subdivision or not?

(Refer Slide Time: 54:50)

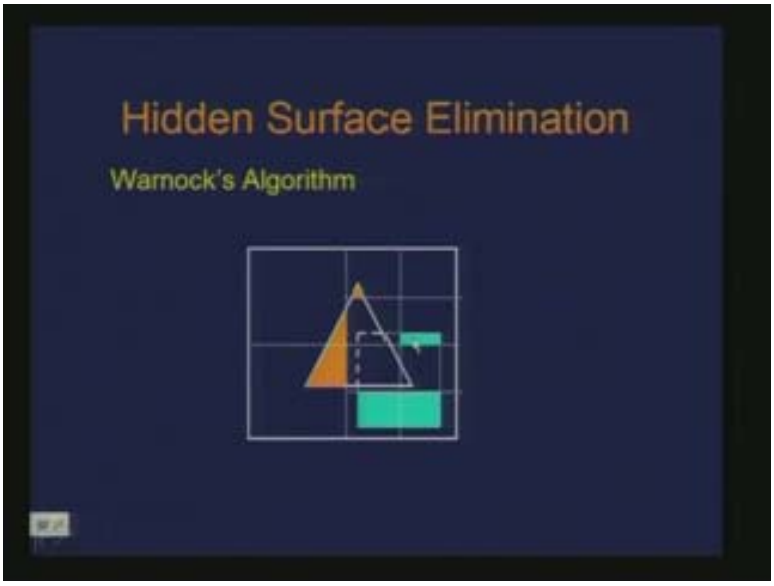


(Refer Slide Time: 55:43)



Actually there is only one intersecting or contained polygon but here there are two of them so you subdivide. So, if you subdivide this is the situation. Now what happens is, with respect to this area you figure out that this portion is the only intersecting part therefore you paint it with the color of this polygon and similarly here. But again here in these two segments you require further subdividing so you subdivide further. In this case you again resolve it because there is only one intercepted polygon, here also you have only one polygon so you color it with the color of the polygon, here also you have only one polygon but these cells need to be further subdivided and so on.

(Refer Slide Time: 58:44)



Now, what is the maximum subdivision you may have to do?
The maximum subdivision here has to be done to the pixel size. So the maximum subdivision you have is the pixel size. And what happens if there is more than one polygon with respect to that pixel? Then you use the depth or Z. So here the problem could be because we are actually dividing equal quantities every time so that may not be a desirable thing in the sense that sometimes for just this portion we need to subdivide. So small portions also sometimes demand subdivision because we are just subdividing the areas without looking at the polygons spanned within that area and always there is an equal subdivision. If there is a scheme which you can design with unequal subdivision then perhaps this could have been subdivided here so that this portion is in one part and the other portion is in another part.