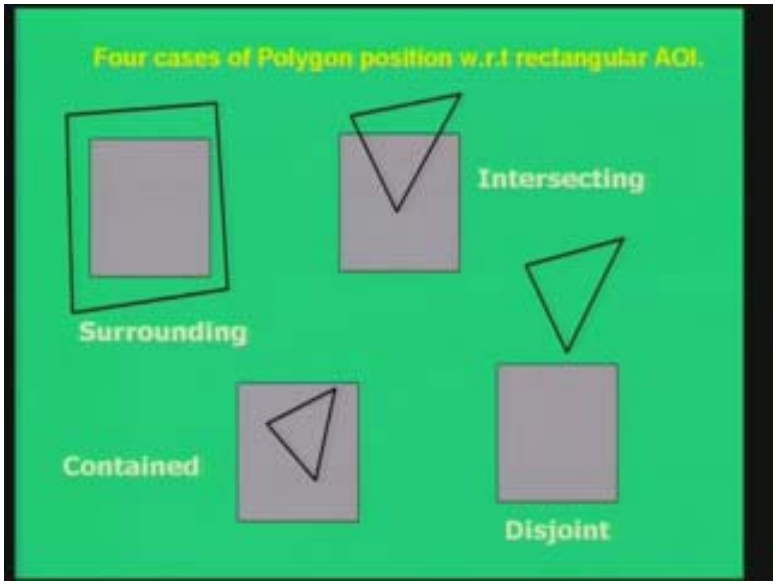
Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture # 31
Visible Surface Detection (Contd...)

Hello and welcome everybody again. We have been discussing in the last class the concept of area subdivision method for VSD algorithms or Visible Surface Detection algorithm. And this is one of the sequences of half a dozen algorithms which we are going to discuss in this lecture series. To just recap; we have discussed back face culling, then the depth buffer or Z-buffer based method and then of course the scanline based method and then the depth sorting or Painter's algorithm. And at the end of the last class we discussed the area subdivision method.

So, just going back to those illustrations which distinguish these four possible relationships between a rectangular sub-image and a polygon based on which we either render or split a polygon, split rectangular area of interest into four equal parts is the following. So if you look back into the slide these are the four cases of polygon position with respect to the rectangular area of interest.

(Refer Slide Time 02:17)



You can have a surrounding polygon, you can have an intersecting polygon, you can have a contained polygon or a disjoint polygon with respect to a rectangular area of interest which is marked by the grey color inside this. And as mentioned in the last class you must remember that you do not have to worry about polygons which are disjoint, you do not even consider them under any circumstances, they contained an intersecting or considered as same identical it means when we mean identical we said that one of the points was that if you find there is only one polygon which is intersecting or it is

contained within the area of interest what you have do is paint the rectangular area of interest with the background color first and then paint the polygon with the corresponding color of the polygon and that is over. Then of course we had the other case of a surrounding polygon.

If there is one surrounding polygon of course use the color of the polygon to render that particular area of interest mind you not the entire polygon must be rendered only that rectangular area of interest which could be the entire image or a part of the image under consideration to paint you must paint with that particular polygonal color if that polygon is surrounding the entire area of interest. Now these are the cases which we have seen in this slide earlier if there is only one.

Now the case comes when there is more than one polygon. When there is more than one polygon that is the case which we have to consider. Of course in that in those cases do not consider any disjoint polygons and still if there are more than one that is two or more polygons within the rectangular area of interest and various relationships exist of course again only surrounding contained and intersection relationships are to be considered disjoint case or not consider keep in mind.

Now you have more than one polygon. Under that you have a special case where there is one surrounding polygon in front which if you can find out by the help of the z coordinate intersection test with respect to these four vertices of the rectangular parallelepiped viewport parallel to z axis if you visualize rectangular cube parallelepiped and you are viewing through a viewport and you are visualizing that there are so many polygons inside and there is one in front of you. The closest to you is obscuring all other polygons then you should use that particular polygon to paint that rectangular area of interest. Otherwise see if none of these cases happens that means you are not having a single polygon case number one, you are having multiple polygons and out of this you are not able to find out one surrounding polygon which is in front of you. So if none of these cases happen what you do you split the rectangular area of interest into four parts that is what you do and keep on sub dividing. This is why it is called a divide and conquer strategy working in the image space and you keep on sub dividing starting from the root node.

Remember the quad tree representation which we talked about for an image take, the entire image split into four parts four quadrants if necessary, you did this in the Bresenham's line algorithm case, ellipse and circle as well and we talked about a quad tree representation and an extension of that was Octrees in the case of solid modeling in 3D. So I repeat again, quad tree representation four parts each of those sub images or four quadrants could be again sub divided if necessary you keep on doing that till you may reach in a certain case if you keep on sub dividing into the maximum resolution the maximum height of the level of the depth of the tree which you get is just one pixel resolution.

You might reach one pixel resolution at the maximum level of the tree that if you keep on splitting and splitting and so on you may reach one pixel where you cannot sub divide

any further. And that case of course what you have do is you have to render that pixel with what polygon is in front of you corresponding to the particular pixel. That is what we were discussing and pick up a few case studies as examples of area sub division method.

Couple of examples as we did for other cases also, here we pick up a few more case studies. Let us look at this simple example. Stage one to be analyzed where within the entire image there are two polygons but in this case there are two triangles. So if you have passed those entire four tests first of all there is no single polygon within the area of interest.

What is the area of interest A Y here? The entire image, the entire image is your area of interest and you find that there are more than one polygon and there is no single surrounding polygon. there is no single surrounding polygon which is in front of you so there is no other option but split this area of interest in this case it is the starting image which is split into four equal parts or four quadrants. So do that and let us see what happens.

When you split this is what results in. Of course you split and do some processing and that is the result which you get if you split this entire image into four parts. Let us say we have four quadrants northeast northwest southwest southeast we will take this is quadrant number one two three and four that is four children of a root node, the entire tree which is split into four parts four children is what you have and those are the quadrants one two three and four. So quadrants one two three and four, four parts you can see here and you start analyzing them one by one.

Let us start with quadrant number two. It is a very simple case of an intersecting case intersecting contained the same. What you do when you find an intersecting polygon a single one? You paint with the background color which is in this case white as I have chosen here but it could be black does not matter and the rest of the part of that quadrant you paint with that intersecting overlapping polygon. This concept holds good for quadrant two three and four. All the three quadrants will have a case of intersecting polygon and a single one and we know what do in all of these cases which is the same case you first paint with the background color and then paint with the polygon which is overlapping with that particular AOI or area of interest.

We are talking about quadrants two three and four and not quadrant one. The quadrant one will be here so two three and four is what you are looking and for all these two three and four quadradents you have an intersecting case and that is what you have done. So the quadradents two three and four had been processed.

Quadrant number one is a special case what happens in quadradent one? If you carefully watch it contains two polygons again. Like the original case of two polygons and there is no surrounding polygon in front so quadrant number one is the only quadrant out of these four after the first split. After the first split you got four quadrants and the three out of those four was an intersecting case intersecting polygon. Only quadrant number one had a

scenario where you had two polygons and there is no surrounding polygon in front so you have to split quadrant number one again into four further sub divisions which results in a scenario like this.

I repeat again, quadrant two three and four have been already processed and you do not need to subdivide further, quadrant number one here in stage number two is split into four equal parts as in stage number three into four other sub quadrants. And again as you can see in this split now you have four area of interest this sub quadrant one is a case of a disjoint, it is only a background color, so you do not have to use any polygon but simply put the background color in that and that process is this particular sub region.

What about this sub quadrant? That is an intersecting case so you put the background color first and then put the overlapping part of the intersecting polygon. Similarly quadrant number four is the same case. Quadrant three is a problem because you have two polygons and again none of them is in front so that is the one which you need to split further. So when you split that particular part this is what you will result in. That final part here which is not processed at stage number three will be processed at stage number four which will give you four sub quadrants again at level 3D composition at the fourth stage first, second and third level of decomposition here you have two sub quadrants at that stage which will be disjoint cases fill it by the background color and this will be filled with a probably a surrounding or an intersecting polygon as the case may be and of course this is definitely a case with the brown color rectangular patch will be an intersecting polygon. So at the end of this stage you will have no further sub divisions necessary because the entire image has been covered with smaller sub divisions, quadrants or regions of interest.

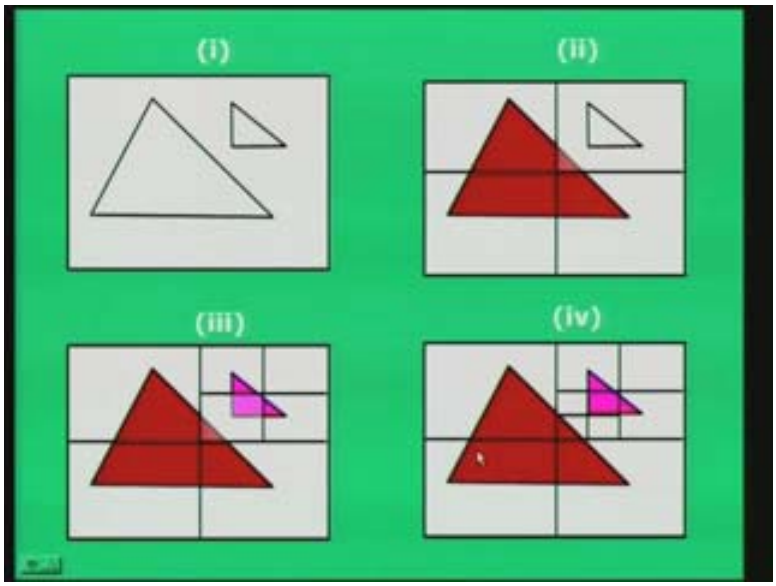
Each of them are either sub divided or at the end of the maximum level of resolution you have smaller sub images or region of interest which will only have the background color or single polygon case, surrounding, contained or intersecting, disjoint of course is background color. So if you look back again as I rewind this stage here this was the initial case, split into four parts, three quadrants are handled, one quadrant left, first quadrant split again into four parts you will again have three quadrants processed very comfortably, only one quadrant having two is split again into four parts and that is the final result which you will get after subdivisions. This is an example only with two polygons where we have to go for subdivisions up to three levels. I leave it as an exercise for you to alter these positions of these two triangles or even the shapes of these two triangles and position it in a graph paper at different positions and start splitting and exercising.

Take this as a home exercise where you interchange these positions of these two triangles or take two different triangles with two different shapes and start splitting if necessary. You can take a small pencil, scale and a paper and workout this problem yourself with two polygons, with two different shapes at two different positions. That is what you do in the area subdivision method.

But an important lesson to be learnt to here is that if you just carefully watch the figure again if you compare the original scenario at stage number one before any subdivision and the final result which you have got if you compare stage one and four you see at the maximum number of subdivision is occurring at situations where their adjacent polygons close by and that too your operating more at near the edges of the vertices of the polygons.

More the overlap more will be the number of vertices around a certain area of interest so you need more and more subdivisions. And the complexity of this algorithm in image space subdivision is a very difficult case. Of course the maximum number of subdivisions, if you have an image of resolutions by m pixels by n pixels n could be 1024, 640 by 480 or up to 1024 whatever the case may be. Of course the maximum level of decomposition which you can go is log in step that is of course the worst case. The case of course an average case can be computed but it is very difficult to compute because it all depends upon the number of triangles which you have or polygons which you have, the relative positions of those polygons, relative positions of those edges and vertices the more closer you have the edges and vertices of two different polygons closer together or overlapping more the more subdivisions you will have and there is where you need to spend more time.

(Refer Slide Time 14:13)



As you can see here when you compare stage one initial stage and stage four final stage the subdivision is somewhere around over here where there is more of protect overlap. In other places where there is a background or only one single polygon you need not subdivide. If you add a third polygon you can visualize that you need to have more subdivisions necessary in cases where the edges of that third polygon will be overlapping to nearby edges of the other two polygons. And you can almost visualize such a stage that you will go on subdividing more and more.

Of course you cannot go beyond the pixel resolution but if you have to go to that pixel resolution the complexity of subdivision and testing goes on more and higher and higher which you cannot avoid but you have to do and that is the main degree of complexity of the area subdivision method, the main bottle neck of computational time necessary is subdivision and testing and more number of polygons overlapping their edges and vertices coming closer together will cause more subdivisions.

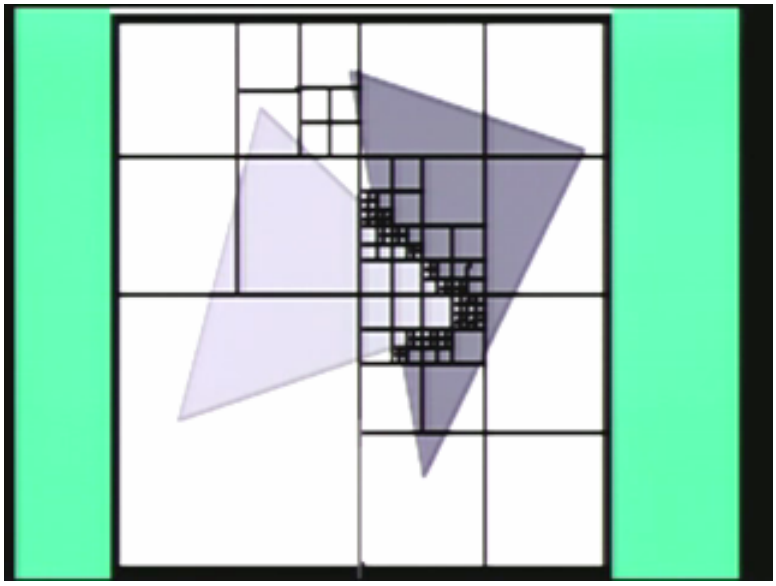
I will just give you a final example based on which you can visualize that if you look in this particular case just two triangles again completely overlapping as much as possible. As you can see here this is the final result of subdivision. So where do you find subdivisions getting more? It is at those edges which are overlapping those areas and the vertices which are overlapping certain areas. So, as long as the two triangles or polygons would have been separated out the number of subdivisions would have been but it would have been less. But the more closure you get more overlap you make more vertices and edges overlapping and then coming closer the more the subdivision necessary.

If you look at this particular figure you can visualize that the first edge is of course four quadrants and the simplest case out of these four sub quadrants which you will get first level of sub quadrants sub images is the third one. The third one is the only case where you have one intersecting polygon. I think it is clear, the sub quadrant is the only one third where you have no other polygons but just one intersecting polygon easily paint with the background color and then that part of the polygon which is overlapping and that part which is the third one is done.

First, second and fourth polygons have to sub divided. If you examine the second one that is also little better in the sense that you subdivide further into four parts this part is the background as pointed by an arrow here and then you have this part which is intersecting single intersection, here also you have a single intersection which is only in this sub quadrant one out of the second one where you have to divide into four parts because a small part of this polygon here and another polygon there which is getting into the picture.

That is a problem you have, there is a triangle here and there is a deeper polygon triangle which is on the back side and a frontal polygon here which I am drawing on the screen, then what you have is these two polygons which is split into four parts, and the background here is intersecting here and intersecting here. This means you split it into four parts but you do not have to split any further beyond this because you again have two background disjoint cases and if you watch these two carefully there are small sections of intersections which handles quadrant number two and three. Quadrant number one and four massive amount of splitting are necessary with maximum level of decomposition. In some cases it may even go to one pixel along the edges and near the vertices the splits are more and more as necessary.

(Refer Slide Time 17:55)



If you go back to the figure here in some case the splits will be handling the case where you are intersecting here as well as in these three cases but look at the sub quadrant and this part here you have more and more splits and more so along the edges and near to the vertices. You can visualize this yourself because if you workout it will take long time so I leave this as an exercise so you try it out yourself starting from scratch taking to overlapping triangles and start splitting and as you can see the splitting mostly occur along the edges. This is the scenario with only two polygons with part overlap.

Of course the previous example which you have taken was also two triangles but since there was no overlap the subdivisions where two stages but it was less. In this case if you see two polygons quite a bit extent of overlap, huge number of splitting of the area subdivision is necessary and more so along the edges and the vertex which is overlapping another polygon which is two only but if you have three or even more and as the number of polygons increase more and more you can visualize the complexity.

It is possible that almost the entire image or entire sub images the quadrants up to the first level of decomposition have to be split further and further and more so have to go to the maximum resolution of one pixel for most of these splitting cases. Most of them have to be split to the maximum level of resolution of one pixel in certain cases. Of course you will be lucky to get a background only on one intersecting case. This was the last example we talked about area subdivision method and you can almost visualize the amount of complexity which you will have and it is working in the image space using divide and conquer based method. It is not very popular method but conceptually the algorithm is very nice.

We move on the next section called the BSP tree based method or Binary Space Partition tree. BSP trees or Binary Space Partition tree is based on the concept of subdivision but in the object space. Let us look at this particular diagram, Binary Space Partition trees we

have three polygonal surfaces and I have purposely taken a curved surface. This is a three dimensional figure you need to visualize that there are two triangular polygonal patches and there is a curved surface as well, do not worry about it right now which is labeled D the C here and this polygon we will see is marked as A B we will see why. Now what is done is the following.

You imagine a plane P_1 with a surface normal pointing towards the brighter side, remember this brighter and darker side of a polygon. So that is the brighter side which is pointing towards the viewer and this P_1 plane is splitting your three dimensional space into a front partition and a back partition. This concept should not be new to you; you have seen this is many cases. A line splits a two dimensional space into two equal half planes, we have done this in clipping and we have even done this in Bresenham's line algorithm where the implicit functional form become positive zero at the line and negative below the line.

You can have a vertical and horizontal line in clipping which we talk of an inside half plane and outside half plane also. That is in 2D then in 3D also we will talk of clipping or not. You can visualize a plane and that plane will be splitting your three dimensional space or volume into two semi infinite sub volumes. So we will talk of that as a front partition and a back partition. The space in your room let us say you have to imagine a plane and that plane splits your space into a frontal and a back.

What is front and back, it depends upon the viewing direction or you have put a normal vector to that surface which is splitting the plane and that normal vector is computed based on the concept of brighter and darker side with respect to the viewer. Remember that concept of anti clockwise ordering of the vertices to get the correct interpretation on the surface normal to interpret the brighter side not the darker side. So with respect to the brighter side you have the front partition of the space in 3D and the backside on the back partition on the darker side. That is what is done in this figure where we split the three dimensional space. This is a three dimensional figure the plane P_1 splits the three dimensional space into a front or a back partition. Once that is done what will happen is this is what will result that under P_1 with P_1 as the root node.

I can visualize the binary tree with the links and the edges between the children of P_1 with root node as P_1 labeled as front and back depending upon we talk of a front partition an a back partition and under that we have the surfaces on one side of P_1 and the other side of P_1. So what is on the front side of P_1? A and C are in the front side which is the left children of P_1 and B and D are on the other side. So it is possible that this triangular polygon has to be split into two parts A and B. A will be on the front partition and B will be on the back partition. This of course is done by using a plane to split a polygon into two parts. We have done already in one of the other VSD algorithms in the last lecture.

If you remember where we used it, we used it in the case of Painter's algorithm, depth sorting where we wanted to avoid infinite loop and we used a polygon to split another polygonal plane into two parts that is what you also done here where the P_1 is used to split this bigger triangle A B into two smaller partitions and we will say that is A and B.

And then again what I do, I use another imaginary plane P_2 with this vector pointing in the front brighter direction and with respect to P_2 also I will have a front partition and a back partition. So I have a P_{2F} and a P_{2B} which basically means I am worried about the back side so I use the same plane on the frontal side so P_{2F} is on the part of the plane P_2 which is on the front side of with respective of P_1. This P_{2B} indicates part of the plane P_2 which is on the back side with respect to P_1. So P_{2F} and P_{2B} is nothing but this plane P_2 but front and back with respect to plane P_1 because I will use P2F a plane which is on the frontal side of P_1 or the front partition of P1 and I use it to sub partition on space further.

So what I am basically doing? Let us say I imagine a plane here with this entire class room split into two parts. I have a plane in between. So I have let us say this is the front plane on your right hand side and this is the back plane which you have on your left hand side. So this is your right hand side which you have the front plane and this is the back plane which you have to your left hand side.

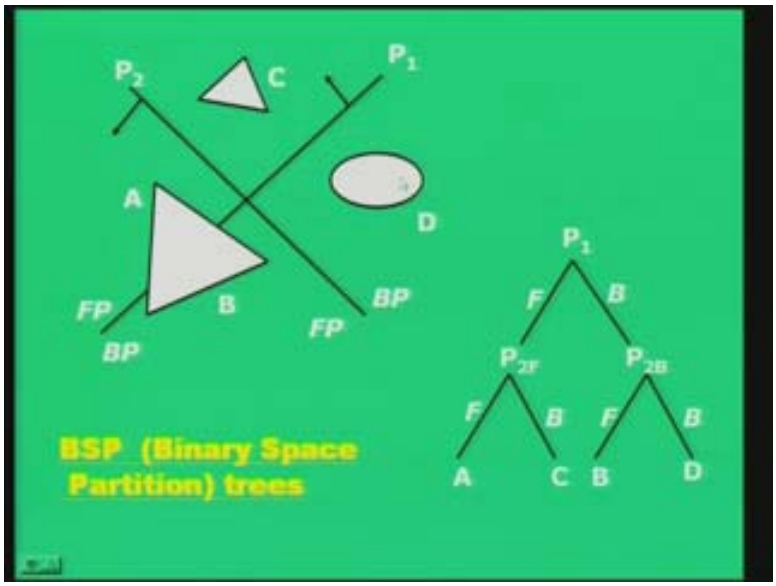
What I do now is after taking this as P_1 I use a P_2 another plane. Let us say assume for the timing that is orthogonal although it could be any arbitrary plane which is non parallel to P_1 that should be intersecting them. So what will happen? This frontal side with respective to P_1 will be further split into two parts and similarly the back side with respective to P_1 the backside which you have will also be split into two further parts. So you have four parts now something like your quad tree representation but in this case in this object space if you project this space and partition it into a two dimensional scenario projection plane you can visualize this to be something like a quad tree type of splitting in 2D. So one plane splits into two parts, select another plane and these two again into two and so on.

Hence I can use a binary tree type of representation unlike a quad tree but purposefully I use quad tree because each root nodes and other siblings of all nodes will either have polygons which are inside the 3D scene or partitions or split parts of the polygons or it could have some imaginary surfaces setting which are dividing this entire space or a sub space in 3D the front or the back at any level of decomposition in 3D into two more further parts. So each node is a plane or a polygon, the polygon plane could be a polygon out of those, we will see that and that is what is usually done and that plane is used to split that zone or three dimensional space into two further parts. And since it is always two parts which you are splitting by any plane a binary tree is a very straight forward and a simple way of representing this type of splitting.

Automatically you can simply use a binary space partition and that is why the name is given and binary tree is a good data structure which can represent this splitting of this 3D space into two parts. So let us complete this figure, as you see here the imaginary plane P_1 is splitting the space into two parts front and back and in the frontal part user P_{2F} under which you have the frontal side this A part of this bigger triangle and that is why you put A and the backside with respect to P_2 which is a right child of P_{2F} will be this polygon C.

What is the on the backside with respect to P_1? This part is where you have B and D. So what you do you use the plane P_{2B} which is this plane the backside of P_2 plane, split this plane into two further parts and the left child you have the frontal part which is B which is put here and the curved surface D is put on the right child which is because it is on the backside with respect to P_2.

(Refer Slide Time 27:50)



I hope with this simple example you are able to understand the concept of binary space partition that is representing 3D space with respect to binary trees. At the leaf nodes you will have polygons or the split parts of the polygons. All higher-ups you could have imaginary planes or it could be these polygons which also could be used to split the plane so that is what is used.

Salient features of BSP: BSP based trees or VSD algorithms are the following:
Identify surfaces that are inside or different. These words are used interchangeably inside front and outside back. So you can use inside outside or front and back as complimentary partitions. Either you can use inside same meaning as front or vice versa, outside or back will be used interchangeably. We will keep the word front and back but you can also used inside or outside. You remember the inside and outside for the line within 2D space or a plane with respect to 3D inside half plane outside half plane. So the similar concept here front and back or inside outside are used interchangeably.

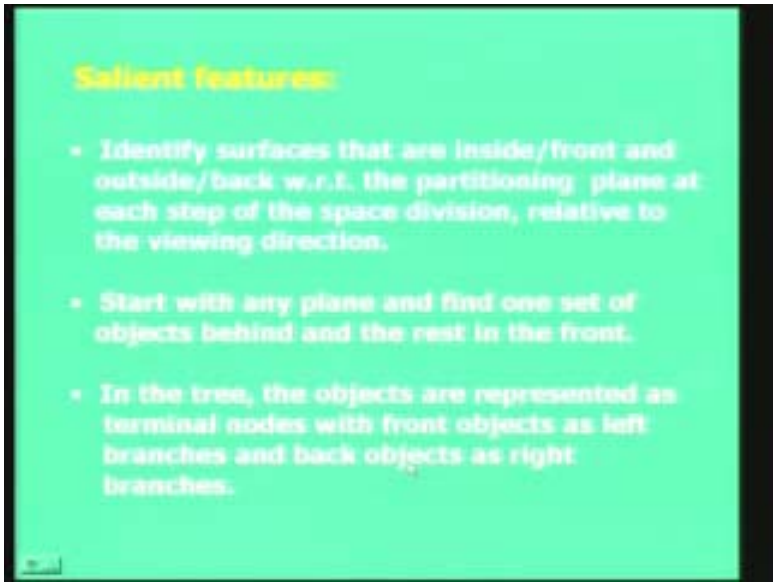
So coming back to this sentence again, identify surfaces that are an inside front as and outside back with respect to the partitioning plane at each space of sub division relative to the viewing direction. I repeat again, identify the surfaces that are in front and the back of the partitioning plane at the each space of the sub division relative to the viewing direction. And start with any plane and find one set of objects behind and rest in the front.

Now, we talked of this previous example as an imaginary plane P_1 and other imaginary plane P_2. But nobody stops you from taking one of the n polygons in the 3D scene as your imaginary plane at the plane which you are using as your root node and split the entire 3D space into two parts and after you have classified or categorized or grouped the remaining planes the n minus 1 planes into two halves you use one more plane each from each of these categories to split into further subdivisions to construct this binary space tree or binary tree as it is called. So the imaginary plane may not exist in the focus and that will add to the computation. You have to actually come up with equation for the planar surface so why you do not use the polygons which themselves could be planes, so that is what is used. Start with any of these planar polygons as a plane to split and find one set of objects which are behind that is one category and the rest is the another category which is in the front.

In the tree the objects are represented as terminal loads with front object as left branches and back objects as right branches. So I already told you because the binary tree will have a left sub tree or siblings in fact not siblings is a left sub tree and a right sub tree and the left sub tree will contain all the polygonal phases which are in the front with respect to that plane.

We are talking of a sub space now. It could be the entire 3D space or a small sub space in object space and you have taken that small sub space which could be a spherical structure or a rectangular parallelepiped or anything you can visualize in this case. But of course it will come as a rectangular parallelepiped here, we are doing like an area subdivision in 3D by splitting into two parts and we have some planes on the front side and some planes on the back side with respect to a clipping surface either at the root node or any other node which had children and then what you have on the left branch of the left sub tree of that particular node you have the polygons, this will be your left child and this will be your right child and then you will have the faces of the right child which are on the back side or outside of the polygonal surface. I repeat the sentence, in the tree the objects are represented as terminal nodes with front objects as left branches and back objects as right branches.

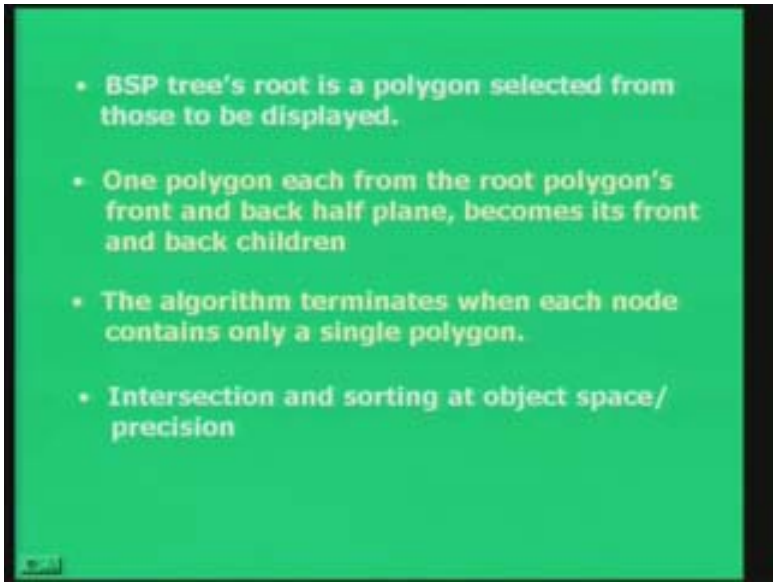
(Refer Slide Time 32:01)



Continuing with the points of BSP tree, the BSP tree's root is a polygon selected from those to be displayed. You do not need to imagine a polygon even to start with or any time that is later on when you are further subdividing with respect to planes. Take one of the existing polygons and the BSP's tree root is itself one of the polygons. So the BSP trees root is a polygon selected from those to be displayed and one polygon each from the root's polygon front and back half plane becomes its front and back children. This is what I was just saying that you do not have to imagine a polygon.

Select one out of the n and the remaining n minus 1 is split into two parts, two categories. So within these two categories select one in the front and one among those in the back and that becomes the corresponding children of the root node. And this process continues recursively till you stop only at one area, just one phase to concurrent the node of phases something like a area subdivision but in 3D. So one polygon each from the root's polygon front and back half plane becomes its front and back children. The algorithm terminates when each node contains only a single polygon, this is true. Intersection and sorting at object space precision is the key idea.

(Refer Slide Time 33:26)



In area subdivision method we worked in the image base this is completely object space. We do not worry about the image space at all except looking at the viewing direction that decides the frontal and the backside sort of thing. Of course back face culling can proceed BSP tree to help reduce the number of polygons which you have to process but after that you just worry about the viewer direction and that viewer direction dictates what is going to be your brighter side to calculate the surface normal and after that you just use one plane into split into two parts and split each of these planes further by the help of one polygon each selected from each of this category and so on and then you keep dividing till at the leaf node to your left. Leaf node cannot be a null point, in this BSP tree it must be a single polygon.
It should not be two polygons, a single leaf node must be one polygon from the list of n polygons and that is what you have. So you can visualize that then for the average case for a BSP tree you can have login complexity whereas in the worst case you can actually go to n and we will see that. Intersection and sorting is your object space precision.

Let us take an example now in the list. As you can see here there are basically five polygons numbered one two three four and five. Remember now we can look into this 3D scenario from an arbitrary direction, polygonal planes as if you are not looking always into the viewer direction that is not the case here. But I have purposefully shown that the planar face is not like this in the case of as you have done for area subdivision method but as if you are looking into polygonal faces in this fashion. The viewer direction can be anywhere involved in talking a planar surface and splitting the object space or the entire 3D space into two parts so that is what we do with an arbitrary number three.

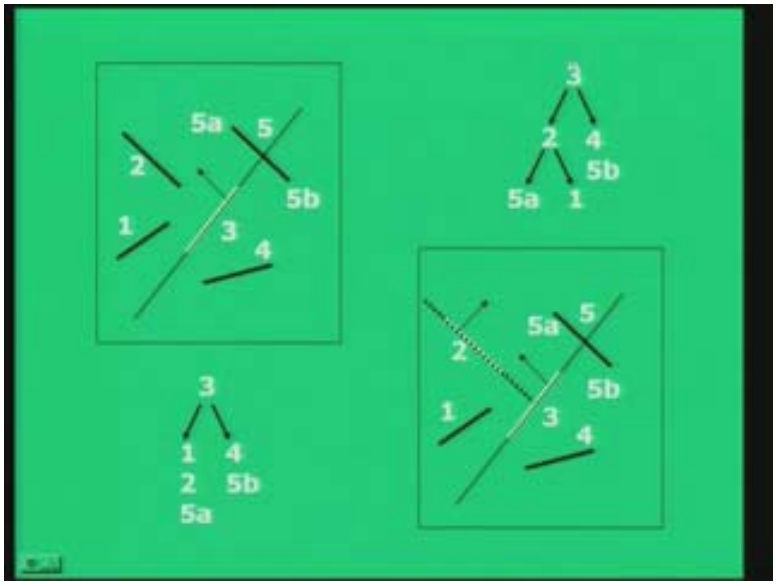
We select the plane number three and split the space into two parts and if this is the frontal vector the surface normal of the tree pointing to the brighter side with respect to a viewer direction then what you will have? On the frontal part of the tree you will have polygons one and two and you will also have to split the polygon five this was the polygon five into two parts which will result in five A on the frontal side and 5b on the back side.

So on the left after the first subdivision with respect to polygon three as the root node you will have one, two and five A as the children of the tree, right now it is not a binary tree in that sense because there are three siblings and so what will happen is on the back side of the tree you will have 4 and 5b that is what will be the result of this. It is a tree with one root node as the polygon tree and the left branch contains the polygons which are on the front side which are 1, 2 and 5a, I am repeating which I just said sometime back and the right pointer or the right sub tree will have 4 and 5b.

So you need to split further because the nodes do not have single polygon. So what you do? You take this structure once again and look into the left hand side which as 1, 2 and 5a and use the polygon number 2 to split the space further the frontal part of the sub space the sub space which is on the frontal side with respect to polygon number three as given here you split it again into two parts with the help of the polygon number 2. So with respect to the frontal side of 2 becomes the left child of root node 3 you have 5a to your left and one as the right and that is what you have.

So that is what you will have with respect to the 3. If use 2 as your left child for splitting the sub space into two parts you will have 5a to your left and one on the back side, that is what you will have and still the construction of the sub tree is not yet complete because the right child of the right sub tree in fact of the root node 3 is not split, it has to be split further because it contains more than one polygon. If there was one polygon only for 5a or 5b that would have been the end of the BSP tree construction but you need to select one out of 4 and 5a as your node for the root corresponding to the that part of the sub tree which will be the right child for the root node three.

(Refer Slide Time: 00:38:06)



So that is what we do, 3 has been selected to split first then the 2 and then we select 4and what will happen the left, it is interesting to note may be I made a wrong comment

earlier, the frontal part of two will be a null pointer because we do not have any node polygons to the frontal side of 4 back side you have 4b that is what you have. So the node will not have any left child because there are no polygons to the frontal side of 4 with respect to this normal vector so that you can have a null point with 4 and then you have the 5b to the back side. Do not worry about this 5b vector normal here that is not use in this calculation but if necessary you could have used that as well.

I hope this construction of this BSP tree is very clear to you as how you take one split into two groups and then in the first group take an other polygon split into further branches take the other partitions split into other branches and so on and you construct a binary tree it becomes a binary space tree or a BSP as it is called because it is representing your three dimensional structure into two equal halves and sub dividends and so on, that is what you have to do.

The question which will come to your mind now is what happens to the BSP tree if I had not started if you did not start with the polygon 3 as your root node. If you have not started with polygon 3 as your root node of course you will have a different BSP structure. Remember, the scene does not change, the scene does not change but you may result in a different type of a BSP structure completely. So let us look at a particular example but before that this is the final thing which you will have that when we are trying to consider now these three prospective polygons for splitting, remember the previous problem is over where this was the BSP tree constructed out of these by choosing 3 first then two and then four in that sequence you are choosing and this is the BSP tree which you get.

Now I choose a different sequence. What is that sequence? I have leveled some of these with a white color plate to indicate that is what I will choose. I will choose polygon five polygon labeled 5 as my first root node and that you can immediately visualize what is going to happen because if the front phase of the 5 with respect to the viewer direction is pointing as shown in the figure whether no polygon is to the front of 5 all the rest of the polygons will be towards its right sub tree. The left sub tree will be a null pointer for this particular BSP construction if you choose 5 and the front either if you choose the frontal phase as given or the reverse one of the sub trees with respect to 5 would have to be a null pointer it will not avenge any source and that is what will happen.

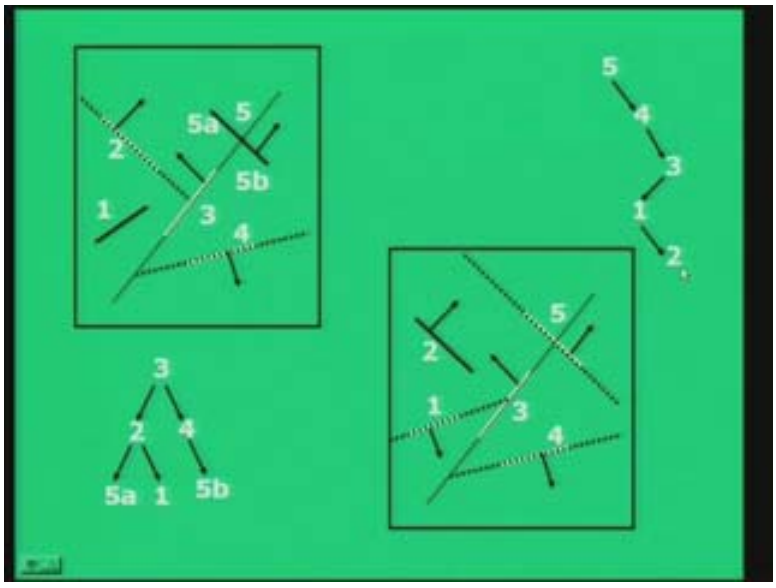
We will continue the sequence and then construct the tree, you choose 5 first then we choose 4 to further sub split then we choose 3 and then we choose 1 that is the sequence. Just remember the sequence, I will play it back for you, I choose number 5 first please note down this sequence that is the one which is blinking right now after the plane 5 or the polygon 5 has been chosen as your root node to construct the BSP to start with I will then choose 4 to split further then polygon 3 or the third polygon to further subdivide and then the polygon 1 to further subdivide this is the sequence.

I leave it as an exercise for you to visualize, we have already done the first step where I chose polygon number 5 where with respect to the left sub tree will be null and the right sub tree which is your right pointer will have the rest of the other polygons and then we

choose 4, 2 and so on and you will be having a structure like this. It is very very interesting as you see here, remember the arrangements of the polygon in 3D space remaining the same depending on the order which you choose you will be reconstructing or constructing different types of BSP trees is to represent the same structure and this is easy for you to visualize because I said earlier with respect to 5 its left sub tree will be a null pointer you will not have any children there and then everything is towards its right because it is in back plane.

Similarly, with respect to 4 everything is on its back plane nothing on the front so left pointer will be a null pointer for the left sub tree for 4 with respect to 3 everything is in front which is 1 and 2 so the right pointer at the right sub tree will be a null pointer for the node 3 here and then with respect to 1you again have its left sub tree which is a null pointer and finally finish with the 2 which is to the right child the only leaf node available in this construction with respect to node 1.

(Refer Slide Time 43:32)



Remember, in the previous construction as given here you had 5a 1 and 5b. There are three polygons which where at the leaf nodes and in this case there is one polygon which will be in the leaf node. So it all depends on the order which you are choosing. That

means one point you must keep in mind here which we will again discuss later on may be in the next class that BSP tree is not a unique representation for a given three dimensional structure.

Of course you change the structure the BSP tree changes but for a given set of 3D polygons you do not alter the viewing direction and the positions and the orientations of n set of polygons. Whatever be the value of n you can have different BSP trees or Binary Space Trees constructed from these depending upon the choice of the order in which you choose the polygons. It depends upon the first one which will choose get subgroups and again within the subgroups if you choose different combinations you can have various possibilities of different BSP trees constructed. Which one is the best one? Why do you need a BSP tree to do a Visible Surface Detection that we should look first after the BSP tree construction and then for this example we will come back and find out which is the best one.

So what you do? Display list for a BSP tree, you need to display this list of a polygonal surface in some order for a BSP tree to implement Visible Surface Detection and to do that the BSP tree must be traversed in what we call as a modified in order tree walk. I hope you remember your data structure and algorithms concepts where in one of the lectures you would have seen an in order traversal. In order traversal is what you do and then we will see what is your binary tree and we will see what is a modified in order tree. So come back to this concept here where we talk of a BSP tree may be traversed in a modified in order tree traversal walk to yield a correctly priority-ordered polygon list for an arbitrary viewpoint to implement the VSD.
What is this modified tree? If the viewer is in the roots polygon front half space that means we assume the viewer is to your left child the algorithm for modified in order traversal must first display all the polygons in the root's rear half space that means towards the right child first then the root and finally all the polygons in the front half space. What was in order traversal for a tree? Recursively define in terms of left sub tree first then the root and then the right sub tree and this is recursively applied to the sub trees.

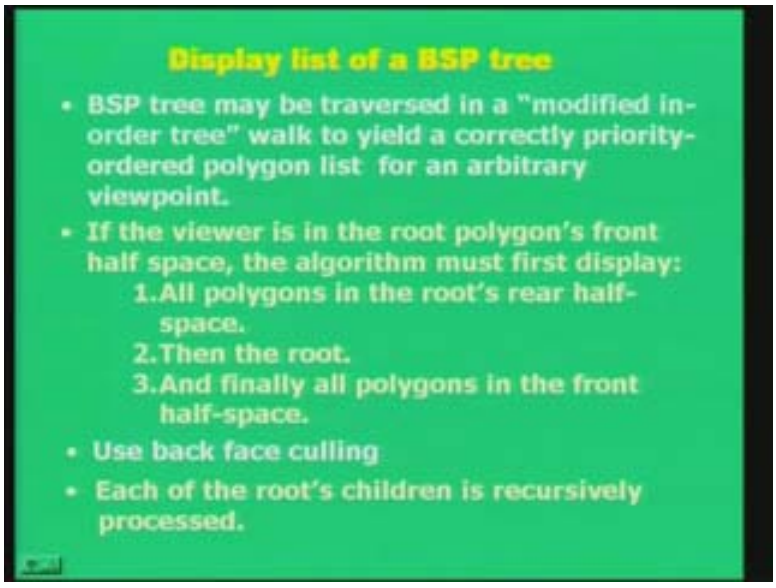
I repeat it again, in order traversal talked of left sub tree hierarchically, recursively then the root node and then the right sub tree. Modified is just the reverse where I first traversed just the reverse path. Again recursively, hierarchically right sub tree first which is the rear half plane polygons first because they are at the back side with respect to the viewer.

You are assuming here that the viewer is with respect to the root polygon which you have selected which is in the front half space that is what we used to define the normal the brighter side of that polygon surface normal which you used to define and we discussed about that when you selected the polygon and talked of this brighter side darker side select the surface normal and based on that select the front partition and the back partition.

So you or the viewer is in the front partition with respect to the root polygon in the BSP tree and if that is done display the ones which are behind your VSD, the z ordering, depth ordering and so on so we display the one which is behind then the root and then which is in the front. So you make the modified in order traversal walk by doing your right child then the root and then your left child in your modified space 1.

In the other case in the original in order, remember I repeat again, you would have had the left child first then the root and then the right sub tree or child whatever the case may be. In the modified in order the reverse happens, you start with the rear half plane then the root and then your left sub tree. Visualize, we will do that if the viewer is in the root polygons front half space the algorithms must first display all polygons in the roots rear half space then the root and finally all polygons in the front of space. Of course you must use back face culling prior to this through out all the back faces then construct a BSP tree then make a modified in order tree walk and then each of the root's children is recursively processed in the modified in order tree walk.

(Refer Slide Time on 00:48:07)



So this is the concept of the display list to construct a BSP tree and then walk through it modified in order traversal you have the correct ordering of the surface. BSP tree is not an order. Finally you must have a sequential ordering. BSP tree gives a structural

representation in a binary tree form of the space and that is what is called Binary Space Partition tree or a BSP tree. You have to do a modified in order traversal or a walk to have the correct ordering of the surfaces with respect to the viewer and in this case remember we are assuming that the viewer is in the front partition with respect to the root polygon. That is what you have.

Disadvantages:
More polygons splitting may occur than in Painter's algorithm. Yes, because you are using polygon to split one another just like that without checking the area subdivision method. Of course it may not be as bad as area subdivision method but with respect to Painter's algorithm you may have more splitting and appropriate partitioning the hyper plane selection is quite complicated and difficult. This is the key idea, I just talk about a scenario where you may not have a unique representation of a BSP tree that is fine.

So what is optimal? It is very difficult for you. You can spend a few minutes to discuss this because based on the root polygon which you select to create the BSP tree and then the further polygons in the different sub groups which you select that is going to dictate what is your BSP tree which will come out. Whether it will have maximum depth or an optimal depth log n or even order in for n polygons will depend on two facts.

Of course the distribution of n polygons in space is number 1, number 2 is your choice, you can have a random choice nobody stops you from doing that. BSP tree construction does not give you any guidelines to follow you can randomly or abruptly choose any polygon to construct the root and then the further sub trees you please go add and do that but you have to pay the price two, two prices. First of all you do not know the final outcome of the BSP tree which you are going to construct and that could cost your polygonal space walk for the modified in order traversal that is number one. And the computation burden in constructing the tree could be more costlier if polygons are not chosen in an order form.

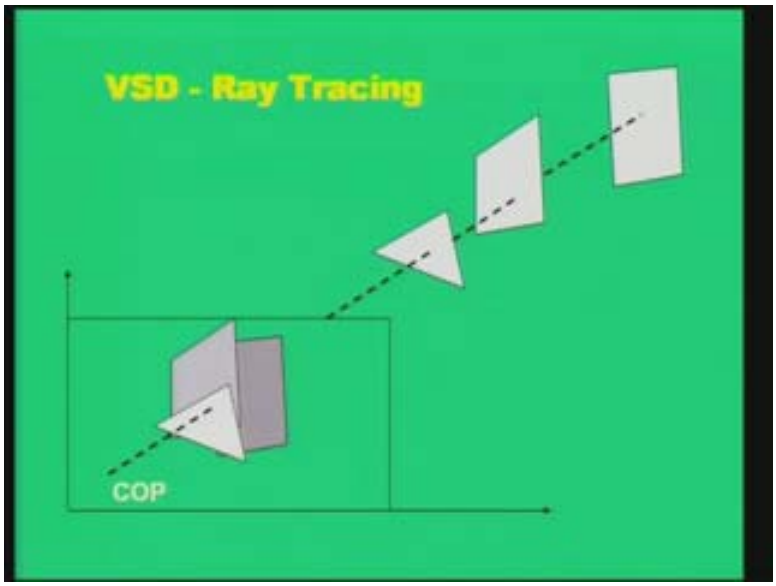
How do you choose in order form? Then you have to compare each polygon with rest of the n minus 1 polygons that could even be more costly if n is very large. If you need to compute whether you have an optimum polygon positioned somewhere in the middle with respect to the other polygons if you have to find out computationally where is that polygon to construct an efficient BSP tree that will be further costlier because recursively you have to keep doing that for the two sub groups further sub groups and so on. That is why BSP tree does not say do not try to find at optimum polygon, use a random one.

Of course there are some efficient algorithms that should construct an efficient BSP tree outside the scope of these lectures in terms of research and BSP trees and partitions but if you choose that it could be a very big computational burden. You want to avoid that because that could be costlier than doing this costlier modified in order traversal walk. You better consider the sub optimal tree or a very long depth tree which you have to walk more that will be easier for you than trying to find out or to construct an optimal BSP tree and at each stage finding out an optimal polygon to let the root of your sub tree or the main tree.

So BSP tree does not give any guidelines you can have a random one. So depending upon your random choice, depending upon your same construction you are not going to have an optimal or a unique BSP tree that does not exist in any unique BSP tree for a 3D object representation in numbs of n polygons and not only that the BSP tree which you construct is going to be random. It is not only going to be unique it will have different structures based on your random choice and that will give you different walks but that walk could also result in different sequence of ordering but it will give the correct depth representation.

I leave it is an exercise for you to check with those two examples which you have taken with 5 polygons where we have two different ordering of different surfaces as a root nodes and we have constructed two different trees. Apply, modify in order walk on both of these and see the order. See the order of the display list at the same amount. Take this as a home assignment with those set of polygons or even modify that structure of the polygons in 3D come up with two different types of BSP trees construction based on random ordering and then have a modified in order walk and see what is the result.

That is where we conclude the discussion on BSP trees with the small amount of time left. I will introduce the most important concept of VSD algorithms which is called the ray tracing and we will discuss most of these in the next class. Ray tracing is one of the most sophisticated methods. And if you see this particular figure you will see that this concept is similar to the concept of Z-buffer or depth buffer algorithm which we discussed earlier. Depth buffer also consisting of n polygons but you worked in the object case.
(Refer Slide Time: 00:54:24)



In the case of this ray tracing it is a combination of object and image spacing but mostly we work in image space. What is basically done is, this ray in depth buffer which you are talking about as a light ray coming out in an canonical view volume orthogonal

projection case can be now visualized to be a ray coming out of a pixel from the projection plane x y and passing through all these polygons. And that is what we do we actually reverse trace the light path which is coming out of surfaces and projecting on to a projection plane we reverse the path of the ray of light and come out of the projection plane and enter into the object space and that is what we look at.

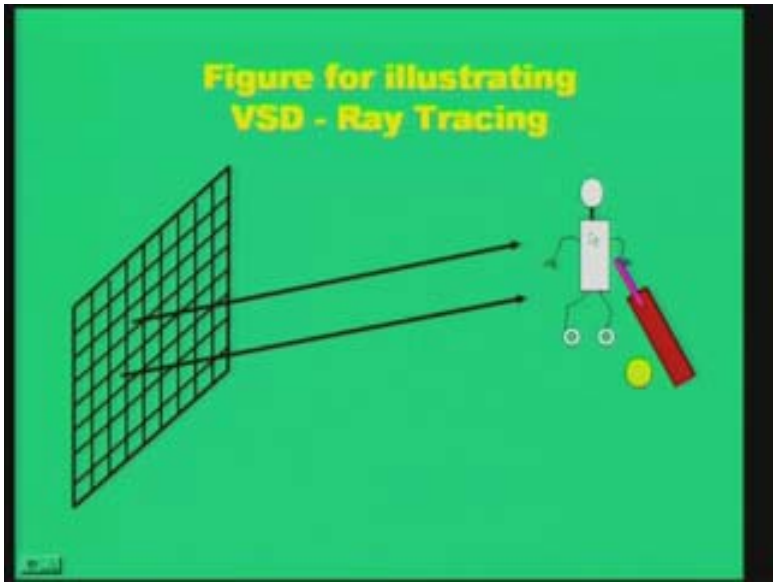
If you look at this particular diagram where I am using a funny creature for illustrating a Visible Surface Detection ray tracing algorithm we are talking of an image space on the left hand side. Grid of pixels here and for each of pixel I have only drawn two rays but if there are n into n amount of pixels depending upon the resolution of this image buffer there could be n horizontal rows and m columns or vice versa.

You will have m multiplied by n different pixels in the image buffer and you have to reconstruct all the m multiplied by n different rays. So in this case I am only considering this figure two out of these m into n different rays and for each ray I drag out the ray in the reverse from the pixel array from each individual pixel, trace the reverse path of the light direction. Remember, the light will fall on the object, we will talk of shading models but you can imagine the light rays will fall on this object and move towards the pixel array but here I am going into a trajectory in the linear path in the reverse direction of the light.

Remember, the light will fall on the object and hit the image plane but on image plane I am moving towards the object in the reverse direction of light source and find out hit the object, hit the polygon. Hit the polygon and find out which polygon is hit first and from that polygon find out the light source and shade it.
Of course we will not talk of any shading algorithms here which we will be pushing when you talk of illumination and shading. But right now ray tracing involves taking a ray out of a pixel and going and hitting a polygon which will you hit first. Let us say there is a polygon one, polygon two, polygon three and so on. So you take a ray out and find out which polygon is hit first and shade that polygon. Do not shade that entire polygon shade that particular pixel with respect to that polygon’s illumination properties. So that is what you will do here as you see.

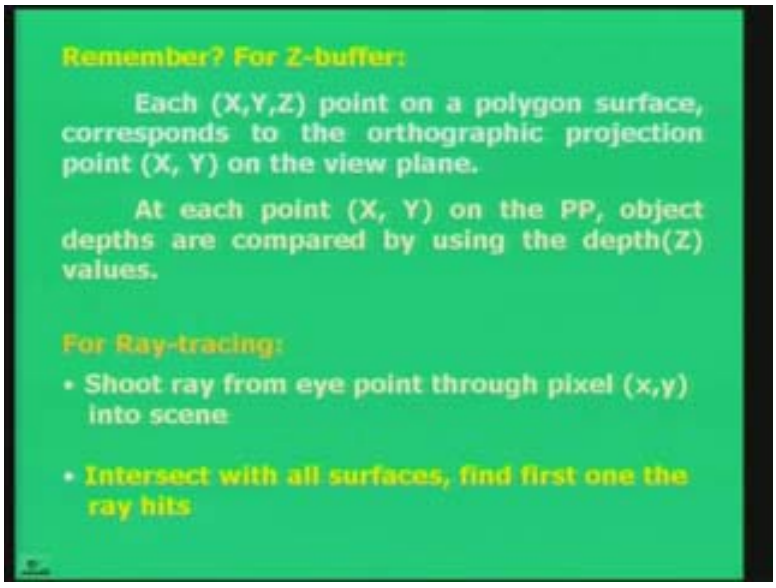
We will conclude this lecture with these two examples and talk of more points in the next lecture when you talk of visible surface rendering here a ray coming out of a pixel orthogonal or perspective projection does not matter and out of all these n three polygons here you have to find out which is the first polygon which will be hit first because that ray will not go and hit the other polygons and that polygon will be used to render that particular pixel only. So, for each pixel you will have a single ray and that is what will be used to render that particular polygon.

(Refer Slide Time: 00:55:20)



We will just wind up with a few points for Z-buffer for each (X, Y, Z) points on a polygon surface which corresponds to the orthographic projection point (X, Y) in the view plane. At the each point (X, Y) point on the projection plane the object depths are compared by using the depth buffer values. And for ray tracing shoot ray from eye point through each pixel (X, Y) into the scene, intersect with all the surfaces find the first one which the ray hits and that is the surface which is used to render that particular pixel.

(Refer Slide Time: 00:56:55)



We will start from this slide onwards in the next class and discuss details of the ray tracing which is the most sophisticated among all the different VSD algorithms which we have discussed so far, thank you very much.

.