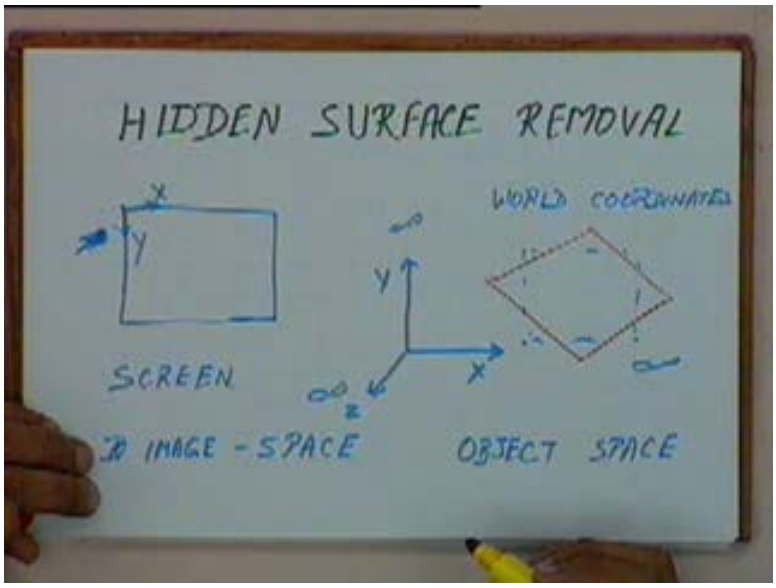
Computer Aided Design
Prof. Dr. Anoop Chawla
Department of Mechanical Engineering
Indian Institute of Technology, Delhi
Lecture No. # 12
Hidden Surface Removal (Contd.)

Today we will be talking of hidden surface removal. In the last class I had introduced this topic. Today we will see a set of algorithms which can be used for removing hidden lines.

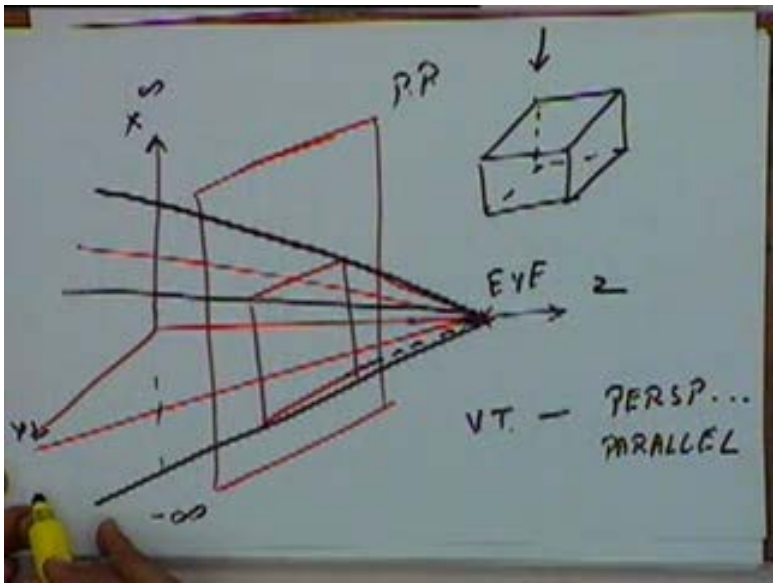
(Refer Slide Time: 00:01:28 min)



We will see two types of algorithms, one type of algorithm consists of those algorithms which operate and what are called the device coordinates or the image coordinates or they operate in the image space. The other object, the other operates and what is called as object space. Typically when we are talking of the object space, we are talking of the world coordinate system where the space is infinite in all the three directions. This is the world coordinate system. If an algorithm is operating in an image space we mean that we are talking of the screen coordinates, a portion of this world coordinates has already been transformed on to the screen.

We already know that a finite portion of this world coordinates is going to be mapped onto the screen and then the hidden surface removal algorithm will operate on this transformed image. The two, they are two different types of algorithms. Student: Suppose you have defined a polygon in the world coordinate system which is large enough and you want to transform it on the screen. So you want the polygon or the whole polygon to be transformed or the few portion of it. See if we have an object in this world coordinates like this, since we were talking about three dimensional space, this object will be clipped according to the volume that is going to be displayed.

(Refer Slide Time: 00:03:25 min)



Let's say we are looking at the world coordinates system from this point in a perspective transformation, in a perspective view. A view is somewhat like this and we define that on this projecting plane, only a finite portion of it we are going to see which means that if we take rays starting from here, these are arrays like this. If you look at this pyramidal structure, only whatever is inside this pyramid that will be visible whatever is outside this pyramid will not be visible. Is that all right? So this coordinate system is my world coordinate system, so let's say this is x y and this is z direction.

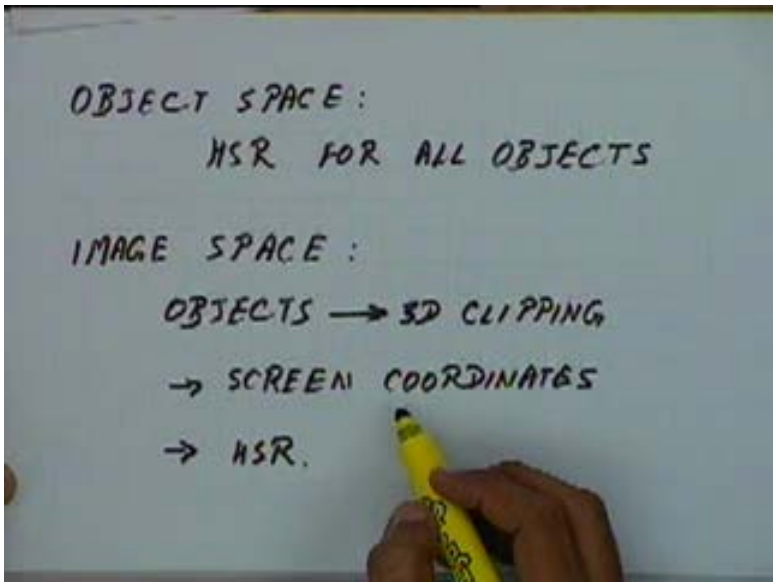
Now in the world coordinate system I have defined a finite volume or I have defined a finite portion on my projecting plane, whatever lies inside this projecting will be visible that will correspond to a pyramidal volume. So an algorithm which is operating on the object space, we first look at the object in the world coordinates, it will clip it in three dimensions such that only portions inside this pyramid are visible. Sir, we are going to choose our own plane in our own way. All that will be specified, this projecting plane, position of the eye in the case of on the perspective projection or the direction in the case of parallel projection. All that will be specified, once we know the viewing direction, once we know the projecting plane only then we can talk of removing hidden lines.

for example if we have any object, if we have a block if you locate it from this direction or you locate from the other direction, in both the cases the portion that will be hidden will be different. Is that all right? That is why whenever we are talking of hidden surface removal, whether you are talking of the world coordinates or the screen coordinates you are always talking of with respect to a specified viewing direction. If you have an object like this, if I am looking at it from a direction which is from that side then a set of planes which will be visible will be totally different. So if I have my world coordinate system, whenever I am trying to remove hidden surfaces or hidden lines I have to specify in my viewing transformation. So the viewing transformation you have already seen that can either correspond to a perspective view or it can

correspond to a parallel view, whichever view we take the viewing transformation has to be specified.

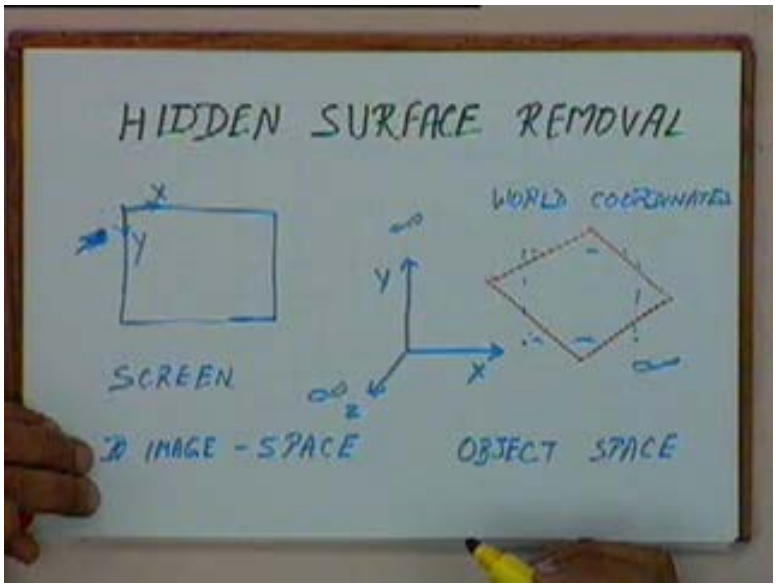
Once these viewing transformations is known then we will talk of a hidden surface removal. An object sorry an algorithm which operates in object space, we will look at all the objects in the world coordinate system, in the world coordinate system a space is infinite in all directions. It will extend from plus infinity to minus infinity in x direction and in the other direction as well. It will locate all the objects and then decide which objects are closer to the eye. While when you are talking of an algorithm working in screen coordinates, it will take a finite plane, a finite portion of it, a finite portion on the projecting plane, get a three dimensional pyramid and it will only look at objects which are inside this pyramid.

(Refer Slide Time: 00:08:09 min)



So an algorithm which is operating in object space this will do hidden surface removal for all objects. While an algorithm operating in image space, I will take the objects and these objects will undergo a process of 3 D clipping. 3 D clipping means only objects inside this clipping volume will be displayed, this pyramidal structure or this pyramidal volume in the case of perspective view is called the clipping volume. Only objects inside this clipping volume will be considered and those objects will be transformed to the screen coordinate system, the screen coordinate system is this coordinate system.

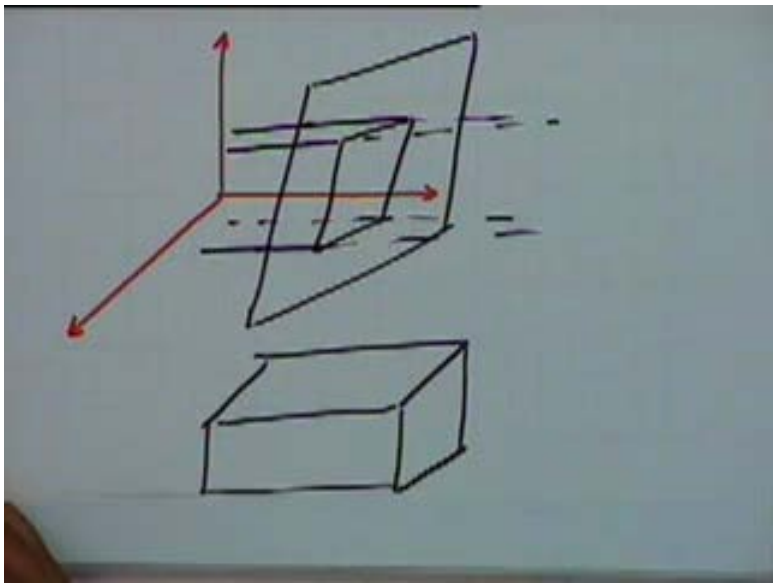
(Refer Slide Time: 00:09:27 min)



That means if the resolution of the screen is 1 0 2 4 cross 1 0 2 4, all these objects will be defined between coordinates from 0 to 1 0 2 4 plus there will be some z value. So when an algorithm operates in image space, the objects first undergo 3 D clipping then they are transformed to screen coordinates and then we decide which objects are closer, which lines are closer and which lines are at the back. So hidden surface removal in image space is carried out after 3 D clipping, after transforming all the points to screen coordinates. While when you are operating in object space, hidden surface removal is carried out for all the objects and then will carry out a process of clipping which will be in two dimensions now. Because we have an infinite world, from the infinite world we have taken a projecting plane and decided which objects are closer.

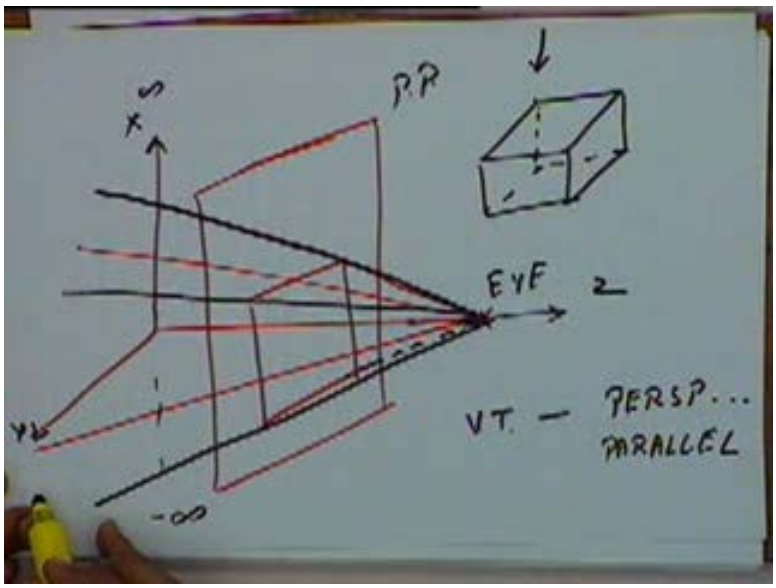
We have transformed everything onto the projecting plane now. So now the operation of clipping that will be involved will only be a two dimensional clipping. So in this case only 2 D clipping will be involved and this case we carry out 3 D clipping and then hidden surface removal. So I have two different types of algorithms in these two cases. One thing you should keep in mind in the case of 3 D clipping, right now I have shown a pyramidal clipping volume, it need not be a pyramidal clipping volume. If you are talking of a parallel projection then the clipping volume will be a block that will look like this.

(Refer Slide Time: 00:11:56 min)



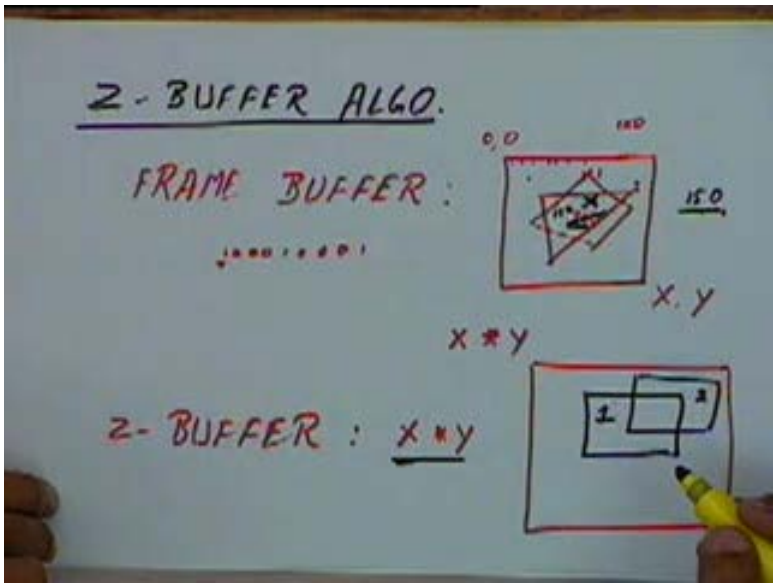
This is a projecting plane and we define a window on it like this, my clipping volume would look like this. That will be a block kind of clipping volume and typically when we define a clipping volume like this, we also define a front plane and the back plane you will see all objects behind that back plane will also be clipped off. So the clipping volume in this case will look like this. What is need for this back plane? the need for the back plane is just that in some cases, we might like to say that all of this plane and certain plane are not be considered, it's just that nothing beyond that, we need not have it. For your back plane and the front plane, all objects in front of a plane that also you might like to clip them out. So we normally define a clipping volume which will consist of 6 planes like this.

(Refer Slide Time: 00:13:25 min)



In a case of perspective also we will have the same 6 planes but they will follow a first term of a pyramid, there will be a front plane and there will be a back plane.

(Refer Slide Time: 00:13:43 min)



Now let's look at some clipping algorithms. The first algorithm that we will consider is what is referred to as a Z-Buffer algorithm. Now this Z-Buffer algorithm, it uses what is an extension of a frame buffer. You remember what is a frame buffer? What is a frame buffer, anyone? They store the information which is read by the controller of. What information does it store? Different entities. Different entities. No. Student: the information could be drawing where you could have this thing and say in line vector. Coordinate thing, no you all have forgotten everything. No. which pixel is off and which pixel is on? 1 0. A frame buffer stores the information about each and every pixel. So if we have a screen like this, this is from 0 0 to xy, this is an array of pixel which is x times y.

For each pixel we have a bit which describes whether that pixel is going to be on or off. If it is 1 it is on, if it is 0 it is off that is in the case of monochrome display. In the case of a color display you will have multiple frame, multiple planes so this is what is a frame buffer. Now when we are talking of a Z-Buffer or a Z-Buffer, the same formula the same concept is extended and instead of a bit, at each pixel will store the depth of the pixel also. That means let's say if you have a plane which is being displayed like this, at this pixel let's say the depth of the entity that is being displayed is 100 units. At some other pixel the depth might be different let's say at this pixel, the depth is 200 units. So we will also maintain a Z-Buffer which will store the depth of the pixel being displayed at each pixel. So the resolution is x times y will use a frame buffer which will be of the size of x times y.

Sir, this stored information only for the 0, the pixels which are off. For the pixel which are off the depth that is being displayed will be some default value at infinity. The minimum z value that we have that will be stored initially for all the pixels. Then what we will do is let's say this is one polygon that we have to display and we have another polygon to be displayed like this. This is

polygon number one, this is polygon number two. A further polygon number one when we came at this pixel we decided that it has a depth of 200 then we are looking at polygon number two and again we are trying to display the same pixel, let's say the depth of this pixel is equal to 150. This value 200 is available in the Z-Buffer. So while displaying polygon number two we will compare the depth of this pixel that is 150 with the depth which is available in the Z-Buffer. Where do you store 150? See 150 is the depth of this polygon, we are trying to display this polygon. When I am trying to display this polygon, I know the equations I know the equation of the plane so I can calculate the depth at each and every point and this value there will be stored in the z value.

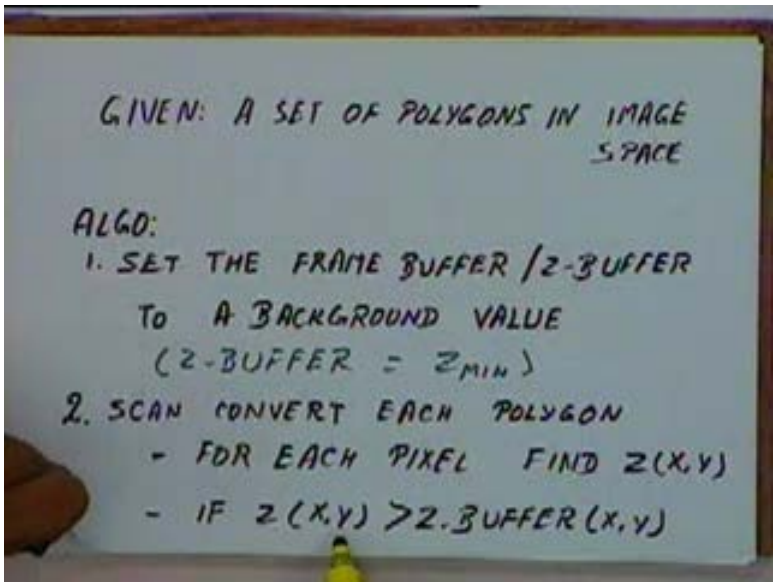
Now I will compare this 150 with the value which is stored in the Z-Buffer and decide which pixel is closer to the eye. 150 is closer to the eye, so I change this to 150 and I will decide that at this point pixel number two is to be displayed. So we use an array of the size of x times y, x and y are the resolution of the screen in the x and the y directions. In the x direction the resolution is x, in the y direction the resolution is y. So we will use buffer or memory space which would be of the size of x times y and it. Again. Clipping are they are storing the information regarding which polygon it is. Yeah, which polygon and the display information with respect to that will go onto the frame buffer. Student: And this will also go. That information will also go. If I am displaying polygon number one let's say that is of a different color or it has a different intensity that information will be stored in the frame buffer.

The moment I update it by the information of a second polygon, the display characteristics of the second polygon will be put in to the frame buffer. So let's just see this, the steps involved in this algorithm. Student: Type of contains the depth or the nearest point. The nearest point that is currently being displayed at that particular pixel. Sir, one minute see. No, for all the points it doesn't compare and then store the nearest point. What you mean all the points, comparing this point with this point. Student: No, comparing this point with the same point which would say be there in other. So it will compare this, the depth at this pixel for polygon number one with the depth at this pixel for polygon number two. So at this pixel it will compare the depth of each of the polygons and whichever is the closest to the eye that will be displayed. Sir, you were saying something. Student: Sir, suppose we have suppose the first polygon is a rectangle. Then every pixel which is off outside the rectangle will have the value infinity and inside the pixel which are all inside the rectangle both values it will have.

See this way your screen and you have this rectangle to be displayed. When I am displaying this rectangle all the pixels inside this, they will take a z value which will correspond to the depth of the polygon of the rectangle at that position. No, when I am talking about a polygon I am talking of a planar surface, I am not just talking of a single polygon. We are talking of planar surfaces, if we are only talking of a wire frame kind of situation then hidden surface removal is not relevant. In a wire frame kind of a situation even if I have a rectangle behind it, even that has to be displayed. Whenever I am talking of polygons, I am talking of planar surfaces bounded by these polygons. So, all the points which are inside this rectangle, the Z-Buffer for those pixels will take a value corresponding to the depth of that polygon, depth of that rectangle. Sir, in the Z-Buffer we have finally showing for the intersection between polygons only the nearest points. That's all.

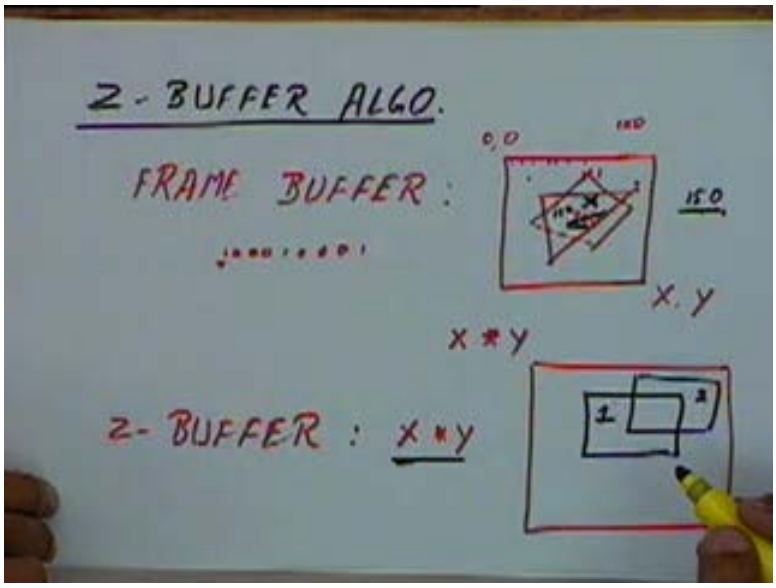
So what will happen is for this portion, the depth of the first rectangle will be shown. For this portion, the depth of the second rectangle will be shown and for this portion the nearer of the two will be shown but they can be intersecting. So if they are intersecting in that case where part of it, one polygon will be shown for the other part of it a second polygon will be shown. They can have an intersection which is something like this because we were talking in three dimensions. So let's see the different steps involved in the algorithm.

(Refer Slide Time: 00:22:58 min)



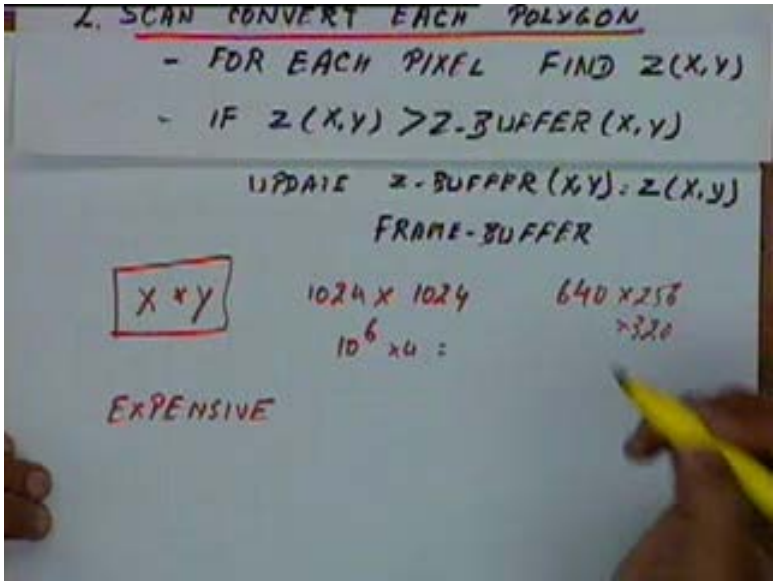
What are we given? We are given a set of polygons in, are you working in image space or in object space? I have described the algorithm, are we working in image space or in object space? Image space, we are working in image space in this case. So we were given a set of polygons in image space and what do we have to do? We have to remove the hidden surfaces and display all the polygons and what is the algo involved? The first step that is set the frame buffer and the Z-Buffer to a background value. A background value in the case of a frame buffer, it will have some background intensity or background color. What that basically means is initially we will set the whole screen to correspond to as just nothing is being displayed. That means whatever is the color of the background, whatever is the depth of the background that will be initialized to all the pixels that is the first step. This means that the Z-Buffer will be put equal to some minimum value and the frame buffer will go to the colour and then intensity of the background. Then what we have to do is for each polygon to be displayed, we have to decide the colour intensity as well as the depth of each pixel. Like in this case we first have to decide for this polygon what is the colour intensity and depth at each pixel, how do we do that, what algorithm we will use?

(Refer Slide Time: 00:25:28 min)



Polygon filling or scan conversions, so we will use a scan conversion algorithm. In this scan convert each polygon and while scan converting we will say for each pixel, find the depth at that point. And if z of xy is greater than the Z-Buffer, Z of xy is greater than the Z buffer, right now what I have done is the Z-Buffer I have initialized to a minimum value that means the point which is the farthest from the eye I am taking that to have a negative z value.

(Refer Slide Time: 00:27:15 min)



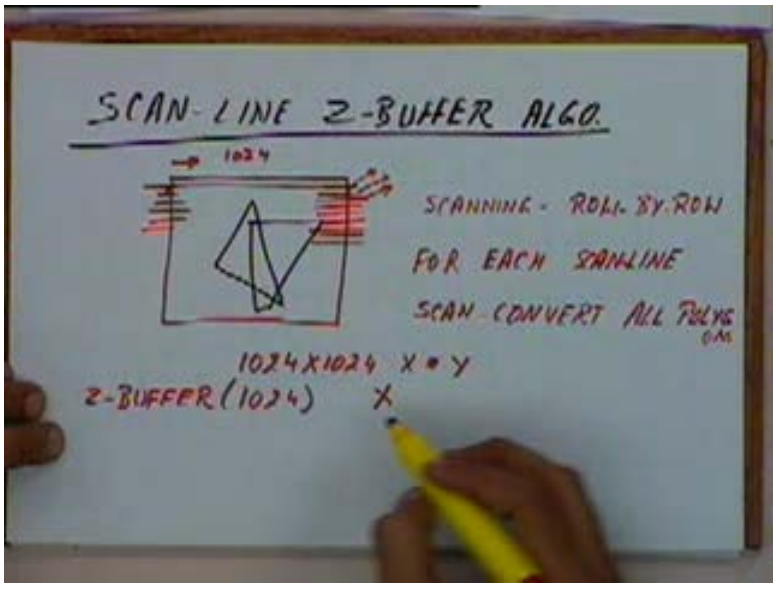
So z of xy is greater than Z-Buffer in that case we will update Z-Buffer and we will update the frame buffer, Z-Buffer at xy. Student: updation before for any change in colour. Yeah, for any change in colour intensity, so Z-Buffer at xy will be equal to Z of xy. In the frame buffer will be

updated according to the colour and the intensity of that polygon. So both Z-Buffer and frame buffer for that polygon will be updated, if Z of xy is greater than Z-Buffer of xy. In this process we will repeat one by one for every polygon. When we complete this whole process all the polygons then we would have decided for each pixel, if you have a set of polygons like this, if you have a set of polygons to be displayed like this at the end of the algorithm you have decided for each pixel what is the depth of the polygon being displayed and what are the colour and the intensity being displayed at that point.

Effectively that means at each pixel we would have decided which polygon is being displayed. So this is a by scan converting each polygon one by one and maintaining a Z-Buffer, we are able to remove the hidden lines and display only those polygons which are closer to the eye, this algorithm is called the Z-Buffer algorithm. Now from this algorithm it should be obvious that we have to scan convert each polygon that means if we have a number of polygons for each pixel, we will have to decide the z value and compare. And the amount of space that we are going to use is going to be x times y. x times y, if you have a screen which has the resolution of this. we need space which corresponds to roughly about 10 to the power 6 integers which is 10 to the power 6 into 4 which is equal to 4 megabytes that much of space would be required to maintain the Z-Buffer. So in this algorithm the space involved is very large, the space involved is proportional to x times y.

The space involved is very large and is typically of the order of a few megabytes especially with high resolution screens. And we have to scan convert every polygon so that is expensive in terms of time. Of that order. In fact in the VGA monitors, PC's that you have screen of this order. The older ones, the CGL or earlier ones they used to have a resolution of 640 cross 256 or 640 cross 320 those kinds of resolutions but not, these kind of, resolutions of screens is quite common. Sir it is a bit higher for graphics. It's a bit higher normally 13 24 or something. So this algorithm takes a lot of space plus it is also expensive because scan conversion has to be done for each and every polygon one by one. Any question about this algorithm, the Z-Buffer?

(Refer Slide Time: 00:32:25 min)



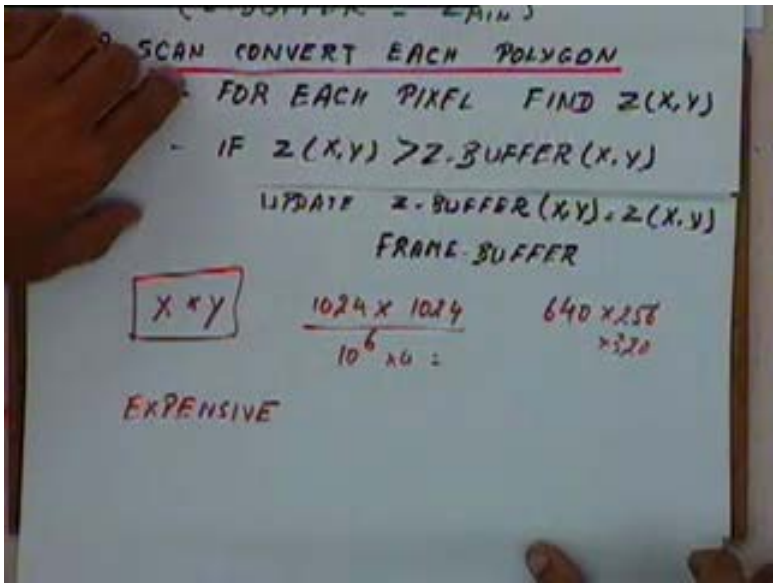
Then we will go on the next algorithm which is a slight modification of this algorithm, this is called the scan line Z-Buffer algorithm. So if you notice in the Z-Buffer algorithm this is the screen and if I have a polygon like this, I am going to scan convert this polygon and scan conversion involves scanning it row by row. So I first scan this polygon row by row then I am taking the next polygon which looks like this and now I will start scanning this algorithm row by row sorry this polygon I will start scanning that row by row and as I am scanning, the display coordinates the display characteristics of each pixel I am going to store in the Z-Buffer as well as the frame buffer. Instead of that what we will now do is we first look at one row, look at all the polygons which are inside that row which are crossing that row, in the sides of this row which are the all we display.

So I will do my scanning but I will do it row by row or a scan line by scan line and for each scan line, I first complete all the polygons. So essentially for each scan line, I will convert or I will scan convert all polygons. If I take one scan line, I complete all the polygons for that then I take a next scan line and I complete all the polygons for that. Then what I can do is first I will maintain a scan buffer which will be, the size of that will be equal to the resolution in the x direction. The resolution in this direction is 1 0 2 4. I will maintain an array of size 1 0 2 4, using that array I will scan convert the first row and dispose all the polygons that are crossing that. Then I will give the same array and use that for the second row.

So once I have decided which polygons to be displayed for the first row, I don't need the Z-Buffer for that anymore. So now I will use a Z-Buffer whose size will only be 1 0 2 4. Earlier I was using a Z-Buffer whose size of 1 0 2 4 times 1 0 2 4 or its size was x times y. Now I will use a Z-Buffer whose size is only x, using this Z-Buffer I will first scan convert the first row then the next row and then the next and so on that way this is referred to a scan line Z-Buffer. It's a similar algorithm except for the fact that the scanning will be done scan line by scan line and I will maintain the Z-Buffer only for one row, I won't maintain a Z-Buffer for the complete screen. This way I will be able to cut down on the space requirements, the space requirements will be much smaller. Again. I mean scan line we update the buffer, at each scan line I will update the buffer. Sir, where you transform the information, the previous information's where are you storing now. The previous information you mean. I mean side by side you are displaying the.

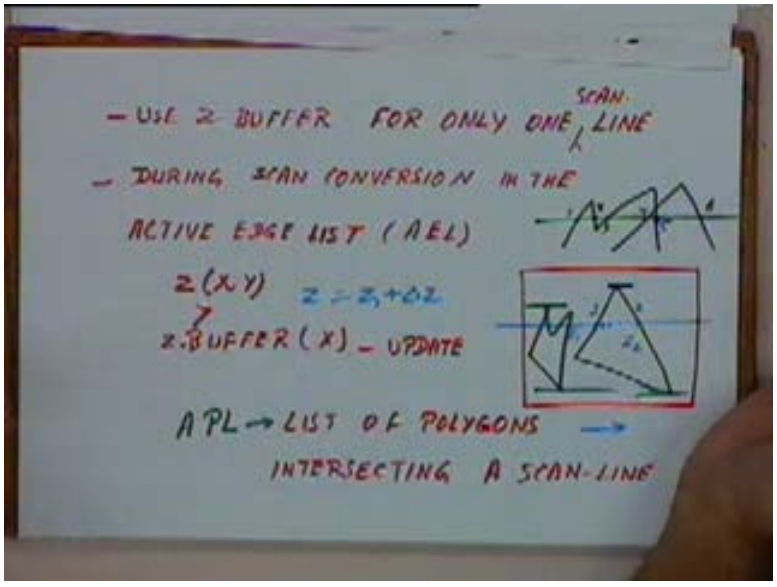
See I first look at, let's say I am looking at this scan line. For this scan line let's say I decide that in this portion first polygon is to be displayed, in this portion the second polygon is to be displayed. Once I have decided that now we move onto the next scan line. When I go to this next scan line I don't need to maintain a Z-Buffer for this because they have already decided that here which are the pixels, which polygon is to be displayed and here which polygon is to be displayed that information will be stored in frame buffer. Student: Side by side you are transferring the whole information to the frame buffer. See your display information is always being stored in the frame buffer, if you look at the previous algorithm, the moment we decide which polygon is to be displayed I am updating the frame buffer for the display information, the Z-Buffer will only store the depth.

(Refer Slide Time: 00:38:16 min)



So the moment I have decided that in this particular line, in the first in this scan line which polygon have to be displayed, I don't need the Z-Buffer for this line anymore. I only need the frame buffer; the frame buffer is available to me anywhere. So the Z-Buffer for this, I will update that to get the Z-Buffer for this. I will use the same array, I will change its value so that it will now show a Z-Buffer for this scan line.

(Refer Slide Time: 00:39:28 min)



So let's see this algorithm in detail. What we will do is in this we would use the Z-Buffer for only one line or only for one row of pixels and then during scan conversion, you remember something called an active edge list.

When you are scan converting, you are maintaining an active edge list. In the active edge list we will also try and calculate z of xy. However you are maintaining an active edge list, this is the screen, this is the polygon to be displayed in the active edge. In the active edge list at a particular line or at a particular scan line will contain the set of edges which are active. That means let's say this is the edge number one and this is the edge number two. Now we will use this active edge list to calculate the z value at every point. How do we do that? Between these two edges, we will have only one polygon. So for this polygon we want to find out the z value at each pixel. Let's say the z value here is z_1 and the z value here is z_2.

If the z value here is z_1 then as we move in the x direction from one pixel to the next, the z value will change by a constant amount. Since we were talking of planer surfaces, as you move from this pixel to the next or from this to this, the change in z value will be constant. So if you know the z value at the edge then the z value after that, for the next pixel can always be calculated just by saying z will be equal to z plus some delta z, z will be equal to z_1 plus delta z. The next pixel will again being, the z value for that will be altered by a magnitude of delta z where this delta z can be completed according to the slope of the plane. So if we know the z value here and we know the equation of this plane then as we move in the x direction, we can complete the z value at each edge and as we compute the z value, as we compute z of xy, we will compare z of xy with Z-Buffer at x.

In the previous algorithm, we had Z-Buffer of x, y. Now we will have only Z-Buffer of x. We compare these two and if z of xy is greater than this then we will update both the Z-Buffer as well as the frame buffer and this process will be completed for all the polygons at this scan line. Student: Does it mean that at each scan line, the computer searches as many number of time and the number of polygons are there or at each point it is see how many polygons are intersecting and accordingly gives. No, what we will, I was about come to that. What we will do is see, we have one polygon like this and we have let's say another polygon which is somewhat like this. This polygon is starting at this scan line and is ending at this scan line, this polygon is starting at this scan line and ending at this.

So earlier if you remember we were maintaining a list as to where our edges are starting, we were maintaining an active edge list. In addition to that we will now also maintain an active polygon list. So this polygon we are going to start at this scan line. So we will maintain an active polygon list as we go from one scan line to the next, we will keep maintaining what are the polygon which are added, which are getting added to the active polygon list. So we will have an active polygon list which will contain the list of polygons intersecting a particular scan line. Sir what does it mean, active edge list is enough? No, for every polygon you will have a different colour, different intensity and we will first for this scan line let's say if you just maintain an active polygon list you might have a situation like this, we have one polygon like this and another polygon like this.

You are looking at a scan line like this. This is intersection 1 2 3 4 5 and 6. Now 1 and 2 correspond to one polygon, 3 and 4 they do not correspond to the same polygon, 3 and 5 will correspond to the same polygon and 4 and 6 will correspond to the same polygon. So we have to be very careful in taking care of this information. Sir, that's not by active polygons. No, we have to know as to what which are the polygon which are active on this particular line. We have one

polygon and a second polygon, all of them are active. So both of them are active, so we first take that one active polygon do the scan conversion for that, then take the second active polygon, do the scan conversion for that at that particular scan line. You get that? We may need two steps, we are supposed to do it in one, isn't it?

We will do it as many times as the number of polygons in the active polygon list or we can maintain a common active edge list and with each edge also keep track of which polygon it corresponds to. We can combine the active edge list and the active polygon list that is immaterial but if you have a set of set of polygons and for each polygon we have a separate active edge list and do the scan conversion at that particular scan line. That means regarding each scan line it will go as many number of times of the number of polygons at each scan line. At each scan line, if there are n active polygons, so it will go n to n, n to n times and n times it will update, n times it will update, but it will update only in the portion which are intersecting.

See whether you do it n time or whether you do it once, what you have to do is for each of these segments which are to be displayed but we look at each and every pixel and decide the z value that will you have to do anywhere. So whether do it once or whether do it n times, it is going to be the same amount of effort because if let's say in this case if I do it once from 1 to 2 I will decide the z values and from 3 to 5 I will decide the z values and then for 4 to. Back one, so we will store from 3 to 4 for the previous one and then 4 to 5 for the third one. They can be intersection between 4 and 5. No, there is a polygon like this and a polygon intersecting it like this. 4 is the edge of this polygon and 5 is the edge of this polygon and the two polygons can be intersecting like this.

So we will take care of that in the next algorithm but for the timing, what we will do is when we are scan converting we will go from 1 to 2, from 3 to 5 and from 4 to 6. If I am maintaining an active polygon list and I first take at the first polygon, I will scan convert from 1 to 2 and from 3 to 5. Then I will take up the next polygon and scan convert from 4 to 6. Whether I do it in two steps or in one is the same amount of effort. So typically you can maintain an active polygon list and then for each polygon, you will have an active edge list which will help us scan convert at that scan line. And then when you move from one scan line to the next scan line, we will have to update the active polygon list, we will have to update the active edge list and we will have to update the z values. From this point we will see in the next class, we will complete this algorithm and then see how to make this algorithm even more efficient. We will stop here now.