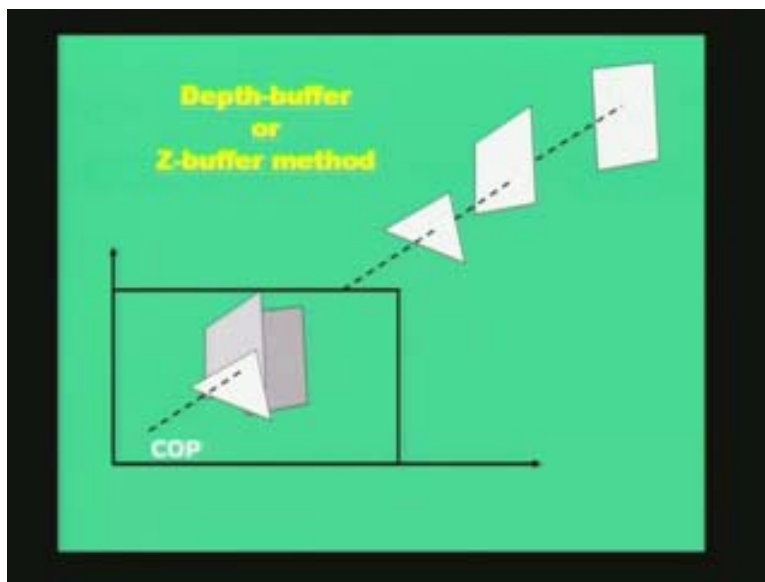


Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture # 28
Visible Surface Detection (Contd...)

Let us continue the discussion we were having in the last class, first of the VSD algorithms or Visible Surface Detection algorithms. In the last couple of classes we have been discussing concepts of VSD algorithms based on back face culling and then of course the first of the VSD algorithm that is the depth buffer or Z-buffer method.

(Refer Slide Time 02:00 min)



So if you look into the slide you can find that basically the concept of VSD algorithm or any such algorithm is to basically compare depths and that is what we are doing here that we are comparing the set of polygons, the depths of those polygons at a particular point (X, Y) and this is a scenario of orthographic projection where we are looking at the rays coming out of points out of a polygon and it is coincident at a particular point (X, Y) on the image plane and at that point whichever polygon has the minimum depth we have to paint that particular pixel with that corresponding depth and intensity.

We also went through some of the calculations and algorithms necessary for the steps of processing. We initialize the two buffers used in the algorithm; one is the depth of (X, Y) or Z_{max} for all (X, Y) and a similarly refresh buffer for the intensity with the background intensity and once this is done that initializing process is completed. We process polygon after polygon in any order and for each position on each polygon surface what we do is we need to calculate depth for each position (X, Y) on the polygon. And then if you find that depth is less than the initialized value or the current value of the buffer at (X, Y) point and if it is so remember we are looking into the negative Z direction then depth of (X, Y) is equal to Z .

(Refer Slide Time 03::08 min)

Steps for Processing:

(a) Initialize

$I_B = \text{background intensity}$

$\forall (X, Y) \begin{cases} \text{depth}(X, Y) = Z_{\max} \\ \text{refresh}(X, Y) = I_n \end{cases}$

(b) For each position on each polygon surface:

(i) Calculate depth Z for each position (X, Y) on the polygon.

(ii) If $Z < \text{depth}(X, Y)$ then

$\text{depth}(X, Y) = Z;$

$\text{refresh}(X, Y) = I_n(X, Y)$

We actually assign the depth at that particular point (X, Y) with the value of the depth computed at that particular point of iteration and the intensity is used to update the refresh buffer at the same point. So these are the steps for processing and we will revisit it again during this class. But before going forward we also discussed in the last class the concept of computing depth. As you can see here the computation involved and the complexity basically once is the number of polygons multiplied by the depth calculation of the size of each polygon sort of a thing. So if you see the depth calculation is the key step in this process which can overburden your CPU or computational time then we need to devise a mechanism by which we can simplify this computation of depth. We have studied that in the last class and we will go to that quickly once again. Before that of course is the projected intensity value at each position XY for which we are not interested in the calculation of the intensity at this point of time.

We leave it to the future classes and it is at that corresponding minimum value of Z at the current stage of iteration. So calculation of Z is our interest as mentioned just before and we see that the equation of the surface is given by AX plus the equation of the planar surface and the depth value is obtained by simplifying the equation and that is expression of Z .

Now, if you repeat this calculation for each (X, Y) point assume that the parameter of surfaces are known, we know how to do that as we had seen that equation couple of classes back; that if you are given a set of n vertices of a polygon then we will be able to calculate the surface normal n based on that expression of a summation of the product terms and that gives us the coefficients A B C and D then on the right hand side you can also substitute the values of X and Y at a particular pixel on the image plane and calculate Z . But these calculations which you have to do for all polygons and all points for each polygon then that is going to be a very huge computational load on the processor and it will take a lot of computational time because in a typical complex scenario as mentioned earlier forget about few hundreds to a few thousands of polygons in a very complex picture you may have about a million polygons to be rendered or shaded and you have

to apply this VSD algorithm, back face culling on so many polygons and imagine so many polygons each of those having few tens or hundred pixels each coverage in terms of a two-dimensional coverage you can visualize the amount of computational time you need to calculate Z for each of these points. So we need a mechanism by which we can calculate this Z in a simplified manner.

So we look back, that is the expression of Z which we have and what we are interested to know is as we traverse using a scanline based method this scanline polygon filling algorithm if you remember then the equation of Z can be simplified by remembering the scanline polygon filling algorithm where we are using the edge coherence. If you remember the edge coherence property of a 2D scanline polyfill algorithm for the next scanline the corresponding point X prime that means given a point (X, Y) at the edge of a polygon for a current scanline Y the same edge will intersect the next scanline y plus 1 at the corresponding point X prime which is given by X minus 1 by the slope of that line.

(Refer Slide Time 07:02 min)

I_x is the projected intensity value of the surface at position (X, Y), which has the minimum value of Z, at the current stage of iteration.

Equation of the surface:
 $AX + BY + CZ + D = 0;$
 $Z = \frac{-AX - BY - D}{C}$

Calculation of Z:

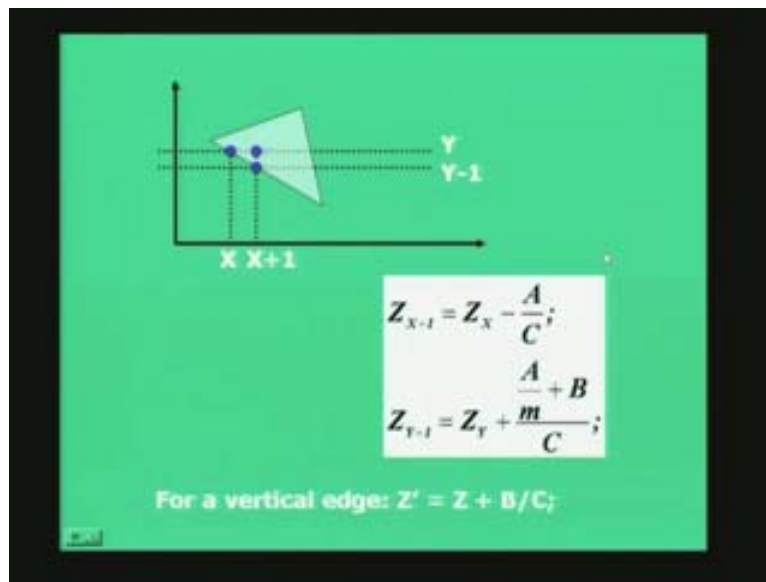
Equation for Z at the next scan line:
 Remember scanline/Polyfill algorithm?
 Using edge coherence, we get for the next scanline (Y+1):
 $X' = X - 1/m;$

So we know that from the edge coherence which we used in the scanline polygonal algorithms all those concepts are valid as per 2D is concerned. Of course there was no depth which we consider there, I repeated this in the last class but we use all the concepts of 2D here as well.

All the concepts about scanline, fill in between pairs, detect vertex processing and concepts such as integer based calculations to obtain the next increment for the next scanline are valid here as well. So just remember this formula. If you remember this formula we are trying to visualize the scenario where if depth of this particular point say (X, Y) is known what should be the Z of this point and the same scanline for that next X plus 1, Y and also for these corresponding edge when you move from scanline Y to Y minus 1. If you do that and the next intersection assuming it to be known by as X prime which is X plus 1 by M depending upon the 1 by M you will increment it here or there may not be any increment depending on Bresenham's concepts here on line algorithm but typically use integer based calculation to get to the next increment point.

If these X and Y point are known the Z also can be obtained very simply as a simple calculation. That means Z_x and Z_y are nothing but at this particular point the value of Z is known, the Z_x or Z_y which is depth at this particular point (X, Y) is known then the depth of the next point X plus 1 is represented as $Z_{X \text{ plus } 1}$ which is given by the next expression and the Y plus 1 or Y minus 1 depending upon whether you are incrementing or decrementing the Z value with respect to the Z Y the previous one will be available in this particular form. That is very simple and of course if you have a vertical edge and the value of slope is infinity you can substitute the limiting case of n terms to infinity this expression will become $Z_{Y \text{ plus } 1}$ will be Z plus B by C factor.

(Refer Slide Time 08:27 min)



So if these are the three simplified expression which you see are much simpler to compute and calculate than the expression for the Z. That means you can apply some incremental method by which for a current point if the depth value is known either it is on the edge of the polygon for a particular scanline or if it is any point with in the polygon itself then the corresponding depth for the next X plus 1 point which is the next point on this scanline or the point just below this scanline when you traverse to a scanline and moving to the next point depth the corresponding Z can be easily obtained without doing the calculations entirely once again if you remember this formula.

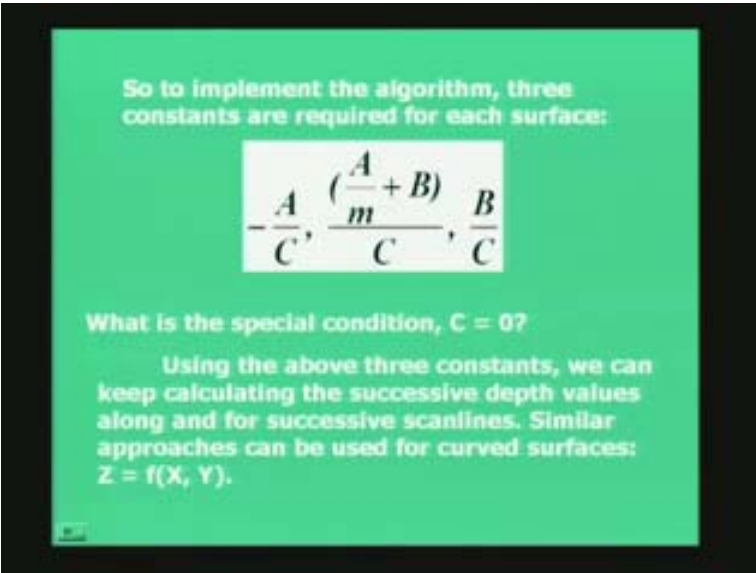
So, if you keep this formula in mind and keep this coefficient which is A by C on the right hand side in this factor also and then B by C then you will remember that to implement the algorithm you just have to remember these constants for each particular surface. And these are constants for a particular surface because given a particular polygon with n Z of vertices n could be 3 in the case of triangle or it could more for any other polygon arbitrarily these coefficients A B C are related to the direction cosines of the surface normal which can be computed straight away and of course M is the slope of the particular edge which you bothered about to compute the intersection of the edge with the next scanline. In that case you need the slope of the line N. M of course could be infinity if the line is vertical then of course you need this particular coefficient B by C.

Also, of course the special condition when C is equal to 0 you cannot use this coefficient. I hope you understand the scenario when the C is equal to 0. What is C? We did these in this case of back face culling. You remember back face culling? C is the Z components of the direction cosine of the surface normal. I repeat again, C is the Z component of the direction cosine of the surface normal. And if you remember back face culling also we discuss that when C is equal to 0 we do not need to actually bother about painting the surface normal we do not neglect that particular condition when C is equal to 0 because if you use that suddenly what will happen is this coefficient will vanish. In fact they will probably shoot up to a large value. So avoid that condition when C is equal to 0 and move ahead.

So using these above three constants we can keep calculating successive depth values along and for successive scanlines as well. I repeat, using the above three constants, these are the three constants here we can keep calculating successive depth values along and also for successive scanlines. Similar approaches can be used also for curved surfaces. Z is equal to function of X, Y. This is very interesting here because this incremental algorithm can be applied for a planar surface may be for a curved surface also. I leave it as an exercise for you to find out that if you have a curved surface then given the depth of particular point the next point also if the surface properties are known that is the surface curvature or the properties which the define the curve or the surface instead of a polygon you should be able to obtain this side.

Of course when you deal with curves and surfaces we will see nonlinear equations which are used to represent curves and surfaces. But you can visualize a typical case like a cone or cylinder or even a sphere, that is the simple example and you know the equation of a sphere. Z square is equal to a function of X and Y X square plus Y square is very simple scenario if you take SV around at the origin. And what will happen if at a particular point X and Y the Z can be computed in fact the next point also could be computed as a function of the previous Z without using this square. I leave it as an exercise it is such a simple task.

(Refer Slide Time 12:07 min)



So to implement the algorithm, three constants are required for each surface:

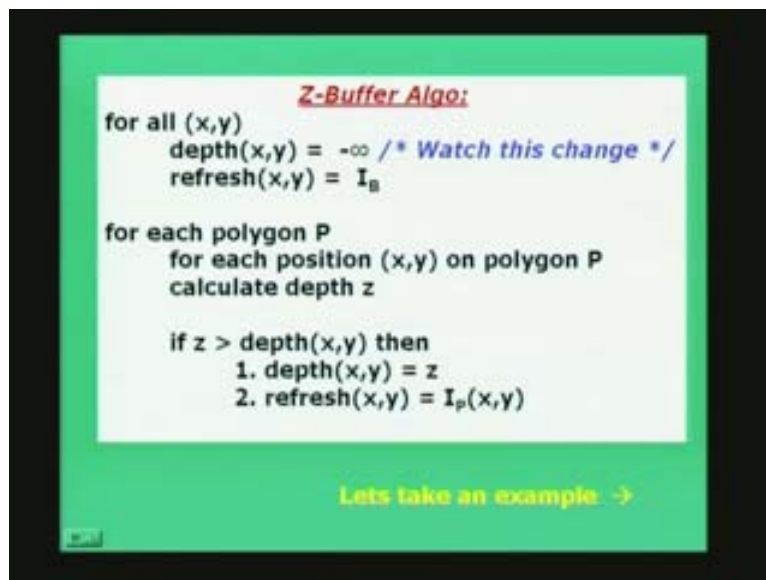
$$-\frac{A}{C}, \frac{(\frac{A}{m} + B)}{C}, \frac{B}{C}$$

What is the special condition, $C = 0$?

Using the above three constants, we can keep calculating the successive depth values along and for successive scanlines. Similar approaches can be used for curved surfaces: $Z = f(X, Y)$.

This is what you have for the incremental calculation which can be used to simplify the computations of Z for a particular polygon, successive depth values for a particular scanline and for successive scanline the intersections of the edges is what you need. If that can be obtained by incremental calculation then the calculations can be simplified. So we will re-look at the Z-buffer algorithm now. Since we know the method to simplify the calculations of Z we must always keep in mind that the calculation of Z involves not the equation which was given as numerator divided by a denominator term that is not the way. Henceforth we will do which was basically given as **X plus b Y divided by C, no**. We do not do that because the computational burden will be quite large. We will use the computational of successive depths that is the current Z value is known the next value can be immediately obtained using one of those three constants which we just discussed about. So keep that in mind.

(Refer Slide Time 16:55 min)



Whenever this algorithm comes we calculate Z basically we use the, of course you need the first Z value at the initial point at some point of this scanline the initial value you need as to when the scanline starts whether from the top or the bottom depending upon whether you are going up or coming down. But once the initial value is given the next successive values for the entire polygon is computed by incremental calculation based on these three constants of the surface. Keep that in mind whenever we mentioned said we talk of this incremental calculation.

Let us go to the Z-buffer algorithm here where we initialize as mentioned earlier the depth values for all (X, Y) as well as the refresh buffer with the background intensity. You just watch this change here where we change the Zmax value to a minus infinity. We change this depth value to minus infinity. In the previous case we were looking into a scenario where we were looking into the minus Z axis or you assume a positive Z axis looking towards origin or from origin towards minus Z. So initialize to Zmax which is typically minus infinity type of scenario is what we are typically doing Zmax is now minus infinity is what we are looking at and we are looking for values which will now greater than instead earlier we were looking towards positive Z and assigning a positive value then we are looking for negative or lesser values in the previous

algorithms. It is a question of implementation, the way your viewing axis is whether you are looking towards positive Z or minus Z the initial value could be a large positive value or a large negative value.

It is something like you are assigning a value beyond the back clipping plane. You remember the back and front clipping plane of a viewing pipeline. So what you basically do is initialize Z beyond the back clipping plane depending upon whether you are looking towards positive or negative Z axis will have a certain value. So that is the change I am pointing out here which is we are looking towards negative Z axis so we are assigning it to a minus infinity, typically it is a large negative value beyond which there will not be any depth values for the polygons you are analyzing.

Of course the back ground intensity could be black or white depending upon whatever you want. For each polygon P for each position (X, Y) on the polygon calculate depth Z this is again based on an incremental algorithm.

Expect the first point on the (X, Y) on the polygon you need to calculate the Z using the formula. the rest of it is all incremental algorithm and now look at this change which also has happened Z is greater than depth (X, Y) because it is initialized to minus infinity. This change is coincident with this. Remember in the previous algorithm presented in the beginning of this class today and the previous class we put a Z_{max} instead of minus infinity and this condition was Z less than, now both have to be changed together. This will go hand in hand whether you are looking towards the positive Z or negative Z .

Please be careful that you are assigning the initial value and the condition also correctly. I repeat again; that means if you assign it to a positive Z_{max} then you are putting a condition Z less than depth (X, Y) whereas in this case as I am presenting right now if I am looking towards negative Z and the initial value is minus infinity then we are looking at depth greater than. So come back and watch this change that I have put a comment here for you to visualize and then we talk of depth (X, Y) is equal to Z and the refresh buffer will be assigned with the corresponding intensity which has to be computed at point (X, Y) .

So let us take an example to visualize this scenario. A very simplified example is what I can take with may be two or three different polygons in a very simplified and limited number of pixels is what I can take and let us take a few examples. Let us start with one example here as you can see. Well we will update the Z -buffer values to start with and then will of course look at intensity. Let us take a simple 4 into 4 frame buffer.

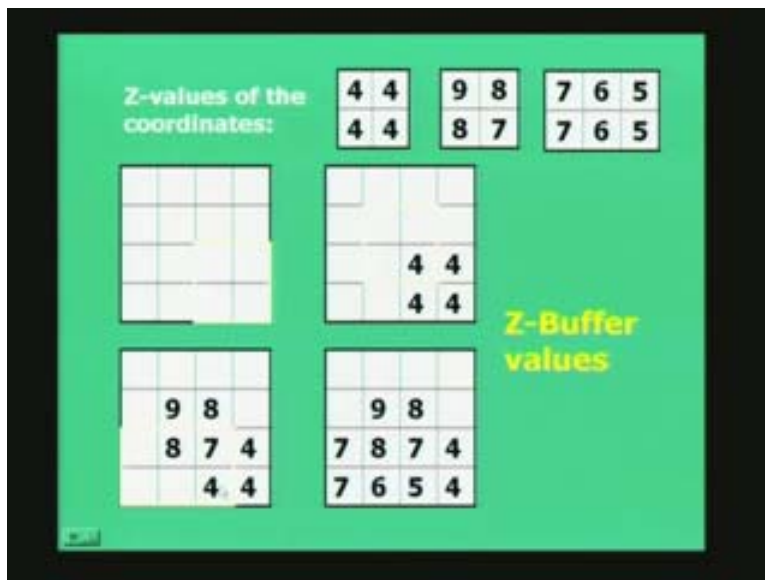
I am taking a 4 into 4 pixel and you have to visualize that this is in general will be a very simplified scenario which is what I can accommodate on the screen because you have seen this show and also what the individual pixels are and they are being operated with the certain Z -buffer or a particular color. So simple 4 by 4 of course you can take a slightly larger one 8 into 8 or maximum 16 by 16 but four 4 by 4 is even visualized very easily. And the first polygon which has come is a 2 into 2 polygon it is a square shaped polygon with depth buffer 4 and it occupies the lower part in the screen that means the corresponding (X, Y) values of each pixels are occupying the lower right corner of the frame buffer.

These are the four pixels it will occupy so the Z-buffer values will be updated with those because the initial values will be some Zmax and background intensities will be there, you do not worry about the intensity for the time being. Depth buffer values were found to be greater than or less than the initial value which you have put and in this case that is updated by the Z-buffer values here. So let us look at the next polygon. The next polygon having the values are given as 9, 8, 8 and 7. So we follow the previous algorithm where the depth value is more, depth value at a particular pixel overlapping with another pixel is more we will update. Otherwise we will replace the background depth value with the corresponding values.

Hence, if you look here these values again of a similar polygon has come as 2 into 2 and the depth values are 9, 8, 8 and 7 and they will sit here at the center of the frame buffer. If you see the first row 9 and 8 since you have the background depth values they will just be replaced by 9 and 8 respectively. Look at the lower row 8 and 7 values have to be occupied here, 8 will not have a problem it will come and sit here because the background depth value will be minus infinity so 8 will be more than that. You have to only compare 7 and 4.

When you compare 7 and 4 you will find that the 7 is more than 4 so the value 4 at this point which is the top left corner of the previous first polygon and the lower right pixel of the second polygon will be compared and the 4 will be replaced by 7 and hence you will have the current depth buffer values as this. I hope these two steps if you have followed is very crucial. Although I bring in a third polygon I hope you understand that I have initialized with a background depth values and of course intensity. All the intensities are not shown here that will come in the next slide. But the depth values are set to minus infinity and whenever you start comparing you are looking for a value which if it is more we just substitute it.

(Refer Slide Time 21:39 min)



So let us take the third polygon which is a 2 into 3. It has two rows and three columns and let us say these are the values 7 6 5 and 7 6 5 are the depth values of each particular point (X, Y). So you see that this is a planar type of a structure all these are plain of course the first one did not

have depth variations it is almost orthogonal to the viewing direction but the second one is second and third polygons do have such variations. And if you look here I am saying that this (X, Y) coordinate of the 2 into 3 sized polygon which is rectangle is coming and sitting in the lower left part. So you have to make lot of comparisons now. Not in all cases but three pixels are still having the background depth so the left column 7 and 7 will come and sit here that should not be any problem. Follow my cursor on the screen; I hope you are able to see that. The left column of the third polygon which is under processing 7, 7 will come and sit her that is not a problem.

If you compare the second column which is 6 and 6 the lower 6 value will come and sit here because the background depth is still around here, 6 compared with 8, 8 will be more than 6 so 8 will still be here and 6 will not be visible. Last column 5 and 5 will be compared with 7 and 4 respectively. Since 7 is more than 5 so 7 will be still be here. The last column, last row 5 depth value will replace the value 4 so four will be replaced by 5.

If you have followed this analysis I repeat 7 and 7 on the first column, 8 and 6 in the middle column and third column will have 7 and 5. So if that is the thing this is what you will have. This is the change which will happen after the Z-buffer values after the third polygon is applied. I hope you have followed this logic by which the depth buffer values or Z-buffer values are updated. You can take this example take a fourth polygon any arbitration anywhere within this 4 into 4 and take triangular type of a structure in a arbitrary polygon put some arbitrary depth values within this range and try for yourself and also ask your friend or colleague to work it out and then compare your values to ensure whether you got the correct result. **I hope you have followed; I will roll it back once again.**

If you see this was the first polygon so that will be filled up here so all the values are 4. A second polygon is sitting in the center of the image raster frame buffer so the 4 value will be replaced by 7, the rest of the value 9, 8, 8 will occupy the respective positions compared with the background depth buffer intensity depth buffer basically. And then the third polygon here has to come and sit here in fact wherever you have background depth they will come and sit here only one such value will be changed here because the value at 5 will be more than the value 4. I believe that you have followed this logic about how to update the depth buffer value using the simple example.

Now what I will do is I will try to use this same example again and put intensity and see how this will look like in terms if you have these polygons to be shaded with certain intensity do not worry about the intensity value because that computation will be done later on. What I have done is taken exactly the same example and I will now paint the frame buffer with certain color distributions. So let us go forward and look Z coordinates of the values here, that value will come and sit here if you remember since the background intensity is black or if it is green you are able to see through that and this is what will happen after you paint it with a corresponding intensity.

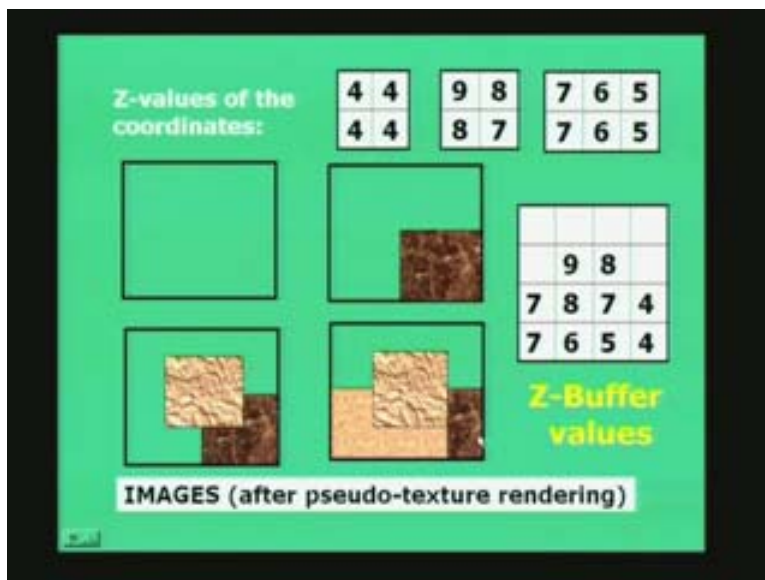
Do not worry about the intensity which I have put as a texture instead of using a constant color I have purposefully put a texture on that to just give a better visibility and good visualization because that is the key for computer graphics. Although you have to put realism on that at every point of time do not put texture everywhere but here I purposefully put texture and that is what I

say in this case my image is rendered by a pseudo texture. That is natural not the intensity value which may be computed it could be but in this case what I am saying is I am trying all that and an entire polygon is rendered by a particular texture to give it a good visualization so that portion is rendered and that is the depth buffer currently which is active. That is what you have and then you have the second polygon which is coming here and the second polygon comes and sits here. You know what will happen that the second one will come and occupy even this portion. So when you update depth buffer values the corresponding intensity values will also be updated in a similar manner.

Again I repeat; these images are rendered with pseudo textures that means I have assigned a particular texture for each polygon instead of the constant intensity purposefully just for the sake of visualization. Otherwise the intensity probably does not have any key role to play in updating. Remember, intensity of whatever constant shade or texture or whatever you are going to give does not have any role to update. The update is basically only happening based on depth comparison. You compare depth Z with the corresponding depth of (X, Y) at a current point of iteration for a polygon at a particular point (X, Y) and that is what you use to update. You never use intensity to update intensity could be any constant color or suited texture as given here for a particular polygon. So let us move ahead and there was a third polygon which was here and it has to sit here.

Now this is an interesting part because the depth values at the first column 7 and 6 and this 7 and 7 and this 6 will come and sit here. So wherever you have background intensity you will have the color of those new polygons coming and sitting here. Whereas the three other values 6, 5 and 5 will be compared with the corresponding depth buffer values here. And if you remember I will bring in the depth buffer values and the intensity will appear to be something that is sitting here. This is the depth buffer values which we got finally and if you remember carefully now these are the three polygons one in turn based to the other that means you have one in the front other somewhere in the middle of the first and the third one.

(Refer Slide Time 26:45 min)



So I hope you definitely have an idea. The interesting one is the third polygon. If you understand how the third polygons squeezes itself between first and second one with respect to its depth values also remember 7, 7, 6, 5 so if these depth values are the current ones the corresponding intensities also will be visible here. Whereas in this case 8 and 7 it corresponds to the second polygon hence we have the shade here and of course in other parts we have the intensity of the first and the second polygon.

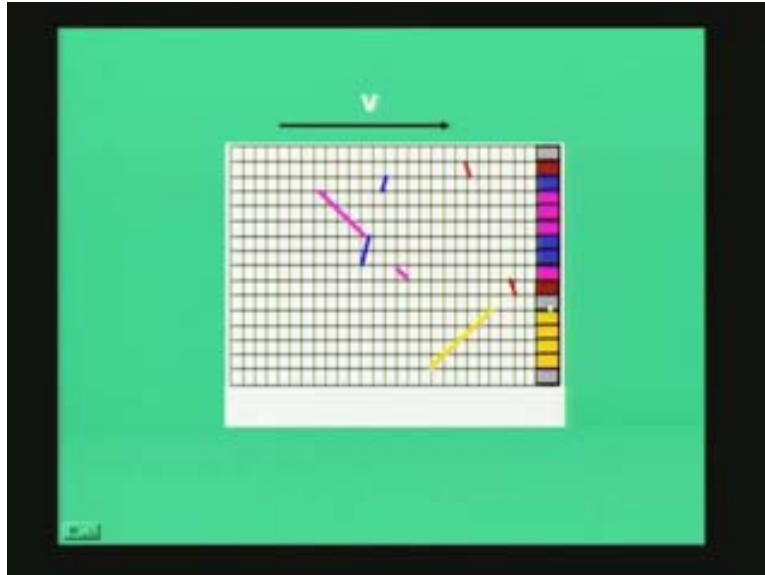
I hope this gives a visualization of how the image will look like after rendering based on depth comparison and gives you a true feeling because it is a case where we had three polygons the first one came at the back side, the second one came which was on the front and the third one which came was in middle. I will roll the values back for you once again. This was the first polygon which will be rendered, the second polygon will come and sit here and it will be rendered in the front then you have a third polygon which comes and sits here basically and that is the location of (X, Y) pixels, it is between the first and the second and that is what it will occupy based on the depth buffer values here. This is what you have as an example of Z-buffer Visible Surface Detection algorithms depth values and intensity for the case of three polygons here. You can take a fourth polygon and compare the depth buffer values yourself.

Let us take another example from a slightly different view point. Now here if you see I have taken an example of about 6, there are of course couple of polygons very closely aligned here and the intensity values are painted. You are actually looking into a ZX or ZY plane. So the three dimensional scenario is what you are looking at and you are looking into the ZX or ZY plane.

So view direction is towards negative Z and the XY plane is in such a manner that it will be along your view direction so you will actually have the projection as a line and now what you can basically do is compare this depth by depth. You can compare this depth by depth at each particular point here and if you see the first pixel it will be blank, the second row of pixels you be compared you have that which is a brown color polygon then you just have to visualize that the first part of the polygon which is in front of you, will be used to paint the refresh buffer.

The refresh buffer is now painted on the right hand side column of pixels which is some background intensity and when you do this Z-buffer rendering I will give you the final answer straight away this is what you will get. When you blow it up this is what you will get and the first row was still with background intensity, the last row of pixels also with the same and you have the second one with brown, the third one with blue and the fourth, fifth quite a lot of them with this pink color and so on. This was the previous scenario. You have to just look into scenarios where which of the polygon occurs first when you are looking from the left hand side towards the right hand side.

(Refer Slide Time 30:42 min)



So at the very top you have the brown color here, the blue color here lot of pink again a blue again a little bit of pink and then quite a lot of here along the bottom that is what you have. This is the scenario which will happen in the sense that it is not that the polygons are purposefully split into several parts. I have just given you a section of that of a particular polygon is used to render a particular portion of a row of pixels.

Again you have to visualize this scenario where you are looking into this plane the Z X or Z Y plane and this is the viewing axis and this is the XY plane and they require a lot of different polygons aligned in a different manner and you are looking towards from your left to the right and when you see that you will have to just check for each row which is the first polygon you go and strike and that is what will be used to render the particular color. This was the scenario which we have with the background intensity at the initial phase on the right hand side. This is what you have on your extreme right column the image buffer painted with background intensity and then this is what you will have after rendering.

At each row the corresponding polygon is rendering the corresponding refresh buffer. You do have background intensities at certain point one at the top, one at the bottom, one at the middle where there are no polygons which are along the viewing direction. You do not have polygons obscuring that particular pixel. This is what we have finally as the examples of refresh buffer being painted with depth values and this is what we discussed about the Z-buffer not depth buffer based method for the first of the VSD algorithms.

So, in the remaining time we move on to the next algorithm for VSD which is called scanline algorithm. And it will be closer to the polygon filling algorithm which we discussed earlier and in fact we will see that the scanline polygon filling algorithm which we discussed in the class earlier in 2D these scanline algorithm for Visible Surface Detection or VSD or the other name called Hidden Surface Elimination we can also call as HSE algorithm which is the Hidden

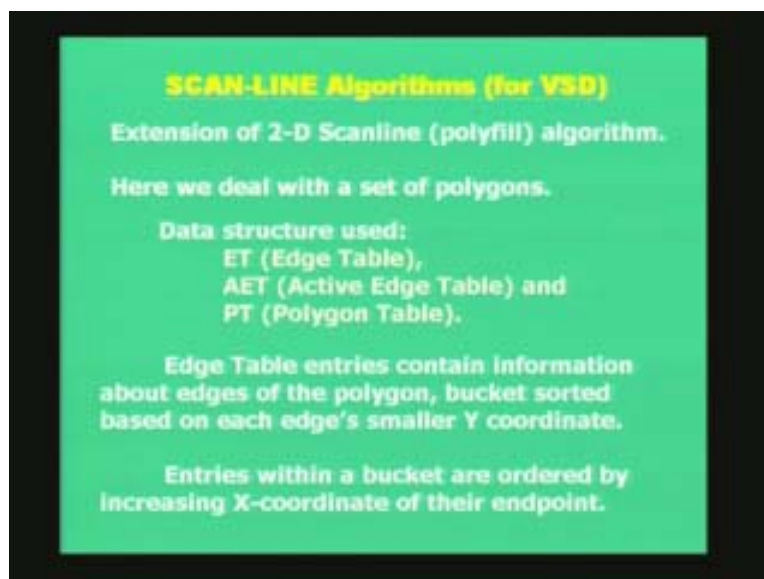
Surface Elimination algorithm. But typically VSD algorithms are Visible Surface Detection is the common term which is being used.

In fact when we talk of hidden lines which we are not discussing here, the Visible Surface Detection or VSD algorithms become Hidden Line Elimination algorithm or HLE algorithms. So, hidden lines can be easily eliminated once you know how to eliminate hidden surfaces. So we are discussing Visible Surface Detection algorithms and after the Z-buffer which we have discussed yesterday in the previous class and also in part of this current class now.

We move on to the second algorithm which is called the scanline algorithm. If you look it is basically an extension of 2D scanline polyfill algorithm as I mentioned earlier and here we simply deal with a set of polygons but of course in 3D and view the same data structures as we used for the scanline algorithm you also got a edge list or edge table which was bucket sorted. We studied data structure, we had an active edge table or active edge list, we also had the additional one which will handle the depth which is called the Polygon Table or PT so we have a ET and PT and edge table, active edge table and a polygon edge table, polygon table as a data structure to be used and the edge table entries contain information about edges of the polygon bucket sorted based on each edges smallest or smaller Y coordinates so this is the same as we have done earlier.

You need to have a static structure as your edge table which will be bucket sorted and it will store the information about not just one polygon but several polygons in this case but each polygon will have this particular data structure made active. Let us see how to bring in several polygons which have to be compared with Z. So entries within a bucket are ordered by increasing X coordinate of their endpoint. I repeat entries within a bucket are ordered by increasing X coordinate of their endpoint that is what you have.

(Refer Slide Time 33:38 min)



The structure for each entry in the edge table this is what you have. I think this structure is also known to you. You have the X coordinate of the n point with the smaller Y coordinate, you have the Y coordinate of the edges other endpoint, you have the delta X value which is 1 by n and you have the polygon id that is the extra which you have. This was the edge table entry structure which we talked about earlier but the only extra entry which will come along with X value Ymax delta X is the polygon id. And of course you have a pointer which will point to the next edge of that scanline or it will point to null if there is only one edge or if it is the last edge in the list.

Structure for each entry in the polygon table this is the new structure which you have. Structure for each entry in the polygon table will be coefficients of the plane equations so you need that to compute depth. So somewhere you have to store that so it is stored in the polygon table structure. So you have the coefficients of the plane equations, you have shading or color information of the polygon and you have a flag which is initialized to false. And that is the key part we will see the significance of this flag which is a Boolean value it could be set in or out or true or false, out means false and in could be true either way. So 0 and 1 and this is the structure which you have.

Of course you have a polygon id as you had in the case of an edge table and then of course you have the plane coefficients of the plane equations used to compute the Z value, you need the shading information of the polygon we will not worry about these coefficients right now in terms of shading algorithm of information we will keep those coefficients of shading till we discuss concepts about shading algorithms and concepts about illumination and shading. And of course the last flag which is the in or out or the true of false you can have a single bit there which could be 0 or 1.

(Refer Slide Time 35:03 min)

Structure of each entry in ET:

- X-Coordn. of the end with the smaller Y-Coordn.
- Y-Coordn. of the edge's other end point
- $\Delta X = 1/m$
- Polygon ID

X	Y_{max}	ΔX	ID	•
---	-----------	------------	----	---

Structure of each entry in PT:

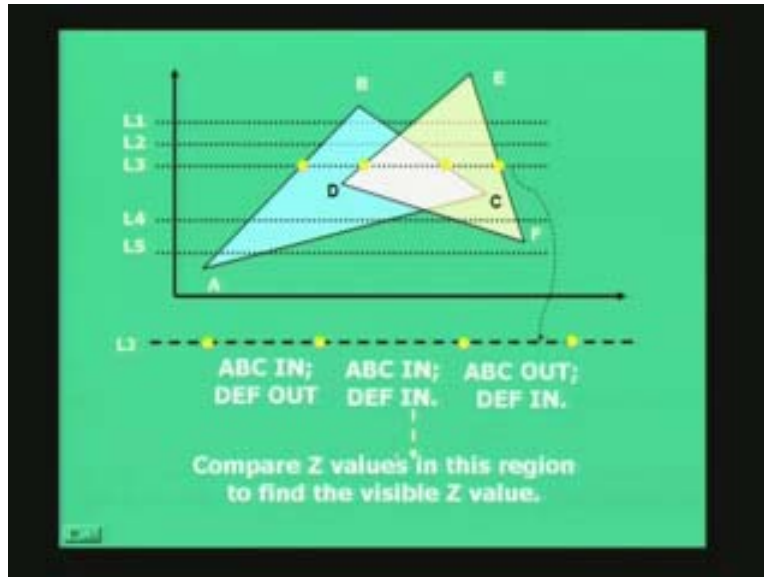
- Coefficients of the plane equations
- Shading or color information of the polygon
- Flag (IN/OUT), initialized to 'false'

ID	Plane Coeffs.	Shading Info.	IN/OUT
----	---------------	---------------	--------

So this is the structure for each entry assuming that you keep this structure in mind and copy it in your notes. Here we look at a typical example where we compare two polygons. If you feel to compare two polygons successfully you can compare any number of polygons. So this is a case of two polygons A B C is one polygon of course we have taken a simple case of a triangle A B C is a

triangle and D E F is a another triangle. And I have taken examples of five different scanlines which pass through this particular polygon.

(Refer Slide Time 37:15 min)



And we will consider only one of these scanlines and analyze that and if you are able to analyze once such scan line in that case I have take L3 then you will able to analyze the other scanline as well. Let us analyze scanline L3 which intersects the triangle ABC or the polygon ABC at two points here and also the triangle DEF at two points that is what you have. So we will enlarge that particular scanline L3 which has four intersections two each with each of the two triangles or we will land up two pairs of intersections and of course you have to fill within the pairs there is no doubt about it. But the question is how you do and where do you fill between polygons and that is what you have.

And as you see for the line L3 as you move from left to right for this scanline initially you will have the background and there is no polygon and you will mark and level these X structures saying that between the first pair of intersections for which the scanline is within the triangle ABC and it is out of the triangle it is not yet in or within the triangle or polygon DEF. So the corresponding flags which you have seen earlier for the polygon table as you keep scanning a scanline the polygon entries and edge table entries are updated. So what you will have is when you are at a particular point of a particular scanline and depending upon the number of polygons which you have the polygon table structures for some of the polygons will be in that flag that right hand side bit or the flag true or false in and out which we talked about. It will be in for a certain polygons and it will be out for a certain polygons. So for this scanline as you move from left to right both the flags will be set out first then between the first pair of intersections the flag for the polygon ABC will be marked as in and for the polygon DEF will be out. And when you are between the second and third intersections the flags for both these polygons ABC and DEF will be set in whereas in between the last pair of intersections here which is the case between this you will have the flag marked for ABC as out and DEF also.

Now if you look here it means that when you have to shade the left hand side of the scanline L3 with a particular intensity and the extreme right hand side use paint with the background color between the first pair of intersections you paint with the intensity for the polygon ABC for the last pair of intersections here you have to paint with the shading of the polygon DEF which is light grey in color and for the first pair it is light blue. Only when you are in between the second and third intersection point when both the polygons show that the flags are in you have to compare the depth values of ABC and DEF so that is what you mean.

You just go through all the polygons for a particular point on a scanline and check how many flags of how many polygons are set to in. So for those many polygons you basically compare depths for a particular scanline as those polygon table entries will be marked as in and only for those polygons you compare depth and the minimum depth which you have will dominate and for that intensity of that polygon is with you used to label or mark or shaded color that particular pixel. So this is the logic which goes through for each particular scanline.

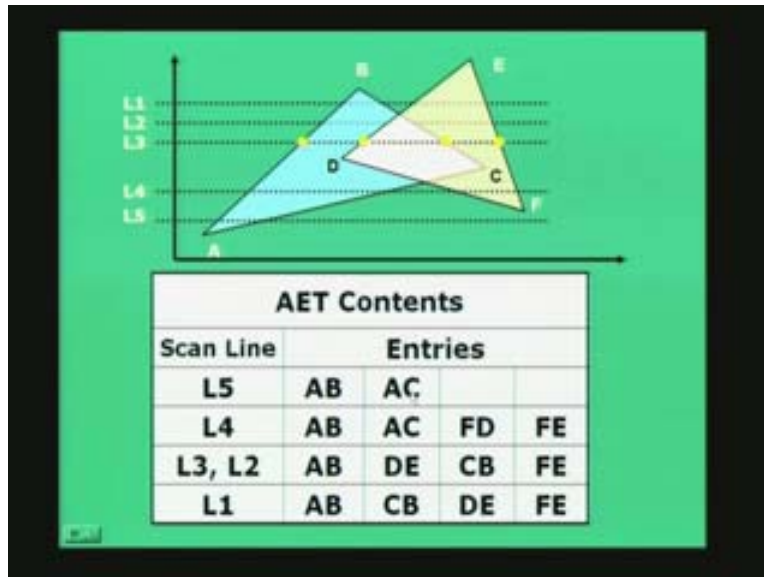
And of course you can use a concept like incremental computational depth and all that but to compare depths of particular polygons as you go through but here once you use those depths mark the particular polygons as in or out and then use them of course you do not use depth to mark the flags in or out. You basically go and check whether that polygon is covering that particular you can use inside or outside test that is what you do and then the polygon table entries are marked for X as in or out as you move from left to right for the polygon and you just compare the depths of all the particular polygons which are marked as in for a particular point (X, Y) and then use the minimum depth.

Again it is the depth sorting which you always use. It is a question of how you compare these depths and at what point you compare depths and what you update. Here you update when you move along a scanline from left to right for a particular scanline from increasing X coordinates then you are basically just checking whether the polygons are marked as in or out. That is what you do and then you compare depths of all those polygons. I am repeating again and again, if you compare the depth of all those polygons which are marked as in for that particular (X, Y) take the minimum depth and for that minimum depth use the intensity or the shading of that particular polygon to color that particular pixel.

This is the logic which is used. Again you are going back to that slide so you basically compare the Z values only within this region to find a visible Z value where you will find more than one polygon marked as in. This is the case of two polygons you may have n different polygons in a particular scene which you have defined and out of those n a subset n of those let us say n of those polygons will be marked as in within a pair of intersections or for a particular point within a scanline. So you compare the depth of only those n polygons which are marked as in not all the n that is the time which you save and then for those visible Z values which is the minimum Z value that is what you use to compute the intensity for any particular point. Let us say in this particular case when you are within the first pair or the last pair only when one of the polygon is marked as in you do not do any depth comparison.

You just use the shading and the color of that particular polygon to shade the pixel for that polygon which is marked as in if there is only one in. Once there are two or more polygons marked as in there is no other choice but you have to compare depth values.

(Refer Slide Time 43:33 min)



This is the concept which you use for all other scanline as well to compare. I have taken this scanline L3, if you remember here the scan line L2 and L3 will have almost similar scenarios but of course the scanline L1 will have a slightly different one. We look at the active edge table structures as you go along for the same example.

The active edge table entry is if you look at the last line L5 which intersects the polygon only ABC at the two edges AB and AC the scan line L5 will have the active edge table contains as AB and AC. That is what we will have and the polygon LT table will show that ABC will be in between these two particular scanlines and when you are outside the triangle or the polygon the ABC DEF both will be marked as out so you paint with the background for the rest. Look at line L4 which will be interesting it will have edges AB and AC for the first polygon ABC and also it will have the edges FD and FE for the triangle or polygon DF. Therefore, all those entries will be there in the active edge table for L4. This concept is valid as you do for a 2D scanline algorithm. Only the structure is similar but the entries will be more because you will have more than one polygon in general.

There will be many polygons and this scanline might intersect several polygons several edges of several polygons all those have to be brought into the edge table structure. The static one an active edge table will be updated has to be updated based on the contents of the edge table brought out. And of course you have to throughout some of the edge entries based on the concepts of what X processing which we did. Only the point which you have to remember here is you must have the same concept of this scanline polygon filling algorithm as we did earlier but for multiple polygons in this case. So, coming back L4 for active edge table entry will have contents of intersection for edge AB AC here then FD here and FE here. Look at L2 L3 said that both these lines are similar

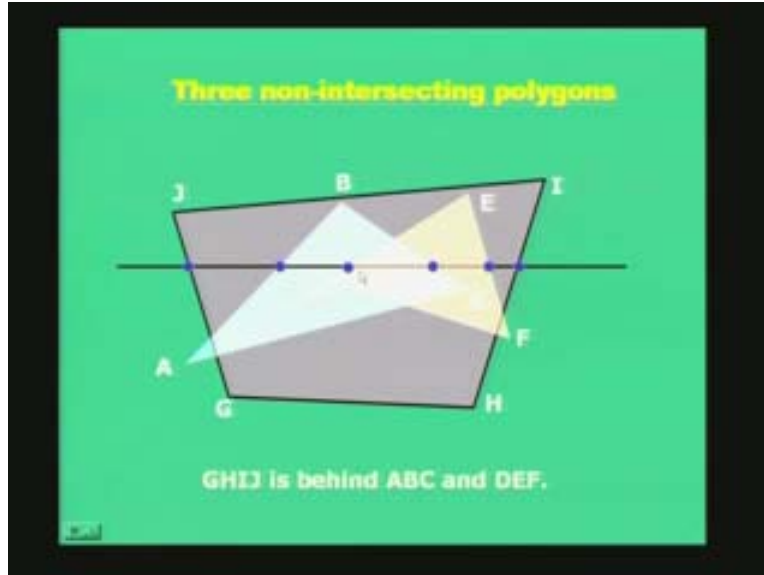
their intersection will be AB then DE then CB and then finally FE. So L2 L3 are same as given here AB DE CB and FE. Look at line L1 that also intersects with four edges but the order is different because it intersects AB first then it intersects CB remember we are talking about clockwise or anti-clockwise processing of course with leveled edges and then of course we have DE and then we have FE. That order is very important as to how you intersect the intersections. That will be done when you are bucket sort of edge table entries.

You have to update the vertices and then when you move from one scanline to another the sequence of the active edge table entries with intersections pointing to one another in term of eight entries have to be updated for each scanline. If necessary at the vertex you need to bring in a new edge or throughout an existing edge. Those concepts whatever we studied are still valid in terms of the Z-buffer processing, processing to compare the depth values and to shade it with the intensity we have seen in the previous slide as to what we have to do.

You check if the polygon table entry for a particular part of the scanline or a particular pixel has just one entry as you may use that otherwise if all of them are out you shade it with background intensity which could be an initializing step. Then only when you have more than one that is two or more polygons marked as in for a particular pixel or a particular section of this scanline between a pair of intersections what you need to do is compare the depth values. Compare the depth values and then choose the minimum depth value that is what you do. To compare the depth values there is one important point which we must also see, we will take this simple example of three non intersecting polygons. This word non-intersecting is very important. We will look at an intersecting case later on which is a special case. Scanline is not a very efficient method to handle intersecting polygons but we will see non intersecting polygons.

There are three polygons, if a very large polygons GH iJ which is a quadrilateral basically we will say that it is behind it is at the background somewhere. Although that is not the background intensity but it is a background polygon. Then we have two other polygons which are triangles ABC and DF which are in front of GH iJ that is why it is written GH iJ that polygon is behind the other two polygons ABC and DEF.

(Refer Slide Time 48:09 min)



This is a polygon instead of a triangle although we have taken triangles and quadrilaterals to simplify this scenario. In fact we have to visualize that DEF also is in front of ABC and ABC is in front of GH iJ or GH iJ is at back of ABC or ABC is on the back side of DEF. And look at this particular scanline there are so many intersections. It intersects GH iJ at two of the edges GJ and Hi it intersects ABC also at the edge AB and BC it intersects DEF at the edge DE here and EF there. As you can see there are six intersections of this particular scanline and you will have the polygon entry tables marked as in and out depending upon as you traverse on this scanline. Of course the active edge table entries will have all the edges which are intersecting.

Now let us try to visualize a particular scenario here. Concentrate on the intersection of the scanline at this particular point where it is intersecting the edge BC of the polygon ABC at this particular point. So when the scanline leaves the edge BC, we will go back to that figure. We are looking into that particular scanline which is leaving the edge busy because we are traversing this scanline from left to right and shading each particular point inside a particular polygon inside this scanline. So let us say we are at this intersection point at the edge BC and leaving it. When it leaves the edge BC it is still inside the polygon DEF and GHiJ. If you are leaving this point BC and moving to the right we are somewhere here just the next pixel although it is leaving the edge BC and the triangle ABC will be marked as out.

Remember to the left of this intersection, this intersection I repeat is the intersection of this scanline which we are considering with the edge BC. When we were before the intersection to the left all the edge, all the polygons ABC DF and GHiJ that is all the three polygons were marked as in or labeled as in or true. So you were comparing all the three and at all the three points you would have found that the polygon DEF is in front. So the depth and the shade of the polygon DF are used to shade all the points within that scanline just prior to the intersection at BC so the intersection at this point.

Hence, the DEF was used to shade all these points prior to the intersection BC. But once it leaves BC ABC will be marked as out, once you leave this point ABC will be labeled as out but DEF and DHiJ will still be labeled as in. So when the scanline leaves edge BC it is still inside the polygons DEF and GHiJ as mentioned just now. And assuming for the time being the polygons do not intersect depth calculations and comparisons between GHiJ and DEF can be avoided, why? Since we know that GHiJ is always behind the other two polygons ABC and DEF remember we are talking on non-intersecting so it is always behind.

If you are sure of that based on the depth calculations then when you leave that point BC it would be better that you do not start comparing the depths of the triangle DEF and DHiJ again. Remember, at each point when two polygons are marked as in did say that you compared depth but comparing depth at each point if the polygons are more will result in huge amount of computation you have to bypass that to save time to reduce the amount of computational burden on the processor you have to reduce the amount comparisons if possible.

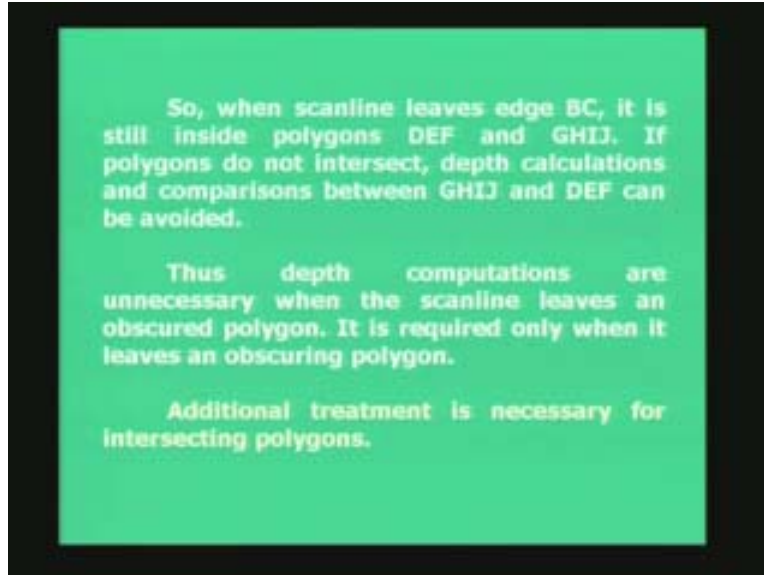
So you use some little bit of intelligence here and if you know that the comparisons between the polygons can be reduced if you are sure that when you are leaving a polygon and the next polygon which is marked as in is definitely in the front of the remaining other polygons provided they are non intersecting of course then you can avoid depth comparisons till the next intersection comes and only use the first polygon to shade those pixels within that scanline.

So again as I said before, if polygons DEF and GHiJ do not intersect which are marked as in, after we have left the edge ABC or after we have left the triangle ABC and moved to the right and the polygons DEF and GHiJ are marked as in and if this polygons do not intersect and still they are marked as in, then the depth calculations and comparisons between GHiJ and DEF can be avoided and you use the depth of only DEF triangle and its intensity to shade those particular pixels on that scanline after you have left the edge BC. So in other words this means depth computations are sometimes unnecessary when the scanline leaves what is called an obscured polygon. It is required only when it leaves an obscuring polygon otherwise it is not required.

Additional treatment is necessary when we are talking of intersecting polygons. So when we are talking of non intersecting polygons we are talking of whether you are leaving an obscured polygon which is obscured or it is required only when it leaves an obscuring polygon. In this case depth comparisons are required when you are leaving obscuring polygon. But in this previous example if you go when you are getting past this edge ABC you are basically getting past an obscured polygon because DF is already in front.

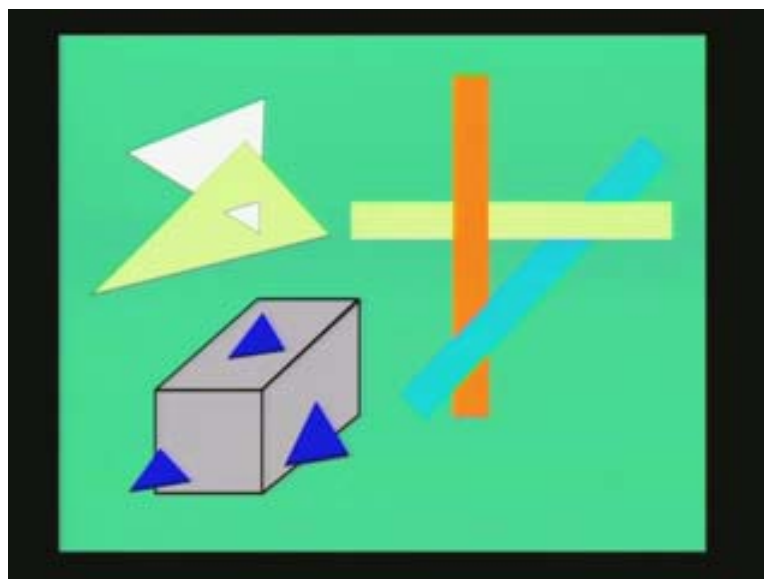
You are leaving an obscured polygon and hence the ABC was obscured before this edge BC here you are only involved in depth and intensity of this triangle DEF and you are leaving an edge BC in which you have never played a role just prior to this for the depth calculations and intensity. So if that is the case why compare again, use the same triangle and use it for shading and that will help you to reduce the amount of comparison.

(Refer Slide Time 55:01 min)



We will read the concepts again, depth calculations are unnecessary when the scanline leaves an obscured polygon. Remember, when you are leaving an edge BC you are leaving the triangle or the polygon ABC which was an obscured polygon. The triangle ABC was obscured by DEF, remember the figure. Triangle ABC was obscured by DEF so you are leaving an obscured polygon you do not do depth comparisons. It is required only when it leaves an obscuring polygon, obscuring means something which is hiding the other one. So, if you are leaving a polygon which is in the front which was obscuring or hiding the other polygons on the back side of you and you are passing through a scanline leaving a particular edge or a particular point here then what will happen is you need to start comparing depths again, so that what it is means.

(Refer Slide Time 55:38 min)



The last point of intersecting polygons is, let us take this particular example. This is a case of course of not intersecting polygons but the polygons where it is one is obscuring the other one. If you have visualization of three thin strip rectangles it is a case of not intersecting it is non intersecting case it is not intersecting at all and the scanline polygon will work here. But similar treatment cannot be done when a polygon enters a phase.

Now this is not happening in reality but you can have virtual reality when one polygon enters the other one. So this is a case when you need additional treatment the same thing is the case here you can visualize a triangle is lying within a cube and three portions are coming out. So in these two cases scanline algorithms are not very efficient in handling where you have to compare depths very carefully and to ensure that the similar concept which we treated with the previous example with three polygons which were non intersecting this ABC DEF and GHIJ the back side were non intersecting that is why the depth comparisons could be avoid in certain cases when you are getting obscured or not getting obscured.

But in this case scanline algorithm need additional treatment we will not discuss that but for the time being we will end with a short note that scanline algorithms are not that efficient as Z-buffer especially in cases when polygons intersect. We will look into various other sophisticated methods in the next class beyond this which are very very efficient and realistic and can handle also cases when polygon intersect one another.

We will stop with this particular case where we will see that the scanline algorithm is a 3D version of a 2D polygon filling algorithm but it handles multiple polygons, it can handle cases when polygons do not intersect one another and depth comparisons can be improved. But however if the polygons are intersecting you need additional treatment and things are not simple. We will continue in next class with a new algorithm on Visible Surface Detection.

Thank you very much.