Computer Graphics

Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture # 27

Visible Surface Detection (Contd…)

Hello and welcome everybody to the lectures on computer graphics. We are in the process of discussing different methods of Visible Surface Detection or algorithms for VSD as it is popularly called. And in the last lecture we had seen the concepts of coherence properties which are used in the VSD algorithms. And also we had discussed about back face removal or back face culling.

In fact we are in the process of back face culling where we found out that when the dot product of the two vectors is positive then it is a back face which basically means when you are looking along the positive Z direction then you look into the Z components of the direction cosine of the surface normal of the polygon at the face and look at the sign of that Z of the direction cosines and see if that sign is positive then it is a back face otherwise it is a front face which you have to shade. So, the concept of the back face culling or back face detection is used as a preprocessor or a front end processing or a front part of the algorithm for any of the VSD algorithms.

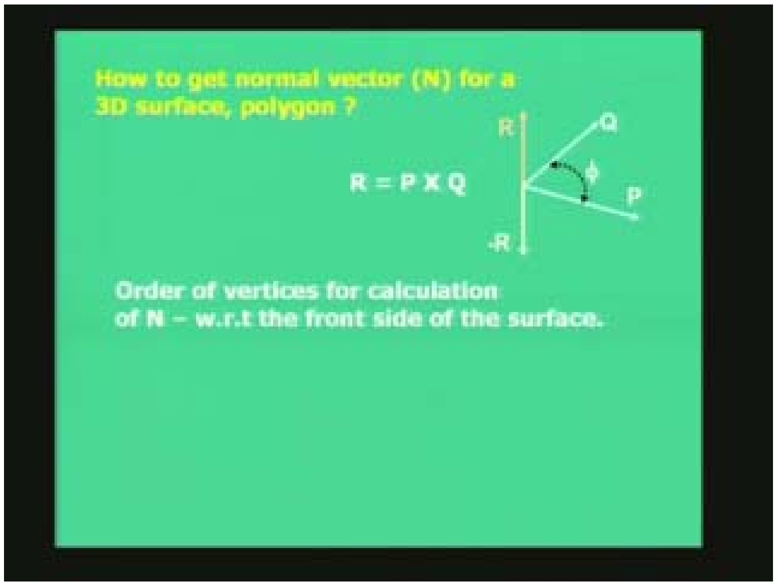
In fact you have to use back faces because we had seen that with an example in the last lecture that it helps you to remove approximately about 50 percent of the basis. And if you are able to do that your computational burden in terms of drawing the polygons which are not to be seen by the viewer is appreciably reduced. You will have the overhead to draw polygons which are at the back of the object and not visible to the user. And if you need to do that then you will overload the system with lots of extra computations which are unnecessary or redundant. So enforce back face culling, if you have a simple algorithm you use that and remove those back faces from the list do not render them at all and show it to the user and only draw the faces which are fully or partly visible.

Coming back to the exact point in back face culling or back face removal at the end of the last lecture the computation involves the computing of the surface normal of the polygon and that too within that surface normal the Z component of the direction cosines are important. And if you get that then it is easy and straightforward for you to find out whether it is a back face or not. So, coming back to the slide which we have seen in the last lecture, the problem was how to get the normal vector n for a 3D surface of a polygon. So assuming that the polygon is represented by a list of vertices or edges and a planar surface there are two such edges are of the polygons are given the P and Q which we visualize as two 3D vectors. And if we take the cross product of these two 3D vectors you are going to get a vector R which is of course going to be definitely the normal to the polygon because it will be normal to the plane on which P and Q raise and of course the fi is important because greater the pie greater you get the magnitude of R and things like that.

But the choice of the sequence of these two vectors P and Q is very important because if you take the cross product in the reverse order that means you take Q into P you will land up with the normal as minus R and is it important because the order of vertices for the calculation of the n the order of vertices will dictate whether you are going to take P into Q or Q into P because you will choose any arbitrary two edges of the polygon out of the set of n different vertices.

You can use visualize small n to v number of vertices or m to be the number of vertices because capital N is what you have used for the normal vector, in this case you can use R or N they are the same and we talk of the front side and back side of a surface. Now you should not confuse this with the back face. We are talking of in respect to back face culling. Back face culling we meant always by the front side of a surface but whether it is visible to you or not.

(Refer Slide Time 05:03 min)


How to get normal vector (N) for a 3D surface, polygon ?

R = P X Q

Order of vertices for calculation of N – w.r.t the front side of the surface.

Now remember here, when we talk of the back face of a particular polygon, a set of polygons form a polyhedron which enclose a solid. So, if you take a particular polygon which is lying on the surface of a solid let us say this one then it will always have a frontal and back side which will never be visible to the viewer because that is inside the object. A solid object is bounded, of course it could be hollow but visualizes a solid cube and visualizes once such side of it or one face of it or one polygon of it or a quadrilateral or a square in this particular case.

That square will have a frontal face which will be on the outside of the solid object which may be visible to the user or not depending upon the location of the side with respect to the viewer or viewer respect to the side. But that surface will have an inside part which we will call the non frontal of the back face of it will always be in visible. So a polygon will always have a frontal side and a back side of it, always a front and a back. The back is inside, this back side is inside the cube or a solid object which you have and you have a frontal side which will be visible and the back side will not be visible at all. So we need to find out and compute the n which is coming out of the frontal face not coming out of the back side face. Otherwise you will be computing the wrong direction cosines of the surfaces. I hope I am able to give you the visualization, a better visualization comes on from the slide which we see here.

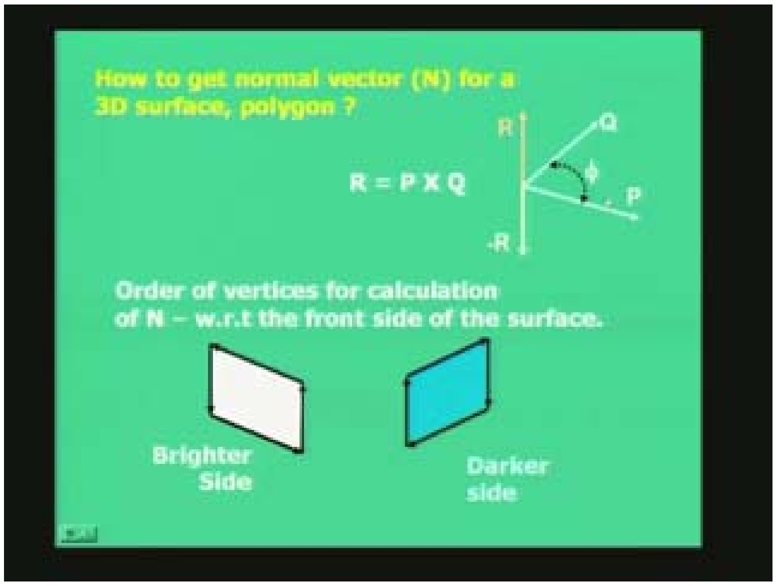
Let us say, if you see this particular three dimensional rectangle, of course it is a plane in 2D but assume this to be the brighter side, why do i say it is a brighter side? Well this is the outside part of the solid object which is visible to the viewer and I will say this is the brighter side based on a particular condition. What condition do I impose here? This was not discussed in the previous lecture. I say, I take the order of the vertices in such a manner that my edges form the vectors of the polygon as shown here by the arrows go in a counter clockwise arrangement.

Basically it means I take the vertices of this polygon in a counter clockwise arrangement and order and when I do that I will say I am looking into the brighter side or the facing side of the polygon. Now there is a reverse side of this as well. If I turn the polygon in the reverse side whereas the reverse side or the opposite side of the brighter side is visible to you this is what I will call as the darker side. I will not call it front or back anymore, we will say it is a brighter or the darker side. This darker side is inside the object and it will never be visible. So I paint it with a color which is non-bright or dark. And if you see the order of the polygons are clockwise because that is what you will see.

If you see this polygon from the darker side which is the opposite to the brighter side this was the brighter side and I am asking to you look from the other side of this polygonal face which will be appearing like these the darker side you see that the vertices or the edges appearing in a clockwise arrangement. Now, in a particular surface there will be a brighter side and a darker side. The brighter side is the one which will always able to see depending upon of course whether it is a back face or a front that is okay, depending upon with respect to the viewer position.

But there will be a darker side which will be inside the object always and will never be visible because it will be always enclosed by other parts. Why we are talking about the brighter and darker side? It is because we want to compute the end very accurately such that it is coming out of the brighter side not getting into the brighter side and coming out of the darker side, so they are very important here. If you look back you are looking into this P into Q which will give this R rather than minus R which could be accidentally obtained by Q into P. To avoid these accidents in terms of the order we say whether we should take P into Q or Q into P because given two arbitrary vectors we will not know what is the correct face in terms of the brighter or darker side.

(Refer Slide Time 09:35 min)



How to get normal vector (N) for a 3D surface, polygon ?

$R = P \times Q$

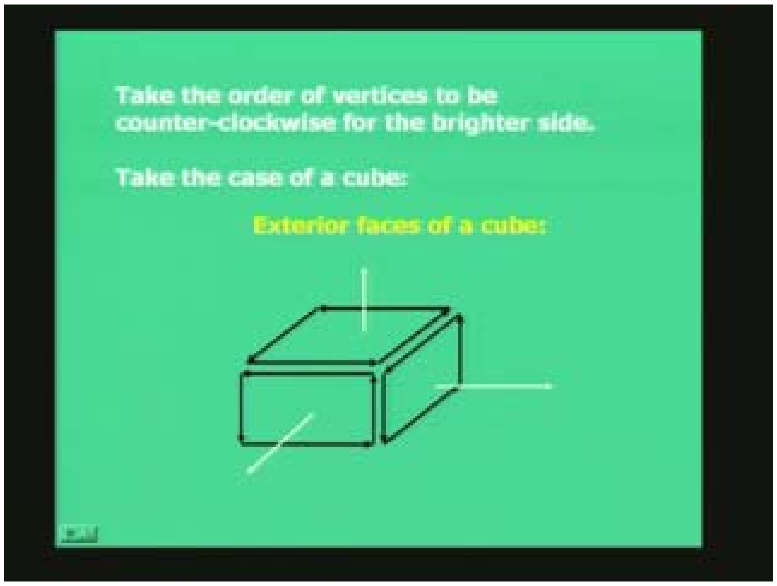
Order of vertices for calculation of N – w.r.t the front side of the surface.

We follow the following notation that the brighter side of a face is visible when we are seeing the face and take the vertices in a counter clockwise arrangement. The reverse side will be clockwise that means if you take the clockwise arrangements of pixels you are basically looking into the darker side and if you are taking the arrangement in a counter clockwise arrangement it is a brighter side. Henceforth when you make the vertex table or the edge list or the polygon surface table and also store the corresponding vertices or the edges inside you must be very careful in the sequencing of the corresponding vertices or edges of the polygon because you must store it in a counter clockwise arrangement to show that it is the brighter side which you have seen.

Accidentally if you have stored the darker side the algorithm will reverse it and take the counter clockwise arrangement and make the darker side better which will cause confusions for you. So this is what I meant by the brighter and darker side when you need to take the vertices and hence the edges in a counter clockwise arrangement to see the brighter visible part of the surface. Then of course depending on the n that the brighter side may be a frontal face or a back face which is involved in the back face culling or back face removal algorithm that is fine and that is the main part. But here we are bothered about to know the correct order of the sequencing of the vertices or edges of a polygon to compute n with respect to the brighter side which will be visible and not the darker side which is inside the polyhedron inside the object which will never be visible under any circumstances.

It is something like if I take the surface of a ball football which is a sphere, a very good example, that surface has an inside, inside the ball itself we have a surface which is never visible unless you tear-open the ball, tear the leather of the ball, open, it never will be visible, always the outside surface of the ball or the sphere will be visible to you. The inside part of the polygon, inside part of the surface will never be visible that is the darker side which we talk about, the light will never enter, the light always come and hits the brighter side and that could be a front face or a back face which will be shown. So I hope the concept which I am repeating several times is clear to you now that to compute design we compute the n with respect to the brighter side and that is done by taking the order of the vertices in the counter clockwise for the brighter side.

(Refer Slide Time: 12: 50 min)



Take the order of vertices to be counter-clockwise for the brighter side.

Take the case of a cube:

Exterior faces of a cube:

Let us illustrate with the help of a case of a cube. This is what we call as the exterior face of a cube. What I have done is I have taken the three visible parts or faces of a cube which are visible to you. And these are the brighter sides of the faces of the cube. These are the three polygons, the three rectangles for you and the surface normals are shown with respect to the brighter side.

Remember, there is a darker side of these faces which you could have taken if you have taken the order of the vertices in a clockwise arrangement. If you go and try to look inside you will find that this order will appear clockwise with respect to the darker face. Whereas with the brighter face if u take a counter clockwise you are actually looking into a brighter side and this is how you compute the surface normal which will always appear to be coming out.

If you do not take any two orthogonal vectors P and Q and C then try to take a cross product of that you will see that you will get this end which is not for the respective polygon. This is how you compute the n. You do not take two arbitrary vectors you basically take an order and then compute. So you need to have a formula where you just do not take to arbitrary vectors but you need to computer normal based on all the vertices because you accidentally pick up two vectors which are parallel vectors and then you cannot compute a cross product of two parallel vectors.

You can choose any two vectors typically of edges to obtain the normal to a surface you can do that but there will be lot of risk. Can you think of any risk in such a case if you randomly choose any two vectors? As I said before P into Q after the counter clockwise arrangement is done. Somebody can just use any two arbitrary vectors P into Q in that sequence after it is arranged in counter clockwise arrangements. You can do that still obtain n with respect to the brighter side of the face and then check whether that brighter side is becoming a back face or not. That is fine. But is there any risk to randomly choose any two vectors P into Q? There is a risk.

If you think and if you are able to visualize the first and important problem it is very easy to visualize if this P and Q are accidentally parallel vectors like the case in the previous one here. In any face you can choose two parallel vectors P and Q and you know when two parallel vectors are chosen the angle fi is equal to 0. Then of course the normal product itself will be 0 and you cannot obtain that. What you do? Can you think of the situation? Then of course you choose non parallel vectors. Still do you think will there be any risk in choosing?

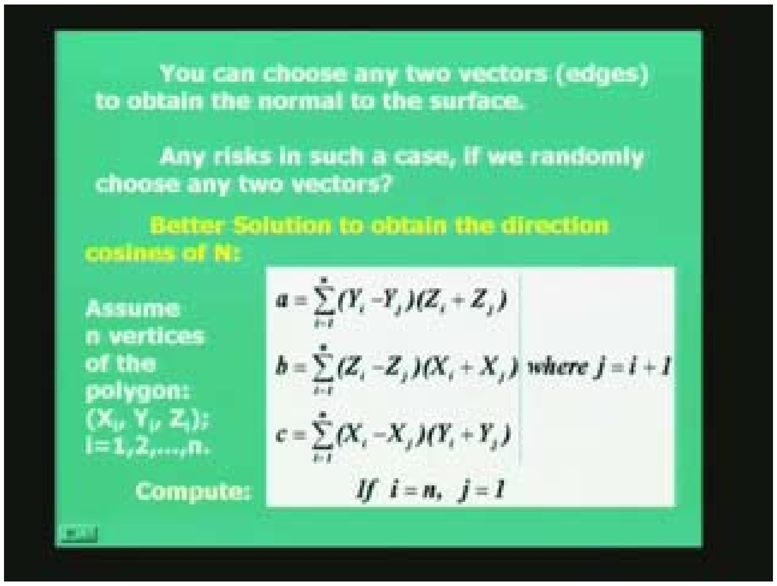
Now, there will be a risk because even if they are non parallel if they are accidentally very close to becoming parallel that means the fi angle between the two vectors P and Q which you are choosing should be perfectively wide apart, much larger. But accidentally if they are very close if not 0 but even if it is any small value what this will result in? The computational value R or the normal n of the polygon which will be very small in terms of its magnitude and I would you like to avoid that small is not a problem in a computer but a too small number is a problem.

I would like to lead with larger numbers where accidentally I would not allow numerical errors to creep in, in terms of truncation and round of. If possible I would like to have larger values to play even if the computational time could be larger because you keep looking for two vectors which are not only non parallel but not close to becoming parallel also will pick up additional computational burden for me if I have to keep choosing pairs out of n different vertices or n different edges.

So I would like to avoid this computational burden and look into a formula which will handle all cases of parallel or not in fact it should take all n edges or vertices into account and come up with the formula which will give me the n rather than keep choosing vertices or and then edges which are not parallel but almost orthogonal if preferable.

So let us look at this, yes there is a problem. So the better solution is to obtain the direction cosines of n is this. We assume there are small n vertices of a polygon, capital N is normal to the plane of the polygon and there are n vertices of the polygon, the X_i Y_i Z_i or the coordinates of the ith vertex and i keeps running from 1 to n as given in the left hand side here I am reading here, assume n vertices of the polygon and each vertex has coordinates X_i Y_i Z_i because it is a plane in 3D, vertices have 3D coordinates X_i Y_i Z_i, i runs from 1 to n or 0 to n minus 1 because there are n vertices of the polygon. If that is given to us you compute these three coefficients a b c. Are you able to recognize these expressions?

(Refer Slide Time 16:40)


You can choose any two vectors (edges) to obtain the normal to the surface.

Any risks in such a case, if we randomly choose any two vectors?

Better Solution to obtain the direction cosines of N:

Assume n vertices of the polygon: (X_i, Y_i, Z_i); $i=1,2,\ldots,n$.

Compute:

$$a = \sum_{i=1}^{n} (Y_i - Y_j)(Z_i + Z_j)$$

$$b = \sum_{i=1}^{n} (Z_i - Z_j)(X_i + X_j) \quad \text{where } j = i + 1$$

$$c = \sum_{i=1}^{n} (X_i - X_j)(Y_i + Y_j)$$

If $i = n, \; j = 1$

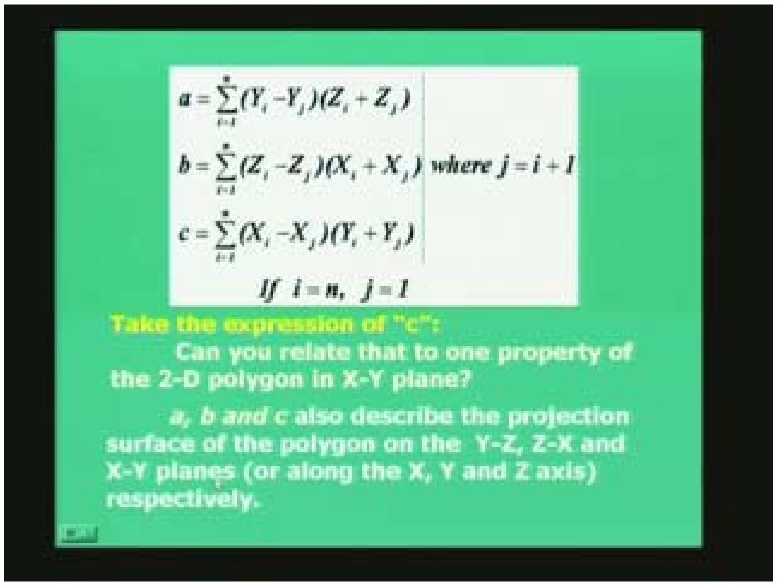
I think we have got this expression some where earlier in a previous class. Can you guess where you have got this expression? If you think carefully we exactly got the same expression to compute the surface normal of the given n set of vertices earlier. But that point of question I did not talk about all these logic about in a direction cosines by picking up any two vertices arbitrary early and then you know the brighter and side face because those are related to visible surfaces reduction algorithms. But these are the direction cosines.

If you remember these expressions what we got earlier j is equal to i plus 1 when i is equal to n of course we are saying that j is equal to 1 that is come back to the first vertex and that is what you do to compute the surface one. So this is the expression we have and we know how to compute the surface normal and to show some more light on these expressions now let us go a little bit fast.

Take the expression of c, this we also did in the last class and we break open that expression and took the case for a triangle and you can relate that one property of the 2D polygon in the XY plane in fact we did it for a triangle and we can also verify for any polygon that a b and c also describe the projection of the surface of the polygon on the respective coordinate planes YZ ZX and XY planes. That means a b c are related to the projection.

Remember, the polygon in 3D plane, so you project it either on XY plane or YZ plane or ZX planes the three orthogonal axis that means perpendicular to one of these axis and you will find that the projected plane will only have two out of the three coordinates either you are throwing out Z, X or Y. So that is what you get the formula a b c. You also describe the projection surface of the polygon in the respective Y_Z that is for a, you have Y and Z co-ordinates for b and you have Z and X and for c you have X and Y.

(Refer Slide Time 19:15 min)



$$a = \sum_{i=1}^{n} (Y_i - Y_j)(Z_i + Z_j)$$

$$b = \sum_{i=1}^{n} (Z_i - Z_j)(X_i + X_j) \quad where\ j = i + 1$$

$$c = \sum_{i=1}^{n} (X_i - X_j)(Y_i + Y_j)$$

$$If\ i = n,\ j = 1$$

Take the expression of "c":

Can you relate that to one property of the 2-D polygon in X-Y plane?

a, b and c also describe the projection surface of the polygon on the Y-Z, Z-X and X-Y planes (or along the X, Y and Z axis) respectively.

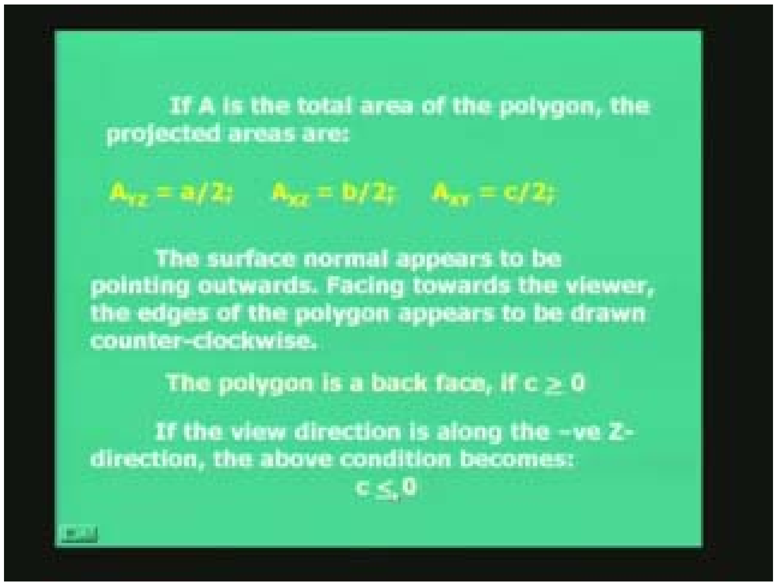
So, if a is the total area of the polygon, this is a new concept which you can keep in mind then the projected areas of the polygon on the respective orthogonal planes is that the area will be a by 2. When you note the formula for a which you have seen this is the a b and c which you compute by these three summation of the product terms expressions the sum of product terms and the projected area on the YZ plane is this a by 2, projected area on the XY plane is b by 2, projected area on the XY plane is c by 2. That means a b c are nothing but we can also visualize them they are twice the respective projected areas on the respective planes. When 'a' is the direction X component of the direction cosine, I repeat; if that coefficient a b c is what you are computing the small a b c, you are computing the direction cosines of the surface normal to the polygon.

Here a is the X component of the direction cosine, b is the Y component of the direction cosine, c is the Z component of the direction cosine, of course that c is very important for back face culling I do agree that you all know. But if you look at a only let us say to start with, a is X component of the direction cosine of the surface normal and it is twice the area, it is double of the area which will be obtained by projecting these 3D polygon on the Y-Z plane, compute the area that area will be double the area which you will get and that area you will get the X component of the direction cosine. So it is proportional to the X component direction cosine. So a is twice that area on the Y Z plane, b is the Y component of the direction cosine so it should be twice the area of the polygon projected on the X Z plane.

Similarly, c is the Z component of the direction cosines of the surface normal as you can see carefully here on the slide, if c is the Z component of the direction cosine of the surface normal then that is twice (X, Y) that is twice the area projected of the polygon on to the XY plane. So remember this important relation and then you can test it with a simple triangle. So if the surface normal is computed by the above three coefficients by the above formulas we have seen in the formulas of a b c, they appear to point outwards facing towards the viewer and the edges of the polygon appear to be drawn counter clockwise. This is what we mean by a back face, front brighter face and a darker face and we have already seen this. I repeat again; the surface normal appears to be pointing outwards from the face facing towards the viewer the edges of the polygon appears to be drawn counter clockwise, that is what is the situation which we know. And the polygon is a back face you need to just compute the c, check its sign if it is positive then the polygon is in back face. So this is simple.

In fact when testing a back face I did mention sometime earlier that you do not need to compute the X and Y components of the direction cosines you just need to compute the c which is the Z component of the direction cosines check its sign and that is what you will get as your back face or not. The c is positive for a back face provided you are looking then you have to just do the opposite, look into the slide there is a small note in the positive Z direction. If your view direction is negative in view here which you can make that if which is flashing on your screen which reads if the view direction is along the negative Z direction the above condition becomes c negative less than equal to 0.

(Refer Slide Time 23:14 min)


If A is the total area of the polygon, the projected areas are:

$A_{YZ} = a/2$; $A_{XZ} = b/2$; $A_{XY} = c/2$;

The surface normal appears to be pointing outwards. Facing towards the viewer, the edges of the polygon appears to be drawn counter-clockwise.

The polygon is a back face, if $c \geq 0$

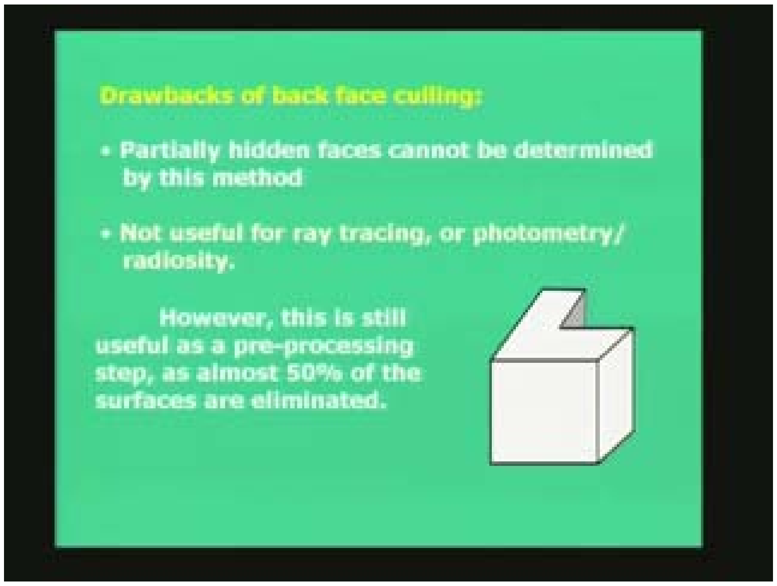
If the view direction is along the –ve Z-direction, the above condition becomes:

$c \leq 0$

So it depends upon whether you are looking along the positive Z direction or negative Z direction. You are checking the sign of c to check whether it is a back face. The brighter side always could be a frontal face or a back face that is true. Never have we bothered about the darker side and we have computed the surface normal with the direction cosines n by taking this counter clockwise arrangement and we are assuming that this n is coming out from the brighter side which may be visible, brighter side of the polygonal face.

Now you need to compute only the c that is the Z component of the direction cosines check its sign. If this view direction is along 0 0 Z positive Z direction check if the sign of that Z component of the direction cosine that is c is positive or not. If you are looking along negative Z direction what you need to check? You need to check if that sign is negative or not. That is what you look back, just be careful that the first condition holds good that the polygon is the back face if c is positive if you are looking along positive Z direction. But remember if your algorithm or situation demands that if the view direction is along the negative Z directions then the above condition becomes c less than 0. So I hope this condition is clear and then we move on to the next topic. But let us look into the drawbacks of back face culling.

(Refer Slide Time 22:33 min)



Drawbacks of back face culling:

- Partially hidden faces cannot be determined by this method
- Not useful for ray tracing, or photometry/ radiosity.

However, this is still useful as a pre-processing step, as almost 50% of the surfaces are eliminated.

We discussed about back face culling. The main idea of back face culling is to know whether there are some polygons with back face or not. And it is good, the computations are very simple and you just need to compute the c and check its sign that is all you have to do to compute. You do not need all the components of the surface now and to know whether it is a back face just compute the c. But the back face culling can do certain things, it cannot do certain things.

Let us look at it, partially hidden faces cannot be determined by this method. This is the first drawback of back face culling. Drawback in the sense it is not suited for this task. If you look at this particular truncated cube of course there are few faces at the bottom and to the left and of course the back faces which are two of them which are not visible. And there is a face on the top and the front left and the front right there are three of these white polygons which are visible to you, that is absolutely not a problem. And of course there are few faces at the bottom of the solid to the left back side which are not visible. But look at that part of that solid which is partly visible. You cannot tell using back face culling whether that face is completely visible or it is partly occluded.

You can actually only say whether it is a front or back, whether it is partly visible or not, a situation like this you can say it is a front face but there is something like this here which we also say this is a front face but you cannot compare two different surfaces and tell you whether a surface is partly or completely visible. That is the key essence of VSD algorithms or visible surfaces. So back face culling cannot do that but it only helps you to throw out about 50 percent of the polygon faces and say that is back face, you never compute the normals and do any more process. You do not use this polygon to compare depths for to finding out partially occluded or occluding surfaces, you will never do that. That is done by VSD but you always use back faces and you know that you do not try to find out partially hidden faces using back face because that cannot be done by back face culling. It is not useful for advanced concepts of visible surface detection algorithms or illumination and shading algorithms which involves ray tracing, photometry or radio city. When we talk of ray tracing in the future classes we will see ray tracing, radio city, photometry and why this back face culling does not play a significant role.

However, we move forward to VSD algorithms with the note about back face culling that this is still a very useful preprocessing step as almost 50 percent of these surfaces are eliminated by this method. Already I told you about this earlier that the back face culling is a method by which you eliminate a lot of faces. Typically on an average case about 50 percent of the faces which will never be visible to the user based on a certain viewing geometry or thrown out by back face culling and hence it is always used as a preprocessing tool before applying any VSD algorithms or illumination and shading algorithm.

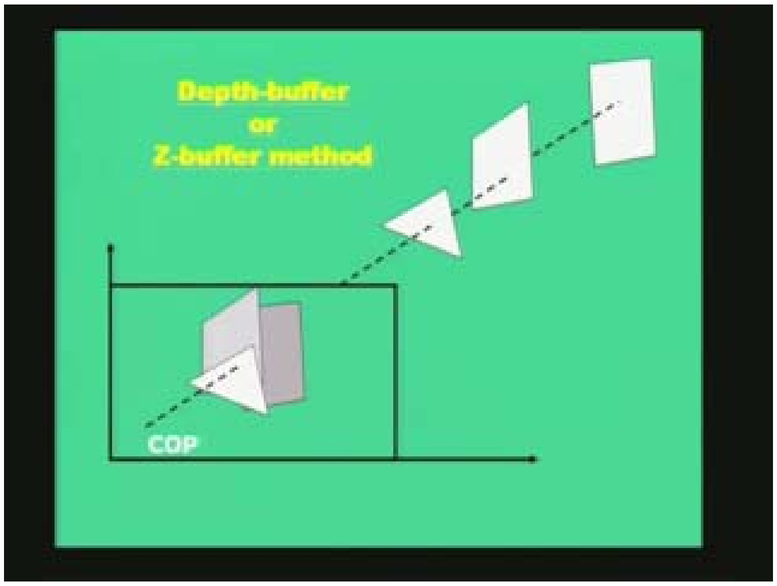
Remember, I discussed about this in the earlier class that we are almost in the last stage of the viewing pipeline before mapping into view port the polygons which will be shaded and rendered, we are throwing out a few back faces, we are finding out the frontal faces we also finding out the part of the occluded faces which are visible to us and then of course we have to compute the illumination and shading in the next chapter or the next section of the class.

All these have to be done before actually drawing the picture on a particular viewport. The last stage of the viewing pipeline is what we are for here and here we are involved in visible surface detection but applying any visible surface detection we have given few lists of those in the last class. We will continue to discuss that in the class today. A few of those algorithms, one of them today and more so in the future classes.

But always you must keep in mind back face culling has to be done. It must be done or it has been done before I apply in the VSD algorithm because the back face culling will throw out a few of those faces which are back face and will not be involved in the computations of VSD. That helps to throw out 50 percent on an average case a number of polygons and you can now visualize the speed up which will get in the feature algorithms of VSD and rendering a 50 percent of those back faces are thrown out. So remember this point that this back face culling in spite of having its drawback it is a very simple computation, the direction cosines, the Z components of the direction cosines of the surface normal has to be computed, check the sign, depending upon that identify if it is a back face or not and then move forward towards VSD.

Remember, it has drawbacks but in spite of all these drawbacks this is still a very useful preprocessing step as almost 50 percent of the surfaces are eliminated. So we will use of back face culling then move over to visible surface detection algorithm which we will discuss next. And first one in the list which we are going to start today is discuss about VSD algorithms after back face culling is called the depth buffer or Z buffer algorithm. Let us look at this slide.

(Refer Slide Time 30:05 min)



This is what we have as a slide for the depth buffer or Z buffer algorithm where we see that we are seeing through a window a set of three polygons which are visible. And one is obscuring the other they are not separated out. They may be separated out but in this case they are not. Let us talk even about orthogonal projection but the same is true for perspective projection. But at this point of time again I bring in the concept that we are in end of the three filling pipeline, normalizing transformations done, clipping is done, scaling and zooming to bring everything under the canonical view volume, you also do the transformation matrix which will transform our perspective projection canonical view volume, the pyramidal structure into a rectangular parallelepiped in a canonical view volume so everything is orthographic right now.

The projection transformation will also handle on the special case of orthographic. And within the orthographic projections all rays are parallel and moving towards a certain center of projection which is at infinity. That is what the rays are going from the object. So this is a case which we have seen here. So the rays will be coming out of the object and intersecting the projection plane, this rectangle image of the projection plane here and is moving towards the center of projection in 3D and as you can see if there is an obscure on an overlap for a particular ray we need to find out which of these three surfaces at a particular point is facing you. That will tell you which is the frontal face, which is partly occluded first and which is partially occluded second. Now this is an example of only three polygons.

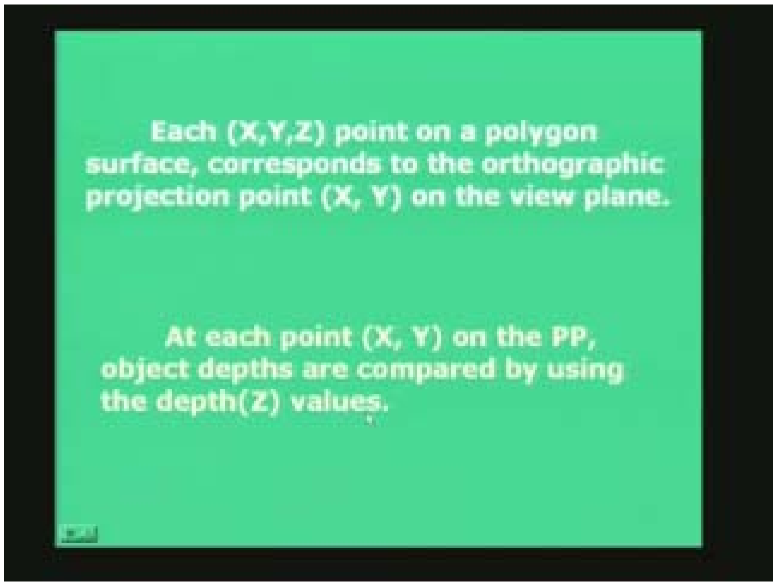
I would like you to imagine let us say in addition to three you have many other polygons, say ten of them or hundred of them or even more than that. Visualize few hundred to a few thousand polygons and one is obscuring the other and so on as you go and at any point of time in an image plane you will able to see one of those polygons and you have to find out which of those polygons is in front of you based on a certain ray which is going or if you take of course the ray costing method is more suited for this task but in z buffer also we will see we have to compare the depths of these three polygons at all instance of time based on a certain pixel point and then find out which is facing you and which is in the front of you.

I did say sometime back in the previous class that visible surface detection is nothing but a 3D sorting algorithm and the comparison of sorting is done based on depth. Depth is the queue or the key based on which the sorting is done. Then the sorting is done with the help of integer values or just floating point you just compare the values and do a sorting. So you need to sort this polygon based on depth and that why it is called 3D sorting algorithm. Let us see how a depth buffer works or Z-Buffer works where we have to start to compare the depths of these three polygons across and over this image plane.

We will say that each X, Y, Z point on a polygon surface corresponds to an orthographic projection point (X, Y) on the view plane. That is what we have seen. So, if you take any X, Y, Z point on a polygon that corresponds to a particular point (X, Y) on this projection plane, that is true for any of these. You can put the line anywhere passing through any of these polygons or may be only one or two or three and you will find that the ray which will be passing through a point on X, Y, Z in a polygon will also intersect to the projection plane at a point (X, Y). That is what this sentence means that each X, Y, Z point on a polygon surface corresponds to the orthographic projection point (X, Y) on the view plane. This is interesting, that is why in the end of the viewing pipeline everything is put in an orthographic way.

Orthographic projection is the most impressed form of getting a projection because if you have a point X, Y, Z in 3D its corresponding point projection point on the projection plane can be easily obtained by just chopping or throwing out of Z value. X, Y, Z take out Z keep the (X, Y) values that is going to give you the point on the projection plane whereas that is not the simple case in the case of perspective geometry. In perspective geometry you have to use transformations, equation and all that. In this case also you can use but you can almost visualize that you do not need anything of those in orthographic projection X, Y, Z point projecting on to a plane. Throw off the Zth value from the 3D coordinates, take the first two components and it is your projection plane keep that in mind that. That is why orthographic projection is often used at some point to simplify the calculations.

(Refer Slide Time 35:21)


Each (X,Y,Z) point on a polygon surface, corresponds to the orthographic projection point (X, Y) on the view plane.

At each point (X, Y) on the PP, object depths are compared by using the depth(Z) values.

Then the reverse is also true that at each point XY on the PP is a projection plane, you remember this term hopefully which you have learnt from 3D viewing pipeline 3D viewing geometry we discussed those issues earlier in the previous class. So at each point (X, Y) on the projection plane object depths have to be compared or compared by using what we call the Z values.

Now if you have say two planes plane one and plane two and if I take a projection plane to your right say at a point (X, Y) so that point (X, Y) will correspond to a point X_1 Y_1 Z_1 or X, Y, Z_1 for an image plane, for a polygon plane as one and for polygon two it will be for a corresponding other point, it will have an another point X, Y, Z_2. So if you have two planes you have X, Y, Z_1 and you have X, Y, Z_2 and Z buffering or depth sorting nothing but compare these two values Z_1 and Z_2 keep comparing them as Z_1 and Z_1 and that is what you need to do in VSD.

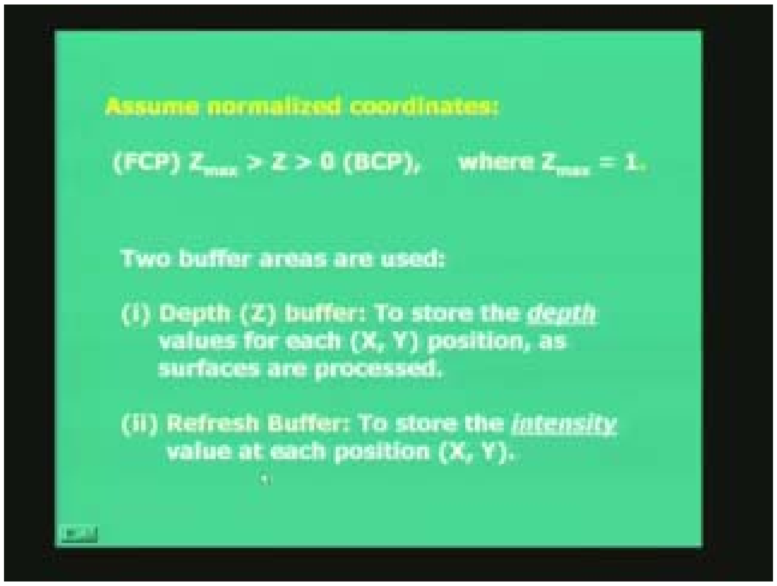
So at each point (X, Y) on the projection plane object depths are compared by using the values in this case I am showing by hand that there are two polygon surfaces but you can more than two, three is already shown in the previous slide but you have tens of those or hundreds of those and all those Zs are compared and you basically need a sorting algorithm to arrange this in some ascending or descending order as the case is required. That is what you do for your depth sorting, this is the key idea.

Let us look at some of the mathematics involved in comparison of the case. So assuming normalize coordinates if you have a Z value and assuming that the Z value for your set of objects is lying between your front clipping plane and back clipping plane you remember these terms when using in 3D viewing pipeline. So we are talking of Z values in the canonical view volume normalized coordinate systems 0 to 1 or 0 to certain Zmax, Zmax could be equal to 1 for your front clipping plane but it could be anything else and if that is so the algorithm uses two buffer areas or two memory areas where one is the depth buffer or the Z buffer as the name suggests in terms of the algorithm which will store. That is used to store the depth value for each (X, Y) position as surfaces are processed. So, we are processing polygon by polygon and we are updating the Z at a particular value of (X, Y) that is what we do. There is another buffer called the refresh buffer which stores the intensity at each point (X, Y). Now although these two buffers are used but in terms of VSD algorithm the depth buffer is the most important part that is what we will see because we will keep the computation involved in computing the refresh buffer or the intensity value for the later time when we discuss the illumination and shading algorithm.

Thus, for the time being we assume that we have an illumination and shading algorithm which will compute at each point (X, Y) on a polygon plane, what is the intensity? That is again a function of the surface normal N and surface normal N was computed earlier when we needed to do a back face. So we are computing only for frontal faces not for back faces any more. And for those frontal faces we sort them based on Z using the Z buffer and compute the intensity and store them in the refresh buffer at the same point of time.

The computation involved in the refresh buffer of i (X, Y) will not be discussed in the VSD it will be discussed later on based on the illumination model which you picked up although it is a significant part of the algorithm. But it does not play any role in terms of comparing Z. So for comparing Z, Z of X, Y at depth buffer is the one which we will used and we will discuss that. So coming back to these two concepts here two buffer areas are used. One is called the depth buffer or the Z buffer which stores the depth of value for each (X, Y) position as surfaces are processed and another of course is the refresh buffer, sometimes people call it the intensity buffer also which stores the intensity value at each position (X, Y).

(Refer Slide Time 39:11 min)



Assume normalized coordinates:

(FCP) $Z_{max} > Z > 0$ (BCP), where $Z_{max} = 1$.

Two buffer areas are used:

(i) Depth (Z) buffer: To store the *depth* values for each (X, Y) position, as surfaces are processed.

(ii) Refresh Buffer: To store the *intensity* value at each position (X, Y).

So, remember that the Z values run from 0 to 1 and there are two buffers which are used for the algorithm. So let us look at the steps of processing for the depth buffer or Z buffer algorithm. We have an initialization step where for all (X, Y) points the two buffers are initialized. The depth buffer or the ZXY you can either it as ZXY or depth XY depending upon your convenience. Depth XY is initialized to the maximum Z value and this maximum Z value is away from you and that is what you must visualize and refresh buffer is assigned to a i b which I will say it is a background intensity color.

Now there is an important point to be visualized here. Your view direction could be along the positive Z or negative Z. There are two things which are usually used interchangeably. And it is important to take any one of these because Back face culling algorithm computation becomes very easy because you have to look into the sign of that c which is the Z component of the direction cosine of the surface normal of the polygon to compute a back face. After all the back faces are removed you just concentrate and work on the frontal faces to do the ZXY or for any VSD algorithm. But for the ZXY now here we are talking of either positive or negative Z, the sign of c for the back face culling could vary positive or negative depending upon whether you are looking positive Z or negative Z. But here also that is an important factor.

When you are looking along positive Z you must know that the faces which are closed to you will have a small value of Z and faces which are for away from you will have a larger value of Z. That is the Z max will be numerically more than the minimum values of Z which are closer to you that is true. But when you are looking along negative Z just visualize negative Zs what will happen is numerically the surfaces which are larger away from you or farther away from you in fact will have a larger value of Z in the negative direction. Larger value, negative means smaller in the number scale they will be lesser value and that means surfaces which are closer to you will have smaller negative values in fact they will be larger in that. So if you take the absolute value in both cases the Z max will be larger.

But if you take numerically the algebra the difference and subtract from the other and say from negative scale towards 0 to towards positive the values are increasing in this form. So if you are looking along negative Z the Z max values will be less. You can look along negative Z sitting somewhere along on the positive Z axis which is you can look towards origin that itself is a case of looking towards negative Z. I repeat; think of an X, Y, Z coordinate system and you are sitting somewhere on the positive Z axis. If you are looking towards the origin your viewing direction itself is negative Z. So in that case the farther values which are towards 0 are less values of Z but they are farther away from me. And values which are larger values of Z are closer to you.

Now this is going to be different when you are view direction changes to positive Z you could sit along negative Z you could sit on the negative Z axis and look towards origin or you could sit on the origin and look towards positive Z axis, the situation differs because here larger values of Z will be farther away from you and smaller values of Z will be closer to you. So keep that in mind, I hope you are able to do the visualization of looking along positive Z axis and negative Z axis not only the back face culling condition of the sign of c changes but this concept of Zmax numerically being higher or lower will change.

I hope you are able to visualize that because based on the concept or based on the way we are going to put the algorithms in terms of Zmax looking along positive Z direction or negative Z direction and based on the method you choose for your viewing pipeline, view geometry and all that these Z max could vary. But you have to be consistent in your algorithm. It must not be like today your algorithm picks up this, tomorrow your algorithm picks up positive Z directions and things like that. So we are assuming here that steps for the processing initializes for X all (X, Y), the depth (X, Y) to a Z max value depending upon the negative Z or positive Z that is in which direction you are looking and the Z max could be larger or a small value whatever the case may be.

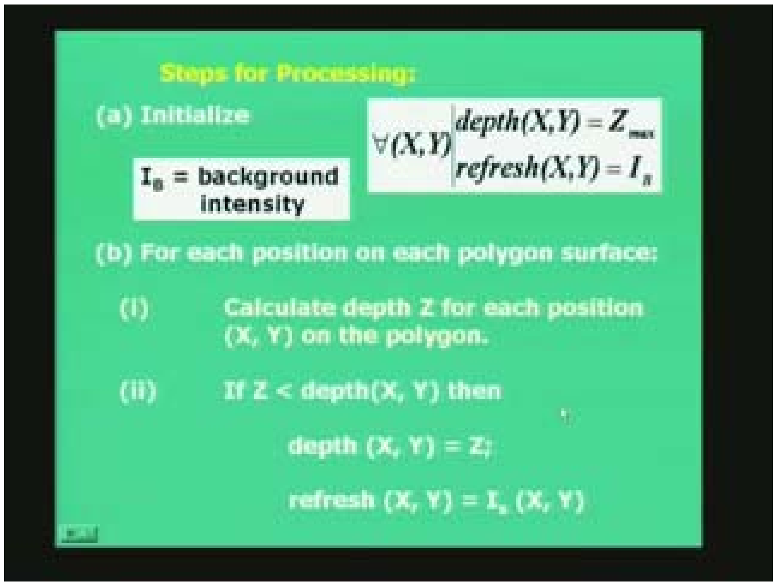
Now, the second step says that for each position on each polygon surface that means you process polygon by polygon. So, if there are P number of polygons you start with a first polygon then the second polygon and so on and what you do for each polygon? This is what you do calculate depth for each position (X, Y) on the polygon. Now it is a polygon filling algorithm, you can always take a polygon in 3D given the n sides or n vertices of the polygon project that on the projection plane which is the XY plane typically, you will get a polygon in 2D.

Orthographic projection mind you just up of the Z values or neglect or omit Z values out, take the X_i Y_i of the set of polygon vertices you will have the 2D polygon. And when you have a polygon in 2D you know how it fill it in a scanner algorithm. Use the same concept of 2D polygon filling and what you get from 2D polygon filling algorithm? A set of (X, Y) point which is to be shared which are inside the polygon and which for each scanline.

So we are talking of those set of (X, Y) points on the polygon in 2D for which the depth Z for each position is known. Why it is known? Either it is known if it is on the vertex or you need to calculate that, that is okay. The Z point for each (X, Y) can be calculated because that is the Z which you are thrown out to get a point (X, Y) of course for the vertices. But for the each point on polygon with genius scan line you can also calculate Z and we will see how. So assuming that you know these calculations to be done to obtain the depth Z for each position (X, Y) on the polygon you compare the Z with the buffer values stored for the depth buffer or Z buffer at the corresponding point (X, Y) and find out if the Z computed in the previous step b_1 here is less than the depth (X, Y) which has been initialized a prairie or by the previous polygon process if that is less then you do the following.

You update the depth (X, Y) to the Z value which is computed only when this condition occurs and the refresh buffer (X, Y) which was probably assigned a background color at the initial stage or it has an intensity of a polygon process earlier we will have the surface intensity corresponding to that point X, Y, Z of the polygon.

(Refer Slide Time 46:25 min)



Steps for Processing:

(a) Initialize $\forall (X,Y) \left| \begin{array}{l} depth(X,Y) = Z_{max} \\ refresh(X,Y) = I_B \end{array} \right.$

I_B = background intensity

(b) For each position on each polygon surface:

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

(ii) If Z < depth(X, Y) then

depth (X, Y) = Z;

refresh (X, Y) = I_s (X, Y)

So these are the three simple and very qualitative steps of the Z buffer algorithm. Of course there is a lot of calculation which goes behind implementing these algorithms but the conceptual is very simple. You initialize, process polygon by polygon and when you process each polygon by polygon for each polygon scan through all the (X, Y) points look into the Z, compare the Z value with whatever is stored in your depth (X, Y) and that is the comparison.

We discussed about depth sorting comparing Z and this is the step in the step number b_2 which you have in the slide, we will see it again now there is where you compare depth and if the comparison succeeds based on a certain condition greater or less it again depends upon where you are looking either in the positive or negative Z direction you update your depth buffer only at that point (X, Y) with the depth value and as well as the refresh buffer with the distance intensity.

So we look at the steps for processing, the first step is initialize with the Z max and the background intensity and process polygon by polygon is step b which has two parts, the first part of step b is calculate the depth value, we look into the calculation in the time remaining in the class today and then when this depth value is found to be less than depth of (X, Y) for the depth buffer, it is the depth buffer, this depth does not indicate the depth of the polygon. The Z is the depth of the polygon but depth (X, Y) is the depth of the buffer which is used to implement Z buffer algorithm.

In that case if this comparison succeeds the depth (X, Y) is updated with the depth value and the refresh buffer at that point (X, Y) is also updated with the intensity. Otherwise do not do anything. So this part of the computation of how to compute the Is we will keep it pending till we discuss illumination and shading algorithms. But for the time being we know that there is a method to compute Is which is used to update the refresh buffer and we have to only see how to compute the depth value.

I hope the key ideas of the refresh buffer is what you have seen in the slide just now is very initialized and process polygon by polygon for each polygon compare depth and update Z and based on the comparison update ZXY and also if you are updating ZXY update the intensity refresh buffer also. So Is is the projected intensity value of the surface at position (X, Y) which has the minimum value of Z at that current stage of iteration. So this method of computing the intensity value at a position (X, Y), we will discuss is later on but this will be updated only when the Z (X, Y) buffer is updated for a particular polygon.

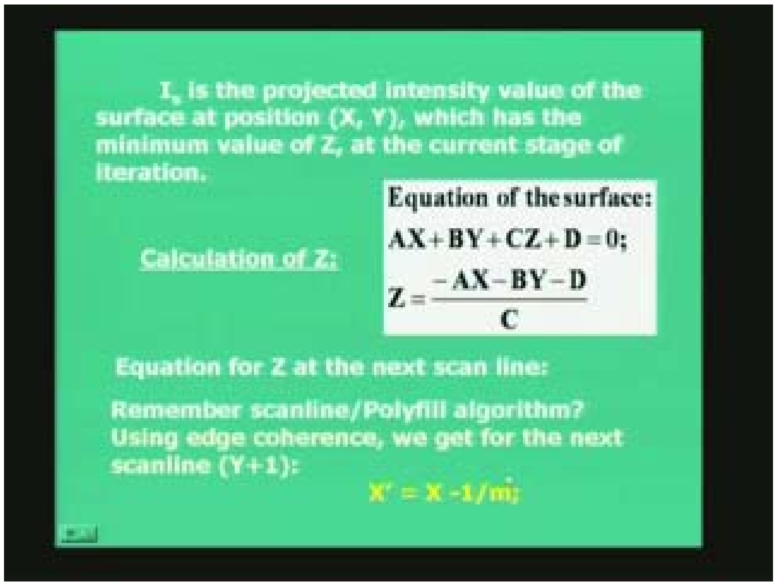
Now let us look into the calculation of Z so that is the most important part in this process. Let us look at the equation for a surface; we know we have seen these equations earlier when we discussed about solid object modeling and you can write the expression for the equation of a surface with direction cosine A B C as AX plus BY minus D you know that. And henceforth you can write the expression for Z as this given here so the numerator divided by the denominator C.

Now if you see A B C and D are the coefficients of the surface which have to be computed a prairie for a particular polygon and then what you need to do is once you have the coefficients A B C and D with you, you basically need to substitute the value of X and Y compute the right hand side and that is how you compute Z. So the process of computing the value of set for a position (X, Y) on the projection plane involves the computation on the right hand side where the coefficients A B C and D are known to you and then the (X, Y) value is picked up from the point on the raster buffer or on the projection plane substituted in the right hand side, compute denominator by denominator term will give you the value of Z and that is how you compute the value of Z for a particular point.

But there are problems in computing the value of Z for each particular point. All the set of points on the polygon will have different values of Z in general. And if you need to repeat this calculation again and again for each particular point (X, Y) you will be doing lot of computations and that could be very costly.

In fact if you do that we should say here that you are not exploiting the concepts of object edge and face coherence property is of polygon surfaces and scanlines to compute the value of Z. So let us look if you can use some method by which we can do incremental algorithm some incremental values to be added to a previous value of Z to get a new value of Z. So, let us look at the equation of Z for a next scanline and that is what we want. What you mean by that? You remember, the scanline polygon filling algorithm, in that if you remember we used edge coherence to get for the next scanline Y plus 1 a value X prime which is X minus 1 by m. What was this m value? It was a slope of that line in 2D. If you have an edge of the polygon in 2D running from point X_1 Y_1 to a point X_2 Y_2 you can calculate this slope m of that. So, that m in turn will help you to obtain that when you move from one scanline to another one what is the next intersection of that scanline with the edge of the polygon.

(Refer Slide Time 52:06 min)



I_s 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.

Calculation of Z:

Equation of the surface:

$AX + BY + CZ + D = 0;$

$Z = \frac{-AX - BY - D}{C}$

Equation for Z at the next scan line:

Remember scanline/Polyfill algorithm? Using edge coherence, we get for the next scanline (Y+1):

$X' = X - 1/m;$

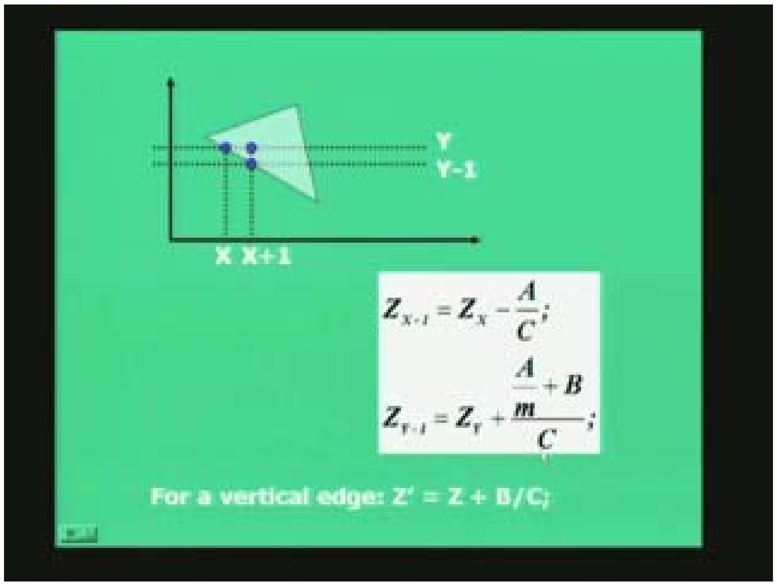
We did this computation and we used integer algorithm even to compute the next X point for X. What does this mean in respect to the polygon in 3D? Let us say, if you know some how the Z at the particular point (X, Y) here so the left there are three dots which are marked here in this figure and this figure says that this is a projection of a polygon in 3D on the 2D (X, Y) domain. And if it so for a particular point (X, Y) if the Z value is known is it possible for us to compute the Z value for the next point X plus 1, Y or even for the next scanline intersection with the polygon edge which will be at some point let us say X plus 1 Y minus 1.

Now remember you can use these values of X and Y at each of these points and use it to substitute it to the expression which we got Z in the previous slide we can use that for computation. But that computation will be enormous because you have to do that for each and every point (X, Y) on the polygon. So we want to save time by seeing if we can simplify these calculations using some incremental logic. That is what we are concerned here assuming that the Z value is known at a point (X, Y) is it possible for us to compute to Z value for the next point in the same scanline or the next point in the next scanline which is the intersection of a scanline Y minus 1 with this edge of a polygon which will be intersecting at X plus 1 or something else that depends on the slope of the line. In fact that 1 by m will dictate whether it will intersect that X or X plus 1 and that you can find by 2D plan scanline polygon filling algorithm. But you are interested in Z to compute the Z value for each of these points.

So if you go back the calculation of Z is this which you want to avoid and this next scanline polygon filling algorithm in 2D which will give you the next X prime intersection of the next X coordinates based on the slope of the line. So if this is the condition that means you assume here that Z at this particular point (X, Y) is known. Then from the previous expression of Z if you substitute Z for X plus 1 I leave this calculation for you to come out that the Z for the next X plus 1 will be given by this expression.

Please try this calculation it will be Z_X minus A plus C and whereas for the next Z_Y plus 1 you have to substitute Y plus 1 for the next one it will be basically Z_Y with this X should incremental by 1 by m which will be this. So these are the two calculations which I leave it to you as an exercise to compute and find out and if you can get these calculations done you can obtain the Z value for the next point X plus 1 of the next scanline using from the coefficients which are nothing but the direction cosines of the surface A C B and the M are the parameter which are known from the 2D and the 3D properties of this polygon.

(Refer Slide Time 55:30 min)



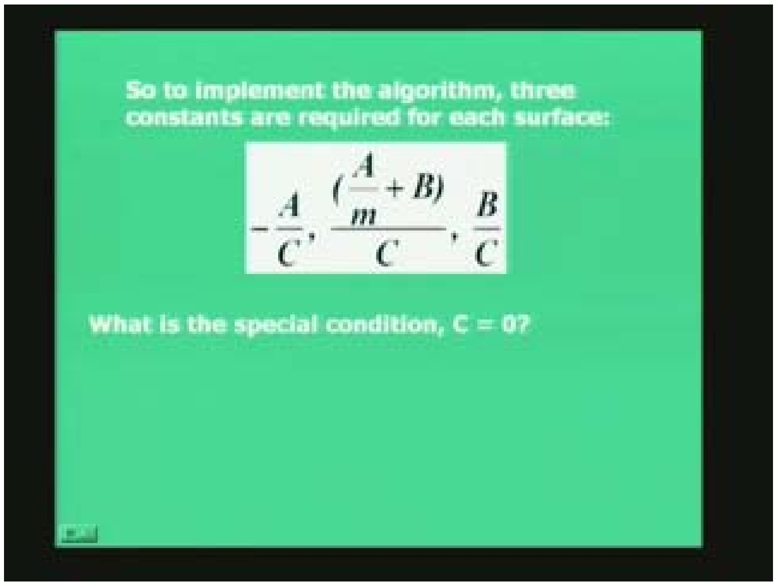
$$Z_{X+1} = Z_X - \frac{A}{C};$$

$$Z_{Y+1} = Z_Y + \frac{\frac{A}{m} + B}{C};$$

For a vertical edge: $Z' = Z + B/C$;

If you look back into the slide once more for a vertical edge you can see that Z prime will be Z plus B by C because slope m will be infinitive so this term will vanish. So you know this is a special case only for a vertical edge when the Z value must be incremented by a value B by C.

(Refer Slide Time 56:20 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?

So in fact you must remember just these three terms A by C B by C A by m plus B these are the three terms which will be necessary for you to implement the algorithm which will be necessary to compute the Z for the next point based on these three constants. Why these three constants are necessary? That is given a Z for a particular value (X, Y) for you and if you are interested to compute the depth of the next point X plus 1 or for the next scanline which is an inclined edge or a vertical edge these are the three cases for which you need to compute the Z for the next point in the scanline if a current Z is known to you. If that is the case then you need these three constants which will help you to increment Z value based on these three conditions which will help you to compute the next Z value.

We will stop with this particular slide here for the class today and continue henceforth in the next class from here on. So to implement this algorithm these three constants are required for each surface and of course keep thinking what is the special condition which will occur when C is equal to 0. Also you should able to find out these computations yourself where how do you get these three constants which are necessary for your Z_X plus 1 that is the depth for the next point which is incremented by X or for the next Y coordinate which will be Y plus 1 or Y minus 1. In fact you substitute these and get these constant out you should remember that these are the three constants which are essential for you for this computation.

We will see examples of how to do this and also for a planar as well as for the curved surfaces here. We will see how these three constants could be used very efficiently to compute the Z for all the points in the polygon itself. Thank you very much.