Computer Graphics

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

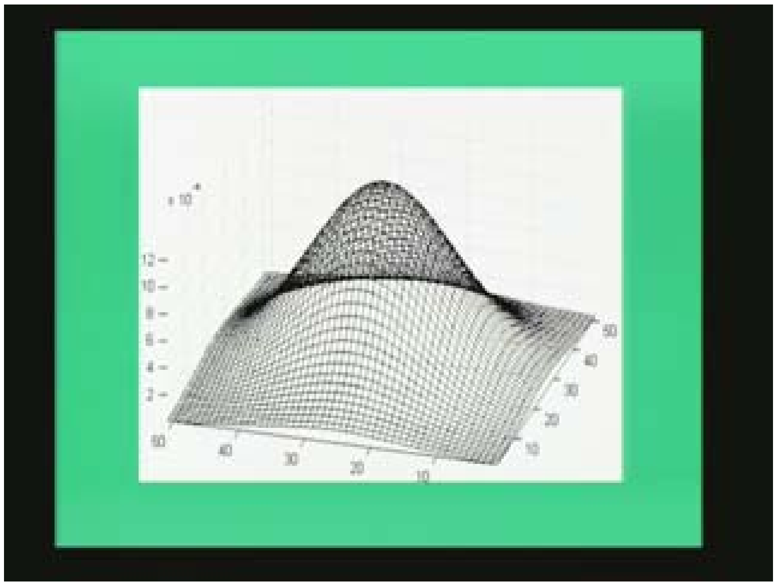
Lecture # 26

Visible Surface Detection

Hello and welcome everybody to the lectures on computer graphics. Today we are going to discuss algorithms on VSD or Visible Surface Detection. Visible Surface Detection is a very key step in the process of the 3D viewing pipeline along with rendering or illumination. Later on we will see what shading is. But if you talk about the 3D viewing pipeline, if you remember, there were four broader steps starting from normalizing transforms and then clipping and then of course mapping to view port and finally the process of rendering. So finally you have in the last stage of the 3D viewing pipeline after you have a model is solid object given to transformation provide the clipping all that before the rendering you need to find out how many of these polygons which are binding the solid are visible to the user and what are there corresponding shades. And it is necessary to do that because definitely when an object is bound by polygons or surfaces there will be a few surfaces which will be behind the viewer and some in the front.

Let us taken example of you viewing me, it is definite in the case that you are able to see my front face and the front side of my body and not on the back side. Similarly, I am able to see you on what ever is available on the front but not on the backside of the solid object, that is true so you do not need to basically render the various polygons on the surface of solid or more than one solids if they exist in the view. This is required because it will save lot of computational time for you and you will be unnecessarily rendering some surfaces which are not visible or going to be at least partially occluded or obscured by certain polygons on the front side. So let us back into the basic concept of Visible Surface Detection which deals with trying to identify which are the polygonal surfaces of the objects which are visible to the viewer or the user.

(Refer Slide Time 3:25 min)



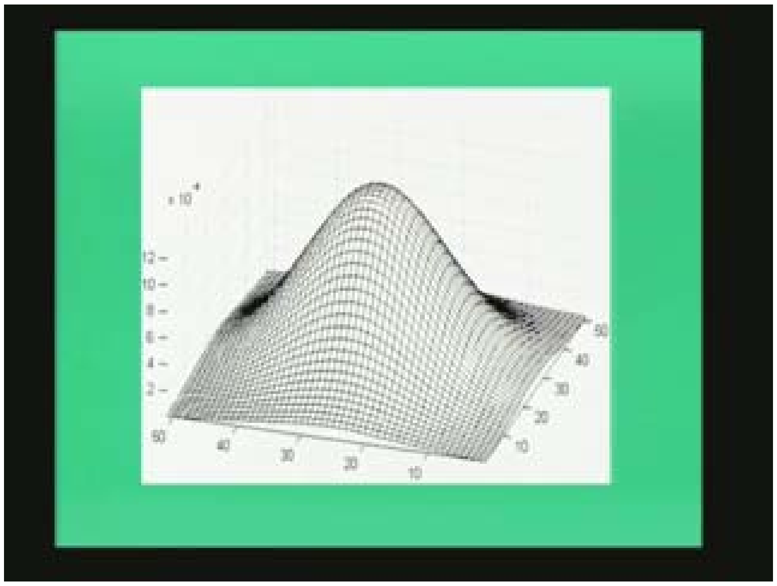
Let us take a simple example, this is a 3D wireframe diagram of a curved surface. It is basically a two dimensional Gaussian surface something like in warded exponential function which we have. So the function 1D will be exponential minus of x square. You can try to view that in 1D or 2D this is an example in 2D. In fact this function is used in many stochastic processes in even in image processing concepts where the function basically is also called as a temple bell. As you can see this structure is appearing like a temple bell and or a mountain type of a structure.

Now if you see here this surface is approximated by simple quadrilaterals generated by the vertices obtained from the wireframe diagram process which can be generated by simple set of points. You can actually compute the heights z as a function of x and y, x and y at the horizontal axis and z is the vertical axis the height of a particular surface at a center it will reach its peak and it will slowly get tapered towards the end. And if you see here it is basically a curved surface, a bell type of structures so there are some faces on the front and there are a few faces on the backside and you would like to view the surface in this form before you provide rendering or shading to the surface.

I am not going to discuss rendering right now I will not provide you a rendering of this diagram but if you see this wireframe diagram itself I would like to view this rather than the original diagram which is this. This contains the lines corresponding to the visible surface which are on the back side with respect to the viewer. So you need to eliminate those surfaces and only view the surfaces which are on the front.

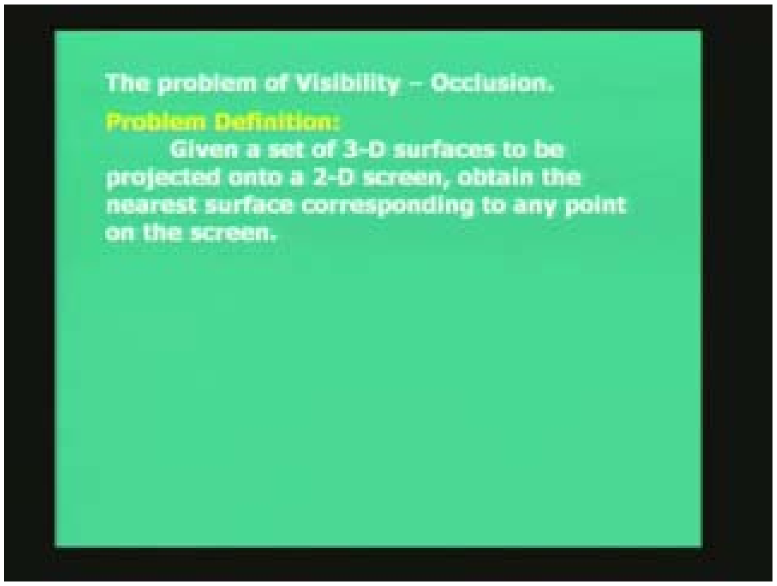
So, if you see very carefully here in this particular example it nicely illustrates that when you see on object you do not need to provide the surfaces which are on the front with respect to the viewer to the user and also on the back and that is true in reality that unless of course the object is transparent in that case the concept will be slightly different otherwise typically you will see only the front side of the particular object. So you need to display the visible surfaces, the name is Visible Surface Detection or Visible Surface Determinations. Henceforth we will call it as the VSD algorithm. There are various types of algorithms, VSD algorithms which are used and we will see them one by one. But if you look back into the diagram just to visualize that this is what we need to see and not this one (Refer Slide Time 6:51 min) because it is the same diagram as you can see, the visible surface or not eliminated here all the polygons all the quadrilaterals are drawn for this wireframe diagram here and only the visible surfaces are displayed and the back side ones are eliminated.

(Refer Slide Time 6:16 min)



So that is basically the task and this explains the problem of what you mean by trying to eliminate the surfaces which are not visible to the camera or not visible to the user or with respect to the viewing direction. You do not need to display the surfaces which are on the back side but only on the front side. So that is Visible Surface Detection for you.

(Refer Slide Time 6:48 min)


The problem of Visibility – Occlusion.

Problem Definition:

Given a set of 3-D surfaces to be projected onto a 2-D screen, obtain the nearest surface corresponding to any point on the screen.

Let us go deep into the problem. So it is basically a problem of visibility and we are trying to deal with occlusion where the frontal faces of the solid which hide or occlude the surfaces which are on the backside should be eliminated and those which are on the front side and visible should only be detected and should be displayed on this screen. So you need not display all the polygons for a particular object. You need to draw the line diagram either or even shade those polygons which are on the frontal face or on the front side of you. So the problem definition states that given a set of 3D surfaces to be projected onto a two dimensional screen obtain the nearest surface corresponding to any point on this screen.

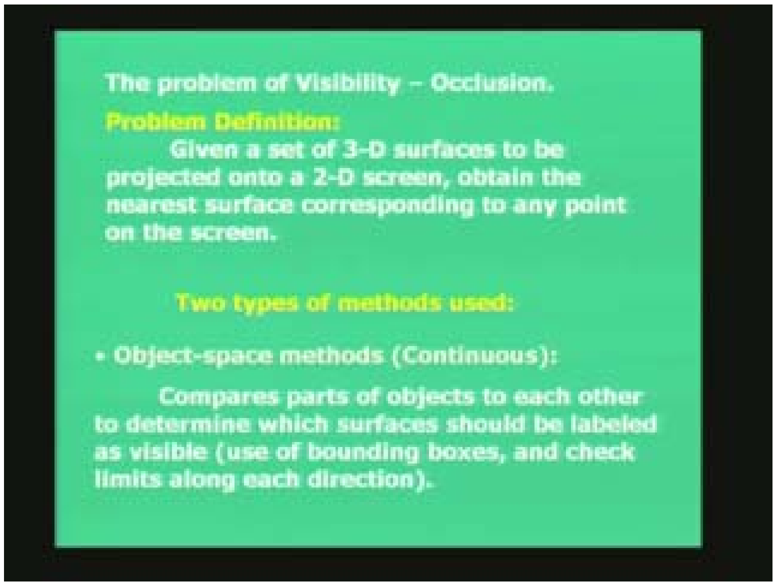
I repeat again, given a set of 3D surfaces to be projected onto a two dimensional screen obtain the nearest surface corresponding to any point on the screen. It is possible that along the viewing direction you may have more than one polygon which lies on a particular line drawn from a particular pixel or within a certain small area through which you are viewing and definitely you will be able to see the frontal face only.

Of course you will be able to see some faces which are on the backside provided they are not of course completely occluded or obscured by the frontal polygon or the polygon which is closer to you and that is true.

You can have two surface as you can see this surface is in front of you, this is on the backside and it could be totally obscured or a part of it could be obscured so you need to detect and find out that you basically do not draw the full surface which is not visible to you but you basically need to draw the frontal surface which is visible to you. So there could be two or three surfaces facing you and you need to basically or definitely draw the frontal surface and some surfaces on the backside if some part of that is also visible you need to draw that as well.

We will look into those special cases later on. So, there are two types of methods which are typically used to solve the problem, there are two broad categories. We will look into the names of the algorithms later on but they fall under two types of categories so two broad categories or methods which are used to solve the problem are one is the object space method which we will call as the continuous domain, continuous operation which compares parts of objects to each other to determine which surfaces should be leveled as visible.

(Refer Slide Time 8:53 min)



The problem of Visibility – Occlusion.

Problem Definition:

Given a set of 3-D surfaces to be projected onto a 2-D screen, obtain the nearest surface corresponding to any point on the screen.

Two types of methods used:

• Object-space methods (Continuous):

Compares parts of objects to each other to determine which surfaces should be labeled as visible (use of bounding boxes, and check limits along each direction).

That means typically you use some criteria like bounding boxes and check the limits along each of the direction x y and z and find out which part of the object should be leveled as visible and which part should be detected as occluded or obscure should not maintain.

So, that is object space method where you looked into a solid object and you know using a bounded representation it is bounded by a set of polygons. so you look at those set of polygons and find out which of these polygons or you compare these polygons of a particular object and find which of the polygons are completely in front you mark them as visible and you need to paint them or draw them completely whereas there are some polygons which are completely obscured and you no need to draw them at all.

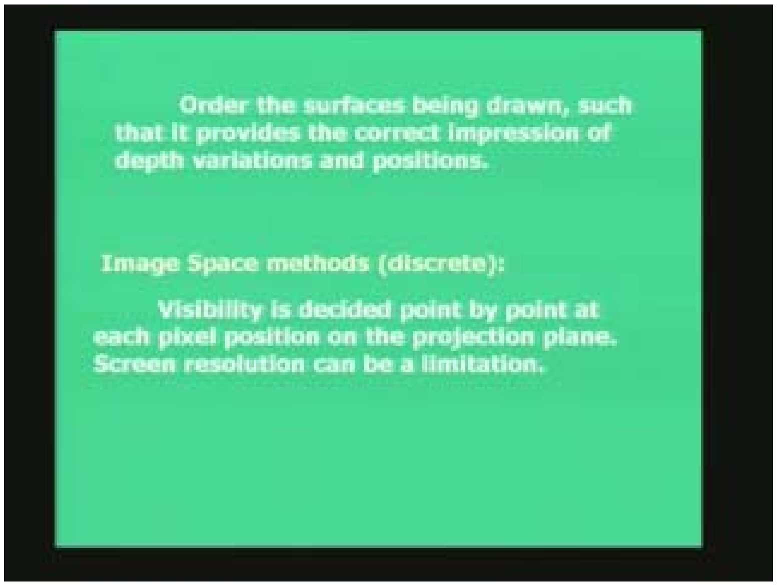
Of course there could be a few polygons which could be partly visible, that could be a partly visible and in those you need to find out which part of that is visible and paint only that part and not the whole part because you will be wasting lot of time. As you could have seen in the case of that Gaussian surface the wireframe diagram which I showed you about a couple slides back we had seen that if you do not draw the visible surfaces typically an average case you are saving about 50% of the computational time if you do not draw the surfaces which are obscured or occluded.

Typically if you take a cube and view from all different angles and take average of all different cases of the number of surfaces which are able to see you will see an average of about 50% of the polygons winding the surface around the backside approximately.

This is a very qualitative figure and so if you detect the surface which are visible that means throw out those polygons under any given condition of viewing are not visible to the viewer and you do not paint them, do not share them, do not draw them then you will be saving about 50% of compression sign. That is why Visible Surface Detection algorithms or VSD algorithms are very useful and you need to use them. So, coming back to the two broad categories one is the object space method or the continuous domain which compares parts objects to each other to determine which surfaces should be leveled as visible and typically you use the concepts of bounding boxes.

When we go into the algorithm we will see how you use bounding boxes and check the limits along certain directions along all the directions to find out which surfaces are visible and which surfaces are not. Then after doing that you order the surfaces being drawn such that it provides the correct impression of depth, variations and positions. That is what typically you need to do in a VSD type of an algorithm. Any VSD algorithm needs to order the surfaces so that it provides the correct impression of depth variations and positions of the polygons binding the solid object. The second category of VSD algorithms works in a discrete environment and hence they are called the image space methods.

(Refer Slide Time 11:55 min)



Order the surfaces being drawn, such that it provides the correct impression of depth variations and positions.

Image Space methods (discrete):

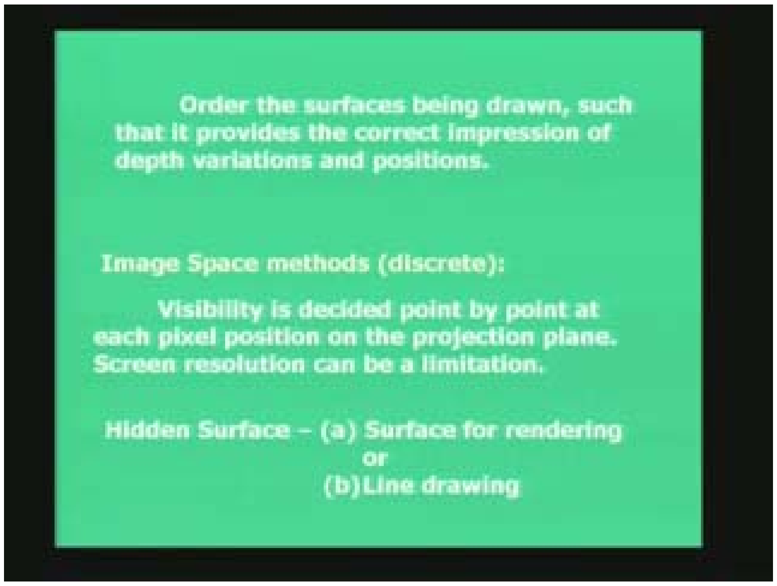
Visibility is decided point by point at each pixel position on the projection plane. Screen resolution can be a limitation.

They work in the image domain where the visibility is decided point by point at each pixel on the projection plane and this screened resolution can be a limitation in this particular case.

As you can see here of course we have to go back the object surface to find the depths no doubt about it but in the previous category of object space based methods we were comparing polygons with depths, here we work mainly at the image level where basically we almost go to each pixel and find out that at a particular point in the image which is the polygon which you are seeing right now, is it the background at this point or is it some part of the object which you are seeing, you basically find out whether that point belongs to a an object at the back or the front and depending upon that you need to occlude.

So you work on the image space method and since you are working pixel by pixel we will see later on using algorithms but you can visualize now that the resolution of the image screen the television screen or the monitor which you are using to sort now that becomes a big limitation. That means if you have a very small image resolution of 100 by 100 the quality of the picture which you will be getting will be very different from a very large image resolution of typically 640 by 480 or 1024 by 1024. So, image resolution is a big factor which plays a very important role here and that dictates the quality of the image which you will get invariant of your the model which you have chosen for representing these solid objects. Whereas the object space method the solid object modeling becomes an important criteria whereas in the image space method in the discretised domain here the image solution can be a serious limitation.

(Refer Slide Time 13:43 min)



Order the surfaces being drawn, such that it provides the correct impression of depth variations and positions.

Image Space methods (discrete):

Visibility is decided point by point at each pixel position on the projection plane. Screen resolution can be a limitation.

Hidden Surface – (a) Surface for rendering
or
(b) Line drawing

And of course the other way of looking at this is that when you are talking of hidden surfaces you are trying to find out the surfaces which should be rendered or should not be rendered and along with that it could be a line drawing. So you need to find out which of this line drawing which at the outlets of the polygons which should be either rendered or should not be rendered. So it depends upon whether you are giving a shaded picture or a wireframe diagram output so hidden surface algorithms are applicable for both. Either you are giving a picture with painting or a simple line diagram the concept of VSD algorithm holds good for both.

In both cases of course you are trying to find out which polygons are visible and which polygons are not. And again of could the third category which polygons are partly visible also could be there. But among all these VSD algorithms there could be two types again where you will say I will find out the hidden surfaces or the hidden lines. In the case of hidden lines you are giving the outlets of the polygon boundaries but not shading the polygon boundaries. These are the two broad categories one is object space image space methods another could be of course classified based on surface for rendering or line drawing. But of course the classification is mostly based on image and object space based methods. There are quite a few coherence properties which are used for Visible Surface Detection algorithms which helps you to do the computation much more faster.

Now some of these concepts of coherence can be correlated with the concept of coherence which we used in 2D scanline polygon filling algorithm. If you remember 2D scanline polygon filling algorithm some of those equations will comeback. Although the main difference you must keep in mind is when we talked about a scanline poly fill algorithm, polygon filling algorithm or poly fill algorithm or scanline drawing whatever, these are the terms which are used interchangeably, we had seen those algorithms in two dimension.

So for x_1 y_1 x_2 y_3 and so on up to x_1 y_1 there are N vertices for a polygon, it was all in 2D and you have to shade that. Now, the concept there was you need to find out if a pixel is lying within a polygon or not. And of course we used inside outside test and then form pairs fill in between and all that, you remember the key words used in scanline polygon filling algorithm. If you have not read it recently please go and read that again. Now when you are filling in between pairs if you remember when we were talking about that algorithm we did not talk about what would be the color of that algorithm.

Of course you did not even think whether that algorithm, whether that polygon the part of this scanline which will be filled is either visible or not. We will talk about this because it was entirely two dimensional and there was just one polygon and we have to fill that. But this is not the case any more. Now the polygon should be viewed as lying on the plane in three dimensions. Earlier it was on a plane which was orthogonal to the viewing direction. This was the polygonal plane this was your viewing direction. So it is absolutely all in 2D now the polygon could lie on any arbitrary three dimensional surface and you have to render it with a color which will be processed depending upon various factors like illuminations, surface normal and all that which we will be talking later on.

But first of all you need to find out whether this surface itself is visible or not. Entirely or in part or not at all visible that is number one and second even if it is visible when you need to shade it with a certain color assume it from one point to another in fact you will find during shading algorithm that the shading code also vary. So there is lot of computation at each point to compute the depth of a particular surface. So if you look at this particular polygon let us say my hand could be represented as a polygon and then if you think of a scanline filling algorithm which you will be filling some parts here each point could have a different depth if it is inclined. If it is inclined each point will have a different depth on top and different depth at the bottom and the surface normals also could vary if the surface is curved like say in the Gaussian function.

In the case of a Gaussian function the shading could also vary from the top to the bottom or from the left to this right and hence you need to compute the depth even for all points in a particular polygon. A polygon could be very large or small but it could be an accumulation of a set of points on at each point you have a different depth. You could have a different surface normal depending upon the surface you are handling and then the shading varies. But the key feature is the computation of the depth at each point and then the computation of the surface normal at each point at the key factors which load your computational time requirement or computational complexity of the algorithm. That means you need to use some property somewhere even of the polygon if it is not in 2D but also in 3D. You need to use some coherence properties extend that concept of coherence from 2D to 3D now to see if you can simplify those calculations.

You remember, we also simplified some calculations earlier when we were moving from one scanline to the other and trying to compute what should be the next exceeding intersection for the next wise scanline and things like that. You use this scanline h coherence and the scanline coherence also. So we will use similar coherence properties here extendable from 2D to 3D to see later on how we use these concepts to minimize the computations. What are the two computations, do you keep in mind? At each point x y for a polygon there is z value now unlike the 2D polygon filling algorithm that was completely in 2D now we have 3D so this x y z for each point you need to compute the z and if you do that computation and again and again for all the z points the computational burden on the system will be very fast you cannot draw your picture at a very fast rate. So you need to use some coherence properties to see how you can compute this z value very fast and also if necessary compute this surface normal at each point.

Of course you need not compute to surface common at all points of a polygon typically but in certain occasions for reality, good visual realism of pictures you need to often compute the surface normal if not at all point but at least more than one point of a polygon although that polygon could be approximating a planar it is lying on a plane but it could be approximating a curved surface or it is a representation of a planar surface.

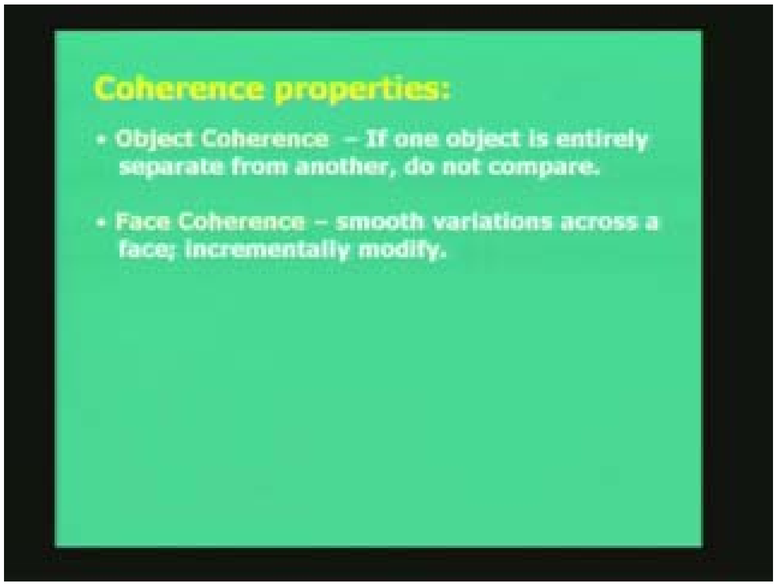
In case of a planar surface you do not have a problem but in curved surface you need to compute the surface normal at a each point as well as the z. So we will see what are the coherence properties used in VSD algorithms now which are extensions of and of course many more. We had only two scanline properties giving scanline polygon in the algorithm there were just two coherence properties here there are quite a few.

So we look at the first case of the coherence properties which is called the object coherence which says that if one object is entirely separated from another do not compare. This is a very interesting observation we are talking about trying to compare different polygons with one another to find out which one is visible and which one is obscuring the other and so on.

You need to do that but if I talk of a polygon here another polygon there and they are completely well separated out whatever be their orientation, depth and all that one could be in front the other could be at the bottom it does not matter basically.

One could be in front, the other could be at the bottom. But if they are very well separated out you do not need to compare these because they are very well separated. There is no way by one could completely obscure or occlude the other like this. So there are two polygons well separated out and hence you need to visualize that either does not matter one of them is very close to you the other one very close to you, you are far away it shows no need to compare if these polygons belong to the same object or different object they are well separated out. Please do not compare to same time so that is what it means by object coherence which tells that if one object is entirely separated from one another object in this case could be either a polygon or polygon to polygons belong to same object or definitely to two different objects.

(Refer Slide Time: 22:06)



Coherence properties:

- Object Coherence – If one object is entirely separate from another, do not compare.
- Face Coherence – smooth variations across a face; incrementally modify.

So you are talking of object coherence there. Now we talk about face coherence where we know that there are smooth variations across a face and you need to only incrementally modify. this is same like the two d polygon filling, extension of 2D to 3D where for a polygon face as you keep a minimum of a 3D structure like this it could be inclined like this and you are moving over a scanline from left to right we will see that using figures and diagrams but I am just trying to give visualization for you.

Let us say it is inclined and you are moving along a scanline. When you move on a scanline of course from left to right or move down from one scanline to the other also you will find that when you are moving across adjacent points. Since the polygons will be approximated planar objects or curved objects whatever the case may be they are small units together which bind this solid.

So the cover has a small area the next pixel will not be so far away from the previous pixel that the change in the depth and the surface normal will be only a small incremental quantity which could be computed. Remember the integer based algorithms for Bresenham's, ellipse line drawing, circle drawing or even scanline and everything the computation was done first by an integer base incremental algorithm in some sets. So we will try to see if the similar thing can be done in 3D where remember I did say that the computational burden of VSD algorithms and shading are involved in computation of depth one and the surface normal.

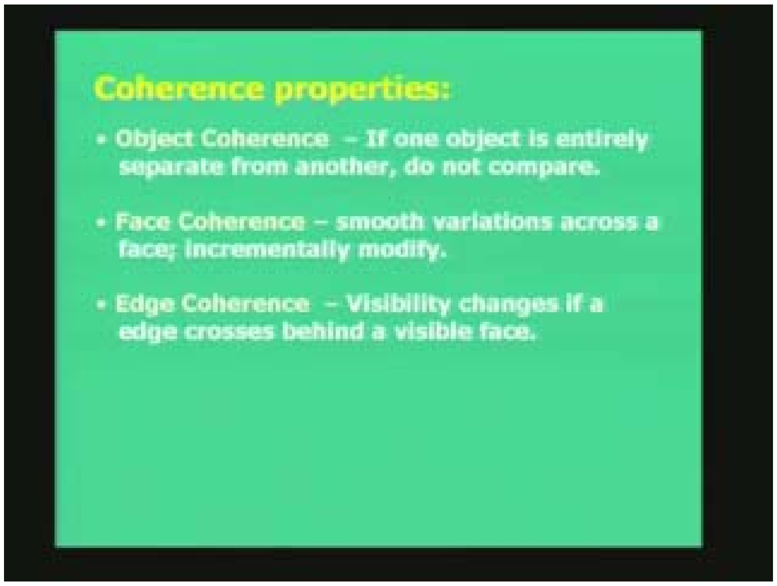
So there are two parts one is a depth another is of course the surface normal. These are the two quantities which have to be computed. so when you move from one pixel x_I y_I to the next pixel x_I plus 1, y_I with the two coordinates differing by just one pixel then the difference in the depth values of those points on the three dimensional polygon surface or the change in the surface normal if at all it comes, change in the surface normal comes only when the polygon is approximating a curved surface otherwise if it is a planar surface definitely there is no point in computing the surface normal at each point.

But however, if you still need that or definitely you need to compute to depth the depth will in general vary. If you spend time computing the depth again by the same formula which you did earlier then you will spend lot of time.

Can we device or come up with a formula which talks of an incremental algorithm which says that if you have this depth z_I at a particular point x_i y_i on this screen for a particular polygon then for the next point xi plus 1, y_i just the next point on the same scanline we get z_I plus 1 could be represented in terms of the previous depth z_i and a small change delta depending upon the surface orientation parameters we will see that later on.

So that is what is being talked about of face coherence. So smooth variations across a face and we typically assume that except when you are reaching the end of a face the face ends there so we can look at algorithms which can incrementally modify the depth values and may be if possible the surface normal itself. But typically the depth value is what is incrementally modified in VSD.

(Refer Slide Time 25:16 min)



Coherence properties:

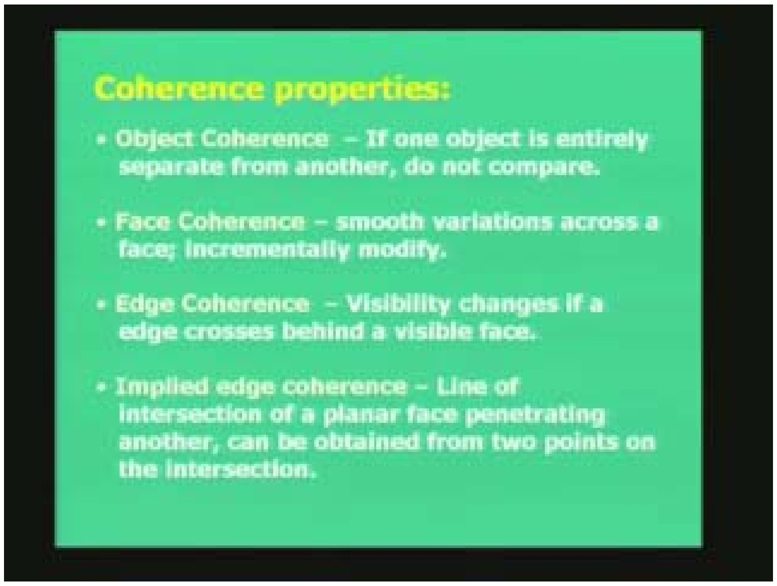
- Object Coherence – If one object is entirely separate from another, do not compare.
- Face Coherence – smooth variations across a face; incrementally modify.
- Edge Coherence – Visibility changes if a edge crosses behind a visible face.

We will look at the next coherence property based on edge coherence which we will say that of course the visibility changes if a edge crosses behind a visible face. This is nice and very interesting. I should probably pick up an object to illustrate let us say if you have an object in front and there is an object sitting behind and it is not visible to you. But if that object comes from behind and rises like this then there could be a case where the part of this planar surface of the polygon will be visible.

It may be completely hidden or it could be partly visible if it comes from behind. So visibility changes across if an edge crosses behind visible face. You have visible face in front and a partially occluded face comes from behind a visible face then of course the visibility changes otherwise from one scan line to the other of course if you travel along on edge of a polygon most parts of the edge which are behind an occluding face, occluding face is the term used for the face which is in front because this is occluding something which is at the behind and the one which is behind this one is called the occluded or obscured face because this is obscured or occluded by the occluding face.

So be careful with this terms which will I will be using so this occluding or obscuring face in the front towards you or closer to you and there is a one which is in the behind which is obscured or occluded. So this occlude or obscure face comes from behind the visible one then of course parts of those edges and other scan line become visible. You just be careful about that and that is what is the edge coherence so visibility change could occur.

(Refer Slide Time 26:57 min)


Coherence properties:

- Object Coherence – If one object is entirely separate from another, do not compare.
- Face Coherence – smooth variations across a face; incrementally modify.
- Edge Coherence – Visibility changes if a edge crosses behind a visible face.
- Implied edge coherence – Line of intersection of a planar face penetrating another, can be obtained from two points on the intersection.

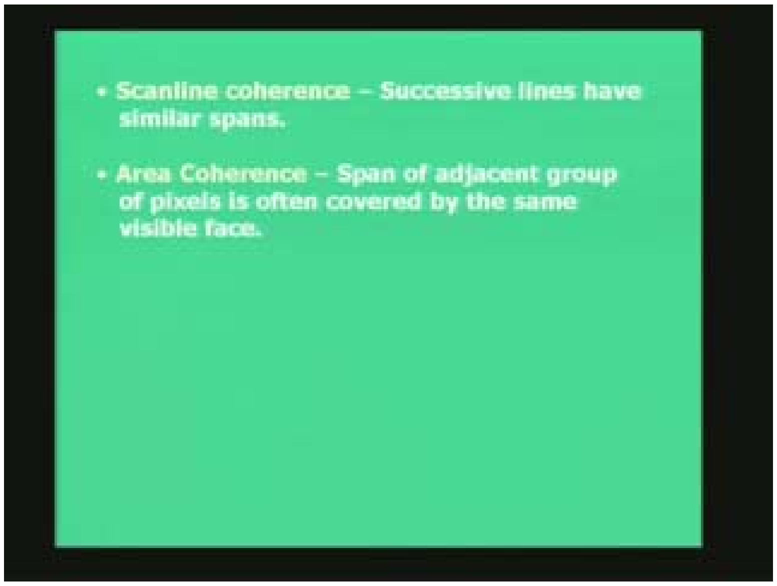
Implied edge coherence means the following, line of intersection of a planar face penetrating another. In this case it can be obtained from two points of the intersection. I repeat this, if a line of intersection of a planar surface in case of one plane penetrating another one this can be obtained from the two points of intersection. That means if there is a face here and there is another face which penetrates adapted it could happen, although we discussed about this when we talked about solid objects and visibility and trying to regularize objects and in solid objects modeling we discussed about Euler's equation constraints about polyhedrons to validator, bounding solid objects but still there could be scenarios where there could be planes getting inside and part of it is here, it is totally visible and when it gets inside a part of it is visible.

So you need to find out from these two faces which is the additional line which comes when a face penetrates another one. You can do that easily of course we need to basically find out the intersection it is basically almost clipping problem where you need to find out an additional set of vertices and lines which will be generated due to intersection. Hence, that is what you find out and that is what you use for the edge coherence. Remember the previous edge coherence we were talking about an obscured coming out here and it is getting in and partially getting occluded, so that is the difference between the edge coherence and implied edge coherence.

Scanline coherence, of course this is known to you because you have seen this earlier which says that successive lines have similar span. This is same as in 2D, 2D scanline polygon filling algorithm had the exactly same property scanline coherence typically when a part of a scanline is visible due to a polygon then of course the next scanline should almost have the similar in general have the same amount of visibility or similar spans.

We look into the next coherence property which is based on the concept of area coherence which says that span of adjacent group of pixels is often covered by the same visible face. I repeat, span of adjacent group of pixels is often covered by the same visible face. Well, as you can see here this is similar to the scanline coherence. We also talked about this in scanline polygon filling algorithm, you take a group of adjacent pixels it is typically offered covered by the same visible face unless you are at somewhere near the edge, edge of a polygon.

(Refer Slide Time 29:16 min)

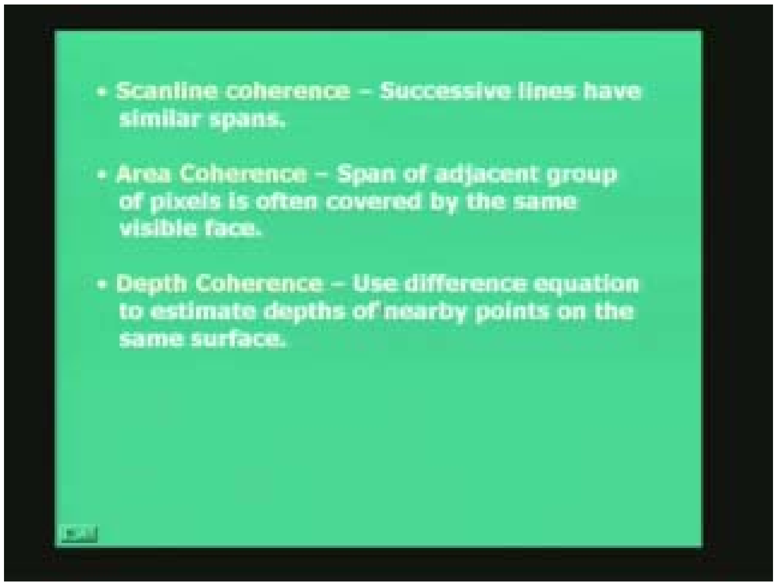


- Scanline coherence – Successive lines have similar spans.
- Area Coherence – Span of adjacent group of pixels is often covered by the same visible face.

Then of course you will probably see a background or an obscured face coming out. Otherwise, if you take on group of pixels and another adjacent one and the next scanline or the scanline prior to that then those adjacent set of group of pixels will be covered by the same visible face.

So this concept can be extrapolated from these scanline coherence properties. Then the next coherence property is based on what is called the depth coherence. This is probably the most important coherence property of Visible Surface Detection algorithms VSD algorithms used this for incremental depth calculation which says that we can use difference equations to estimate depths of nearby points on the same surface.

(Refer Slide Time 30:10 min)



- Scanline coherence – Successive lines have similar spans.
- Area Coherence – Span of adjacent group of pixels is often covered by the same visible face.
- Depth Coherence – Use difference equation to estimate depths of nearby points on the same surface.

I repeat, use difference equation to estimate depths of nearby points on the same surface. This is the key idea of coherence in VSD algorithms because polygons are in 3D and I just mentioned this a few times as to why we talk of coherence and how can we speed up the calculation. Definitely when you move from a point x_i y_i to the next point xi plus 1or move to the next scanline even.

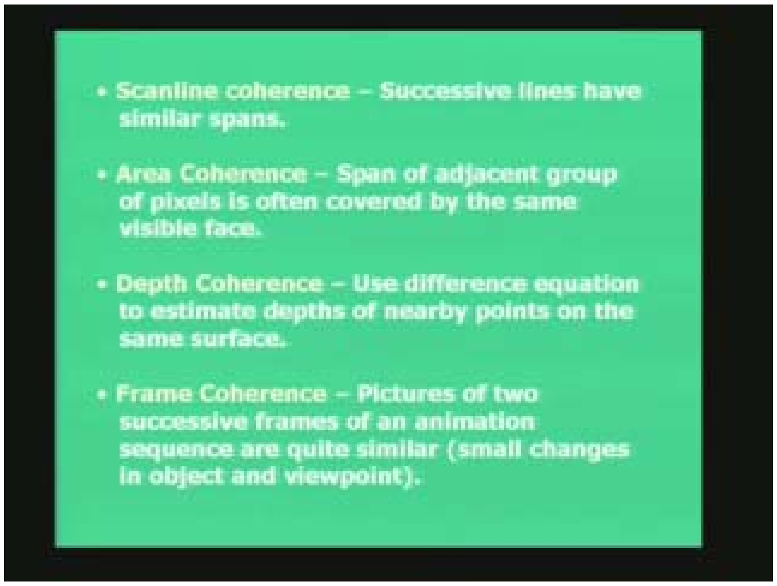
So if there are adjacent group of pixels just nearest neighbors in fact, the most nearest neighbors if you go either moving from left to right or top to bottom just the next adjacent pixel to the right or to the bottom the depth of that point typically if it is on the same surface typically it will be on the same surface except when we are talking about the where you are on an edge you can visualize that in that case things will change. But those are very few edges and very few and far between mostly you will find pixels within the polygon and then within a polygon when you move from left to right or from top to bottom adjacent near to nearest pixel you will find that the z value the depth value will not change much. It will change by a very small amount and to compute that change if you can known that change that means the previous z value is known you can apply that incremental value delta z if it can be computed then you can use that incremental value and add it or subtract it from the previous value to get the new depth of the next adjacent point and that is the key idea.

How to compute that incremental depth value delta z? We have to use the difference equations. Use difference equation to compute this delta z and use the delta z add it or subtract it to the previous z value of x_i y_i z_i and when you are moving to the next point xi plus 1 and y_i plus 1 add that delta z to the z_i to compute the next z_i plus 1, z z_i plus 1 means z of x_i plus 1 or y_i plus 1 whatever the case may be. This is the most important property which we will use in VSD algorithms all through out.

We will see how this calculation gives us a small constant value based on, it is something like Bresenham's line algorithm Bresenham's mid point circle ellipse which is at the scanline. All those were in 2D but still we use some integer based incremental values to increment and find out the next point. In that case of course you are finding out points in 2D there was no depth value incurred or involved in that process. Here our main concern is of course we need go to the next value using integer algorithm if necessary but that same concepts hold good. But to compute the depth of the next point which is essential here we use there also the integer based increment.

It cannot be an integer but we will use incremental algorithm possible. So use difference equation to estimate depth of nearby points on the same surface is what we talked about depth coherence. I read that again before leaving it, we will use difference equation to estimate depths of nearby points on the same surface.

(Refer Slide Time 33:30min)



- Scanline coherence – Successive lines have similar spans.
- Area Coherence – Span of adjacent group of pixels is often covered by the same visible face.
- Depth Coherence – Use difference equation to estimate depths of nearby points on the same surface.
- Frame Coherence – Pictures of two successive frames of an animation sequence are quite similar (small changes in object and viewpoint).

The next one most likely the last one the coherence properties is called the frame coherence. Of course we will not have an example of this in this lecture or in the next successive once may be towards the end when we talk a little bit about animation sequence we will know that pictures of two successive frames of an animation sequence are quite similar and that involves only small changes in object and view point. So we talk about frame coherence when pictures of two successive frames are displayed one after another and the amount of the time difference between two successive frames is so small that either you are changing the object or the viewer location in your 3D viewing pipeline whatever the case may be.

If you have a set of several polygons the typical object can have a few tens to few hundred polygons and you may have a few objects in the scene very complex structures if you visualize animation pictures, movies of a star world, the dinosaur pictures which may have a million polygons available at any given point of time but between two successive frames if an object either moves or the view point moves or may be both, the amount of time interval between two successive frames is very small. If you remember a movie or video rendering shot or it is being played to you it is played at about the rate of 25 frames per second. Of course there are high frame rates available these days but typically you should play it about 20 or 25 frames per second or even that is a normal video rate and so you are talking about 40 or 50 milliseconds time interval between two frames.

And if it is that small the change due to transformation which will be there in an animation sequence, the change in the position of this structure or the view point so in you are 3D viewing pipeline there will be some changes which will be happening that will be so small and negligible that most of the visible surfaces which were visible in the previous frame is also visible now. And those which were hidden in the previous frame or what we will soon come across is the term called back faces they were on the back side the face or they were hidden they will also remain hidden from one to the next immediate successive frame where the time difference is just a few milliseconds, few tens of milliseconds.

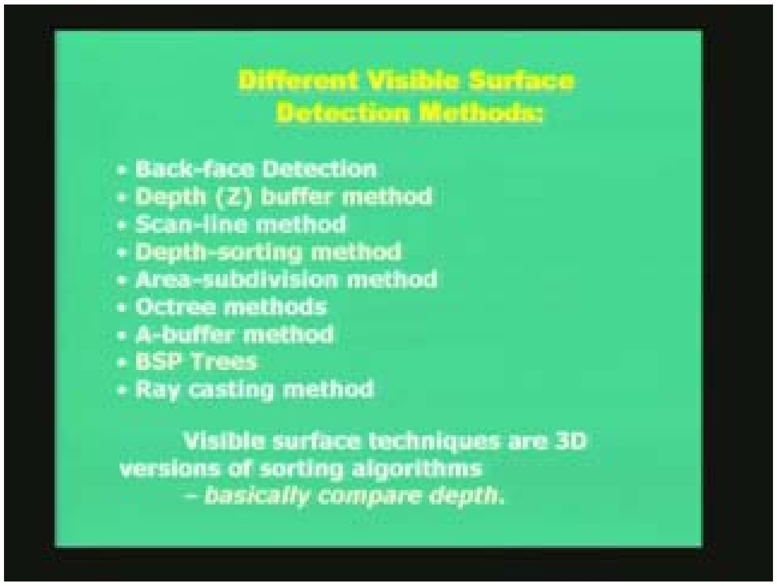
What is the change? There will be change in terms of a just a few polygons and it is very difficult to say how few it is because it depends upon the type of transformation which the objects are going through and the complexity of the object structure and all that.

But just a few polygons definitely which were visible in the previous frame might become back faces are obscured or invisible. And those which were back faces are occluded or obscured might become visible. If an animation sequence causes a little bit of change of course not even that much just a small change you can see there is a slight bit of visibility change from one frame to the another with respect to in this case even just two polygons but you have to visualize lots of polygons some of them become visible which were invisible some of them which where visible become invisible but those are very few compared to the total number of polygons.

The change which occurs that is the number of faces which were visible earlier almost all of them remain visible. Even if you talk of very high speed motion typically it will remain and major part of the picture does not change between two successive frames. and those which are invisible most of them remain obscured or invisible, a part of those, a very small fraction of visible faces become occluded or invisible or back faces as they are called and a few of them which were occluded or obscured may become visible. So that is what we use in terms of frame coherence. Since we are not going to discuss frame coherence now in VSD but when you talk of animation pictures you have to this frame coherence concepts to minimize your computation time and when you use a VSD algorithms from one frame to another you can use concepts to minimize the computation time to find out which of the faces were visible or not.

There are a whole lot of Visible Surface Detection methods and we will try to cover as many as possible in the next few lectures. And of course we will cover one or two today itself in the lecture. And the first one of them which is the most common is called the back face just talked about the back face a few minutes back. And we will see the major preliminary are the front end processing done for any VSD algorithm is called back face detection.

(Refer Slide Time 39:50 min)



Different Visible Surface Detection Methods:

- Back-face Detection
- Depth (Z) buffer method
- Scan-line method
- Depth-sorting method
- Area-subdivision method
- Octree methods
- A-buffer method
- BSP Trees
- Ray casting method

Visible surface techniques are 3D versions of sorting algorithms
– *basically compare depth.*

You just go through the names then we come across one of the most popular methods called the depth buffer method or the z buffer method or the zee buffer method. That is one of the most common limits used. Remember, back face detection is the one which we will cover next and that is the one which has to be used in all cases. Then of course we will also see in the next lecture probably the scanline method. We will also see the depth sorting method, remember, this is different from the depth buffer method.

We have a depth sorting method also. We will also see a concept of area subdivision method which is not that popularly used but the concept is very good and nice so we will look at area subdivision method. We will also see Octree based methods. Although we might just skip this method which is also not very popular since we have already studied about Octrees you can visualize how VSD could be done.

Of course another method is a buffer method and we will also study BSP trees Binary Space Partition trees, you can note it down. This expansion of BSP is Binary Space Partition trees. And finally of course, probably the most popular sophisticated method any method use is called the ray casting method. Ray tracing method or ray casting these terms are use interchangeably it is either called ray tracing or ray casting. And we know that Visible Surface Techniques or VSD algorithms are 3D versions of sorting algorithms. Remember sorting algorithms, so these are 3D versions of sorting algorithms because they compare depth.

So based on depth as a key or queue you sort polygons and that is what you do. That is what you need to do in all these algorithms. But of course you study back face which is a something like a pre processor, it is done at all algorithms to first throw out the back faces and then find out which are the visible ones and among the visible ones some of them may be partly occluded and all that which has to be found out and determined by the help of the other method. So coming back to this slide you will learn about back space detection method which is used first as a pre processor for different VSD algorithms.

We will go through most of these depending upon the time available to us. Definitely Z buffer method and ray casting method will be covered along with a few others. But Z buffer and ray casting are the most popular ones and in fact most sophisticated methods based on even sophisticated rendering or shading and illumination models which of course we will see much later on or based on a combination of Z buffer and ray tracing or ray casting so these algorithms are very popularly used.

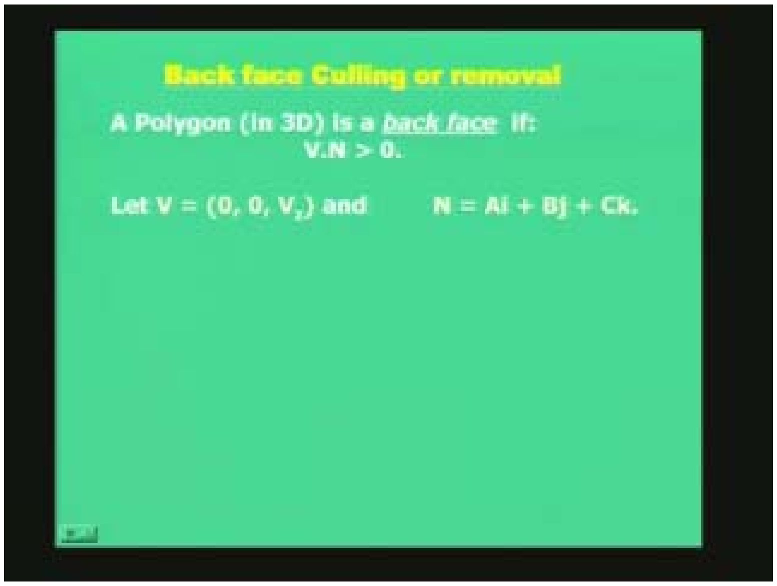
Coming back to the last point in this slide which says that basically we are looking at 3D sorting algorithms which basically compare depth and first of all among this we look at back face removal, back face detection or the most popular term called back face culling.

We look at back face culling or back face removal or back face detection methods where we say a polygon in 3D is a back face if the dot product of 2D of these vectors we have to define them, we will define them very soon but you know these vectors already. The dot product of this is positive, what are these two vectors? They are V and n.

You can almost guess right now these are vectors not scalar quantities V is not vertices which we discussed in solid modeling. V is the view vector, V is the view vector and N is the surface normal of the polygon. This we have also seen when we talked about constructive solid geometry the equation of the surface normal which will come back again. So the dot product of view vector which you have seen in the polygon and the polygon here the surface normal will be coming out in the view vector.

We will see that if the dot product of these two very interestingly is a scalar quantity the dot product of two vectors is basically a projection of one vector onto the other one and the dot product which is a scalar quantity gives some unit measure scalar quantity again if it is positive then the polygon is a back face we do not need to draw, it we do not need to shade it and we do not need to worry about it just forget it. That is what you compute the dot product and if you see if it is positive we will look at pictures back.

(Refer Slide Time 43:05 min)



Back face Culling or removal

A Polygon (in 3D) is a *back face* if:
$V.N > 0$.

Let $V = (0, 0, V_z)$ and $N = Ai + Bj + Ck$.

Let us take an example now where we will see for the case of simplification, why simplification? In fact this is done in the case of the 3D viewing pipeline where at the end basically the V vector looks along the positive and negative z direction.

So let us say that the V vector is 0 0 V_z. It is looking along the positive direction. You can take negative z direction also it does not matter much, only the expression of the back face culling will change a little bit but it could be 0 0 plus or minus V_z does not matter. We will take positive V_z in this case. You can take even 0 0 1 V_z could be one you take any vector on.

And you take the surface normal of the polygon as Ai plus Bj plus ck I j k but what are they, you need vectors along the three orthogonal axis x, y and z respectively. So the surface normal a b c at the corresponding direction cosines of this surface normal of the polygon we are testing whether it is a back face or a front face now you can visualize already that a polygon could be facing you in that case it becomes a front face or it could be a back face where you are unable to see it.

So what is the dot product of these two vectors? Take a dot product of these two vectors V dot N you know how to take a dot product of these two is you have to multiply the corresponding coefficients and in this case the dot product is so easy because there are two elements which are already 0 here so the dot product of V dot N will be V_z dot c.

In fact if you take V_z equal to one it will be only c it will be the z component of the direction cosine that is playing a very important role z component of the direction cosine of the surface normal of the polygon, polygon face N is the surface normal and the direction cosines of the surface normal is a b c and c is the z component of the N which is the z component of the surface normal n.

So that is playing a very important role. Of course you multiply that with V_z but V_z could not be visualized to be 1, you can visualize it to be any vector. So, if V_z dot c is the important part that is the scalar quantity this dot product in this case the dot talks about dot product this dot just indicates a scalar multiplication, we should not have used this sign but do not worry this does is not applicable to similar operations this is a dot product of two vectors this is a scalar multiplication because V_z and c are scalar quantity so just multiply them.

Let V_z be positive as we are talking about so assume that the view is along the positive z direction but do not worry if it is negative which his often the case we will see what should be done. And in this case you have to just check the sign of c because as I said before V_z can be taken to the unity plus 1 and if that is the case you checked the sign of c because V dot N will be V_z dot c V_z is 1 then it is a just c.

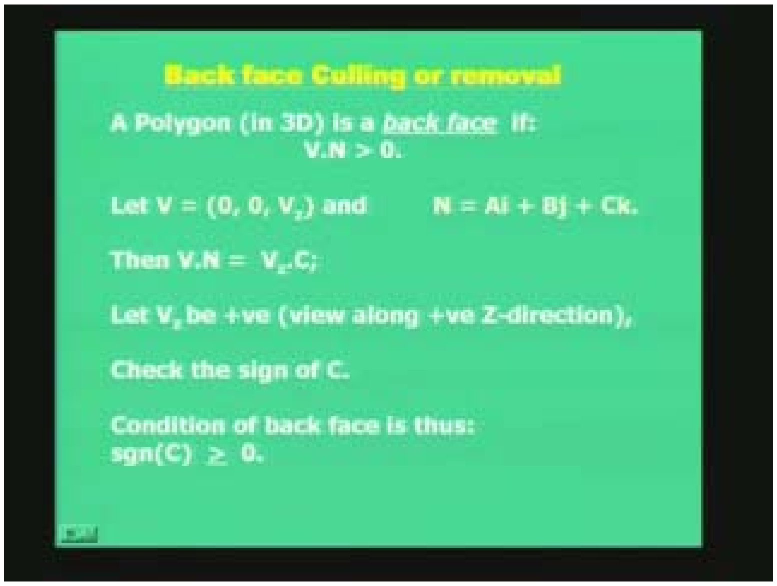
Depending upon the sign of c you will get the dot product to be positive or negative and you want the dot product to be positive for a back face. So condition for back face is just so simple now that the sign of c should be equal to or more than 0. It should be a positive quantity. So the condition of back face has become so simple. This is why you need algorithms to simplify your computations where you can handle why hundreds and thousands of polygons you basically want to handle millions of polygons in a very complex scene. And that too when you talk of animation few of those millions must be processed millions of polygons must be processed from one frame to another and all that.

But even in a static picture you want to draw it very fast you need optimal algorithms, integer based if possible, incrementally possible very simple conditions to be checked such that you can process not one or two not ten but more than few hundred or thousands up to a million polygons if possible within a certain amount of time frame to give real time and if possible on line performance.

That is what computer graphics is and these days if it is on systems both in hardware and software which takes lot of time to compute, lot of time to draw your picture because the computational time involved is very large to draw lines to render people will not accept and purchase your system, you need to have things which are efficient and fast that is what of course gets things involved in research in computer graphics to generate complex which has to bring into reality visual realism.

You need to bring in concepts of reality both from the point of structures and physics of objects but even simple pictures when there are so many polygons to be drawn, animated, rendered and things like that you need to make your computations very very simple. You can see now that the back face culling algorithm has been made very very simple.

(Refer Slide Time 47:48 min)



Back face Culling or removal

A Polygon (in 3D) is a *back face* if:
$V.N > 0$.

Let $V = (0, 0, V_z)$ and $N = Ai + Bj + Ck$.

Then $V.N = V_z.C$;

Let V_z be +ve (view along +ve Z-direction),

Check the sign of C.

Condition of back face is thus:
$sgn(C) \geq 0$.

Look back into this slide as you can see that the back face culling or removal algorithm is a condition for back face which has come down very simple. So do not worry if your direction of view vector is along the positive z direction compute the direction cosines of N and look at a sign of c.

In fact it says that for back face culling you do not need the other direction cosines at all. You just compute this c, that means you compute if possible the a b c are three different computations for you if at all to compute this direction cosines of the surface normal of the polygon face you need to compute just this z component in this case the c value just compute the z component of the direction cosines of the surface normal and when that is done you just do not even need the value you just need the sign of that direction cosine, check if it is positive or negative if it is positive label it as a back face and if it is negative it is visible, it is visible to the viewer and you should be able see it. So that is the condition of the back face.

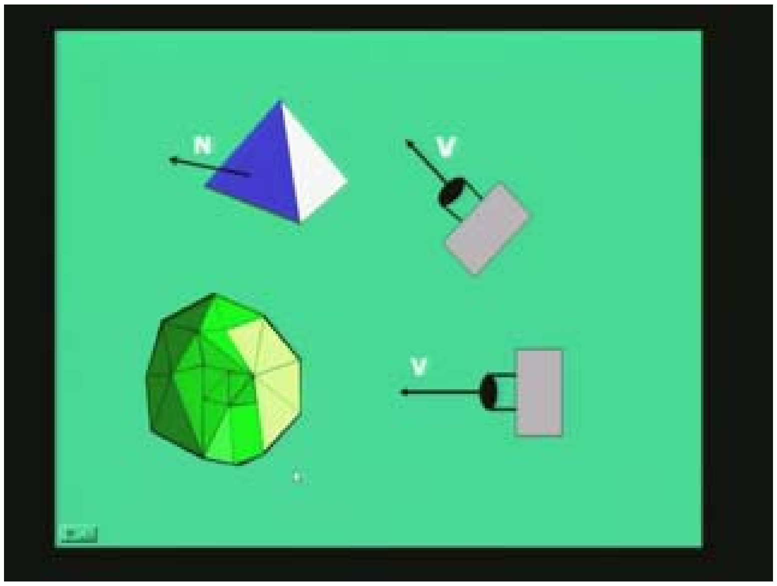
Remember that just compute z component of the direction cosines of the surface normal n, look at its sign by and at a logic, decide whether it is the back face or a front face and go ahead so that is very simple. Of course the other small condition of course when the dot product is equal to 0 then you get the c component also equal to 0 and it is a case the dot product of two vectors which are orthogonal. If you take two vectors let us one here one there and take a dot product this will be equal to 0 and that is the condition which you will have when the dot product V dot N will be equal to 0.

You know when that will happen? You can start visualizing, when will the viewing vector and the normal will be perpendicular, if this is the viewing vector and this is N and they are perpendicular to one another.

I repeat again, if this is the N and this is the V they are perpendicular it will only happen when the view vector does not intersect the polygon at all. The polygon is in such a manner that it is lying on Z_x or Z_y plane or in fact it is lying along a plane which includes the viewing vector at all and that is why the surface normal is orthogonal to the viewing vector and since you might see a line or you may not be able to see the face and that is the case when you get it. You can almost consider that is the back face.

In some cases you need to of course draw a line if possible so that is the special case which you need to handle when the dot product is equal to 0. Otherwise just look at a sign of z component and see if it is positive, if it is positive then it is the back face and if negative it is the front face or not a back face and move back so that is the key idea of back face culling algorithm. Let us look at an example, a very simple one here. That is a virtual camera with a arbitrary view vector in this of course it could be 0 0 V_z that does not matter we are just looking at a scenario where we are looking at a pyramid with a triangular base.

(Refer Slide Time 52:54)



I am able to see two surfaces for the time being assume that when you are seeing these two faces and these triangles are the polygon faces for this structure which is a pyramid one is the blue with the surface normal N shown here and the other is the white triangle which is facing towards the camera.

As you can see that the blue triangle is a back face because the N is along the v. That means if you check a dot product of this N and V you will get the dot product which is positive. Or you can basically check the sign of the direction cosines of the N provided V_z is looking along V 0 0 V_z and if that is positive in this case it will be so that is why it is a dark face and I have labeled it in blue color to show to illustrate that it is a back face in fact you do not have to show it or render it for the camera.

In your case you are seeing it but it is the back face with respect to the camera. The other triangle which is almost rendered or shaded with white color is visible because the dot product of that surface normal and V will be negative it will not be positive. So you can visualize that easily that this blue triangle is a back face and the white triangle is not a back face it is a frontal face.

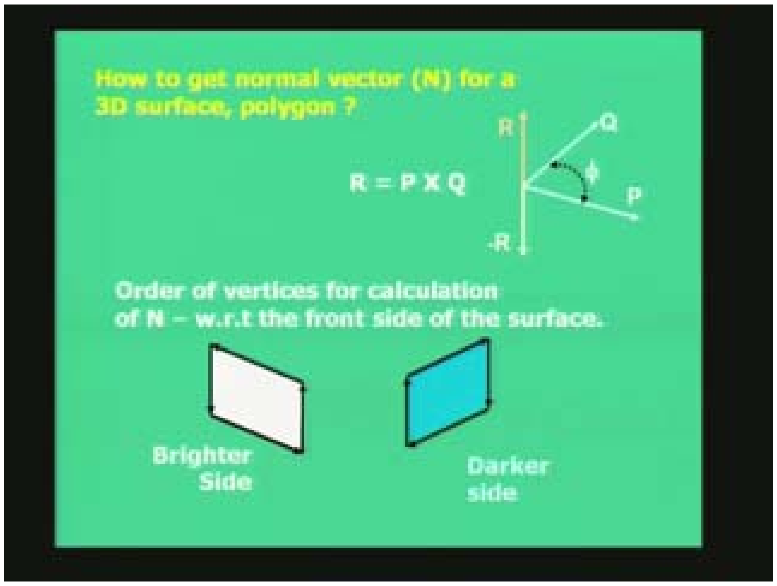
Based on this you compute not only back faces or front faces this is an example where the object has arbitrary triangles it can visualize these two an arbitrarily rocky structure bounded with polygons which are typically triangles here, a very arbitrary crooked structure you can visualize this to be an asteroid. And as you can see here the view vector now looking along 0 0 V_z you will be seeing surfaces which are closer or definitely frontal as you say are not back faces or more brightened in color because they will be appearing much more brighter and the back faces I have put a very darkish color to show that those parts of the object and those polygon faces will not be visible. That is what you have with respect to the frontal and back face of this.

Of course we will talk of shading and rendering later on to see how to shade the frontal faces which are close to the viewer and with brighter color and those at the back. The last part of today lecture we look at a computation concept of the surface normal and for a 3D surface polygon. Now, we know that if you take any two vectors p and q lying on a plane this plane could be the polygon in this case. And we take a cross product p into q this cross indicates the cross product here and p and q are two vectors lying on a plane which could be two edges of the polygon itself and you can see that if you take the cross product of p into q you will get a vector r which is pointing upwards.

Now, if you remember the formula for the cross product it will be p q sin of phi. So sin of phi will dictate the amplitude of r that is true. So if the phi is 0 you get a small r or if phi is very large you get a very larger r.

But depending upon the order you choose p into q if you take the cross product of q into p then you will have a negative of r. Why do you talk of positive and negative r because you will have the term direction cosines but the signs will change but this is very important because the order of vertices which you choose for the calculation of N assuming that you choose vertices to get edges and that will dictate whether you are looking into the frontal side of a surface or the back side of a surface.

(Refer Slide Time 54:34 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.

Just to give an example if you look into this particular scenario then I will say the same surfaces on the brighter side and if you looking into the back side if I turn the same object you have a darker side of the object. Let us say if you look at my hand you will say that I have a brighter side on the back side which is the darker side and if I turn it so there will be only frontal part which will be visible to you, the back side will never be visible to you because it is inside an object and a polygon remember for the 3D solid object does not hang in space it bounds the solid.

You will have a front side which is visible back side will never be visible do not confuse this with the back face as back side will never be visible. We need to find out the surface normal coming out of the frontal face not from the back face. So you need to find out the computational method by which you compute the actual line of the polygon not the back face.

So we stop here how to come up with computations involved to calculate the actual N not by taking just to arbitrary vectors p and q and taking cross product which could give you a risk of having the N coming out of this surface or from the back side surface which is never visible to you. We will see how to come up with method based on the figure which you have seen in the last slide I got a method by which you need to compute the N from the frontal side and not the back side and which should be used to compute the direction cosines of a surface normal not taking any two arbitrary vectors lying on the plane and taking a cross product it is very risky do that.

So we stop with it and will continue in the next class from this state onwards where we looked into methods to compute the direction cosines of a surface normal N of a polygon given N vertices. Thank you very much.