

Computer Graphics
Prof. Sukhendu Das
Dept. of Computer Science and Engineering
Indian Institute of Technology, Madras
Lecture - 32
Visible Surface Detection (Contd...)

Welcome to the lectures on Computer Graphics. In the last few classes we have been discussing various Visible Surface Detection algorithms or VSD algorithms and the first preprocessing stage of any of these VSD algorithms is the concept of back face culling where we removed all the back faces from the scene. We are trying to render a set of polygons and we remove all the back faces or the hidden faces and then of course the Visible Surface Detection will actually show you the surfaces which are facing the viewer partly or in full. And among the various VSD algorithms we discussed we started with the concept of the depth buffer or Z buffer algorithm first and then we moved over to the concept of scanline algorithm which was an extension of the two dimensional polygon filling algorithm now in 3D. And then of course we discussed the Painter's algorithm or the depth sorting algorithm. After that we talked about area subdivision method, the concept of BSP trees this was covered in the last class. And at the end of the last class I gave an insight about the concept of Ray tracing which we will discuss at length in the class today.

Now, if you think of all these different VSD algorithms starting from the Z buffer across the Painter's algorithms to the BSP trees you will notice that in none of these methods there is a talk about light rays which is the main reason for image formation either when you see an image through your eye or you capture a picture using a digital camera or video using a camcorder the concept of image formation from the basic theory of optics and physics says that you have a light source let us say when you are in an outdoor environment during day time the principle source of light energy is the sun and then of course you are in an indoor environment in a class room or in a building the light source are the artificial lamp then condense lamps and the tube lights and various other types of light sources and they help us to view not only the human beings, the animals, the kingdom, birds as well as if you visualize the artificial sensors the man made sensors like the digital camcorder or digital cameras which are used to capture the picture. The concept there is the light rays are emitted by the source they are emitted in some cases in all directions sometimes in a certain direction may be and they fall on the object surface.

The object surface reflects a part or the whole of those light rays towards the sensor or towards the viewer or towards the projection plane in this concept of Computer Graphics. Of course how much of it is reflected in what direction we will take of those models in rendering but assume that the light rays are emitted in full or in part towards the projection plane and those light rays go and hit the pixels. The image set of pixels in the case of digital camcorder or in the case of the human eye will hit the rods and the cones in the retina and that helps us to visualize the image. That is the theory of the image formation and the light rays are the principle cause or the reason why we are able to see why an image is formed. But in all of these VSD algorithms there has been no talk of

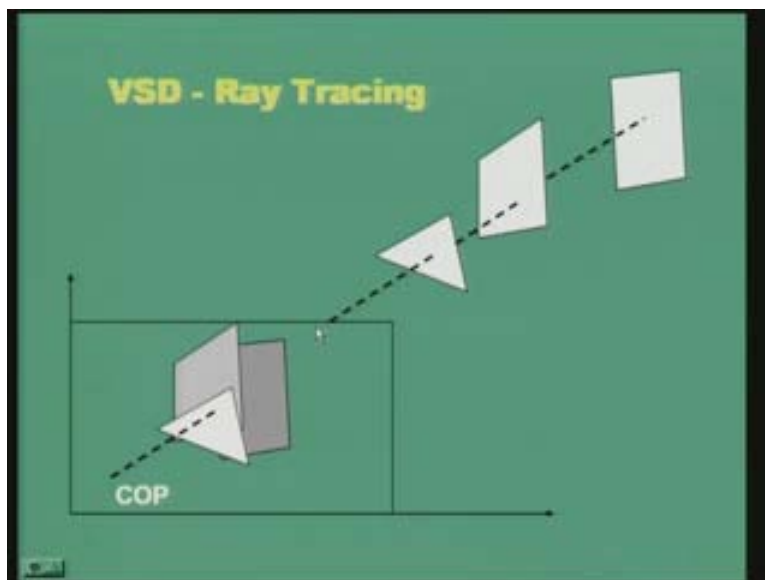
these real phenomena because in Computer Graphics we know that we are creating reality.

It is a concept of trying to create virtual reality, visual realism in the picture and no where the exact concept of this image formation process in terms of optics theory, light rays, coming out of the light source, hitting the objects, then getting reflected back into the projection plane or the sensor is not being talked about. This is the main or almost the only concept which is discussed in ray casting and which tries to model the basic theory of image formation.

Let us look back in to the picture which we saw in the last class Visible Surface Detection algorithm of Ray tracing where we see, of course I must remind you here that we use this same figure when we talked about concepts of Z buffer algorithm and there we were talking of this direction of projection vector or the ray which is moving towards the center of projection from the points on the object surface and you are basically comparing depth along that particular direction here.

These dashed lines indicate a ray. I repeat, the dash line indicates a light ray which could be reflected from object surfaces towards the projection plane. So this square rectangle is a projection plane which is used to view the scene. And you will see that along a particular direction which is the direction of light ray the ray could be deflected by various objects but the object which is closet to you closet to the viewer will actually be reflecting the light towards the projection plane the other light rays reflected along the same direction at the same position from the other objects will be not be seen because they are hidden or occluded.

(Refer Slide Time 07:01)



I repeat again, the closest object surface so the depth comparison is again a key but the concept here is about the light rays and the closest surface corresponding to any point on

the projection plane you will be able to find out which is the closest object and the light ray reflected from that point will be responsible to create that particular scene of the object.

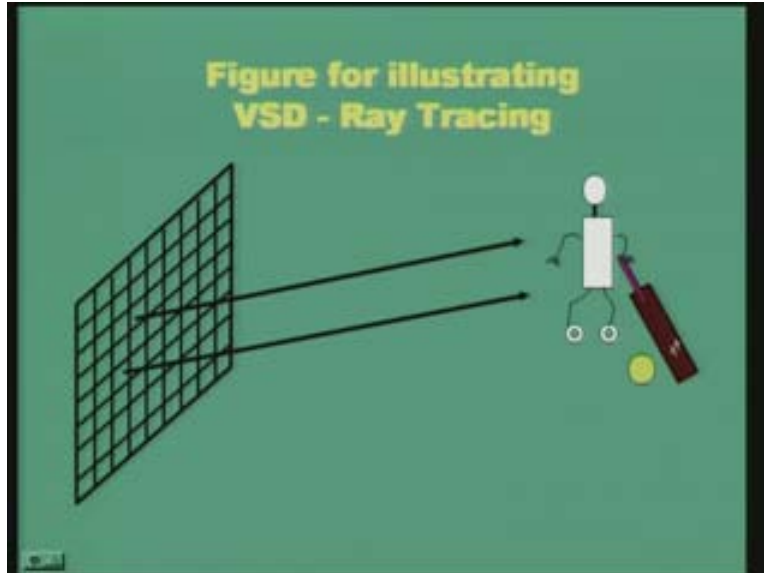
Remember, at any particular point (X, Y) on the projection plane you are going to see only one object you can not see more than one object and that object is closest to the viewer in terms of a set of other polygons which will occupy the same position (X, Y) . Remember, we are talking of the canonical view volume orthographic projection model where from x, y, z set of points on the surface of polygons in 3D you need to just chop of the value of Z and you get the corresponding point (X, Y) which are the projections of a point in 3D to 2D. That is the advantage of using orthographic projection model here. We are talking about the last stage of 3D viewing pipeline.

I kept reminding you of this concept several times in the sequence of lectures when we discussed VSD algorithms and here from $(X, Y) Z$ simply we can get (X, Y) . But of all those polygons or all those planar surfaces or even curved surfaces which occupy that point (X, Y) and various values of Z the one which is closest to you will give rise to the intensity for that point (X, Y) . So what we do now is, since light rays are coming out of the source getting reflected by the object surface and coming towards the viewer in the concept of the VSD Ray tracing we just reverse the direction of the light source or the light ray.

We traverse in the reverse direction that means given a point (X, Y) on the projection plane we actually traverse in the direction of the opposite to that of the light source. that means from the point (X, Y) we move towards the object and we hit a particular object which is closest to the viewer and from that point we try to find out what are the different sources of light which are visible to that object point and based on that we can shade the particular point (X, Y) .

We will talk of though shading and rendering models later on after we finish VSD algorithm but the concept of Ray tracing is just the reverse phenomena of what is happening in the process of image formation. That is why it is called Ray tracing or ray casting and remember we are just reversing the direction of the light, remember light travels in a linear part. Let us visualize this particular imaginary figure where on the left hand side you have a matrix of N into N pixels and I have just shown two light sources in fact the reverse direction of the actual light ray, we are just going in the reverse direction so we ray cast from a particular point (X, Y) towards the object and so we have two points $x_1 y_1 x_2 y_2$ here and two rays will be coming out of that projection plane in a normal direction normal to the projection plane and it will hit certain points on the object surfaces and based on the color of that object surface based on the light source you will be able to render those object surfaces.

(Refer Slide Time 09:56)



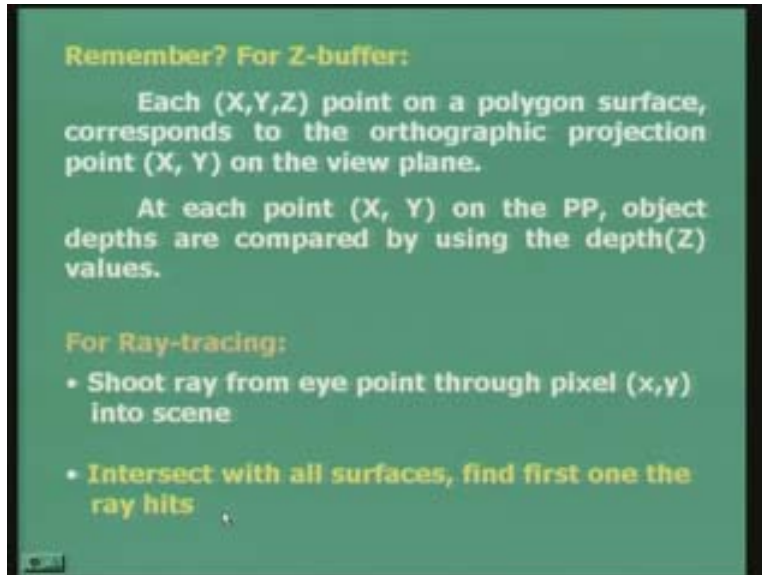
This is what we do where I repeat again in a actual real scenario light rays will be coming out of the surface hitting the object and moving towards the viewer in this case it just the reverse phenomena, ray casting just tries to model the image formation process by reversing the direction of the light. So as if you are not sending the light back but you are casting a ray back in an imaginary ray which is traveling in the direction opposite to what the light would have traveled to create that same image. So you are trying to simulate reality. The field of Computer Graphics tries to create reality and that is why the other term which is used is virtual reality.

We are trying to create a picture as real as possible and give effects of visual realism, ray casting is one of the very most popular and commonly used methods where it tries to simulate reality and gives visual realism. There are concepts of visual realism based on anti aliasing. We know that particular term concepts of radio city which is used on top of Ray tracing. And of course some of the most sophisticated methods might use a combination of Ray tracing and Z buffer methods to really create a very complex pictures which appear very very realistic or real. So come back to Ray tracing, remember for z buffer each (X, Y, Z) point on a polygon surface corresponds to orthographic projection point (X, Y) on the projection plane or view plane.

We already discussed this because any point in 3D (X, Y, Z) you need to just chop of this value of Z and the first two terms will give you the two dimensional projection in case of orthographic two dimensional projection of a point in 3D. So at each point (X, Y) on the projection plane or view plane these terms are used interchangeably PP or VP you can talk about. Object depths are compared by using the depth values. I repeat again; at each point (X, Y) on the projection plane object depths are compared by using the depth or Z values. So, for Ray tracing we basically shoot a ray from the eye point through a pixel (x, y) into the scene.

We just saw that in the picture in the previous slide and then what we do, we intersect this with all surfaces find first one the ray hits. Intersect this ray with all the surfaces and find the first one which the ray hits to find out which is the object we are seeing at that particular point (X, Y).

(Refer Slide Time 12:22)



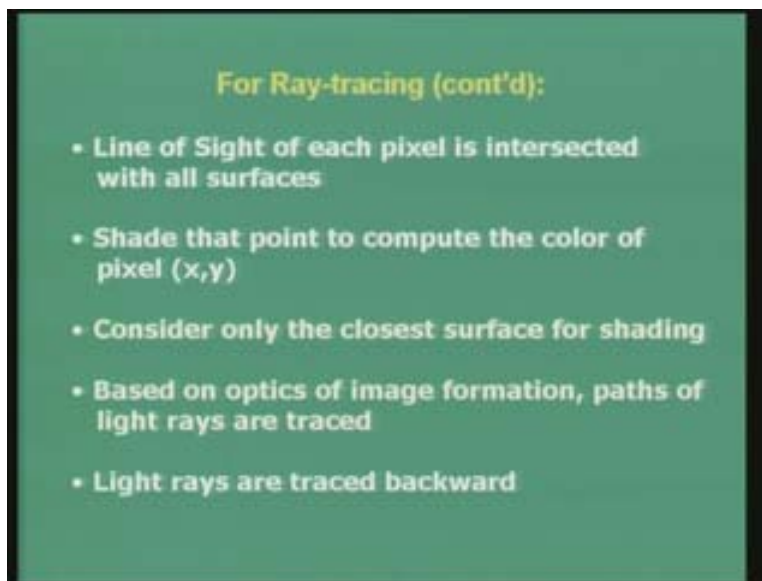
Remember for each point (X, Y) on the scene you have to do this, it is computationally a very expensive method or very costly but as you know that when you want have reality complex pictures you have to pay the price of computation. If you want to do it very fast the other better methods which you can use but you will have a very coarse picture not a very fine and nice picture not looking very realistic it will appear as a virtual in like any video games or a cartoon type of a picture we can easily simulate but you want to simulate reality you have to pay the price of the computation so that is what we should remember, the computational complexity basically order n^2 where n^2 is the size of the image pixels.

If you have an image array of n into n you have to shoot out that many rays from each particular point (X, Y) and for each ray what you do? You find out what are these objects which this ray will go and hit because it will not hit all the objects because all set of n polygons will not intersect this particular ray actually so it will hit a subset of this. It may not hit any object for that matter it could just hit one it could hit two or in fact many more. And basically if you hit one or more such objects then you need to find out which one of them is closest to the viewer because that will be responsible for shading that particular point (X, Y) or pixel (x, y) on the screen. So we keep looking at the other points for Ray tracing where the reverse direction of this ray which is cast. We talk of that as a line of sight of each pixel and that is intersected with all the surfaces. You need to shade that point to compute the color of the pixel (x, y).

We will talk of the shading models in the next class and consider only the closest surface for shading that is what you do before shading in fact. You need to find out the closest surface for shading and based on optics of image formation paths of light rays are traced. I repeat again, based on optics of image formation paths of the light rays are traced. This is a scenario which you may have and there could be multiple light sources illuminating a particular object two or more times. So when you hit a particular object surface and find out which is the closest one after you have traced a ray back from that object surface you hit to find out how many light sources are visible.

It could be a case that some light sources are visible from that object surface some of the others could be occluded by some other object surface. It is possible that you have a light source here but another object surface is actually hiding the particular object surface which is used to compute the shade that is the surface which is the ray as gone and hit at the current point of computation at the current stage of iteration. And of course there are could be other light sources on other directions which are visible so you need to compute the shading or the color of that object surface based on those light sources to render that particular point (X, Y). So find out come back to this based on optics of image formation paths of light rays are traced and then you consider only the closest surface for shading and then shade that point to compute the color of the pixel (x, y). Light rays are traced backward. Traced backward means the direction opposite to the natural flow as mentioned in Computer Graphics. In the case of Ray tracing we are tracing this in the reverse direction.

(Refer Slide Time 00:15:52)



It is also suitable for very complex curved surfaces, we have not analyzed much of complex curved surfaces in the previous VSD algorithms and we will see how Ray casting is very suitable for complex curved surfaces. We will take a couple of examples for these at least one. It is of course computational expensive we already talk about this

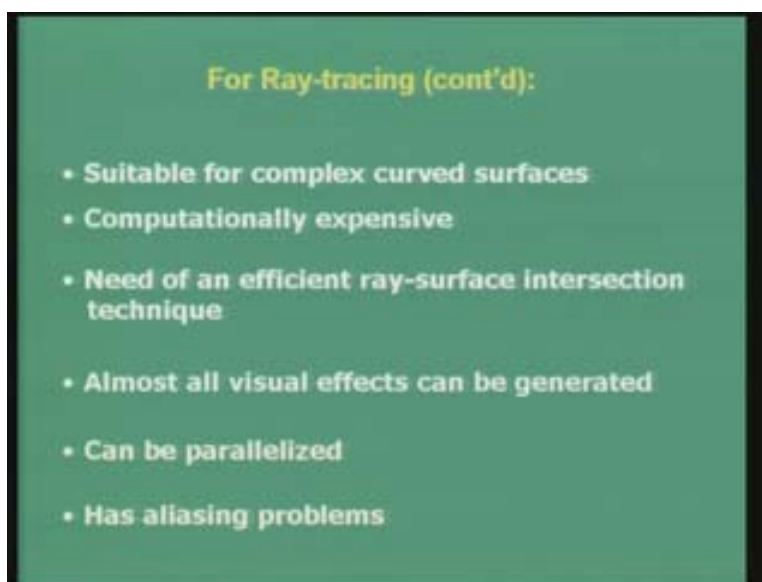
and we also need an efficient ray surface intersection technique to speed up the computation we will see how it is done.

Almost all visible effects can be generating. We had seen that many sophisticated algorithms are based on Ray tracing although it is computationally very expensive but to generate visual effects people to take the help of Ray tracing based methods. And of course if you have a parallel environment in a super computer or a grid accompanying environment we have lot of processors working in parallel or a parallel computing environment lose your **practicable**, it is possible that you can paralyze this environment why because computation of one ray does not effect the other one at does not depend on the computation of the shading of another pixel in this particular case, such a model is not used.

Of course if you use such models you have to take care but typically in case of modeling one ray does not affect the other one. So you can pickup a set of rays for a small sub area the sub window of the image to be generated and give it to a particular processor and the other partition you can give it to a second and third and so on. So you can partition the image into several blocks and if you have that many processors to be used for computation you can share the load by giving each such block to each particular processor and, parallely, you can compute the shading for each pixel.

The single processor is not responsible for casting all the rays and what you do is simultaneously compute for each block. So you can visualize that you can use parallelism to speed up if you have the feasibility. And then of course it has aliasing problems which have to be sorted out. We will talk about these aliasing problems and anti aliasing methods even for Bresenham's line algorithm it could be there. We will discuss this when we talk about visual realism at the end of this course.

(Refer Slide Time 18:11)



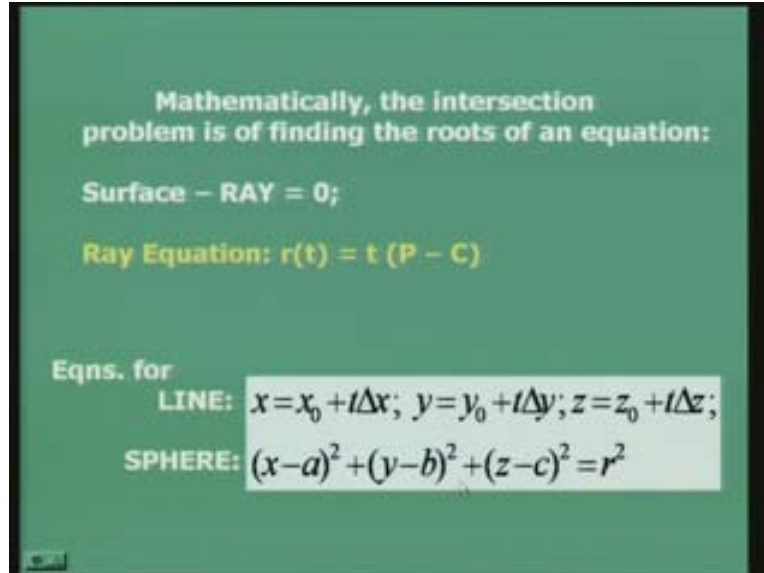
So coming back to mathematics again which is responsible for trying to find out the intersection of a surface or a polygon with ray basically it means that we need to find out the roots of an equation which is basically caused by the intersection of a surface and ray. So we take the equation of a surface and the equation of a ray and then we look at a solution for this particular equation surface minus ray is equal to 0. And typically ray equation can be very simply obtained in a parametric form because it is just line in 3D and it is basically given by T into P and C.

What are P and C? You can visualize the C to be the center of projection from which the rays are coming out and P is the particular point (X, Y) on the image plane or the projection plane. So you join these two points and cast the ray back that means you keep on looking for increasing values of T which will help the ray come out of the projection plane and enter into the scene and go towards the object for larger values of T and that is how you um obtain the equation for a ray. You know equation for a line can be written in this form where you obtain the values of x_0 y_0 z_0 is one particular point through which the ray is coming out and Δx Δy Δz are the other parameters of the line constants for the line which have to be computed from two points. x_0 y_0 z_0 being one point and if the other point is x_1 y_1 and z_1 then you know how to compute Δx Δy Δz we have done these several times both in 2D and 3D Δx Δy Δz will be just the difference of the (X, Y, Z) coordinates of the two points which are joined to form the line. And this 2D or 3D in this case we have 3D because we have all the (X, Y and Z) coordinates for the line.

And this is an equation of a sphere where a b c are the center of this sphere and r is the radius so that is the equation for a sphere. So now what we are interested is to basically render a sphere. And in this case we are interested that we take any ray out of pixel (x, y) and we are trying to find the intersection of a sphere with a line. I talked about modeling curved surface in this case we will see how easy it is. Just visualize that you have a curved surface and you have a ray intersecting that particular curved surface.

We have seen that earlier in cases as well you talked about regularized Boolean set of expressions and of course we took a projection of a sphere in 2D and we got a circle on a square which is the projection of a cue when you talked about union intersections of those structures. So now visualize a ray coming out of an image plane, if you have an image plane the ray coming out and then you have a spherical surface, an intersection of this spherical surface with the rays is what you are computing. This is the equation of the ray, the equation of a line, is the equation of the ray and this is the equation of a sphere. One way is to handle this problem is we just take the values from the top equation which is given by x equals something y equals and z equals and substitute in the equation of the sphere.

(Refer Slide Time: 21:40)



Please try it out, try to substitute the equation of the line into the equation of the sphere. Please try that out and work it out yourself and what you will get when you substitute the first equation and to the second equation, you please try it out and you will get quadratic terms in the expressions. You will get the quadratic terms, please try that out yourself. So when you substitute that yourself you will get the quadratic terms of (X, Y, Z). Try to accumulate those quadratic terms and form a nice expression. If you go into the next slide this is what you will get after substituting the expression of the equation of the line on to the equation of the spherical surface. And then when you rearrange the terms, that is collect the terms and try to write a quadratic in terms of the parameter t.

Why are you interested in the parameter t? It is because that will help you to get the roots of that equation surface minus ray is equal to 0. You are actually trying to find out the value of t for the ray which will go and hit because that will give you the value of (X, Y, Z) of the intersection point between any surface planar or in this case a spherical surface **with its rays**. So assume a spherical surface here now a round strap of a structure and with this spherical surface you are hitting the ray from the outside. It is a spherical surface and you are hitting a ray and the intersection point is what you are calculating. So that can be calculated by obtaining the value of t.

As you can see here please work it out yourself and then check with these expressions whether you are getting this particular term, you are trying to collect the nonlinear coefficients of t square then you collect the coefficients of t and the constant values because x_0 y_0 z_0 (a, b, c and r) are all parameters of the line and the ray which are all known to you we have to just compute the value of t. This is what you will get as an expression for the value of t.

(Refer Slide Time: 23:20)

Substitution gives us:

$$\begin{aligned} &(x_0 + t\Delta x)^2 - 2a(x_0 + t\Delta x) + a^2 \\ &+ (y_0 + t\Delta y)^2 - 2b(y_0 + t\Delta y) + b^2 \\ &+ (z_0 + t\Delta z)^2 - 2c(z_0 + t\Delta z) + c^2 = r^2 \end{aligned}$$

Collecting terms gives us:

$$\begin{aligned} &(\Delta x^2 + \Delta y^2 + \Delta z^2)t^2 + \\ &2t[\Delta x(x_0 - a) + \Delta y(y_0 - b) + \Delta z(z_0 - c)] \\ &+ (x_0 - a)^2 + (y_0 - b)^2 + (z_0 - c)^2 - r^2 = 0 \end{aligned}$$

So I hope you can derive this equation very easily because so far we have done lots of mathematics throughout this course and it should not be any big problem for you. Now, quadratic expression of t and you know any quadratic equation will have two solutions. How many solutions will it have? It will have two solutions. And out of these two solutions both could be real or both could be imaginary. You can have also case of one solution also. In the case when both the solutions are same you can have just one solution. So there are three different cases that arrive for a solution of T , I repeat again; both real roots of this value of T , one real solution and two complex numbers two imaginary solutions.

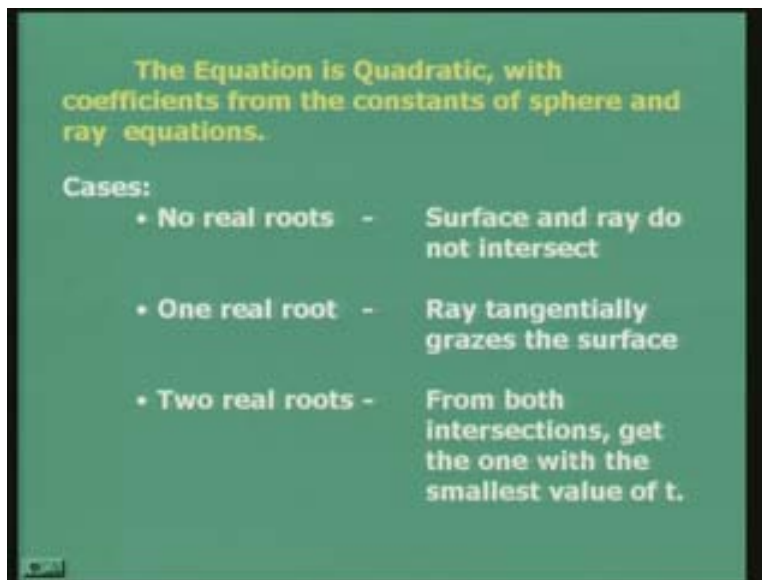
Let us look at these three situations here when you look into the slide the equation is quadratic with coefficients from the constants of sphere and the equations we have seen that in the previous slide. This was the equation as you can see the coefficients of t square t and the other third term are all constants from this sphere and the ray. So you can compute all those and then compute the solution for T and the three cases are, first is no real roots, no real roots means the discriminant of that particular equation for the root of T will be an imaginary quantity, what is the physical significance of no real roots that means the intersection is imaginary there is nothing real indeed. The ray physically does not intersect the surface it is possible because if you have a sphere here unless you actually have the ray close by not only that intersect to the surface then you will actually have an intersection.

If the surface is here and the rays is somewhere on the top or at the bottom and the spherical surface is only occupying a finite volume in your 3D scene and the ray cast is on the top or away from this sphere so there are may not be any intersection it is possible. So mathematically what comes out? You will not have any real roots and you just neglect both the solutions in the imaginary because there is no intersection of that particular surface with that ray. So surface and ray do not intersect with no real roots.

You can have a situation of one real root that is a very interesting phenomena. The ray tangentially grazes the surface. Of course you can visualize that you can have a ray which is tangentially passing through the spherical surface in this case. And what happens mathematically is the discriminate value, if you take a equation of the quadratic form root over b square minus 4ac that part will vanish, that will become 0 so these two real roots will now become just one root and that is the case when the ray just tangentially grazes that particular surface.

The third case involves the real situation of the case when we have two real roots from both the intersections and we need to get the one which is the smallest value of t. So this is the case which is of interest because we neglect the case when we have no real roots the first one when there is no intersection, the second also when we have one real root and the ray tangentially grazes the surface we actually do not have an intensity point for that particular point (X, Y) but we may have a line and the case of two real roots from both intersections

(Refer Slide Time 27:00)



We get the one which is the smallest value of t. Why smallest value of t is because if you have a surface and then there is a ray which is coming out of the projection plane striking the frontal part of this sphere and then of course the rear part as well. Remember, the have a sphere and some part of the sphere is closest to the viewer the other part is on the back side. Of course if you have done back face culling that part of the surface will not be present in the computations but assuming the full sphere to be present and since you are using this equation so you will get two values of T. You will get two values of t two real roots in the case when you actually have two intersections of the spherical surface with the line and what you do is take this smallest value of T and that will help you to compute the values of (X, Y, Z) on the line which is intersecting the surface.

This is what you do is an example of computing an intersection of a light ray with any complex surface a sort of curved surface. In this case you taken a sphere but you can take any other surface, it could be a cone or a cylinder or you can also visualize any curved surface of the form Z is equal to $F(x, y)$. If you take Z is equal to $F(x, y)$ form then what basically you have do this take the re equation and take the equation of the surface as Z minus $F(x, y)$ is equal to 0 take that form and also take the corresponding equations from the parametric form of a line and then substitute back and then what you do is again you basically find out the value of t in this particular case.

You can have an arbitrary surface and you have a ray coming out you can visualize that if you have a sinusoidal oscillatory surface are an exponential decade something like a Gaussian surface or Gaussian function type of a surface or any other type of variations which you have typically used in digital terrain modeling where sometimes curved surfaces are better to represent than piecewise planar approximations.

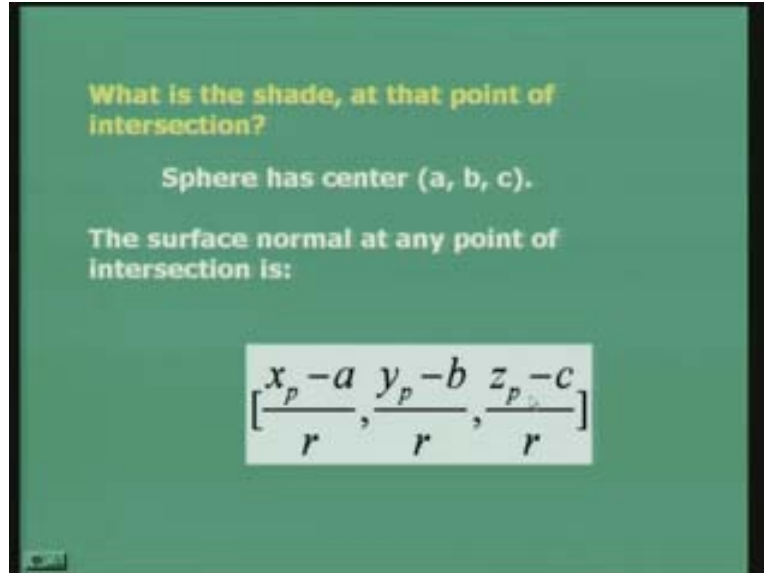
We will talk of coefficient surfaces later on but if you have a functional form z is equal to $F(x, y)$ there should not be any problem for you to calculate the intersection of the surface with a ray. Just substitute the equation ray into the surface form and obtain the value of T and that T helps you to obtain the intersection pattern. So what is the shade at that point of intersection. That is of course a big question and that question will not be answered now.

Although we bypass that question now and leave it to the discussion stage when we discuss shading and rendering algorithm. But we will just discuss a small point here only with respect to this sphere since the sphere has a center we know the A, B, C at the center of the sphere and R is the radius the surface normal at any point of intersection in this particular case.

We are talking only about the intersection of a sphere with a ray and we will later on find out why is the normal has a very important role in computing the shading on intensity the normal at any given point x_p, y_p, z_p on a surface is given by this particular equation. Remember when we talk about solid modeling when we are also talking about back face culling the surface normal was a very important computation which we have to always do.

We have to always do that because that was not only used for back face culling but it plays a very significant role in terms of the computation of shading. So you need to compute the surface normal, I leave it at that point here that is normal will be used to compute the shading or intensity at that particular point wherever that ray has gone and struck that particular surface in this case of a sphere. I am just giving you the expression for the surface normal of a sphere only at that point x_p, y_p, z_p . So given a b, c as a center of this sphere and radius r if you know x_p, y_p, z_p as a three dimensional coordinates of a point on the surface of that particular sphere the normal at the particular point on the surface of this sphere will be given by this particular expression.

(Refer Slide Time: 31:10)



I leave it as an exercise for you to derive this very easily. You should be able to derive this quite comfortably if you know basic geometry where x_p y_p z_p I repeat again x_p y_p z_p are the coordinates of a point on the surface of a sphere which has its center at a b c and the radius is r . Let us take the intersection of the simplest case of a line and a plane. The equation of plane is Ax plus By plus Cz plus d is equal to 0. So (a, b, c) is the surface normal we know that. The parameters of the direction cosines of the surface normal D is the perpendicular or shortest distance of the plane from the origin. And of course we have seen the equation of the line many times where how you get x_0 y_0 z_0 and the interpretation of Δx Δy Δz .

We have discussed already in terms of the parametric representation of a line. So, when you substitute the same thing we get an expression of t which is given as this. All the parameters on the right hand side are known A B C and D are from the equation of the plane and Δx Δy Δz are or obtained as from the parameters of the line so that is what you get. So you obtain D as a single solution in this case unlike you did not have a quadratic as a case of the surface. In fact in case of a curved surface you can have more than one solution more than two in fact. It depends on the type of a surface which you are handling. In the case of a sphere cone or cylinder we might have just two but if it is a highly nonlinear oscillatory type of a surface and the ray is passing through it might be a case that where you can have more than even two intersections of the ray with a particular surface.

I leave this as an exercise for you to visualize when of a surface is not difficult comfortably you can visualize that when you have an oscillatory surface a ray can intersect not only one or two but many many times. In fact sometimes the intersections can go to infinitive. The total number of intersections can be virtually infinitive. But let us come back to this equation of the intersection of the line with the plane and if you go back to the slide the expression of the t numerator by denominator if you see here it is

just simple one solution for t is what you get and there should not be any problem of evaluating the t once the values of A B C and D are given from the equation of a plane and Δx Δy and Δz are obtained from the equation. You will really feel that there will not be any problem of computation of t in terms of computation. You definitely have one value of t there is no doubt about it. You have only one solution for t . But always you will have a finite value of t . If you look back into the expression once again in the slide the denominator may vanish and so you can always compute a finite value of t only when the denominator is not 0. So the denominator should not be 0.

When will the denominator become 0? Look back into the expression once again, look into the denominator term. The term in the denominator of the value of t when will this expression become 0? You should be able to guess it yourself. If you see the term it is appearing as a dot product of two vectors what are those two vectors? A B C surface normal of the plane and what is the other vector? Δx Δy Δz . What is that vector Δx Δy Δz ? You can visualize that is the direction cosines of the vector along the line. So there are two vectors one the line itself another normal to the surface. If these two are perpendicular to one another what happens? The dot product is equal to 0. And the expression at the denominator for the computation of t is basically a dot product of those two vectors. So it will not be 0 only when those two vectors are not orthogonal or normal to each other.

Look back only when the vectors are normal to each other denominator will vary otherwise you will always be able to compute a finite value of t . When will this dot product vanish or be is equal to 0? That is when these two surface normal and line become normal to each other is the case. We discussed about this back face culling when the C was is equal to 0. That is the surface is in such a manner that the ray is actually passing through the surface it will not intersect. It will never intersect in fact it grazes to the surface and that is the case when the normal and the ray are perpendicular to each other. We do not even attempt to compute the value of t because you can also treat it as a back face.

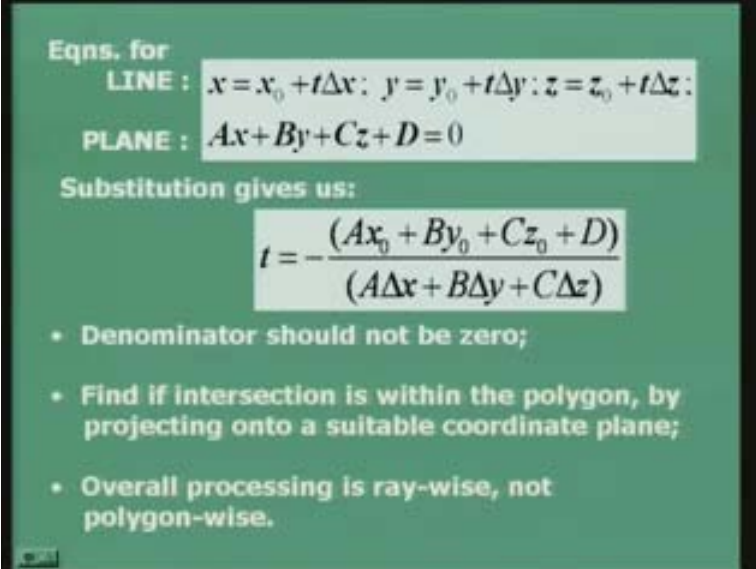
So, if your back face has been handled in the case of planar surfaces polygons there will not be any scope of the denominator vanishing in this case. So let us go to the other point. Find if the intersection is within the polygon by projecting onto a suitable coordinate plane. You will always get a value of t because if you have a polygon and there is a ray the infinite plane will intersect the surface that is always the case. But actually you are interested to know whether the ray actually passes through the polygon because that is a finite area in 3D or 2D. Infinite plane ray will always intersect but if you take a finite polygon or a triangle the ray may or may not actually pass through the polygon.

What you do? It is very simple, project the ray and the polygon into a two dimensional coordinate system xy or yz whatever is comfortable and find out if that intersection point which you have just computed using the value of t you take the projection of that point because that point is a 3D but when you project the point and the polygon on to a two dimensional coordinate system you will have everything in 2D and you just simply apply inside outside test as what we did in polygon filling.

Find out if that point is within the polygon that is very simple in 2D. So that is what you do, find the intersection or find if intersection is within the polygon by projecting on to a suitable coordinate plane so you transform the problem from 3D to 2D. And the last point which you must remember is that did the overall processing is done ray wise and not polygon wise. Remember, we are talking about image space and object space methods in the previous VSD algorithms.

I will say, this falls under the category of image space in some sense because you are scanning the image pixels one by one sequentially row wise or column wise and for each row or column you pass through all the pixels and for each pixel you shoot the ray out. So you are doing in image space ray wise and not polygon wise. For each ray you are computing the intersection for all the polygons or the nonlinear surface which exists in the three dimensional scene and then compute the nearest surface and that is what you should.

(Refer Slide Time: 37:20)



Eqns. for
LINE : $x = x_0 + t\Delta x; y = y_0 + t\Delta y; z = z_0 + t\Delta z;$
PLANE : $Ax + By + Cz + D = 0$

Substitution gives us:

$$t = -\frac{(Ax_0 + By_0 + Cz_0 + D)}{(A\Delta x + B\Delta y + C\Delta z)}$$

- Denominator should not be zero;
- Find if intersection is within the polygon, by projecting onto a suitable coordinate plane;
- Overall processing is ray-wise, not polygon-wise.

So the last few points in terms of comparison of the various VSD or Hidden Surface Removal or Hidden Surface Elimination algorithms or Visible Surface Detection various terms are used interchangeably VSD, HSR or even HSE. Visible Surface Detection or Hidden Surface Removal or Hidden Surface Elimination we have discussed about half a dozen different algorithms in this class. And due to the amount of time left to us we will not be able to discuss a few other algorithms which I expect and request you to read couple of them which is important and the A buffer algorithm and the concept based on Octrees.

We discussed Octrees in CSG solid modeling so CSGs are also used for ray casting. And almost visualize that when you have an Octree representation of an object if you remember that Octree type of structure based on the ray direction looking into the positive or negative Z you can actually label the sections of those Octrees which will be

hidden or not based on the ray direction that is the principle philosophy which is used in Octrees. But the computation is not very efficient and the visualization is also not that easy so that is why Octrees are not that commonly used for Visible Surface Detection algorithm. And also I will request you to read about a buffer accumulated buffer as it is called for VSD algorithm. So VSD or HSR or HSE, Visible Surface Detection Hidden Surface Elimination or Hidden Surface Removal algorithms I repeat the list of algorithms if you have discussed we started with Z buffer then scanline based algorithm then Painter's algorithm which is depth sorting or area subdivision method or BSP trees and ray casting.

We will just take a look at the table when we try to compare these algorithms both in terms of memory requirement and speed but we must keep in mind that the most efficient methods for generating visual realism virtual reality in complex scenes is a combination of various concepts taken from Z buffer and ray casting put together because they are the ones which are popular, easy to visualize, easy to implement of course but except with that point of computational complexity ray casting is really inefficient in terms of the amount of computational term requirement. But since it simulates the reality which I talked about earlier today in terms of trying to simulate the reality in terms of the ray coming out of the light source hitting the object and then coming to viewer we are just traversing in the opposite direction now.

We go along the reverse direction from the projection plane to the object surface and then look towards the source which is giving the light and that is how the ray casting works. And the basic concept is intersection of a surface with a ray which we are seeing the case of ray casting. That is very simple and either it is a planar surface or a curved surface in any form we have taken of course the simple example of only a plane and a sphere, I leave it as an exercise for you to take another type of curved surfaces like a cylinder or a cone and try to take the equation of a surface of a cone or a cylinder and then try to intersect the ray with that also try similar equations and see what are the results of the values of t which you get. Again you will have two values of t you can have situations of complex roots which you have to throw out may be one real root or two.

When you have only two real roots or multiple roots you take the one which is the minimum value of t because the t increases from the projection plane towards the three dimensional scene towards the object so the minimum value of t will tell you that is the closest object which you are intersecting closest surface which you are intersecting. Of course the last step involves either a polygon or a surface especially in the case of a polygon you need to find out whether you are actually intersecting the polygon. This is unlike the case of a surface, when you have a value of T you are actually intersecting a curved surface if it is represented in full.

Of course if it is represented in part it is possible that you will have a value of T but it could lie outside a smaller part of the curved surface which is representing. So you need to find out always whether there is a physical intersection between the part of the planar or curved surface with the light ray which is cast back from the projection plane so project everything including the ray and the polygonal patch curved or real on to a two dimensional scenario the ZX plane or ZY plane and it is a two dimensional problem

where you find out whether the (X, Y, Z) point which you computed from the value of t. That x, y, z will now become a ZY or ZX it is a two dimensional problem and you find out whether a point is lying inside a polygon or not. That is a simple inside outside test which you do for a Ray tracing and we will although compare using a table and different algorithms.

Keep in mind the ray casting and the Z buffer or a combination of both is the one which is the most sophisticated and which is used for generating a real complex scenario as in the case of virtual reality or to simulate concepts based on visual realism based on anti aliasing based on concepts such as radio city which are also used to generate the real picture. In fact you can see light sources along with good shading models.

We will see that it is a combination of Ray tracing, depth buffer, radio city, anti aliasing. These are the three or four common topics which are combined together to keep very good visualization of shading and intensity. Of course we have to discuss shading models which we will start in the next class but before doing that we have a comparison of the various VSD or Hidden Surface Removable techniques for the remaining time left to us as algorithms in the first column, the memory requirement in the second and the speed on the last column.

I have just tried to give a brief idea about the comparative nature in terms of Z buffer the memory requirement is the most because we basically need to process two arrays. Remember, Z buffer requires a refresh buffer and also a depth buffer Z of XY and I of XY. So the memory requirement is the highest or the most in the case of Z buffer and the speed requirement based on the depth complexity which you are handling.

Painter's algorithm or what is called as depth sorting algorithm requires one array in terms of memory requirement and the complexity could be very high A-priori sorting of the polygons will help you to speed up. This is a comment which you can note down that A-priori sorting of the polygon will help you speedup the Painter's algorithm.

What about ray casting? And I have compared the main three or four algorithms only not all of these which you have studied so far. The memory requirement depends of the object data base.

What do you mean by object database? You are representing or three dimensional scene with a set of polygons or a set of curved surface parameters of the curved surfaces which are in the scene. So you can have a mixture of curved surface and planar surface or a combination of both and the complexity of course the time requirement will basically depend on the number of pixels on the screen and the number of surfaces but the memory requirement will be based on the amount of the surface representation, the object representation or the scene representation that is the number of polygons which are used, the number of curved surface which are used will basically dictate the memory requirement. The refresh buffer does not have any role to play much in terms of the memory requirement.

The memory requirement is basically the object database. So ray casting if you look back the memory requirement is on the three dimensional scene. The object database which must be of less important role for the memory requirement speed is highly complex, the complexity depends on the number of pixels and the number of surfaces or objects.

So, if you have N square pixels and P surfaces or S surfaces you can visualize that the computational complexity will be in the order N square into S . Increasing the resolution number of pixels increases the complexity increases nonlinearly and linearly with respect to the number of surfaces which are present and modeled in the surface. Well, in terms of scanline or area subdivision method the memory requirement of course is based on, I put a dash because it depends on both the complexity in the image domain as well as the object space and the speed means it is the slowest.

Why the scanline area subdivision is slowest because as you can see if you remember what we have discussed a few classes back area subdivision is based on splitting into various sub modules and scanline method also fails when you need to partition surfaces and need to compare depths. It is basically a 3D polygon filling algorithm where at each point you need to compare depth and then use it for shading. So this is the most inefficient and lowest that is why it is not that popular although the algorithm exists. I did repeat again; Z buffer combined with the ray casting is the most popular one and that is what is mostly used.

(Refer Slide Time: 00:46:17)

Comparison of VSD (HSR) techniques

Algorithms/ Methods	Memory	Speed
Z-Buffer	Two arrays	Depth complexity
Painter's	One array	Apriori sorting helps speed-up
Ray casting	Object data base	$O(\#pixels, \#surfaces \text{ or } objects)$
Scanline, Area sub-division	-	Slowest

The algorithms and methods in terms of issues of implementation and a remark for each of those four which you have studied so far the Z buffer or the depth buffer algorithm is the one which is based on scan conversion and it is implementable in hardware. There are specific Computer Graphics hardware on the graphics accelerator cards which are based on Z buffer and it is of course commonly used is a remark for Z buffer.

The Painter's algorithm is based on the scan conversion technology and the major drawback or major bottleneck for the Painter's algorithms or what is also known as the depth sorting algorithm is this splitting and sorting. You need to split and sort and that is the one which increases the computational complexity that is the major drawback of Painter's algorithm and that is why it is not commonly used.

Although conceptually it is very nice to visualize how the Painter's concept is used in terms of rendering in terms of ordering depth but it is not commonly used. Ray casting spatial data structures help the speedup so we need to have efficient data structures which can model your solid objects and then excellent for constructive solid geometry or CSG it is used for shadows and transmissions.

So you can see when we talked of shadows and transparency the ray casting is the most efficient one, why? We did not talk of shadows anytime we will talk more of that when we move towards shading and rendering algorithms. But if you visualize shadows how do you get a shadow? You get a shadow on a surface because this surface is basically hidden by some other surface. As you can see when light rays are coming and hitting on the surface you have perfect illumination. But if I hit this light source if I hide this light source by some other object you can see the effect of a shadow.

If I remove that occluding one and the light source is visible you can see that there is no shadow. So how do you implement this concept using in any other conceptual modeling of Visible Surface Detection? You have to incorporate that only with the help of ray casting. Why? When you ray cast assuming that the ray is coming out of your eye currently and hitting this polygon planar patch which is my palm, assume this to be a rectangular or a planar surface as you can see a ray comes and hits so what you see at this point you find that there is a light source which is illuminating that surface. So you illuminate and shade it. But in the case of shadows when you have an occluding surface what will happen from this point if you have an occluding surface in front you will actually cause the ray back towards the light source and you will find that there is another object in front sitting in between the light source and this surface being rendered.

There is another surface sitting in between which is creating this effect of shadows. If I remove it you will see bright illumination and if I bring it you see the effect of light shadows. So you can compute that only with the help of ray casting because a ray comes from you or from the projection plane and then hits its surface then you can cast this ray back towards the light source and when you are doing that in the reverse direction of the light source in terms of the reverse direction of the light ray which is actually coming out of the source hitting the polygon and moving towards you, now when you are ray casting back after hitting the surface when you are moving towards the light source you find that there is another object planar patch which is occluding or hiding the light source from the polygon which you are rendering currently for that particular ray. So do not shade it with full illumination. May be that ray is receiving some other lights from some other direction that is visible.

If all the light sources are hidden it is possible that you will have a complete shadow. Look at this particular case when my face is completely under shadow and I remove one

by one the occluding planar patches the light sources will be illuminating certain polygon assuming that you are representing my face as a solid object then what you will have is if you are occluding polygons then occluding the light source you will have the effect of shadows. You can only do that with the help of ray casting when you cast a ray to the object and then find out if the light sources are visible or not.

If they are not visible you paint it with a lower intensity or do not paint it and that gives the impression of shadows. Shadows can be visualized only with the help of ray casting. What about transparency? We did not discuss about transparent objects and so far we have only talked about opaque objects where the light sources which are impinging on the objects are all reflected back or observed by the surface.

In case of transparent objects some light sources pass through object. Typically glass object let us say you can have translucent or transparent objects with light rays pass through it on objects on the backside of an object, occluded object will be visible in the case of transparent object. That means if you have an occluded object on the backside and a glassy object in the front take a glass in which you used to drink water then objects on the backside of it will be visible. If you carefully see, ray casting helps you to implement not only excellent for Constructive Solid Geometry operations implementing shadows and transparency.

All complex representations can be handled with the help of ray casting and Z buffer is commonly used also over and above if necessary along with ray casting because it is efficient for hardware implementation. Scanline and area subdivision I leave it with a small comment here that implementation is very hard and it cannot be generalized for non-polygonal law model that is for curved surfaces. Curved surfaces, concept of scanline or area subdivision method is very difficult to implement but ray casting is efficient where even Z buffer can be implemented.

Painter's algorithm is exclusively for polygonal objects, you have to do a lot of modification's to make it suitable for curved object shapes that is true whereas for ray casting you do not have to do any modification, you can straight away handle not only complexities in terms of shadows, transparencies and complex solid object representations of Constructive Solid Geometry but also you can handle all sorts of curved objects, planar objects or combination of both.

So, coming to the last point here scanline area subdivision method is very hard in terms of implementation because it cannot be generalized for non-polygonal models. So you leave this slide with a point that ray casting is the most efficient one not in terms of computational complexity but in terms of implementing real scenarios, real pictures and that is the essence of virtual reality or visual realism. This is a small point I want to talk about before winding up this lecture on Visible Surface Detection.

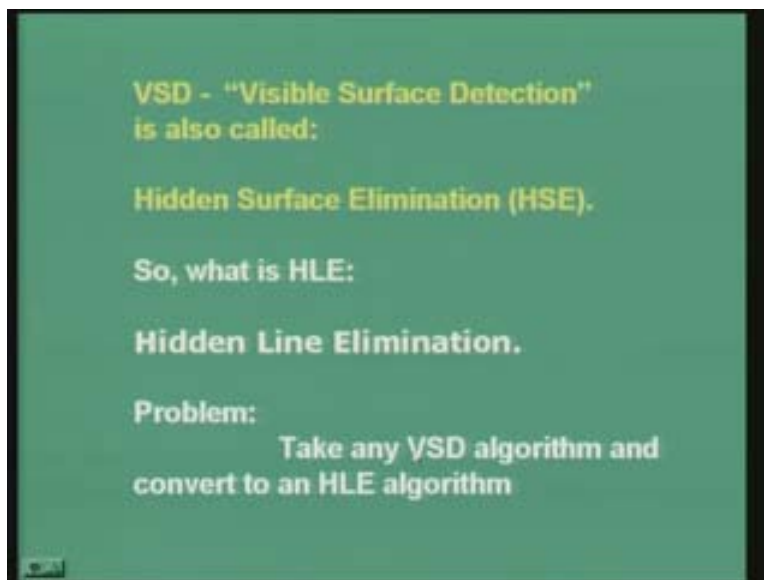
We talked about this point that Visible Surface Detection or VSDs also called Hidden Surface Elimination (HSE) or Hidden Surface Removal or HSR these are the terms which are used interchangeably. There is a term called HLE. Based on these two words VSD and HSC you can almost visualize what is going to be your HLE. It is Hidden Line

Elimination. What is the Hidden Line Elimination? If you remember the first picture when we started to talk about Visible Surface Detection and before back face culling I gave the example of that Gaussian surface in wireframe diagram, it was not shaded with the intensity and the front parts of those surface are visible the back part was eliminated and it was a line diagram a wireframe diagram.

In that case if you remember that picture where while showed both the front and back lines in terms of faces and the second picture the lines for eliminated. So you need a case where you may not need to actually shade the surface but throw out the hidden lines only. So, Hidden Line Elimination is also a problem like a Hidden Surface Elimination or Visible Surface Detection algorithm and it is just a small extension of all VSD algorithms. I will give a simple method by which you can turn any VSD algorithm into a Hidden Line Elimination algorithm.

Let us take the case of scanline or Z buffer in some essence when you are working from one polygon to another you are rendering one polygon and the second one and so on. That means object wise and then polygon wise you are rendering. When you render a polygon either you can render the whole polygon or only the edge. How can you do that? Whenever a pixel on the polygon is on the edge you render that you do not render pixels which are inside the polygon. That means you render only the edges of the polygon and not the inside. Inside you render with only background color and render only the lines of the edges of the polygon with a foreground color. This is the essence by which a Hidden Surface algorithm or VSD can be turned into Hidden Line Elimination or a HLE. That is the key where we say, that is the problem which I leave it for you to visualize but I give you a clue that you take any VSD algorithm and convert into an HLE algorithm.

(Refer Slide Time: 56:00)



The essence of that is to not paint the entire polygon inside part you do not paint at all do not render at all you do not have to work the VSD inside a polygon for points which are

inside the polygon that you can easily test. You only render points which are on the edges and use the VSD only on the edges of the pixels which are on the edges of the polygon. I repeat, you use the VSD only on the pixels which are on the edge of the polygon in 2D or 3D. That is how you implement a HLE algorithm from a VSD algorithm. So that is the end of the lectures on VSD algorithms.

From the next lecture onwards we move on to the concepts based on the rendering and shading which is an essential part of the last stage of the 3D viewing pipeline because at the end of the every VSD algorithm we have said that we will paint using rendering or shading models. That is what we will see in next class and that is the end of today's lecture and the sequence of VSD or Visible Surface Detection algorithms, thank you very much.