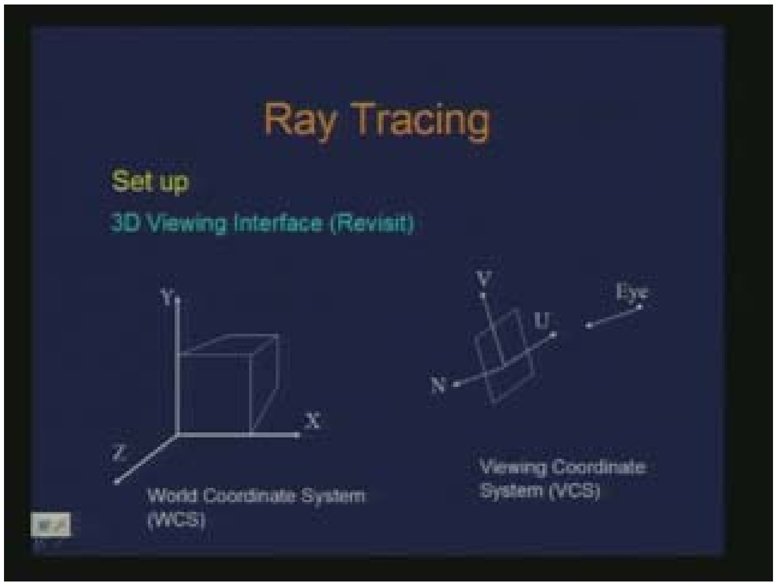
Introduction to Computer Graphics
Dr. Prem Kalra
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 25
Ray Tracing (Contd.)

Today again we are going to talk about ray tracing looking at the interface from the users' prime point of view just like what we did in viewing transformation. So, if we recall what we saw in the 3D viewing interface the scene was basically defined in World Coordinate System and the eye or the viewer or the camera was specified in another coordinate system which was the viewing coordinate system.

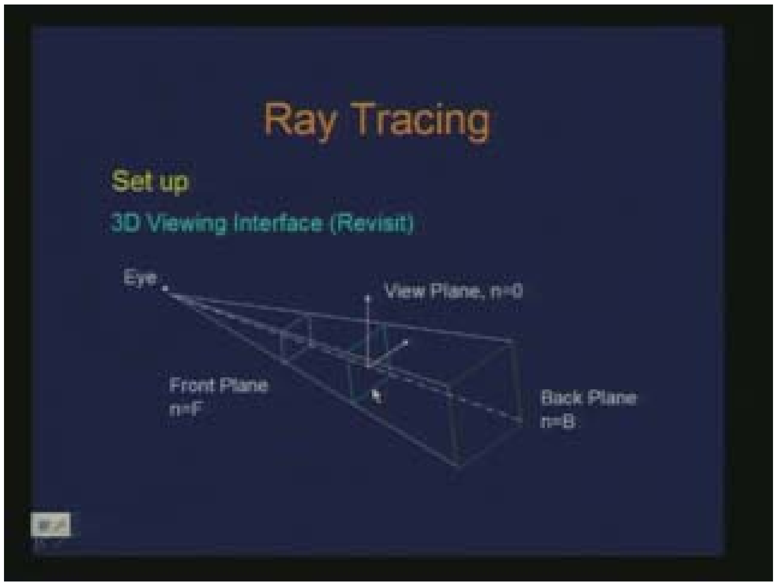
In fact this was a left handed system and this was a right handed system. If you assume that this is the kind of interface we are going to have from the users' prime point of view that means the user is going to specify the position of the eye or the camera in this coordinate system and your world is going to be in this coordinate system. Hence there are going to be issues in terms of how do we specify the corresponding ray definition in one of these coordinate systems so that when we get the intersection of the ray with the world this is consistent with the coordinate system. Therefore again we need to look at issues of transformation from one system to another system and how do we incorporate that with respect to ray tracing.

(Refer Slide Time: 00:02:24)


Ray Tracing
Set up
3D Viewing Interface (Revisit)
Y
X
Z
World Coordinate System (WCS)
V
U
N
Eye
Viewing Coordinate System (VCS)

Here this is the eye and that is where the origin of the ray starts with respect to the ray tracing and this is what we get as our viewing volume.

(Refer Slide Time: 00:04:22)



Ray Tracing

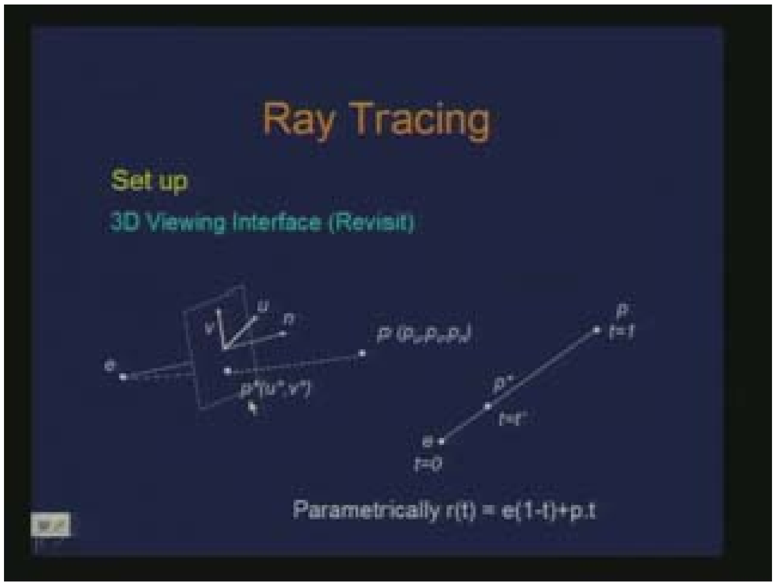
Set up

3D Viewing Interface (Revisit)

So we need to have this in one coordinate system. Therefore whatever scene we need to specify in the viewing volume has to be in the same coordinate system. Now the question is, would you like the ray to be specified in the world to have the intersection done, that is one way but we are insisting that the specification of the eye is in another coordinator system.

So we need to transform that into the world coordinate system so that the intersection what we get is in the world coordinate system. The other issue is that one could have done this whole process of finding the intersection and so on in the viewing coordinate system. So what is the implication of that? At times what happens is that one aspect is to get the intersection point and another aspect is to get the normal at that point.

(Refer Slide Time: 00:06:39)



Ray Tracing

Set up

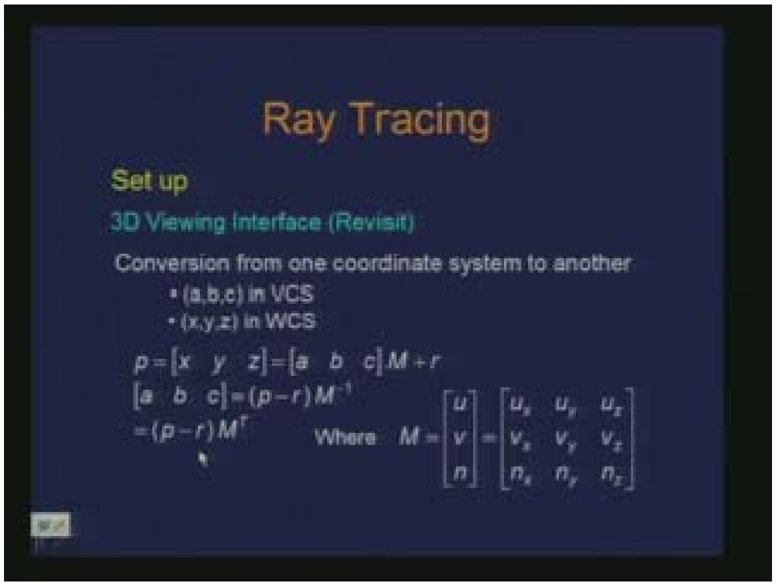
3D Viewing Interface (Revisit)

Parametrically r(t) = e(1-t)+p.t

At times we are even given the normals for the object or for the faces of the polygons of the object. So not only just the coordinates of the vertices are given but also the normals are given so that one does not have to compute that. So in that situation we would like to use that information which is already there rather than re-computing them. But again these normals are in World Coordinate System. Therefore it is better to do this intersection in world coordinate system. Here are some issues to look at with respect to the viewing transformation. So if this point P which has been now transformed in u v n or the viewing coordinate system and if I consider a ray coming from the point E which is the eye then this the point which is on the image plane and which is of interest to us. So this is what I will render here.

Again I can see this as a ray defined in a parametric form with origin as e and this point P as the termination of that. So the same thing can be represented as a line parametrically like this. And we have a similar definition of the ray so there is a match. This eye is in the viewing coordinating system. I am defining it in this coordinator system. I am just looking at definition of a line with respect to the point defined e and this point which I have got after having done the transformation from the world to the eye. This is where I get the image of the point.

(Refer Slide Time: 00:09:11)



Ray Tracing

Set up

3D Viewing Interface (Revisit)

Conversion from one coordinate system to another

- (a,b,c) in VCS
- (x,y,z) in WCS

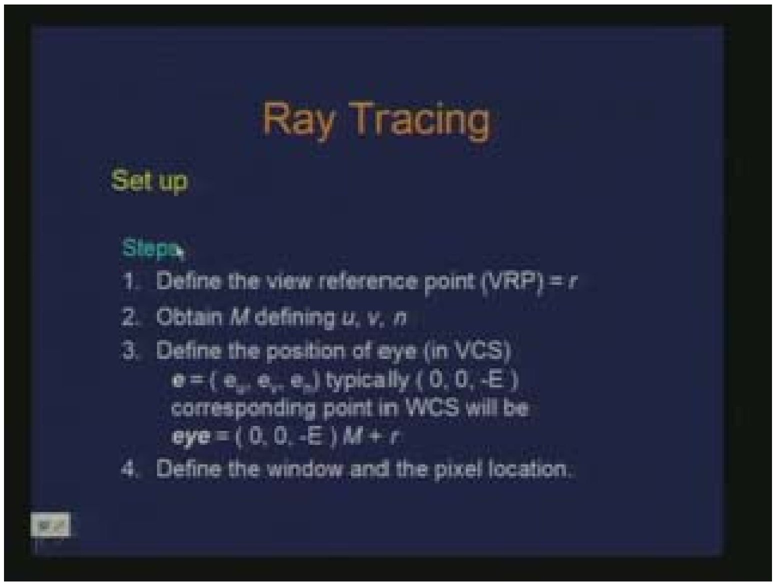
$$p=[x \quad y \quad z]=[a \quad b \quad c]M+r$$
$$[a \quad b \quad c]=(p-r)M^{-1}$$
$$=(p-r)M^{T} \qquad \text{Where} \quad M=\begin{bmatrix} u \\ v \\ n \end{bmatrix}=\begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{bmatrix}$$

Now we need to have a mechanism of transforming the point from one coordinate system to another coordinator system. And we have debated on, that it is better to have a point represented or the ray represented in world coordinate system.

If we consider a point defined as [x y z] in world coordinate system and correspondingly the points [a b c] in viewing coordinate system then what is the transformation which takes the point from one system to another system. If this is the point P we are considering then all it requires is a transmission of the type m which is the top most sub matrix of the 4 into 4 matrix then there is an offset term or the translation term.

And if I am interested in getting [a b c] given m and r from [x y z] I can obtain in this way. Therefore I can get this [a b c] from the point given in the world using this because this inverse of m happens to be a transpose just because we know m is basically formed by the u v and n vectors which are nothing but these and this is nothing but a matrix which is orthogonal and therefore the inverse of this is the same as the transpose. So basically we can get a point in the viewing or the eye coordinate system from the world. Now the question is how this maps to our ray tracer.

(Refer Slide Time: 00:12:16)



Ray Tracing

Set up

Steps

1. Define the view reference point (VRP) = *r*
2. Obtain *M* defining *u*, *v*, *n*
3. Define the position of eye (in VCS)
   e = (e_u, e_v, e_n) typically (0, 0, -E)
   corresponding point in WCS will be
   eye = (0, 0, -E) *M* + *r*
4. Define the window and the pixel location.

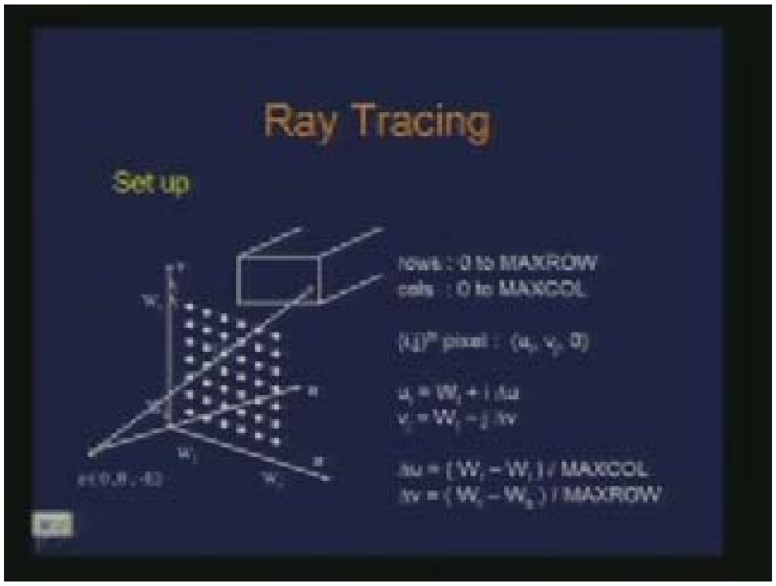
If we have to take the definition of the ray which is coming from the specification of eye in the eye coordinator system to the world coordinator system is what we want. So, if you want to have this set up for ray tracing the first thing we require is the definition of the viewing reference point which in turn defines this offset r which is the user specified quantity. So you either compute in some sense this VRP or give VRP. Hence this is user specified. We obtain the matrix M from u v and n.

There is a way you can either specify or get n and then there is a way you align the v vector in conformation to the up vector specified by the user and from these two you get u so we can obtain the matrix M.

Now if the eye which is defined in the viewing coordinator system as e given as e_u, e_v and e_n is typically defined as a point on the line of sight where the line of sight is coincident to the vector n and typically we observe that the eye is defined as [0, 0, minus E]. Now what we are looking at is that where does this point go in the world coordinator system. So all we need to do is perform the transformation of this through m and r and obtain eye in world coordinator system.

Now we know the position of the eye in world coordination system. The other thing is we need to also look at the definition of the window which might be in again the viewing coordinating system and the corresponding pixel location to the world coordinates. Therefore this is the eye e in viewing coordinator system through [0, 0, minus E] and these are the objects, this is the viewing plane where the window is specified through W_l W_r along u and W_b W_t along v.

(Refer Slide Time: 00:15:39)



Ray Tracing

Set up

rows : 0 to MAXROW
cols : 0 to MAXCOL

$(i,j)^{th}$ pixel : $(u_i, v_j, 0)$

$u_i = W_l + i\,\Delta u$
$v_j = W_t - j\,\Delta v$

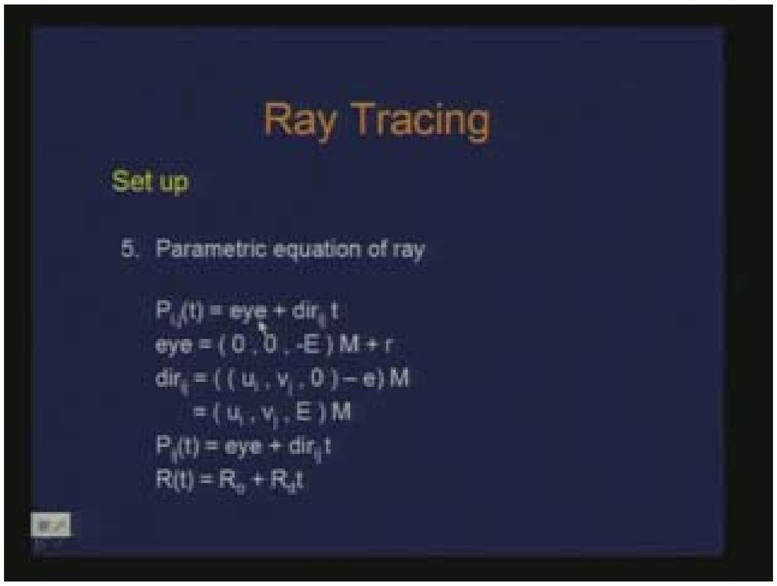
$\Delta u = (W_r - W_l) / \text{MAXCOL}$
$\Delta v = (W_t - W_b) / \text{MAXROW}$

This is just the extends of your viewing of the window. So typically when you specify the zone of the screen which you want to render you would specify the pixels along x on horizontal axis and the pixels in the vertical axis. So typically you will define the region of the zone of the window through pixels basically. This is how we specify that we are looking at these many rows and these many columns on the screen which we want to render. What is the number of pixels we want to render in horizontal axes and what is the number of pixels in the vertical axis, so it is that window. Hence the window is defined as pixel array.

Now the question is, for a pixel located at ijth location in this array would it correspond to the point u_i v_j 0. Let us assume that this is that n is equal to 0 the viewing plane. Then it is just a matter of figuring out what are u_i and v_j. So I can compute this u_i and v_j through W_l and W_t which is just a matter of mapping this into the window size defined through the W_s namely W_l, W_r and Wd and W_t. that is how it is done here. Therefore it is W_i plus j_i delta u and v_j is Wt minus j(delta v) where delta u and delta v are defined through this. All it is doing is it is basically locating the pixel in this coordinate system.

Now we have to define ray. So the point of the origin through eye has been defined of the ray but we have to define the direction now and that is where the role of u_i v_j comes in because that is where it is passing through and that defines the direction of the ray. So we basically look at the parametric equation of ray through the point of origin which is the eye. There is a direction which corresponds to the ijth pixel and this is the parameter t. So it is the same as R_0 R_dt form.

(Refer Slide Time: 00:17:27)



Ray Tracing

Set up

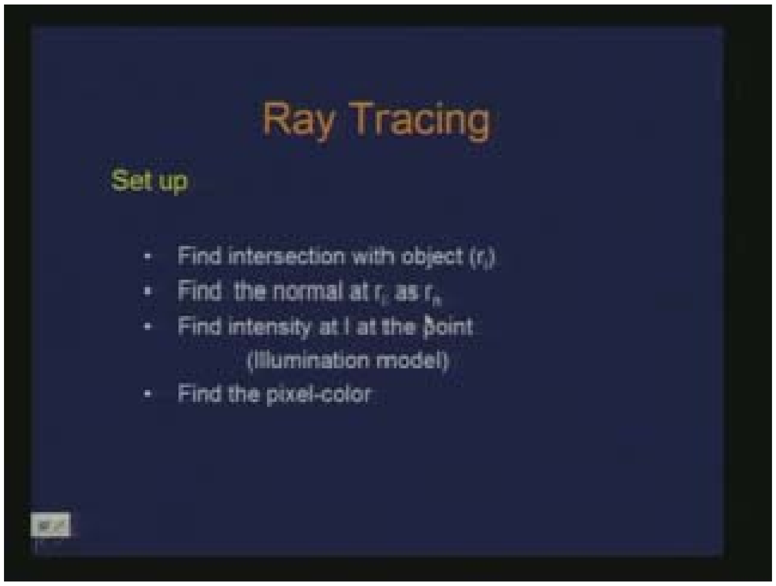
5. Parametric equation of ray

$$P_{ij}(t) = eye + dir_{ij}\, t$$
$$eye = (0, 0, -E)\, M + r$$
$$dir_{ij} = ((u_i, v_j, 0) - e)\, M$$
$$= (u_i, v_j, E)\, M$$
$$P_{ij}(t) = eye + dir_{ij}\, t$$
$$R(t) = R_o + R_d t$$

We have already figured out how we get the eye given the eye point in the viewing coordinate system. The direction is nothing but the pixel position minus e which is the eye position in viewing coordinate system transformed through m. Since it is the direction I do not need r. so this way I get direction i_j. In case if I am considering eye to be [0, 0, minus E] that means along the axis n I get direction i_j as u_i v_j e transformed through M. Therefore now the ray is defined in World Coordinate System which we were trying to achieve and which is in the same format which we have been looking at for performing the intersections.

The next step is to find the intersection with the object that gives you the point here as r_i, find the normal at r_i as some r_n, find intensity eye at the point which is nothing but applying illumination model and then find the pixel color.

(Refer Slide Time: 00:19:23)



Ray Tracing

Set up

- Find intersection with object (r_i)
- Find the normal at r_i as r_n
- Find intensity at I at the point (Illumination model)
- Find the pixel-color

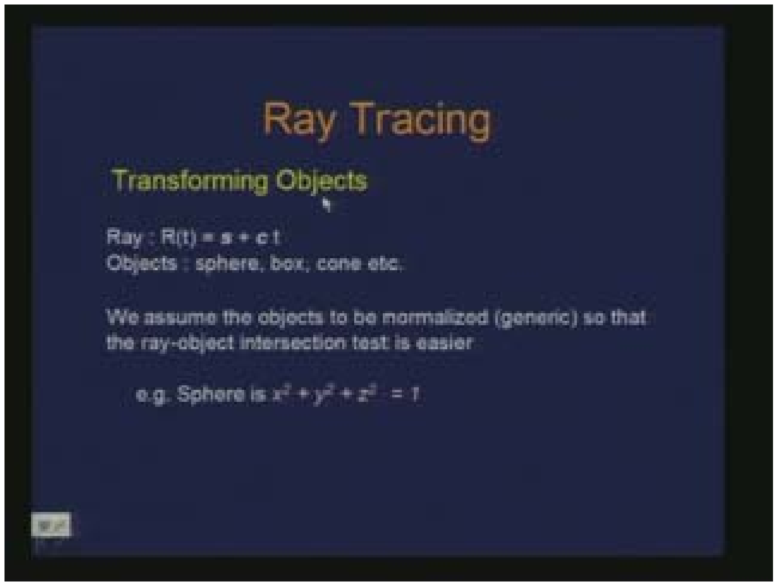
Therefore, what may happen is that when you are computing this eye it may give you a certain range. So one thing is, this eye is a scalar value where you are rendering a gray scale kind of an image single color image with tones of intensity or it could also be a color vector having components of r g b then you have a colored image. And there could be a certain range which you get for your image which you may require to remap to a certain range of your device color maps. Therefore some eye min eye max for this could then be mapped to some 0 to 255 whatever range you have for the depth of pixels in your system. So that way you decide the final color of the pixel. This is how the whole set up is done for ray tracing.

What we have done is only one level of ray tracing in the sense that you are not looking at the secondary rays, it is just an intersection of the ray which starts from the eye and hits the object or a series of object and then you consider the front most object to be rendered. Now there is an interesting issue here, so far what we have looked at in terms of objects are fairly simple objects. Now it might be useful to see if there could be objects which could be derived from those simple objects without doing too much and still be able to use your ray tracing.

If I am using a description of hierarchical scene where the instance of an object is obtained applying a modeling transformation to a primitive then I know how to handle that primitive which could be one of those simple primitives like sphere, plane etc which we have dealt with for ray object intersection. But the question is when I have performed modeling transformation to those primitives can I still use ray tracing without adding too much of a problem.

What I am referring to is that, if we are transforming objects which mean applying a modeling transformation to an object and the object is sort of a simpler primitive which we know how to deal with the respect to ray object intersection.

(Refer Slide Time: 00:23:38)



Ray Tracing

Transforming Objects

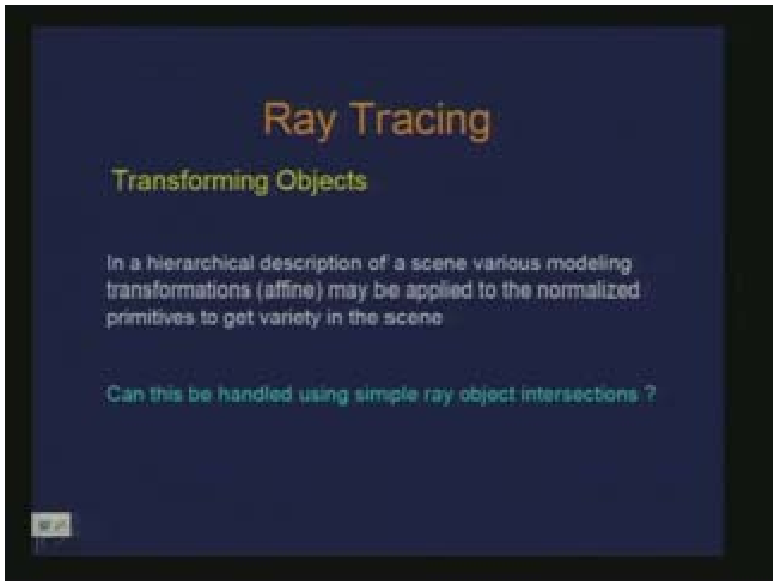
Ray : R(t) = **s** + **c** t
Objects : sphere, box, cone etc.

We assume the objects to be normalized (generic) so that the ray-object intersection test is easier

e.g. Sphere is $x^2 + y^2 + z^2 = 1$

Hence if I look at the ray as a parametric defined line which is s plus c t and the object which we have already looked at are of the kinds sphere, plane, box, cone etc we call these to be some sort of a generic or a normalized class of objects or primitives. For instance, if I am dealing with that object which is a normalized sphere in the form x power 2 plus y power 2 plus z power 2 is equal to 1 then I know that finding intersection is relatively simple. Now the question which is being imposed is that if I take this sphere and perform a transformation to this how does the ray tracing change?

(Refer Slide Time: 00:25:08)



Ray Tracing

Transforming Objects

In a hierarchical description of a scene various modeling transformations (affine) may be applied to the normalized primitives to get variety in the scene

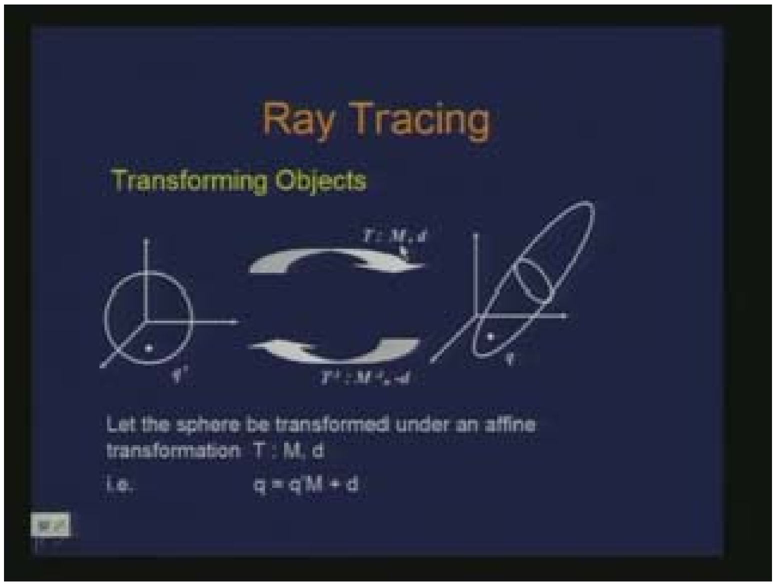
Can this be handled using simple ray object intersections ?

This is particularly pertinent if we have hierarchical description of a scene where we would like to have instances of a simple primitive using modeling transformation. Then the question is, can we still handle the ray object of intersection in the same way?

The kinds of transformations we are trying to observe are affine transformations. The class of transformation which we want to deal with is affine transformation. So, if we see that these affine transformations are applied to this simple primitives what change do we need to be able to perform those ray object intersections so that we can incorporate them.

We know how to handle the ray intersection with sphere which is normalized in the form of x square plus y square plus z square is equal to 1. Now I have some transformed shape which has been applied on this sphere and I know this transformation. But now my interest is to find out the ray intersection with this transformed object because that is what I am going to render. Objects when transformed are not any more this simple primitive. Therefore I cannot apply. So you have already written a ray sphere intersection routine which is dealing with only a simple normalized form of sphere and I still want to use the same routine.

(Refer Slide Time: 00:28:33)



Ray Tracing

Transforming Objects

$T : M, d$
q'
q
$T^{-1} : M^{-1}, -d$

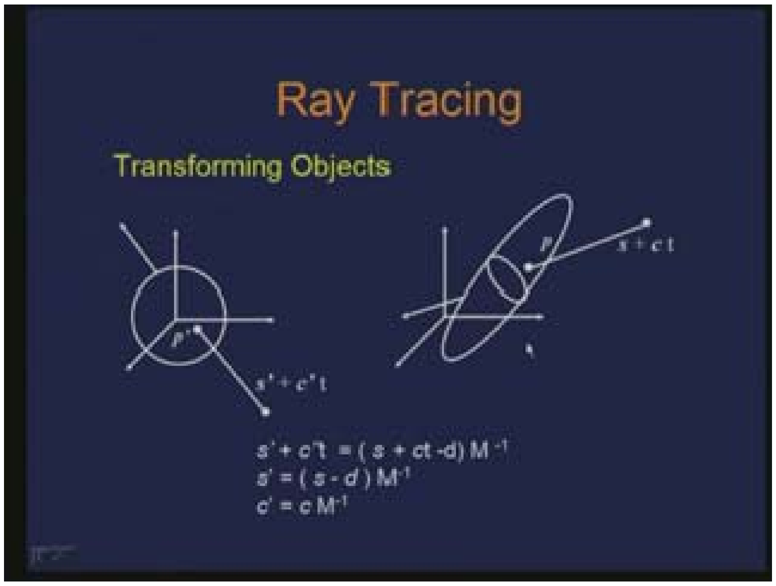
Let the sphere be transformed under an affine transformation T : M, d

i.e. $q = q'M + d$

I have a sphere here and I perform a transmission through t as that is the transformation I have and since I am talking about the class of transmission as affine transformation I can see this to be a composition of some matrix m which is linear transformation and there is some translation d which would give or make this object a sphere transformed in this way.

Now the fact is that, I have this transformation known to me I also know the inverse of this transformation or I can know the inverse of the transmission which is t inverse which is nothing but m inverse in minus t. So from here I can go back here. Hence if I have a point here as q prime for the sphere then I know the corresponding point q through this transformation t as this q is equal to q prime M plus d. Now when I look at what happens to the ray, so my definition of the ray is some s plus ct. This is the ray I have computed by all your set up in world coordinator system whatever coordinator system you were using. This would be the point of intersection p with this ray or this object.

(Refer Slide Time: 00:33:19)



Ray Tracing

Transforming Objects

$$s' + c't = (s + ct - d)\,M^{-1}$$
$$s' = (s - d)\,M^{-1}$$
$$c' = c\,M^{-1}$$

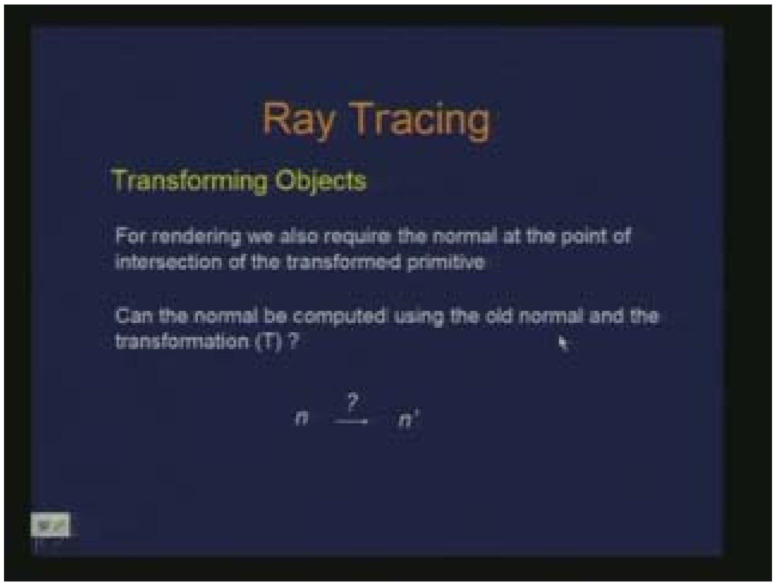
So all I need to know is what point it corresponds to here. In other words, what ray I have which would give me the point p prime which corresponds to the point p. thus all I need is to render the point p which is a consequence of the ray intersection defined ray as s plus ct to this object and unfortunately I do not know how to compute intersection with this object.

What I know is how to compute intersection with this object. So the trick here is that I would like get the corresponding ray to this which would give me the point of intersection p prime corresponding to the point p which I am actually interested in. Therefore the corresponding ray in this for this object which I define as s prime plus c prime t is nothing but the inverse of the transformation which I would apply to this ray which is m – because it is just a ray, a vector, direction.

Then it is just a matter of comparing the coefficients of the two sides and figure out what is s prime and what is c prime. So in a way I have obtained the corresponding ray for this primitive. Now this newly obtained ray can be fed to the standard routine I have for rays sphere intersection. And then I obtain the point p to be rendered. Now there is still a problem.

There is a point q here in this object which has a corresponding point q prime in this object through the transformation t so I can always map q prime to q or vice versa. That is precisely what we want to do and that is done using transformation of the ray. But now there is still some problem. We have obtained the point of intersection. That means we know what point to render and there is something else required which is the normal. Therefore we have been able to do the transformation to the object and now can we obtain a transformation which would necessarily be applied to the normal without re-computation of the normal of the transformed object. Therefore we need these normals to be able to render them for the transformed object. Now the question is, can the normal be computed using the old normal and the transformation?

(Refer Slide Time: 00:35:23)



Ray Tracing

Transforming Objects

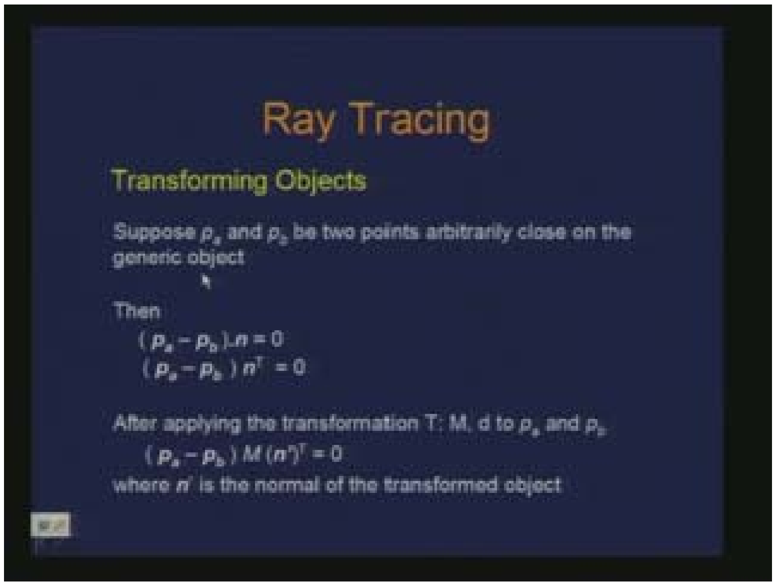
For rendering we also require the normal at the point of intersection of the transformed primitive

Can the normal be computed using the old normal and the transformation (T) ?

$$n \xrightarrow{?} n'$$

So what is that transformation which takes the normal defined for the normalized primitive to give us n prime which is the normal of the transformed object?

(Refer Slide Time: 00:37:44)



Ray Tracing

Transforming Objects

Suppose p_a and p_b be two points arbitrarily close on the generic object

Then

$(p_a - p_b).n = 0$

$(p_a - p_b)\, n^T = 0$

After applying the transformation T: M, d to p_a and p_b

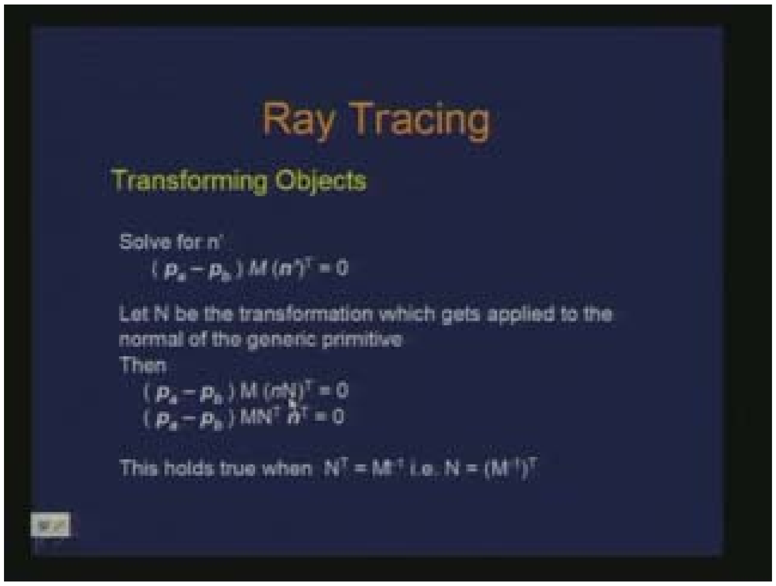
$(p_a - p_b)\, M\, (n')^T = 0$

where n' is the normal of the transformed object

So what is happening is that, if I consider two points P_a and P_b which are very close to each other so it could be applied to any object we are not specifying the type of the object so these are the points which are very close enough then we basically define the tangent of that point and that is why they need to be close enough. So, if have P_a and P_b two points which are close enough then the condition which needs to be satisfied for the normal is nothing but the dot product with this has to be 0.

Now this can be re-written in this format. If I consider a point being defined as a row all I need to have is a transpose here. Now what we are saying is that we apply a transformation t which is consisting of the component m and d which would get applied to P_a and P_b when we apply that to the object which in turn would give you the transformation as (P_a minus P_b) M so keeping the same condition here that the tangent and the normal gives you 0 as a dot product and the same thing is here so this is for the transformed points or they give you a transformed tangent. And if I define n prime to be the new normal then this also should give me 0. Now getting n prime is what the problem is. Hence, basically we are looking at the solution for n prime through this.

(Refer Slide Time: 00:38:54)



Ray Tracing

Transforming Objects

Solve for n'

$(p_a - p_b)\, M\, (n')^T = 0$

Let N be the transformation which gets applied to the normal of the generic primitive

Then

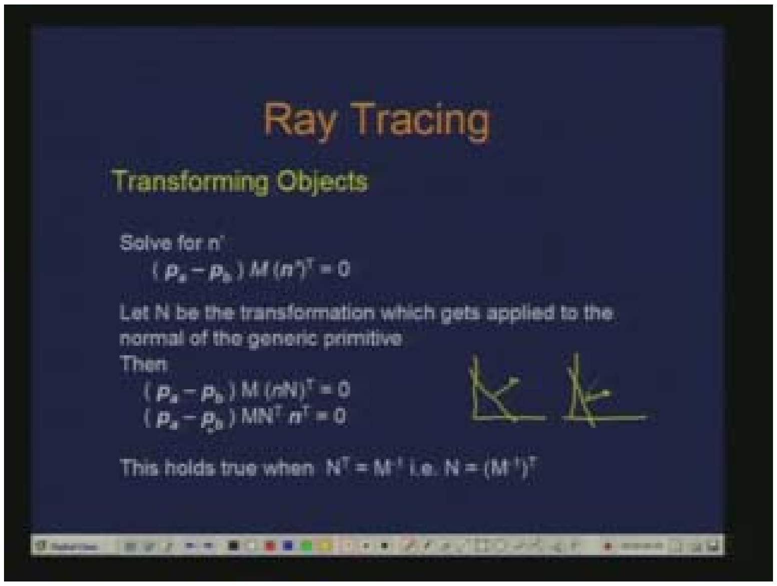
$(p_a - p_b)\, M\, (nN)^T = 0$

$(p_a - p_b)\, MN^T\, n^T = 0$

This holds true when $N^T = M^{-1}$ i.e. $N = (M^{-1})^T$

Now if I define N to be the transmission which is required from the normal n which was for the object which was transformed to n prime which is the normal of the transform. This is the transformation N. So I am just substituting here, instead of n prime I just substituted as n(N) so this gives me this (P_a minus P_b) M n transpose is equal to 0. This holds true when I have n transpose to be the inverse of M. Therefore n which is the transformation I am looking for is nothing but M inverse transpose. So you have the solution of your question here.

(Refer Slide Time: 00:42:23)



Ray Tracing

Transforming Objects

Solve for n'

$(p_a - p_b)\, M\, (n')^T = 0$

Let N be the transformation which gets applied to the normal of the generic primitive

Then

$(p_a - p_b)\, M\, (nN)^T = 0$

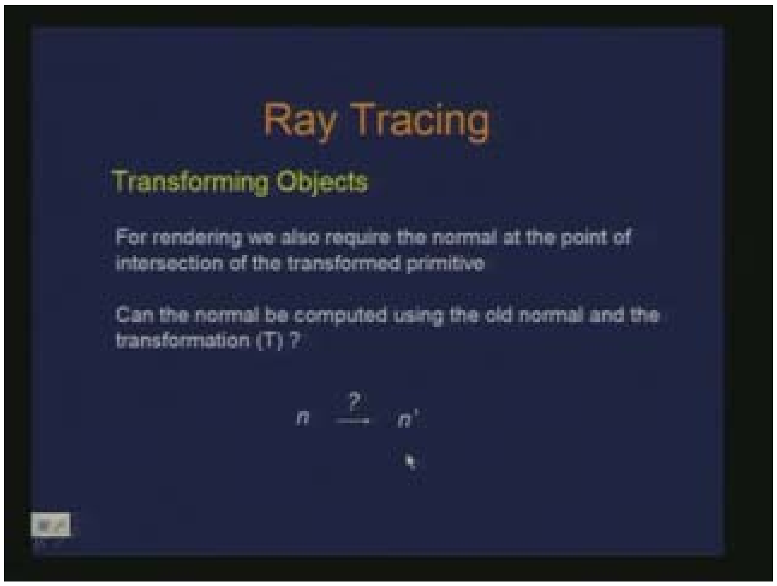
$(p_a - p_b)\, MN^T\, n^T = 0$

This holds true when $N^T = M^{-1}$ i.e. $N = (M^{-1})^T$

If it is orthogonal then it makes sense. If it has a rigid transformation rotation and then if you apply that transformation to the object then the same transformation would be applied.

In 2D this is some plane and the normal of this is this. Now I give some kind of a scaling in one direction so clearly it is not the same transformation which will get applied to the normal. If I do that the normal is going as something like this whereas now I have the normal as this. Therefore that is a powerful thing. Now you can handle transformation applied to the simple primitives. Therefore you can construct much more interesting scenes rather than just having these unit spheres and planes and so on and so forth.

(Refer Slide Time: 00:43:00)



Ray Tracing

Transforming Objects

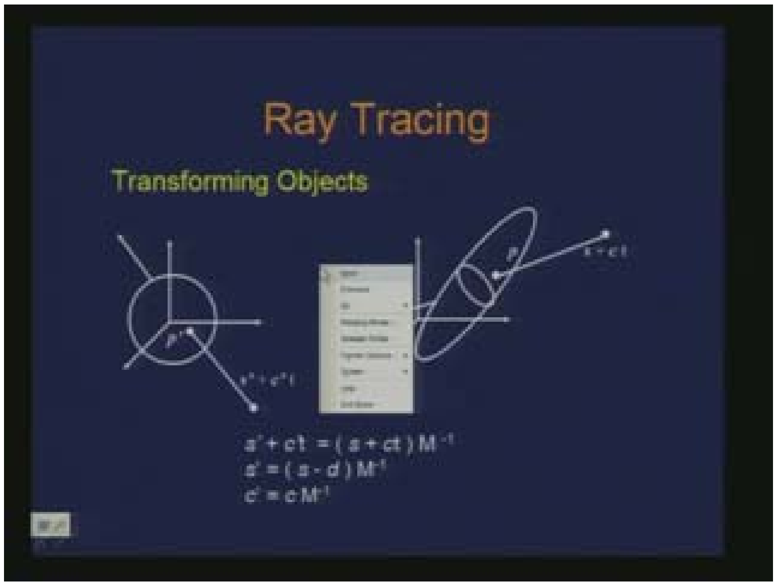
For rendering we also require the normal at the point of intersection of the transformed primitive

Can the normal be computed using the old normal and the transformation (T) ?

$$n \xrightarrow{?} n'$$

This is just a matter of defining a tangent like difference of two points close enough. It is a tangent vector. Therefore it will have two components where one will be in this direction and another will be in the other direction as two tangents and the orthogonality holds with both the normal. Whatever vector I have in that tangent plane the orthonormality holds therefore the dot product is 0.

(Refer Slide Time: 00:43:12)



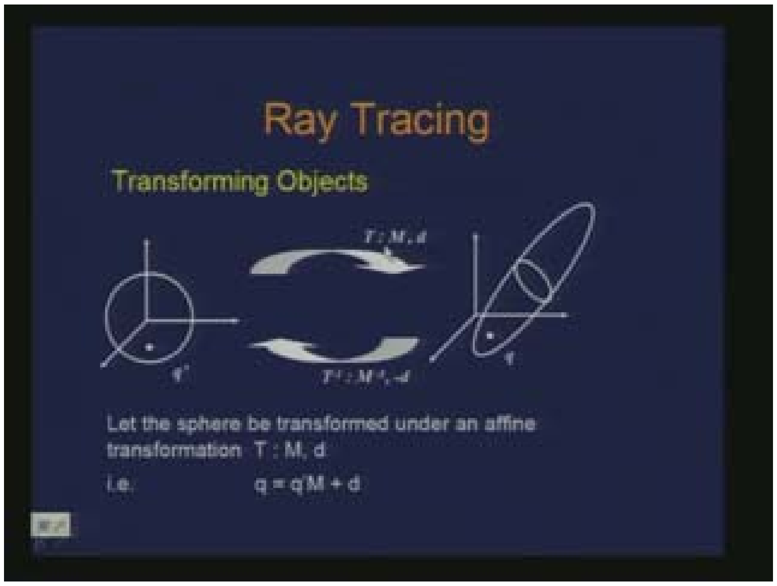
Ray Tracing

Transforming Objects

$s' + c't = (s + ct)M^{-1}$
$s' = (s - d)M^{-1}$
$c' = cM^{-1}$

The d is the offset. When we are dealing with the affine transformation if I define this t using this M matrix is the linear transformation part and d is the offset of the translation which I may apply. So that in combination gives me the affine transmission. If I am talking about 4 into 4 matrix in a 3D case then the top most 3 into 3 sub matrix is the same.

(Refer Slide Time: 00:43:38)



Ray Tracing

Transforming Objects

Let the sphere be transformed under an affine transformation T : M, d

i.e. $q = q'M + d$

When we deal with recursive ray tracing we would also like to consider the secondary rays namely rays due to reflection, rays due to refraction etc. Let us also see the effect of shadow rays when doing the computation of illumination. We will also see that if we detect that the point is actually on the shade what happens to the illumination or what kind of illumination you would use.