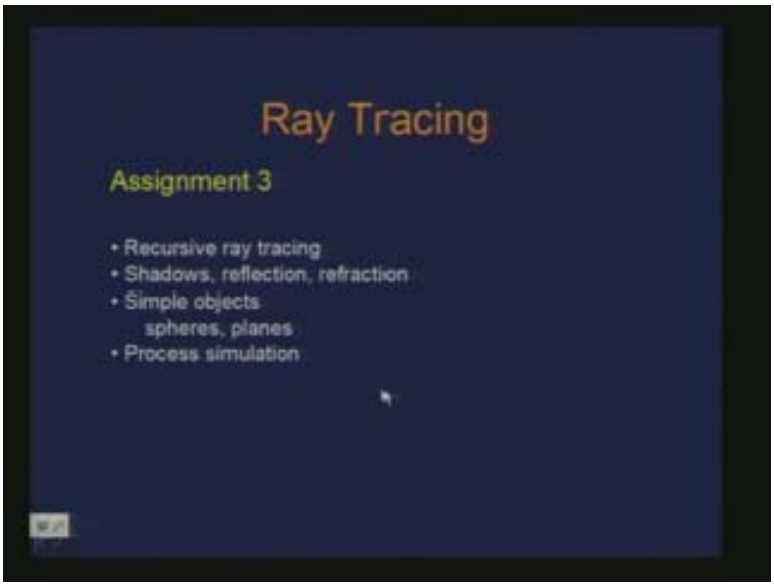
Introduction to Computer Graphics
Dr. Prem Kalra
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 27
Ray Tracing
Assignment 3:

This assignment is basically going to be on ray tracing. So you have to write a program for recursive ray tracing. This is not just basic ray tracing of level one it is recursive which would include shadows, reflection and refraction. As far as the type of objects are concerned just consider a very simple primitive like spheres and planes. And one of the important components of this whole assignment is that you will have to simulate the process. In fact it is something very similar to the first assignment of implementing the viewing pipeline. In the viewing pipeline you simulated the process of viewing pipeline and found the various stages of viewing pipeline. Similarly here you will simulate the process of ray tracing.

(Refer Slide Time: 02:46)

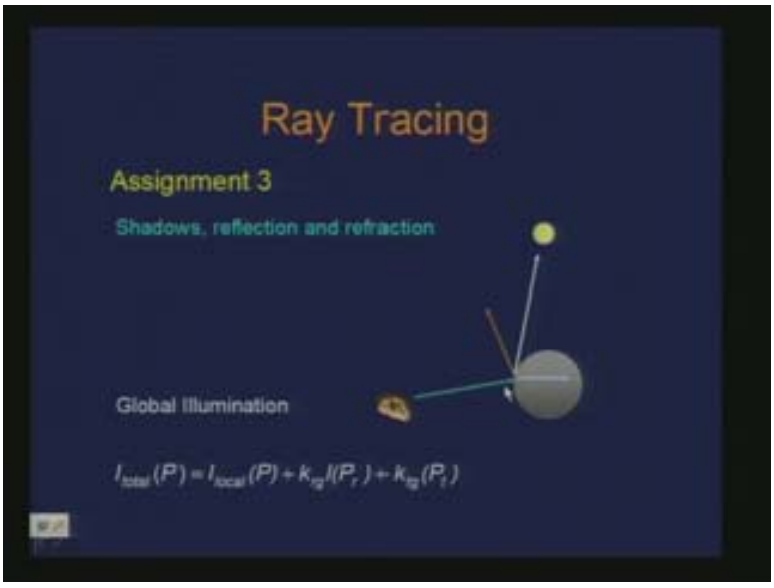


The first thing we need to do is implement the recursive ray tracing where the idea is that you have a specification of the eye, specification of the viewing plane and image plane and there is an entire scene. This scene would comprise simple objects with spheres and planes.

(Refer Slide Time: 04:27)

The idea would be that you will implement how to get the reflective ray, how to get the transmitted ray or the refracted ray and do this tracing till a specified depth of your recursion or when the ray escapes the scene that means it does not intersect with any other object so it is either of the two conditions. Hence this whole thing has to be implemented. So the data structure you would imply would facilitate you to form this tree structure of ray tracing. You also need to have shadows, you need to have reflections and you need to have refractions.

(Refer Slide Time: 06:07)



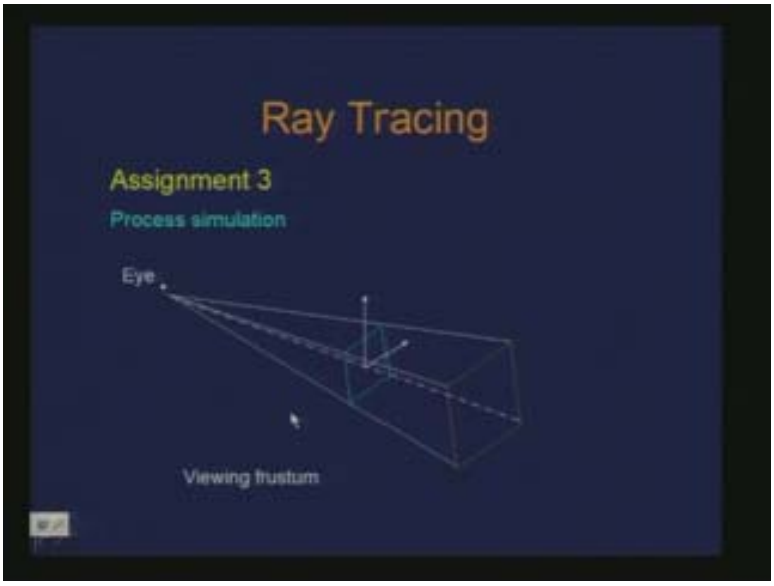
What we are looking at is that given a primary ray through a pixel (x, y) from eye what are the various other rays which may be emanated by the hit point. So there may be a transmitted ray, there may be a reflected ray or there may be a shadow ray so all these rays need to be computed. And then as far as the global illumination or the illumination at this point is concerned you would use this global illumination which would account for the local illumination computed at this point through the various light sources which are there and also account for the subsequent reflective and transmitted rays in any kind of illumination you obtain as a consequence of recursion. So you will have to do a global illumination computation to get the intensity of the color at this point. But as far as the objects are concerned you basically consider spheres and planes.

You need to write the intersection routine for sphere and plane. It is a finite plane and there are extends of the plane so it would also require you to do a containment test. First you will do a test for the point on the plane and then figure out whether that point is contained within the extends of the plane or not.

(Refer Slide Time: 07:18)

Therefore as far as the object ray intersection is concerned you are basically asked to have the object with sphere and plane. Let us look at process simulation.

(Refer Slide Time: 09:12)



If you look at the way the viewing frustum gets formed it is basically with respect to the eye or the center of projection on the camera which you place and there you have this image or the viewing plane. The rays start from here and they go through each pixel you define and then the intersection is performed in the scene. Therefore basically a viewing frustum gets formed with the window that is specified and the eye is specified.

What we are trying to look at as a process of simulation is that such a viewing frustum is now in the world coordinated system because you are going to perform the intersection in the world coordinate system is going to be slightly different from the viewing pipeline. So you would show this kind of frustum with this viewing plane or which tells you that how many pixels you have considered for the window, the location of the eye etc. Then the idea is that some sort of a probing will be done with respect to this pixel or image plane and you have to give the visual display of the trace of all the rays from that pixel. This is what is meant by process simulation. You have this eye, you have specified the extent of the window also which basically is in terms of how many pixels you have horizontally or how many pixels you have vertically.

(Refer Slide Time: 17:47)

And with respect to each of these pixels you figure out the appropriate coordinates u i v j take them into the world where you have the display, the eye is also in the world and then you show this primary ray which is going from the eye to that pixel to the scene and then figure out from the point of intersection the closest point of intersection which is what you display there. Hence these are some of the other objects present in the scene and there could also be some light source.

For instance, you can show light source with some yellow sphere or something like that. Therefore basically we are trying to see what happens to the ray which goes from this pixel as a visual demonstration. As an outside viewer I will first see that the ray starts from the eye and goes through the [ra….] pixel and this is the point of intersection I obtain and then further there may be a ray which is a reflected ray which goes and hits this object and another reflected ray which goes from here and so on. So in the process you have showed all these rays which you obtained. This is known as simulating the process. In some sense it is a visual debugging because you will see exactly what is happening to your scene.

So you have to specify of course the material properties and the color attributes for each of the objects you define. So, in addition to the geometric specification of the object you also need to specify the material properties of the object which includes the specification of the diffuse coefficient, specular coefficient, refractive index and so on. Therefore these material properties also need to be defined. So the idea would be that basically some probing is done. Therefore when I want to know what happens to the ray with respect to the pixels at location (x, y) you will show the entire process to the ray in 3D again by using OpenGL.

Basically we compute the illumination with respect to the light sources we define at this point which is hit. That is the local illumination component and the same thing happens

here. So whatever are the hit points with respect to the secondary rays we compute the illumination and do the attenuation using the reflective coefficients and the transmitted coefficients. So there is a local illumination computation which is directly from the light sources you have and these are the accumulated terms with respect to the secondary rays the reflected ray and the transmitted ray.

This is the primary ray you have and this is the first reflected ray, this is the second reflected ray and so on so this is the order you have. And when you form the tree, this is the root node which corresponds to the eye ray which hits this object C and you have this reflected ray and this transmitted ray. This is the next level of recursion and then I go to a further level down and I accumulate the intensities. And every time when I come from here to here the intensity which was aggregated will be attenuated by the reflective coefficients multiplied by K_{rg}.

Again in this process of simulating, when you probe for any pixel one should be able to see the process of tracing the rays including the secondary rays. Now what would happen in the 3D environment? Now at that point on the sphere on which the reflected ray from the plane comes there might be a direct ray from the eye to that point via some other pixel. So that will be the primary ray or the eye ray with respect to this point when I have to render that pixel. But here I am accounting for what it would have reflected to this.

At a time you show the path for one pixel. Then you are also looking at what is happening with respect to the light sources to decide whether that point gets into the shadow or not and so on. So there is a window which is a 3D window which we call as simulation window where this interactive process is going on and where one can probe for any pixel (i, j) or (x, y) as to what is happening. This is your simulation window. You also have a ray trace image the final image which you would obtain in another window.

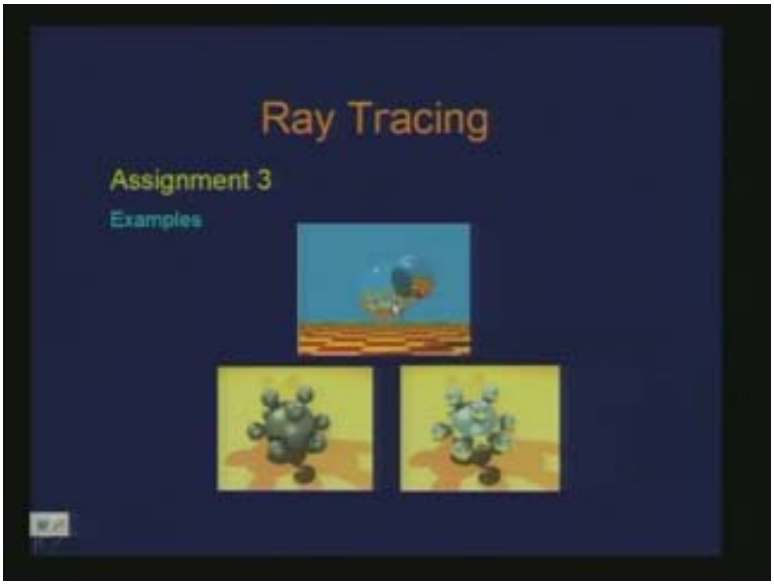
So this is your simulation window where you can see this whole process and there is also a 2D window which just gives you the ray traced image. In fact you can do an interactive building up of this image also. So you scan with respect to all these pixels to find out what is happening and render that point. This complete image is what you would obtain as the final intensity of that pixel. This is the complete pipeline. This is eye, there is a view plane, there is a scene and the rays start from the eye goes to the viewing plane and it sees what happens with respect to the scene. I am able to trace the ray wherever it goes and what all objects it intercepts.

(Refer Slide Time: 22:22)

It definitely corresponds to a point on the ray traced image and that is why I am going to find a ray gender. In order to be able to show clearly on the screen when you display a particular pixel the point on that viewing plane may not capture the true coloration of the pixel. So the best thing is that you have this simulation window which just indicates a point on the viewing plane which of course corresponds to the pixel on the ray traced image which you render. This illumination term is nothing but a tuple. If I am talking about a color R_{bg} then it is an intensity defined as a vector R_{gb}.

I can do some sort of a scanning of the entire viewing plane and simultaneously display what is happening as a result on that scan to this image. So this is a complete interactive process of generating a ray traced image. When I said that the scene should comprise of the object of the kinds sphere and plane it does not mean one sphere and one plane it means that the scene is consisting of several spheres and several planes. These planes could actually help you to develop the scene having these spheres so that we have walls, ceiling and things like that.

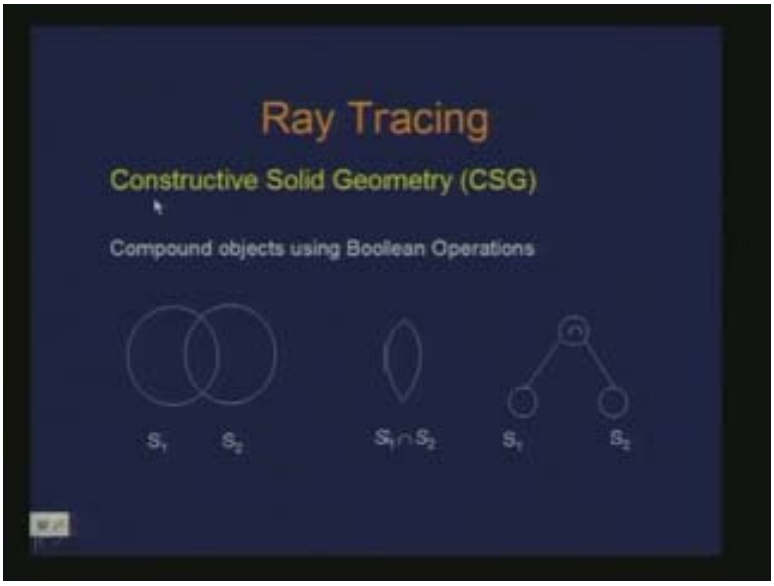
(Refer Slide Time: 24:16)



You can also set other parameters namely input parameters like depth of recursion. So this is a one level ray tracing where I do not need any reflection then here I have reflection and so on. So this could also be given as an input parameter. For instance, you can generate the scene with objects which are transformed from sphere and so on then you need to do transformation of the ray. But more emphasis on this assignment is towards the building up of this simulation so you can see the process.

You can even try to see it as an animation that the ray starts from the eye goes to the viewing plane, goes to the other object, goes to the other object and so on. The objects which you display in the simulation window would be through OpenGL. So simulation window is nothing but OpenGL window basically. Everything is happening using OpenGL. There are also certain other small issues related to ray tracing. We discussed about constructive solid geometry CSG when we talked about solid modeling.

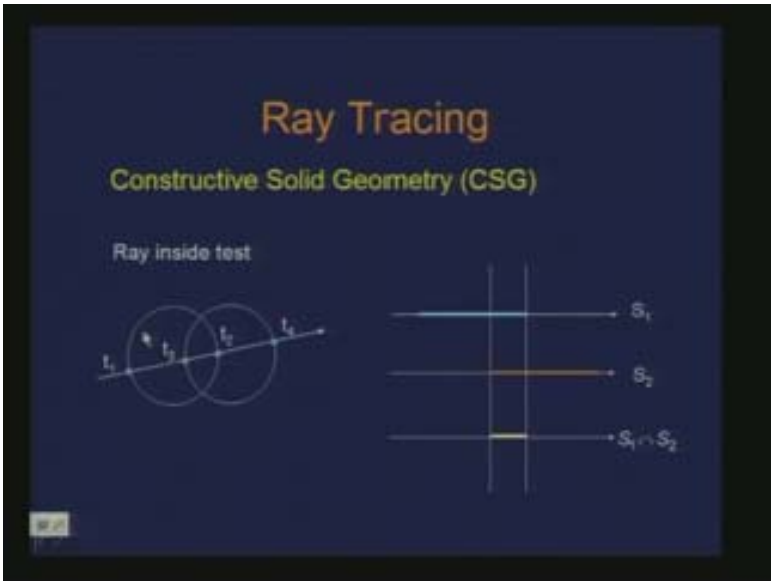
(Refer Slide Time: 29:07)



In fact we can do the ray tracing of these objects also fairly easily. What is this constructive solid geometric? It is basically building up of compound objects using Boolean objects. For instance, there is a surface of object S_1 then another object S_2 and I am interested in getting an object of this kind which is a lens kind of an object then all I need to do is perform a Boolean operation of the type as an intersection S_2 to get this. So as a representation I can see this in the form of a tree where I have the leaves here as the surfaces, the primitives which are being used for forming the compound object and then the node above is the operation which I am performing into these leaves. This is the intersection here. And in fact you can think of also using a transformation T_1 and T_2 which is which could be associated to these objects S_1 and S_2.

Now, given that you are able to do this construction of compound objects using simple primitives and these are the primitives with which you do the ray intersection then if you are attempting to do the model of this which is a small intersection S_2 then how ray trace can be used? If you look at what we try to observe is the ray inside test.

(Refer Slide Time: 32:28)



When I was talking about two surfaces S_1 and S_2 if you look at the ray inside test that means what is the span of parameter t for which the ray is inside the object. So, with respect to this there is an intersection at T_1 and T_2 and with respect to this there is an intersection at T_3 and T_4 and these define the ray inside with respect to the each of these objects. And if one is interested in looking at what happens after you apply a Boolean operation to the ranges I test then it turns out that it is merely a Boolean operation applied on these spans of (t, s). So, for instance, if I look at this as a t axis for S_1 this is the span from T_1 to T_2 which is displayed here in blue.

For S_2 I have from here to here. Now the ray inside test for the object which is obtained as a Boolean operation of intersection is nothing but the intersection of the spans here and here which is this yellow line. So this gives me the ray inside for the compound object and that is what I should be interested in where I am performing ray tracing. One of them would give me the nearest point and therefore I can display it. And these are nothing but the regular intersections with the simple primitives. If I was to find out the ray inside test for the union then all I need to do is take the union of this and this which would give me T_1 and T_4. Similarly, if I am performing an operation of difference S_1 minus S_2 then the same operation can be applied onto the t spans. So, constructive solid geometry can be easily used in the context of ray tracing without doing anything, it is just there.

(Refer Slide Time: 35:28)

So, explicit intersection you are determining only with respect to the simple objects and then it is just a matter of using an appropriate data structure to be able to decide the point of intersection for the compound object. The other feature is texture mapping. Texture mapping itself is a subject not necessarily related just to ray tracing.

Texture mapping is something like when you observe a scene rendered in this fashion where you have objects here and these are the walls so if you just supply the material properties of these objects which are defined in terms of planes and boxes you can generate such a scene.

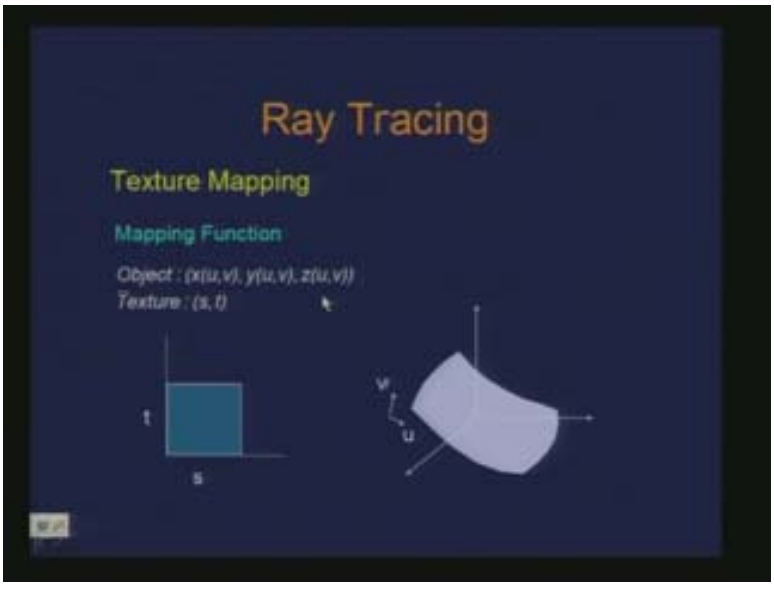
However, if you know that this wall is actually made up of bricks and this material is something which is more like a thermocol then how do you add those details on it which you see here on the right side? Here you see some patterns, here the bricks are shown where you see some kind of a texture on this material and there is some pattern on this, there is a pattern here on the floor etc. The question is, how do we generate such effects?

Since it is a part of modeling when you construct a wall it is made up of bricks which is an element of modeling. Therefore why not do that as a process of modeling? For example, if am interested in getting the surface details like bricks one option could be that I do a detailed geometric modeling of the wall wherein I can define the entire variations and details of these bricks. So, obviously that is a very expensive task and we will require lots of storage. And it may be unnecessary from the point of view of what you want to generate as a visual effect.

(Refer Slide Time: 36:53)

Another way to do is use combination of geometry which is the definition of walls as planes and texture mapping which adds details to this geometry. In a crude form texture mapping is nothing but like a wallpaper wrapping around onto the object and wallpaper is nothing but some sort of an image or pattern which you paste on top of the object. Therefore how do we perform this texture mapping? What are the requirements? So, we have an object which could be defined as a parametric object (x(u, v), y(u, v), z(u, v)) where (u, v) are the parametric domain or the parameters on which the object is defined.

(Refer Slide Time: 39:49)



A texture is nothing but another definition on parameter (s, t) which could be an image or which could be some procedurally defined texture. So this is a two dimensional image through the parameters (s, t). So (s, t) could just be the pixel definitions of the image or the pattern you want to paste on. And the object could be defined in this fashion where you have an object here through the parameter u and v just like your parametric surfaces or your sphere or any other object.

What is it that we are trying to do through texture mapping is to associate a point on this surface the object with the image in texture. Ultimately I want to put this pattern on the top of this. So what I want to do is associate a point here to a point here. In other words I need to have what we call as the texture coordinates (s, t) added to the coordinate of this point or the surface. Hence that is what we basically mean by texture mapping.

In other words, we are trying to look at a mapping function which takes you or which helps you defining the association of u with (s, t) and similarly the association of v with (s, t) through functions (f, g). It is a function mapping of (u, v) and (s, t) and vice versa where you associate (s, t) with (u, v) through functions h and I. So a simple way of looking at is, I have a linear mapping that means u is related to (s, t) through a linear function where I define u as a linear function of s where u is equal to As plus B and v is equal to Ct plus D then I can associate the parameters of the surface (u, v) with the textures. Once I associate this parameter I can associate the 3D point on the surface.

(Refer Slide Time: 41:56)

Basically this is a matter of defining these mapping functions with the parameters of the surface to the texture. So if you want to see it in a little more illustrative form what you are saying is that there is a surface patch or some other parametric surface defined through (u, v) and there is this pattern here the texture (s, t) and I want to take this and put it here which undergoes the projection transformation because ultimately what I am going to have is what I see as the object on the screen so this object which is a 3D object gets

further transformed into a screen coordinate.. We refer to this as forward mapping. I try to associate the object with the texture and then perform the projection transformation to get what happens to the object when we see it on the screen.

(Refer Slide Time: 43:15)

Now there could be another way to look at the process where I see it just in the inverse order. That is, I have the screen, I have a pixel here and I try to look at the pre image of this pixel onto the texture so it is an inverse mapping. This is what you have in ray tracing.

(Refer Slide Time: 48:45)

You start from a pixel and you have the mapping between the object and the texture then you can associate the object to the texture and find out where this pixel would map to the texture. So the issue is what do we do with this? We have established the correspondence as to what this pixel or the point on the surface has got with the texture or a point on the texture, so what do we do with it? Eventually we have to render this pixel and I know that this corresponds to this texture.

In some sense I have to incorporate into the illumination or the color of the pixel. So in the context of ray tracing where you are using the illumination model using diffuse reflection and other terms the material of the object is defined through the diffuse coefficient kd and that is what captures the color of the object. So what you need to do is modulate or replace that kd through this texture information because now the color attributes will be taken from the texture so your kd terms will be modified as we want to capture whatever information is available in the image of the texture to render this pixel. And the mechanism I have is through the material definition of the object which is through kd the refractive coefficient so you can add texture mapping in your ray tracing.

There are certain issues here. If this was a mapping from one pixel to one point in the texture there is no problem. But the small pixel may actually get mapped to a larger portion in the texture. This would in turn result in aliasing. So, what one may try to do is take this information and take an average of the information which is there in this texture and put it there because if you just map to the centroid of this or to one corner of the texture then it may give you artifact. Ideally what we are saying is that this whole area is actually getting mapped to this pixel so I should have a measure of averaging this information onto this entire area and put it here. Therefore in that way I can avoid aliasing or reduce aliasing. But if you are willing to implement texture mapping in your ray tracing do not worry about aliasing. That means you can even consider one of the points here like take the centroid of the area.

Typically if you look at the examples where one uses these textures or patterns of this kind where you have bricks or skin or may be some other pattern pasted on the entire plane or object it requires repeat of texture. And even if you are starting with a small pattern of this kind it requires repeat of texture. In some sense you would require some kind of a tiling. So the issue which is concerning here is, when you do tiling there is a possibility of getting the seams discontinuity where you do the tiling.

(Refer Slide Time: 51:05)

So how do you resolve that? Seaming means if I take this put the same pattern adjoining this pattern and so on just to build a larger texture. That is what I mean by tiling. So clearly at the boundaries I will see a discontinuity, it is not going to have a smooth joining. There are issues related to that. There are ways by which you can almost eliminate this artifact if not fully. So, given a small texture image you can actually enlarge to whatever size and scale you want without having this effect of discontinuous boundaries. Now there is another issue. What we discussed about is taking a texture image and put it on the top of the object just wrap it around. That may not be desirable a thing to do always.

Therefore here is an example of a little hippo. This is shown in somewhat wireframe kind of a display. This is a shaded display and this is a texture mapped display. The issue here is that, if you are given an image of a hippo and you have this model also, remember this image is a 2D image, you wrap it around and when you do this wrap it around you need to establish a correspondence of the features you want to have association in the model in the texture. Otherwise the texture of the nose will go to the eyes in the model but we do not want that. Therefore this is an issue, it is just not taking the image and wrap it around the object but you need to do more than that, you need to establish the correspondence of features, the eye should go to the eye, the nose should go to the nose and so on.

(Refer Slide Time: 52:33)

There are also other issues. We have talked about parametric definition of surface of the object where you could define the object in terms of (u, v) and you have a mechanism of associating this (u, v) to (s, t) on the texture.

(Refer Slide Time: 54:23)

But what happens if you do not have the definition in (u, v)? If you just have a polygonal mesh how do you do the texture mapping for those objects? So there has to be a way of either defining those parametric representations of these objects and then do the association of the parametric to the texture. All you need to do is you go back and look

for the color or the attribute in the texture image with respect to the pixel you want to render.