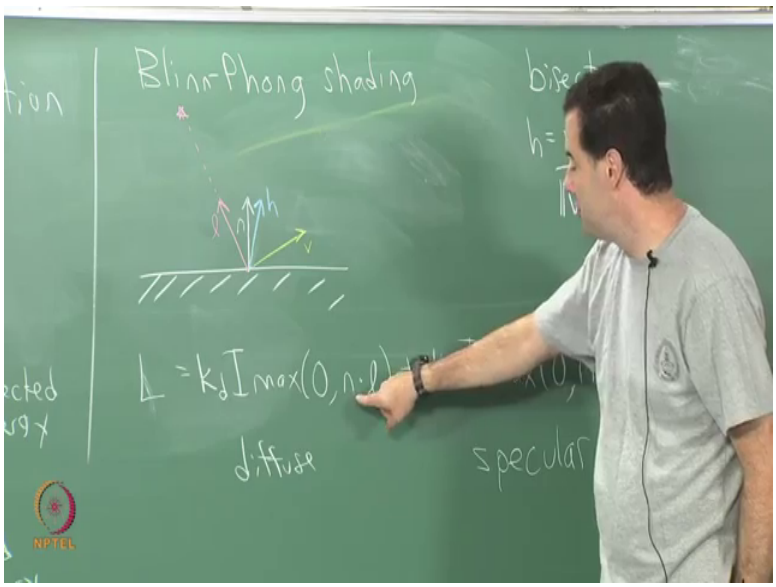
Virtual Reality
Prof. Steve Lavalle
Department of Multidisciplinary
Indian Institute of Technology, Madras

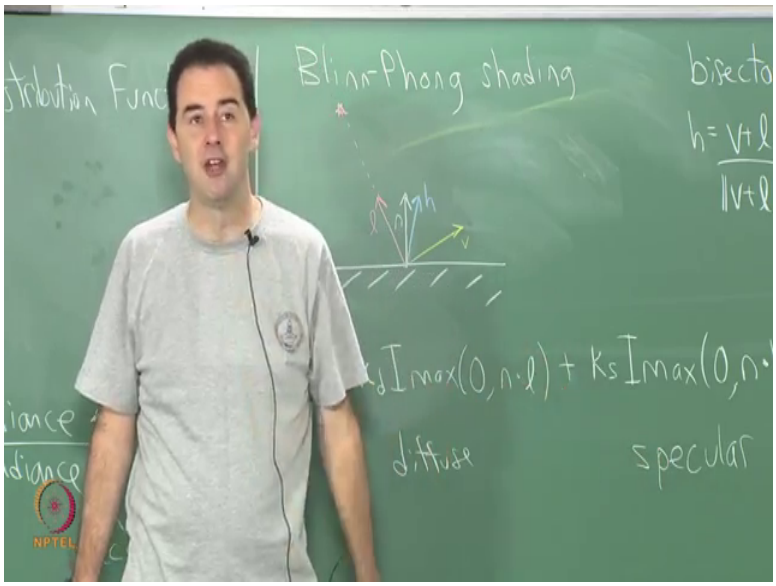
Lecture - 15-2
Visual Rendering (rasterization)

(Refer Slide Time: 00:16)



So, in the case of a Lambertian model, the B R D F is constant, it turns out. So, the only part that is really left is. If this does not look constant, here, it still depends when I put it in this expression. It depends on still the angle with that, the light source makes with respect to the surface, but again, as I said that is, that will decrease the amount of energy coming in and this ratio is based on comparing the amount coming in to the amount going out. So, if I make this constant, then I get a Lambertian model. So, that is one special case of this. So, that gives you the easiest case and then, you can handle all the other cases through various complicated expressions. Here, you can go off and research that on your own if you like.

(Refer Slide Time: 00:39)



But it is going to give you the sense that you know, people do these simple kinds of hacks to generate and render scenes very quickly, but then, they also know how to do it in the complex way, but there is a lot of research going on, in trying to make this more efficient and, it is quite challenging. Especially when we get to the problems of virtual reality, when we want even higher resolution right and faster frame rates, we want higher quality. So, it is going to push us more towards making some kind of hacks, but the hacks end up looking worse in virtual reality because we can move our heads around, we can see a lot more than we could on a simple screen and that some of the details I will get to.

One of the complications that V R causes right away for these kinds of problems is that, the viewing direction right I get one viewing direction on a screen, but if I have stereo rendering going on, I have two different viewing directions, right? So, that is one simple complication that happens right away. One clear complication that happens right away is that, I have two different viewing directions to take into account here each time. So, that is already at least double the amount of work or. So, it seems right? That makes sense.

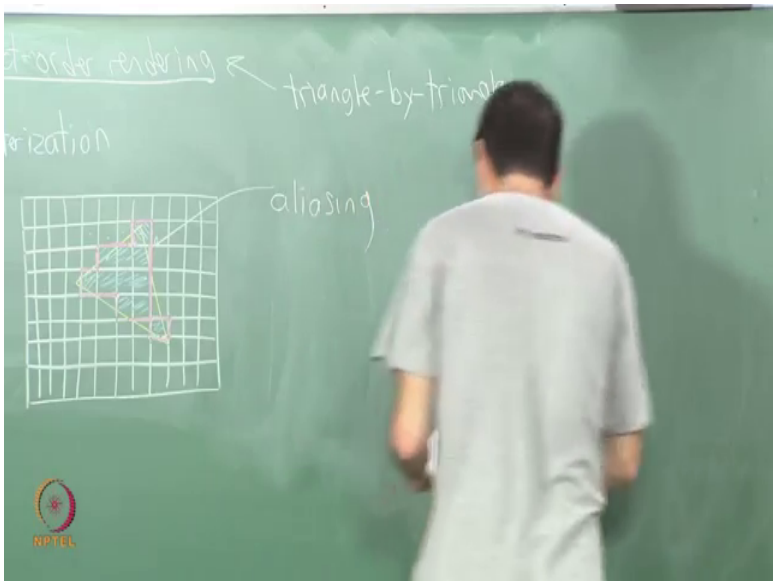
So, stereo just because I am rendering to each eye had different viewpoints for each eye, I told the mathematics of it when I gave the geometry of the transforms that you just perform a simple shift, but already you have to do more than a simple shift here. You have to calculate. Some difference reflectance, if it is diffuse it might not matter, but if

you have a specular component right, if it starts looking very close to a mirror and for that to look right, you will have to very carefully take into account each one of the eyes. The left and right eye should see a different light, should receive different light. This seems correct right?

Questions I want to now move over to the. The next part, which is the object oriented rendering object order, rendering I keep saying object oriented because of C plus plus and other object oriented programming models and things. So, I do not mean that an object order rendering which again is going to be looping over the triangles, first, for us I am still going to have to take into account shading somewhere in this pipeline of processing. So, these concepts still are relevant. I am not going to cheat our way out of this.

So, we are up to object order rendering and if you have had computer graphics background before, if you had a course on it before, try to think about what is unique to virtual reality as we go through these things, Right as we are reviewing some of this material, all right?

(Refer Slide Time: 03:34)



So, in this case, the most important step is rasterization and this is you know, object oriented rendering and rasterization is the most common method in GPU's. So, this is mainly what is going on in hardware. This is the approach that has been widely used in industry, not the ray tracing methods and there is a lot of complications to what is going

on here, but by being object order rendering under, that I said it is going to be triangle by triangle. Let me draw part of the screen here, ok? This is getting tedious, good enough. So, I have some kind of low resolution image here, right and suppose this is what I am going to ultimately render to my display.

This is how the display will look and, we are going triangle by triangle and you might recall the transformations that we did quite a few lectures ago, where we talked about if I define a triangle in the body frame, I move it into the world frame and then, I perform more transformations to figure out exactly where it ends up in terms of pixel coordinates. Well if I do all of that, then I end up with a triangle here somewhere. So, my triangle will end up somewhere with pixel coordinates and as I said when I covered that topic, these pixel coordinates do not need to line up exactly with the integer or integral values of the pixels, right?.

They do not necessarily end up being integers. Just some floating-point numbers and now, we end up with a triangle here and we have to figure out. Now, in the rasterization process, which one of these pixels to color based on the light hitting that triangle, right that makes sense? How do we decide that? Well one simple way to do it and one very common way to do it is to just take a pole at the center of each one of these pixels and then, color in the pixel based on the shading models, whichever you decide to use, if the center ends up falling inside of the triangle. So, you have a inside - outside test that you need to implement to determine whether or not.

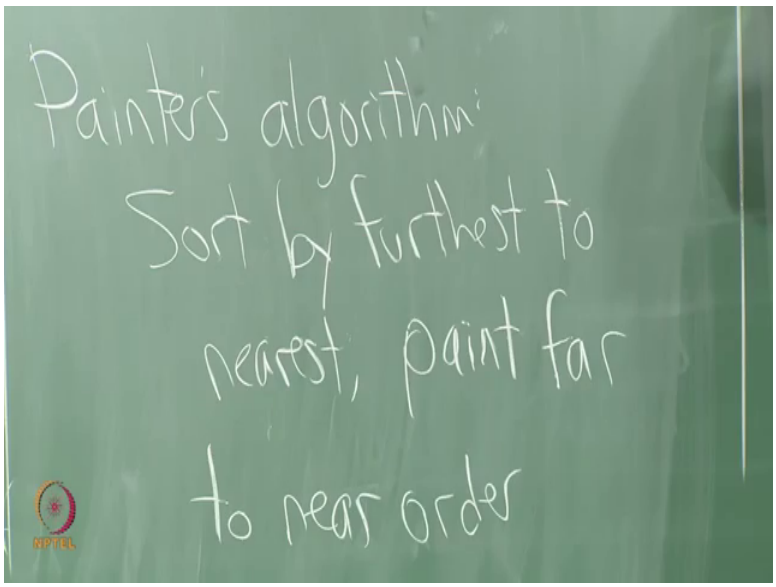
The point a sample point is inside or outside of the triangle and then, you go along and shade it in. I guess we could try this here. So, if I go stepping along, this is outside. This is a bad example maybe. So, maybe this is just barely inside. So, I decided to color that and I will just pick a color here and color it in and, see outside coming along here, ok? This is on the border, you have to decide what to do in those cases and so, you are right. On the border, here I will say these are all inside. So, I color these and coming along again, let us see it looks like, this would be inside. This is, in these are all inside, that is outside. See, this is inside. This one looks like it is inside.

So, it looks like, try to draw a nice outline of this. So, it looks like the rasterized version of this triangle from my somewhat haphazard example, looks like that right. Do they even look like a triangle anymore? Not too much. So, with very - very poor, a low level

of resolution, if that is a very small triangle, maybe they; maybe have a high-resolution display, you end up with a significant amount of distortion of the triangle. This problem is generally referred to as aliasing when you end up with significant distortion, of the, say the geometry of the triangle, we will talk about that as we as we go along, I will talk about anti-aliasing methods. So, if we are going triangle by triangle of course, you probably do not want to scan the entire image to figure out where one tiny triangle has landed.

So, you can develop very simple tests to eliminate most of this image, when performing these samples in the center of each one of these pixels to determine if it is inside or outside of the triangle. So, you can do some simple tests to call away, let us say, large portions of the of this image that you're making., the next problem outside of that after determining, where the triangle falls in terms of pixels.

(Refer Slide Time: 09:13)

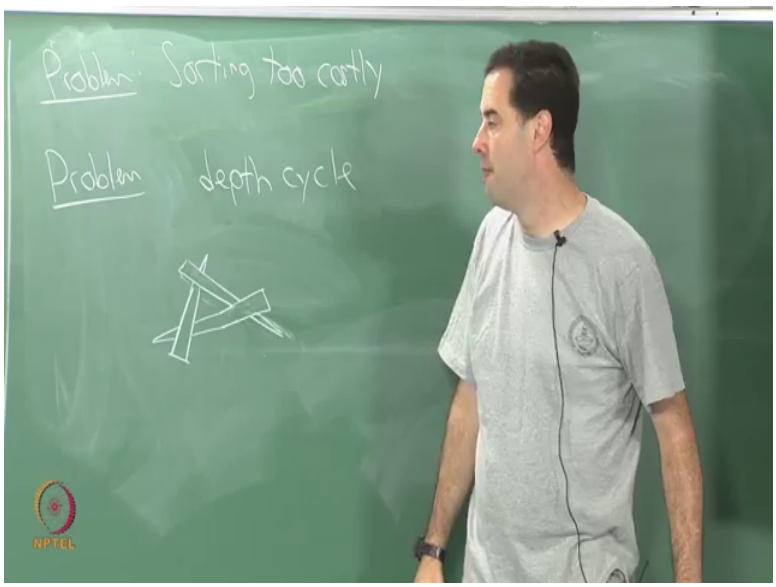


The next problem we have is called the depth ordering problem. In other words, on which triangle is in front, we have the same thing when we talked about ray tracing. I mention that you extend array and you have to figure out which triangle is hit first. So, the problem does not go away. You have to figure out, essentially the same thing here, if there is a kind of virtual camera looking at this image, I need to figure out which triangle do I see first and, especially I would. Let us say, at a particular pixel is this triangle in

front or is there some other triangle that would be in front of this in my scene after I have placed all the triangles into the appropriate place here.

Well, this might seem easy enough. I can apply what is called the painter's algorithm, which is just sort the triangles by furthest to nearest. That is with respect to the eye that is looking at the triangles, right? So, from furthest to nearest and then, paint or in other words, do the rendering, it is called the painter's algorithm. I will say paint, then paint in far to near order all right. So, that sounds really easy and then, I can just quickly loop over all of my triangles and if I have them ordered nicely, I start with the ones that are furthest away and if I start rendering the ones that are closer, later the whole thing should work that makes sense. So, that I just destroy whatever I have rendered before and I do not worry about it, this makes a very simple loop for going through and rendering everything correctly - two problems with that.

(Refer Slide Time: 11:21)



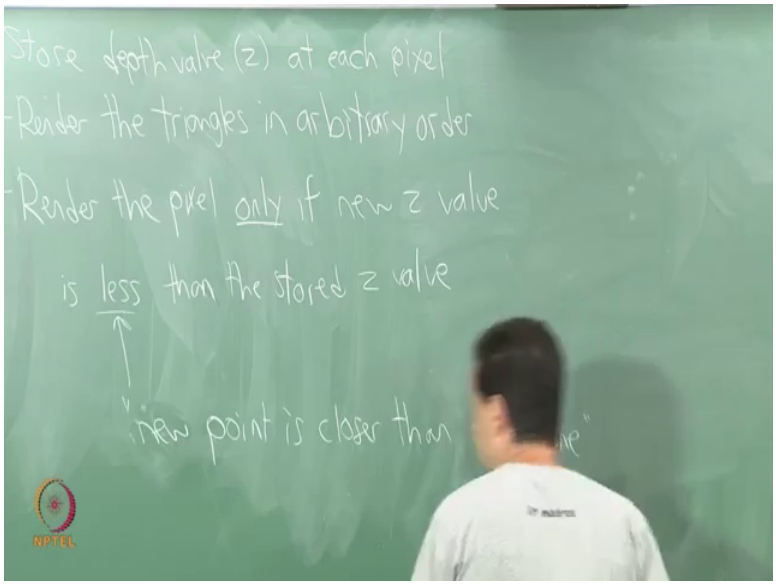
One problem is that the sorting is too costly. If you have millions of triangles you pay all of N log in time for the best sorting algorithms, you end up with. Then too much time spent on sorting. Another problem is what is called a depth cycle. See if I can, if I can draw these correctly here, trying to draw a triangle. So, let us see I have this triangle appears to be in front of that triangle, but then, I make let us see another part here, that is in front of this one. I will erase that. Continue this one on or to make it look. I have fixed

a bit here and now, I think I would like to make this part of the triangle look like it is behind this, is that ok? So, which triangles in front right. So,.

So, in three dimensions, just pointing this does not happen in two dimensions. If you just arrange some line segments, but in three dimensions, there is no well defined notion of in front in some cases, right? So, you cannot simply sort from far to near. You have to take into account these in some special way by breaking them into pieces. So, it is still not a clean solution. So, how is this problem solved in rendering what do people normally use in graphics, because of what is used in a GPU, anybody know?

There is no graphics gurus here z buffers right. So, z buffers end up getting used. So, I will explain that briefly as the main technique and then, we will start to talk about. I will talk exactly about how the, how the pixels are getting shaded in these buffers and then, we will start to talk about the virtual reality problems, but probably the after the break into the next lecture for that, but till we get to that part, ok? So, we are going to give up on the painter's algorithm. For a bit it can be used as I said, if you take into account depth cycles in some special way and if you are not afraid of the sorting cost.

(Refer Slide Time: 14:13).



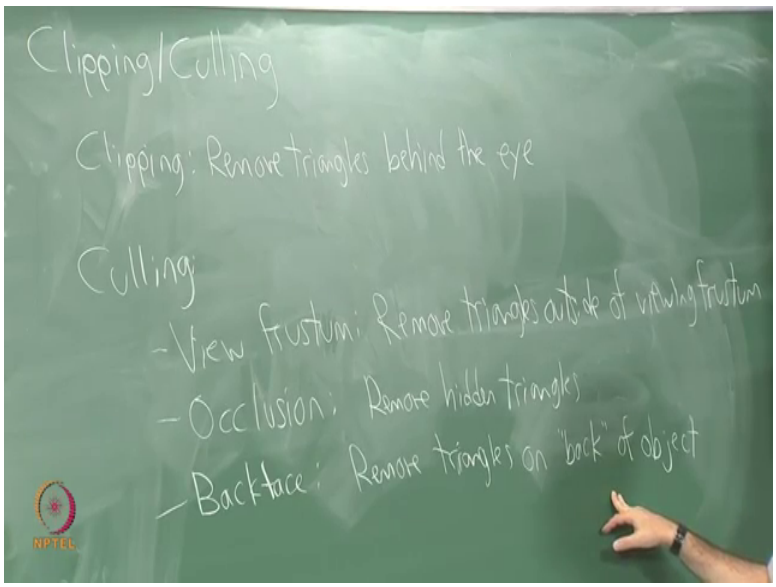
The z buffer and remember that in computer graphics, the z axis has been chosen. So, that corresponds to depth right, which is distance from the viewing direction. So, you could also call this a depth buffer, if you want it is also fine, just the z coordinate is what people always end up using. So, they just identify that with depth and so, what you do in

this case is, you store a depth value, all right? So, some z value at each pixel. So, any image you are making, imagine you decide you rendered a triangle, you have colored the pixels with RGB values, which I have not exactly said, how to do yet, but you do that, but you also store the depth for exactly that pixel, all right? So, you go ahead and do that.

Now, you can render the triangles in arbitrary order. In other words, in unsorted order. Any way, you like and then, you render the pixel. In other words, you assign the RGB values, only if the new z value. The new z value is, what the new z value should correspond to. So, if I have already been drawing on the screen, I have been coloring in pixels. I now have a new triangle that I want to render. I will only render it if it is in front right, so; that means, that it is z value should be less depending on whether Z's increasing or decreasing.

Playing how we set things up, but only after z value is less then restored z value, as it less or greater to be careful there depending on how the coordinate systems are set up. It should correspond to being closer. In other words, the new point is closer, then the stored one. So, that is the important part, not necessarily less, but it actually corresponds to being closer to the virtual eye, right questions about that, ok? Let me mention just a little bit more and then, we can take a break.

(Refer Slide Time: 17:15).



So, I want to talk about clipping and calling. These are both very related and people use the terms sometimes interchangeably, sometimes quite distinctly, but basically it means

eliminating triangles from full consideration in this rendering pipeline here. So, rather than going all the way to the stage of calculating RGB values for every pixel, there may be a quick way to know that you do not need to look at them.

For, example this usually is called simple clipping, remove triangles that are behind the eye right. So, behind the viewpoint. So, there is a particular point at which you are viewing the scene. Everything that is behind why not just do a simple test and you know make a simple flag that just eliminates all those from consideration right. So, we can skip over all of those triangles, then end up being behind. So, that is very easy kind of clipping. I should point out something that is already interesting with respect to virtual reality.

Here, if I am looking at an image on a screen and it is not taking into account the viewpoint adjustments because of tracking our head then. This clipping, let us say, a clipping plane. Imagine a vertical clipping plane that eliminates everything behind me, that remains fixed right when you are just looking at a fixed screen. Now, what should I do in virtual reality? Where should I put the clipping plane? I could put it exactly where my eyes are and then, I could have it move with me. So, that clipping occurs. What if I put it hmm, but even if I do that, I think it means that if I have some geometry, some information, those are some walls or something in front of me, I should be able to put my face up very close to it, right?.

So, that in the virtual world it is only one centimeter in front of my eyes. What is going to happen in that case is, it going to be in focus if I have a head-mounted display and I put my face up to it, then, it is one centimeter away. Let us say, I do that in the real world. I put something up. Once a meter away from my eyes, it does not look like it is in focus to me and if I, if you did the same experiment yourself, my guess is it would not be in focus to you.

Can your eyes converge and give a stereo picture there? Cannot do that either, but in virtual reality, you can very easily render things all the way up to one centimeter away from your eyes. They will remain in focus because if you remember the optics of the screen on the lens in front of your eyes, it will remain in focus, but you cannot converge on it. So, it is a terribly uncomfortable situation. What do you do? Well you could move the clipping plane further away. So, that as soon as anything gets beat within, let us say,

be safe maybe 15 centimeters of your eyes, you just clip it. So, what is wrong with that solution?

Student: (Refer Time: 20:12) vanishes.

Not vanishes. So, now, there is a look too good. If I want to come up and put my face up to the board in VR, I have a virtual board here. The board just disappears, that does not look right either. So, what should we do? I do not have a good answer for them. I am just pointing this out right and if I turn my head at an angle and I start putting it up to the board like this, right? So,. So, so I am turning my head at an angle on it and I have come in like this, then, this eye is getting closer to the board, my right eye is getting closer to the board than my left eye so; that means, the clippings are not going to match exactly and I have to get things matching within stereo with my eyes right.

So, I will have mismatched clippings too to deal with. So, I may have some parts of the board that both eyes can see and in some parts of the board that have been eliminated for one of the eyes, but included for the other. So, so that is a mess right. So, there is already even with something as simple as clipping there is already a miss match problem and some kind of difficulty corresponding to virtual reality because of this vergence and accommodation conflict that we talked about.

And it is always in focus. The fact you can move your head what should happen, all of these come into play. Here, with regard to virtual reality. So, how do we deal with that? Not completely sure, but I am just letting you know this is a problem. So, more complicated kinds of culling operations which I will not cover in detail, but I want to point these out is that there is, the remaining part of the clipping is to call the way or clip away. Everything outside of the viewing frustum, you might remember the frustum from our transformations that we did.

So, remove triangles outside of our viewing frustum. So, that is one kind of calling that goes on. There is also occlusion calling this could be quite complex, but this is if you know that a bunch of triangles are behind some large object, then you can remove them. Now of course, you have to be very complicated. Visibility analysis and reasoning to accomplish that it might be too complex to do the calculations nevertheless in many settings, it is worth considering. So, remove what are called hidden.

And then, another one is back face culling, which in this case is remove triangles on the back of an object. Okay, that looks a lot like the occlusion case, but the back face in particular; the surface normal's are pointing the wrong way. So, they are pointing away from the light source. So, you know they are not going to contribute anything. So, remove these and of course, if you have a problem with your model, which is very common, especially if this model was constructed in some kind of automatic way., using slam for example,, you may have some inconsistencies.

You may have some of the triangles with the normal's pointing in the wrong direction, accidentally and then,, you have to be very careful. You will get these operations wrong. So, remove these. So, these are just several operations that happen, to try to reduce the number of triangles in each step and there is a lot of important work, that is going on. Here, a lot of experimentation, a lot of highly optimized efficient algorithms that get implemented in the graphical processing units in hardware, to do all of this efficiently. So, that gives you the idea of this graphical rendering pipeline, which has been very well optimized for computer graphics on a screen, but not so.

Well optimized for virtual reality and so, we have to go back and rethink about all these things that happen and are they appropriate for us in the virtual reality context. You can use them because they exist right there. You can buy a graphics card, plug it in and start using these operations, but are they right for virtual reality? Some yes, some no. One final thing I should say about z buffers and I will take a break, is that z buffers are also very useful for rendering shadows.

If you want you can move the viewpoint instead of being the viewpoint of the observer, you can make the viewpoint. Be a light source and then, you can take a look at the ordering of objects with respect to the light source and calculate where shadows will fall. So, z buffers are also useful for calculating shadows. I just wanted to point that out, as another side benefit of using z buffers questions on this.