Computer Graphics

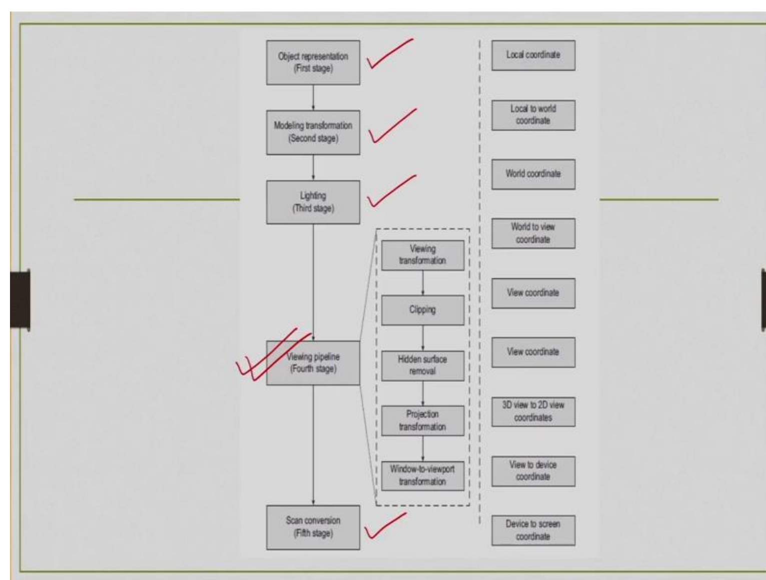
Professor Dr. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture - 23

Hidden Surface Removal - 1

Hello and welcome to lecture number 23 in the course Computer Graphics. So we are currently discussing the graphics pipeline and there are five stages in the pipeline as we all know.

(Refer Slide Time: 00:46)

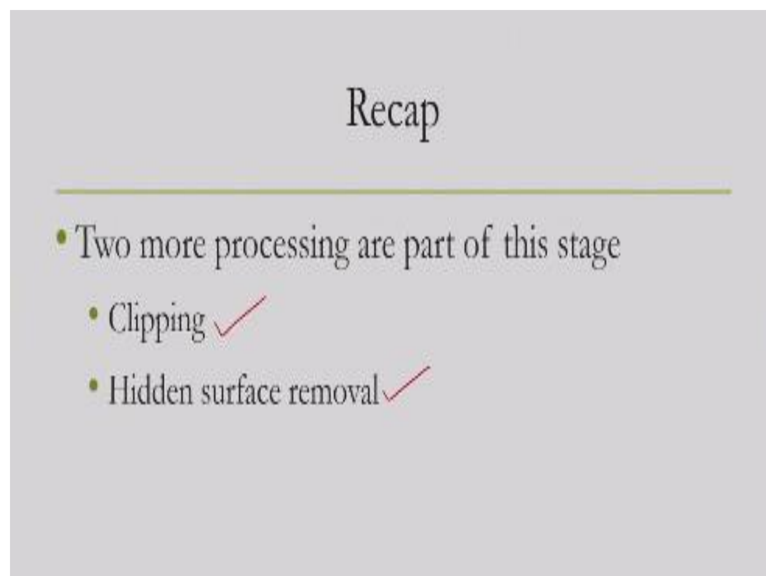


The first stage is object representation, second stage is modeling transformation, third stage is lighting, fourth stage is viewing pipeline, and the fifth stage is scan conversion. Among them, we are currently in the fourth stage that is viewing transformation. The previous three stages we have already discussed.

(Refer Slide Time: 01:16)

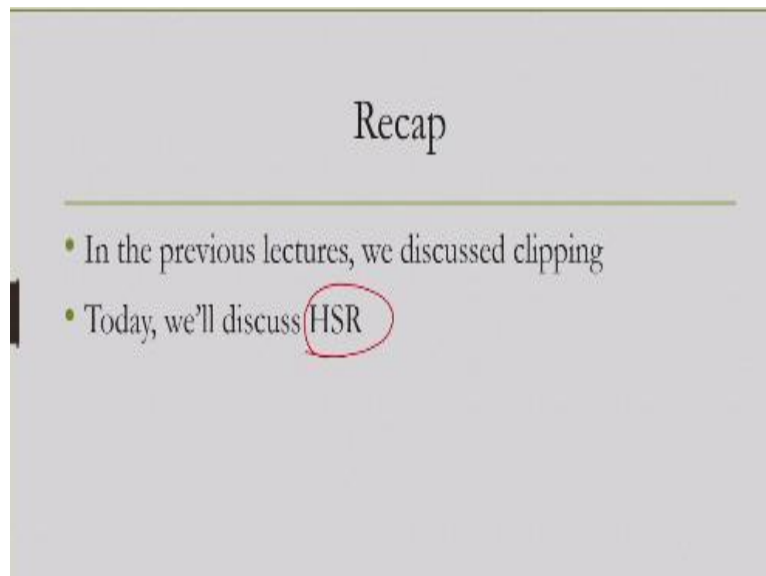
Now, in this fourth stage, as you may recollect there are many sub-stages. So there are 3 transformations and 2 sub-stages related to some other operations. So the three transformations are view transformation, projection transformation, and viewport transformation.

(Refer Slide Time: 01:47)



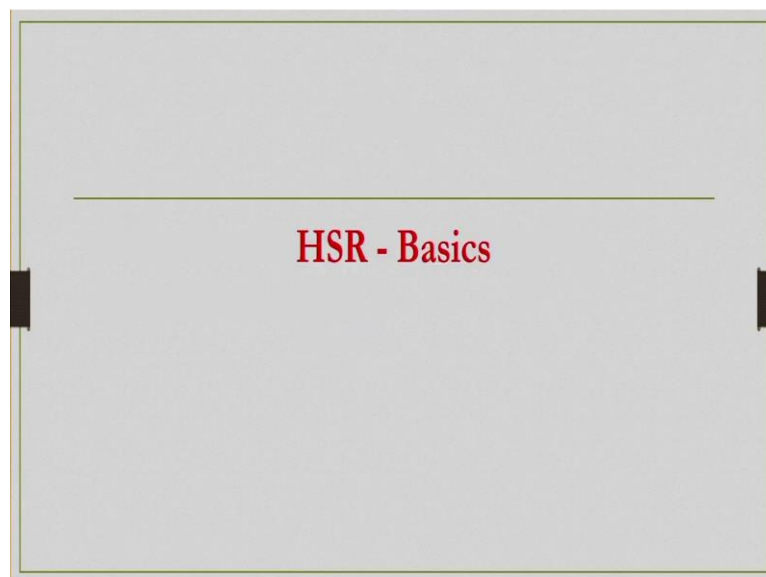
Then we have two operations, clipping and hidden surface removal. Now, among all these transformations and operations, we have already covered the three transformations and clipping.

(Refer Slide Time: 02:02)



Today, we are going to discuss the remaining operation that is hidden surface removal. Let us see what is hidden surface removal, what is the basic idea, and how we can do this.

(Refer Slide Time: 02:22)



Basic Idea

- Earlier, we learnt to remove objects fully or partially outside view volume - using the clipping algorithms
  - Sometimes, we need to remove, either fully or partially, objects that are **inside** view volume

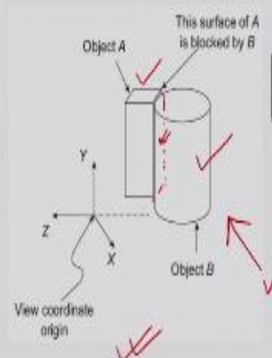
Earlier, during our discussion on clipping, we have learned that how to remove objects fully or partially that are outside the view volume. So those we did using the clipping algorithms. So to note here that the clipping was done on objects that are partially or fully outside of the view volume. Sometimes, we actually need to remove either again fully or partially objects that are inside the view volume.

So in case of clipping, we are dealing with objects that are outside the view volume, whereas in case of hidden surface removal, we deal with objects that are inside the volume. Now when the objects are inside the volume, clearly, we cannot apply the clipping algorithm because clipping algorithms are designed to detect the objects that are outside, either fully or partially.

(Refer Slide Time: 03:40)

Basic Idea

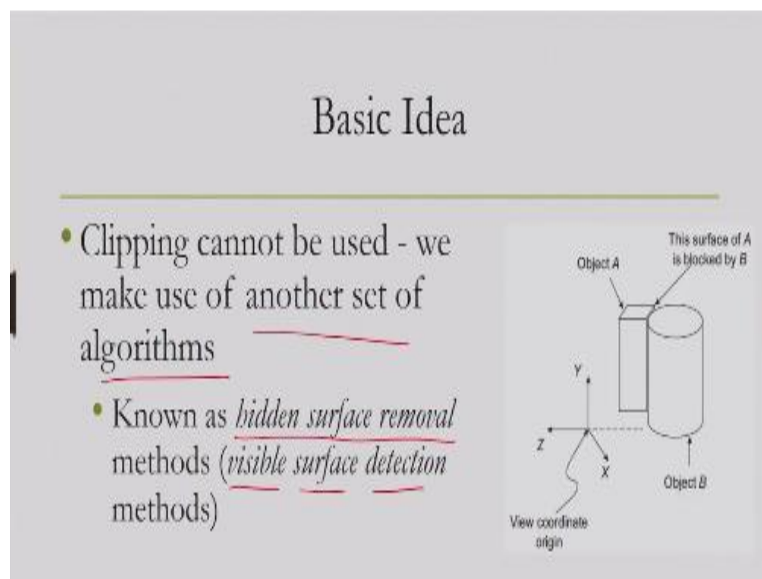
- Ex - object B **partially blocked** from viewer by A
  - For realistic image generation, blocked portion of B should be **eliminated before scene rendered**



Let us see one example. Consider this image. Here, there are two objects; this is the one and this cylinder is the other one. Now ideally, there is one surface here. For realistic image generation, if we are looking at this object from this direction then ideally, we should not be able to see this surface represented by the dotted boundary line, this surface. So if the viewer is located at this point here then this object A surface, which is behind this object B should not be visible to the viewer.

So before we render the image to have the realistic effect, we should be able to eliminate this surface from the rendered image.

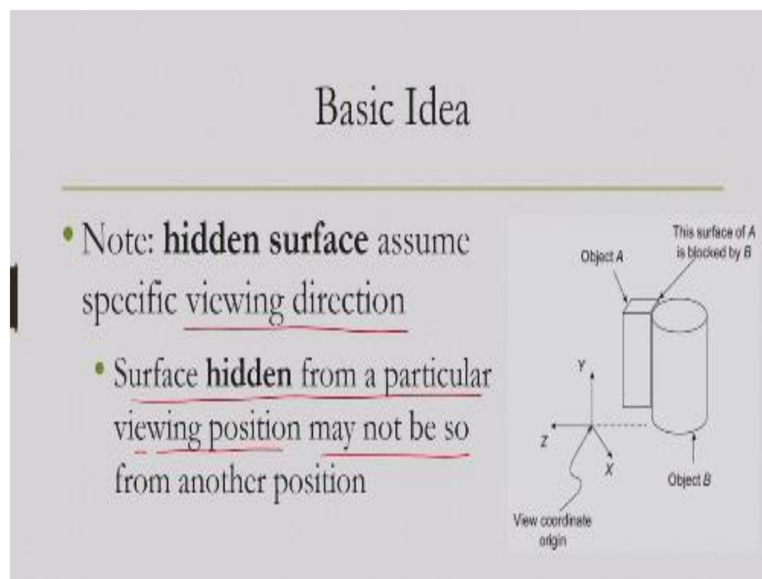
(Refer Slide Time: 05:11)



Here in this case we cannot use clipping, because here we are assuming that both the objects are within the view volume. So clipping algorithms are not applicable. What we require is a different algorithm or a different set of algorithms. Now, these algorithms are collectively known as hidden surface removal methods or alternatively, visible surface detection methods.

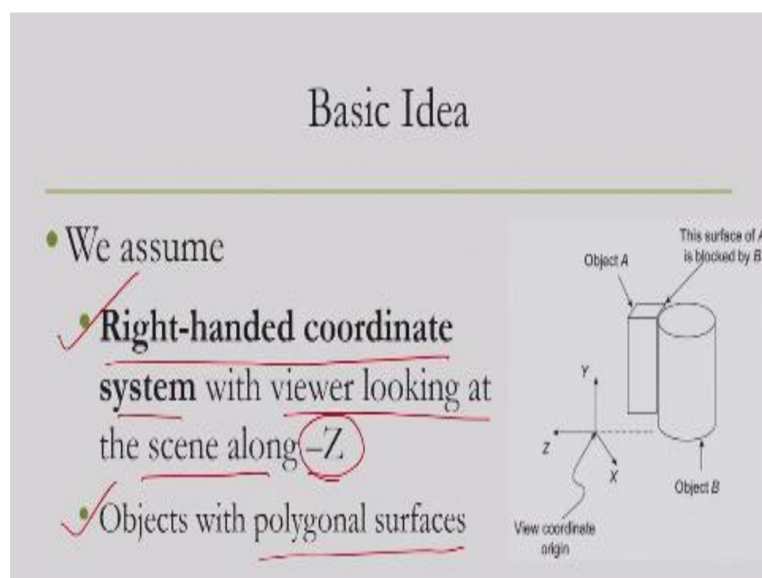
So to note which clipping, what we do? We try to remove objects that are partially or fully outside the view volume. With hidden surface removal, what we do? We try to remove object surfaces or objects, which are inside the view volume but which are blocked from view due to the presence of other objects or surfaces with respect to a particular viewing position.

(Refer Slide Time: 06:17)



So in case of hidden surface, we are assuming specific viewing direction. Because a surface hidden from a particular viewing position may not be so if we are looking at it from another direction. So with respect to viewing position only, we can determine whether surface or an object is hidden or not.

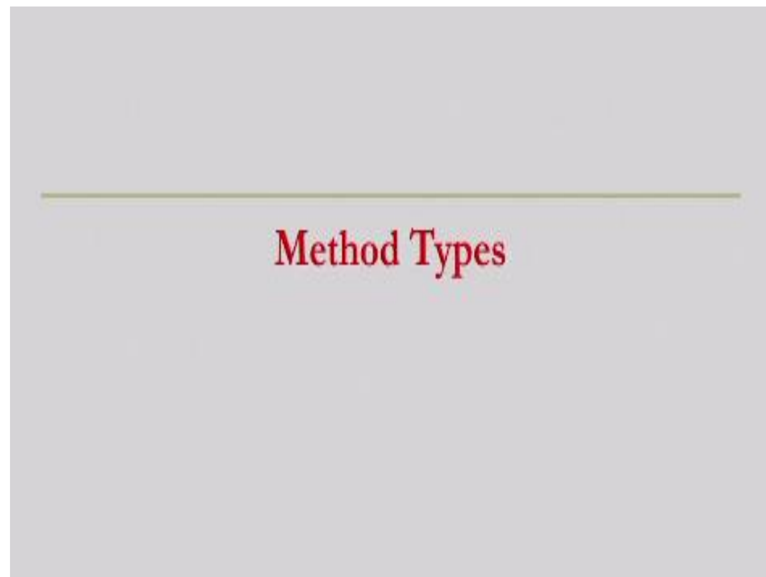
(Refer Slide Time: 06:46)



Now, before we go into the details of the methods for hidden surface removal, we should keep in mind that there are two assumptions that we will be making. First one is we will use a right-handed coordinate system and we will assume that the viewer looking at the scene along the negative Z direction. Secondly, the objects whichever are there in the scene have

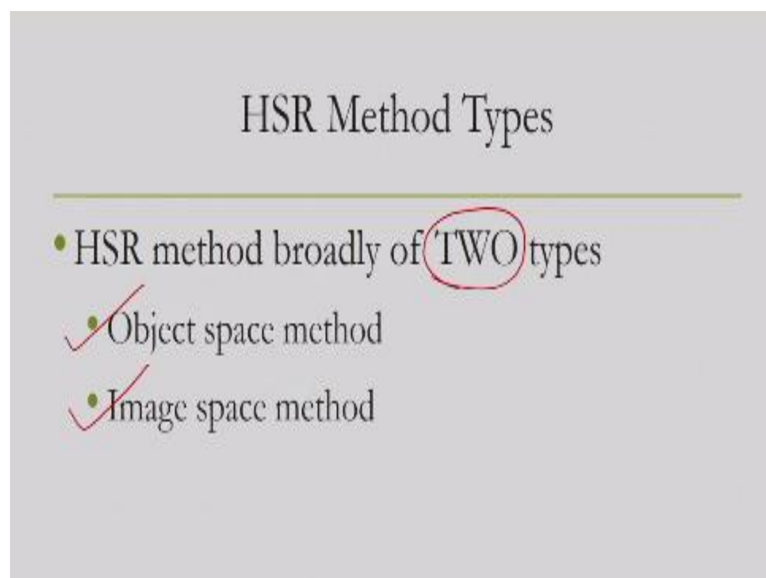
polygonal surfaces, so all the object surfaces are polygonal. These two assumptions we will make in order to explain the hidden surface removal methods.

(Refer Slide Time: 07:48)



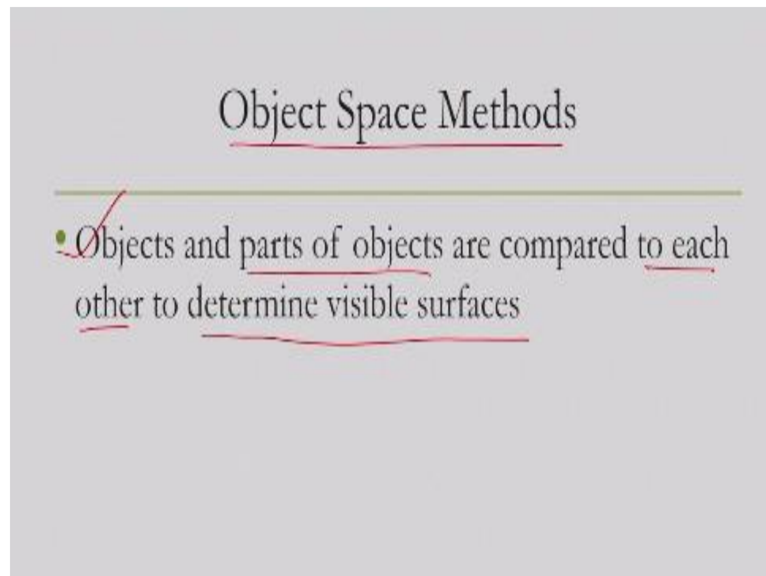
Now let us go into the details of the methods that are there to detect and eliminate hidden surfaces.

(Refer Slide Time: 08:00)



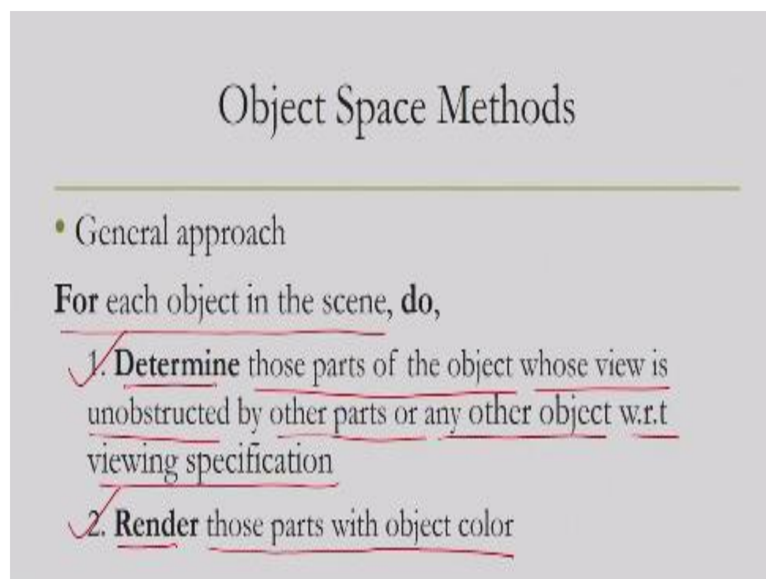
Now there are many methods, all these methods we can broadly divide into two types. One is object space method, the second one is image space method. So what is the idea behind these methods let us try to understand?

(Refer Slide Time: 08:24)



In case of object space method what we do, we compare objects or parts of the objects to each other to determine the visible surfaces. So here we are dealing with objects at the level of 3D.

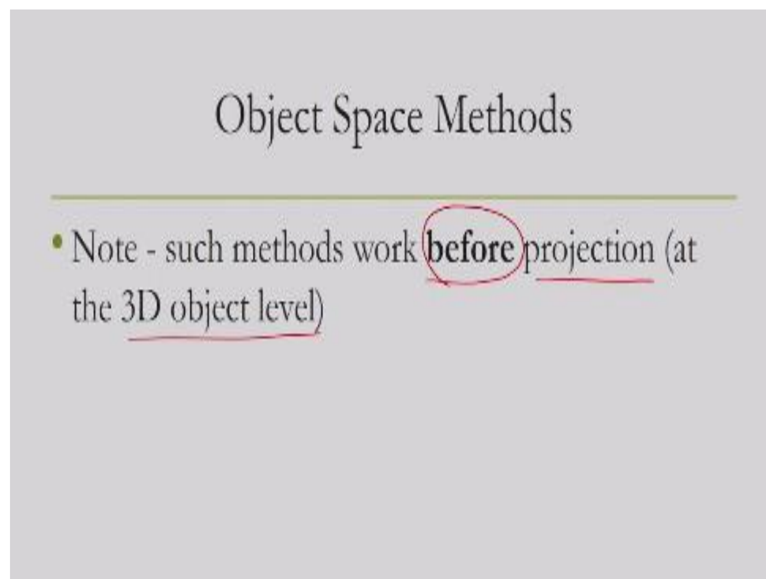
(Refer Slide Time: 08:53)



And the general approach that is followed to perform hidden surface removal with an object space methods consists of two stages broadly. So for each object in the scene what we do is first we determine those parts of the object whose view is unobstructed by other parts or any other object with respect to the viewing specification.

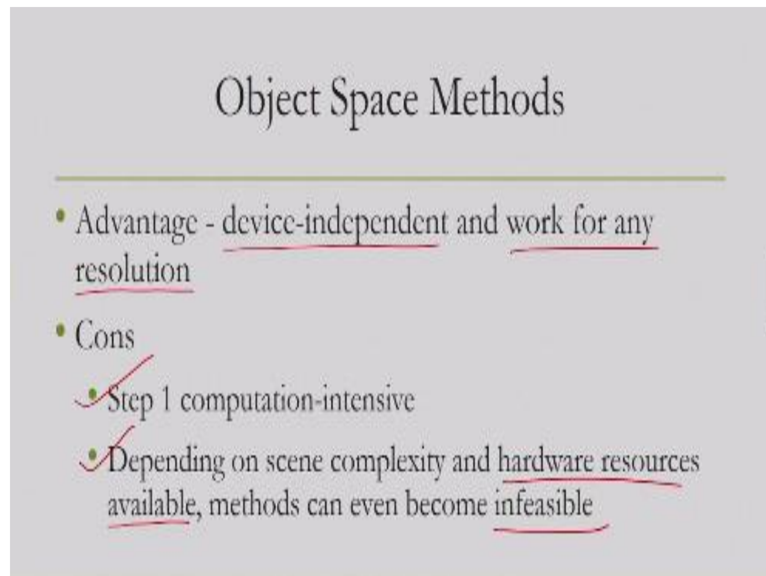
So first stage is to determine the parts that are hidden with respect to the viewing position. And then in the second position, we render the parts that are not hidden. So, essentially those parts that are not obstructed with the color of the object. So these two are the general steps that are performed in any object space method namely, first stage is to determine the surfaces that are not hidden and in second stage we render those surfaces or parts of the objects with the particular color.

(Refer Slide Time: 10:12)



Since here we are dealing with objects, so essentially these methods work before projection, at the 3D object level. Remember that once we perform projection, the objects are transformed to a 2D description. So we cannot have this 3D characteristics.

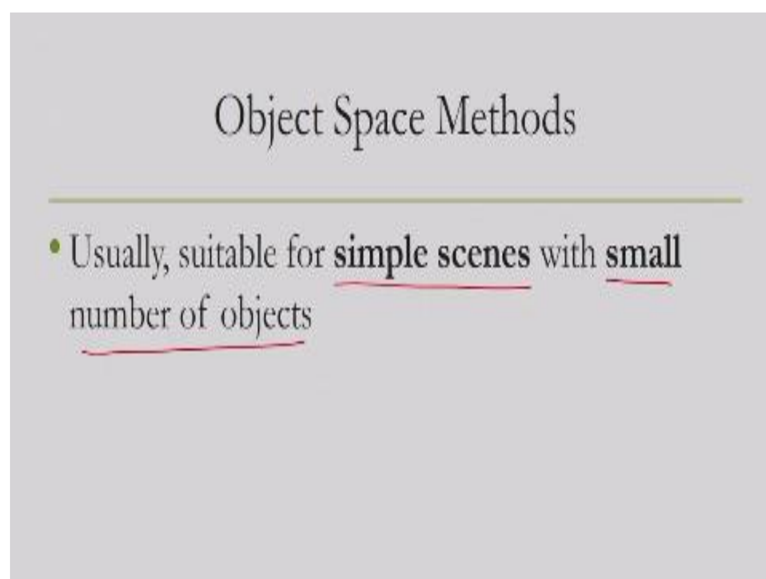
(Refer Slide Time: 10:41)



So what are the advantages? There is one advantage. So this object space methods provide device-independent method and work for any resolution of the screen, but it also has some drawbacks namely, determination of these surfaces that are hidden or not hidden is computation intensive. Secondly, depending on the complexity of the scene and also the resources that are available.

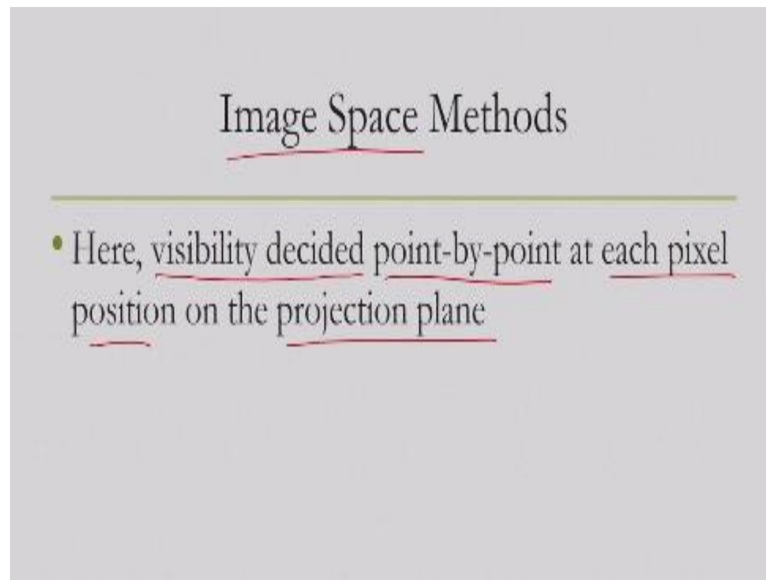
These methods can even become infeasible because they are computation intensive and if the resources are not sufficient then we may not be able to implement them at all.

(Refer Slide Time: 11:43)



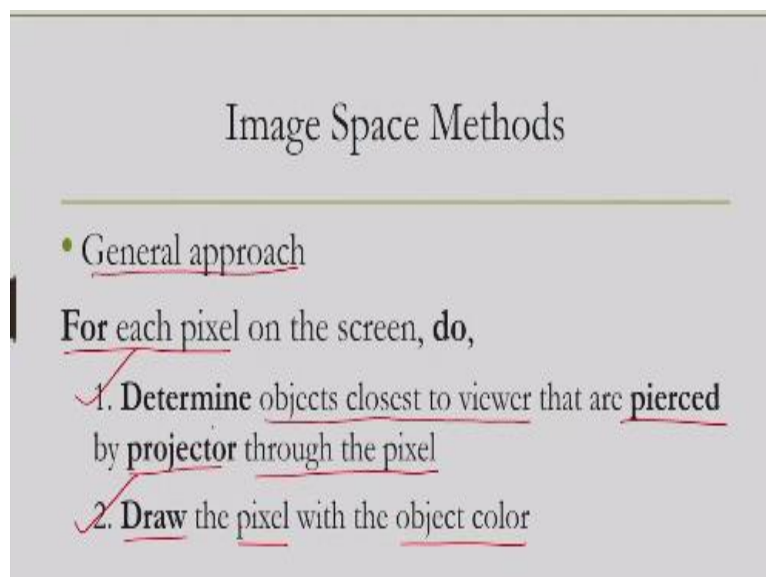
Usually, such methods are suitable for simple scenes with small number of objects. So object space methods are best applicable when the scene is simple and having small number of objects.

(Refer Slide Time: 12:00)



In case of image space method what happens? So as the name suggest, the detection and rendering takes place at the level of image that means after projections. So here, visibility is decided point-by-point at each pixel position on the projection plane. So here, we are no longer dealing in the 3D space, we are dealing in 2D projection plane at the level of pixels.

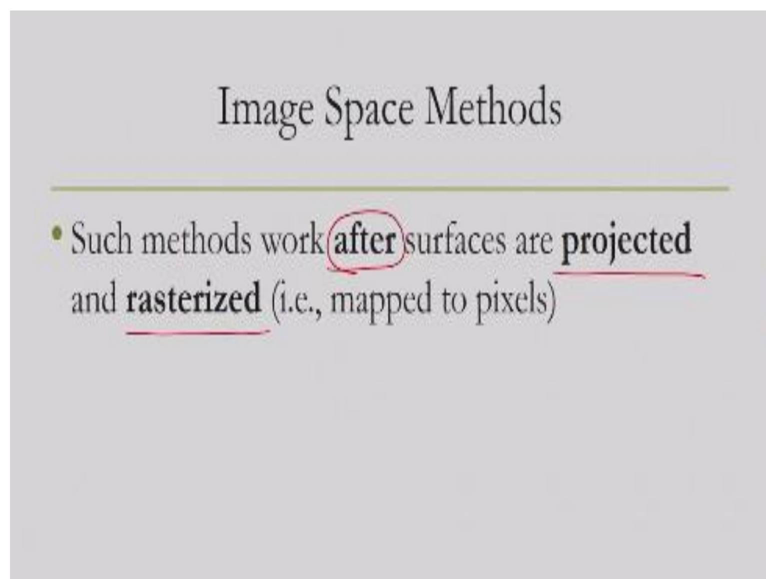
(Refer Slide Time: 12:41)



Again, there are two steps in the general approach. So for each pixel on the screen what we do? We first determine the objects that are closest to viewer and are pierced by the projector through the pixel. So essentially, the closest objects that are projected to that point. Secondly, the second step is to draw the pixel with the object color.

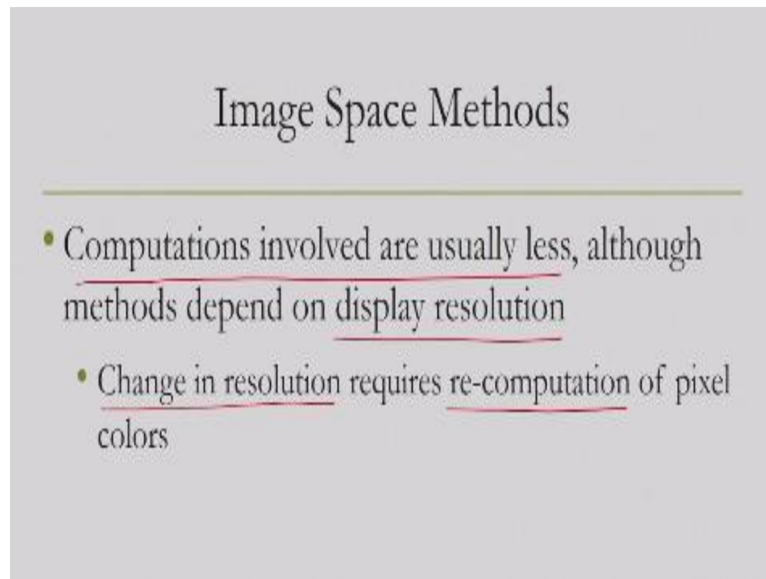
So in the first stage, we determine which is the closest object that is projected on the pixel, and in the second stage we assign the pixel color as the object color and that we do for each pixel on the screen. So to compare earlier what we were doing? Earlier, we were doing it for each surface here we are doing it for each pixel.

(Refer Slide Time: 13:58)



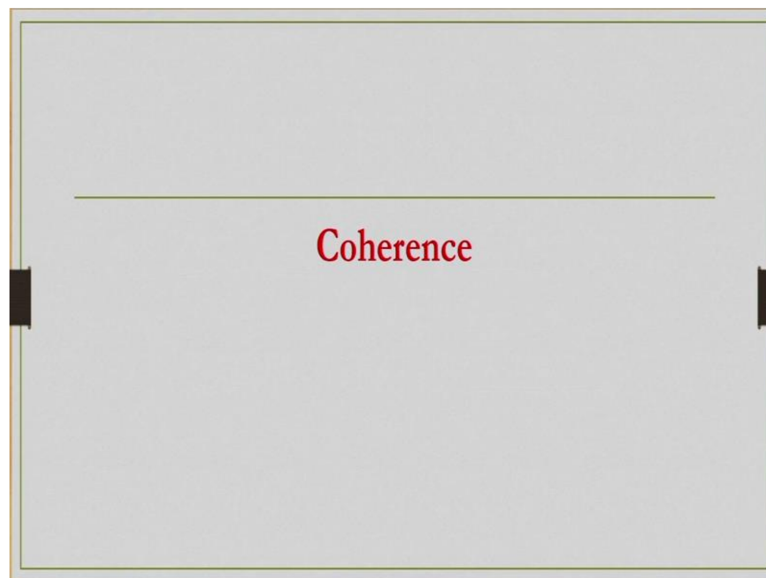
Clearly, the methods work after surfaces are projected and rasterized that means mapped to pixel grid, unlike the previous case where we were in the 3D domain.

(Refer Slide Time: 14:20)



Here the computations are usually less compared to object space methods. However, the method depends on display resolution because we are doing the computations for each pixel. So if there is a change in resolution then we require re-computation of pixel colors. So that is the overhead.

(Refer Slide Time: 14:52)



So broadly, there are these two methods, object space methods, and image space methods. Later on, we will see examples of each of these methods which are very popular. But before going into that let us try to talk about some properties that actually are utilized to come up with efficient methods for hidden surface detection and removal.

(Refer Slide Time: 15:25)

Application of Coherence

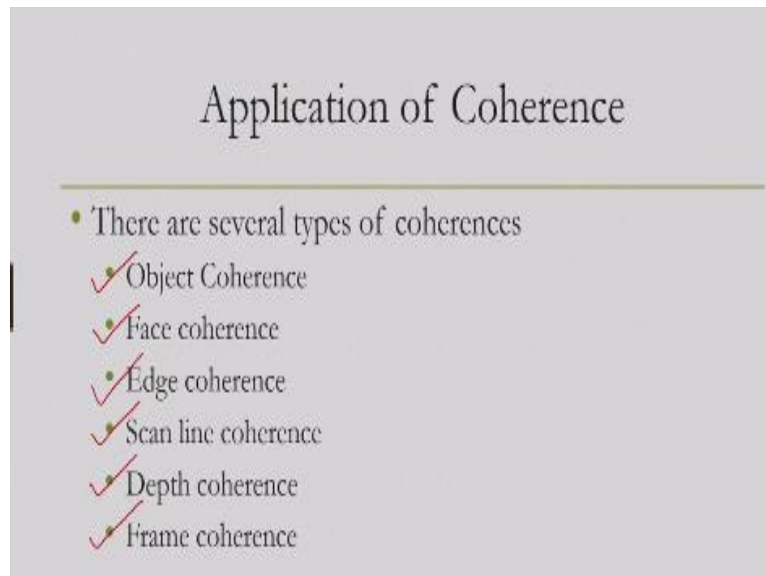
- HSR computationally intensive - one way to reduce computation is use of **coherence** properties
  - We exploit **local similarities** - making use of results calculated for one part of scene or image for the other **nearby** parts

So there are many such properties. Collectively, these properties are called coherence properties, which are used to reduce the computations in hidden surface removal methods. As we already talked about, these methods are computationally intensive. So if we use this coherent properties then some amount of computations can be reduced as we shall see later.

Now these properties are essentially related to some similarities between images or parts of the images and if we perform computation for one part then due to these properties, we can apply the results on other parts and that is how we reduce computation.

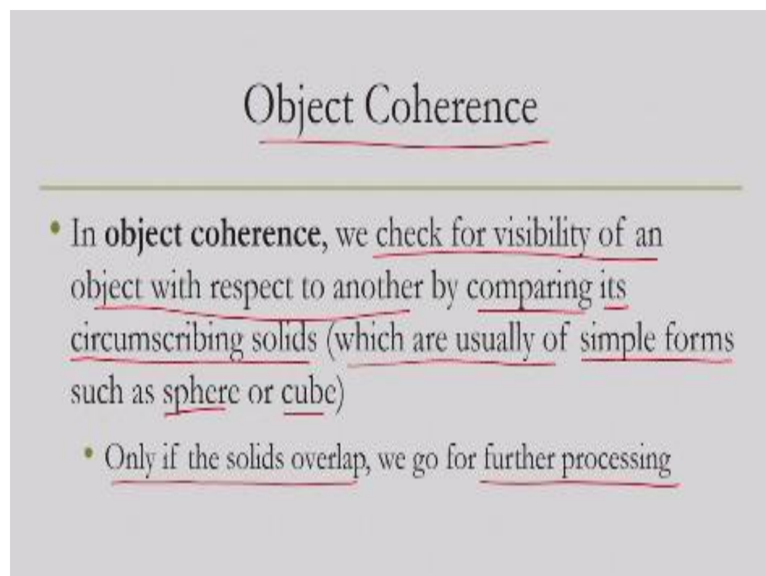
So essentially, we exploit local similarities that means making use of results that we have calculated for one part of a scene or an image for the other nearby parts. So we perform computation for one part and use the result for other part without repeating the same computation and in that way, we reduce some amount of computation.

(Refer Slide Time: 16:54)



Now, there are many such coherence properties; broadly, of six types, object coherence, face coherence, edge coherence, scan line coherence, depth coherence, and frame coherence. Let us quickly have a discussion on each of these for better understanding although, we will not go into detailed discussions and how they are related to different methods.

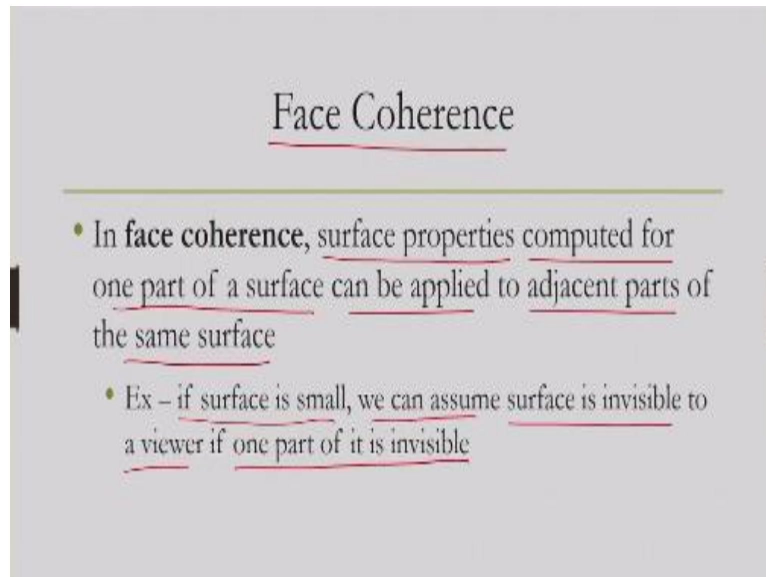
(Refer Slide Time: 17:25)



First is object coherence, what it tells? Here, we check for visibility of an object with respect to another object by comparing its circumscribing solids, which are in many cases, of simple forms, such as spheres or cube. So then, only if the solids overlap we go for further processing. If there is no overlap that means there is no hidden surfaces so we do not need to

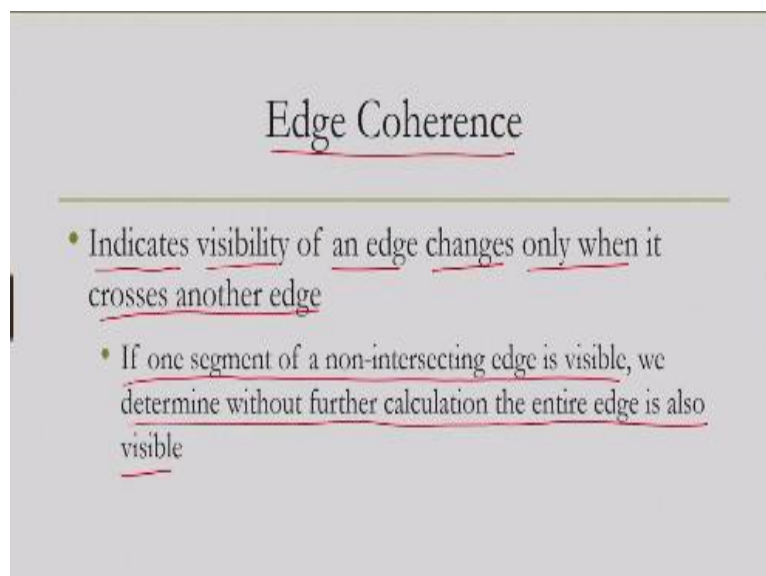
do any processing for further. So this is simple way of eliminating lots of computations due to this object coherence property.

(Refer Slide Time: 18:23)



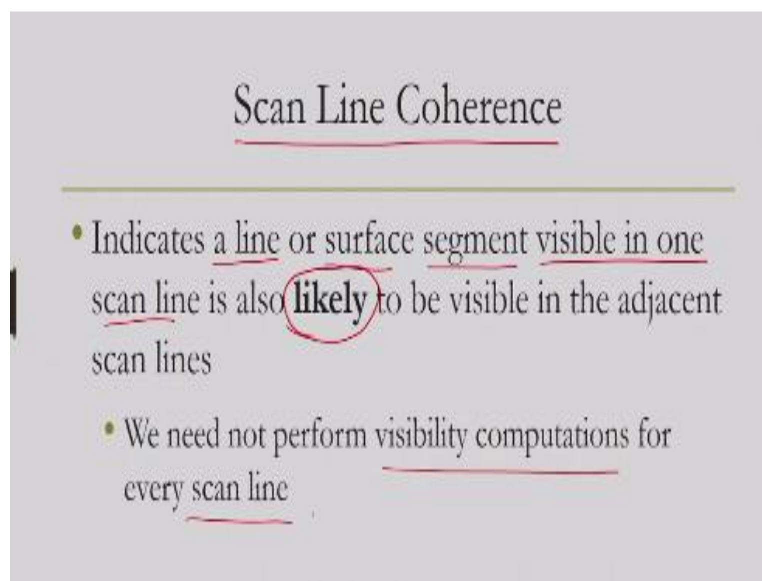
Next, come face coherence. Here, surface properties computed for one part of a surface can be applied to adjacent parts of the same surface that is what is the implication of this face coherence property. For example, if the surface is small then we can assume that the surface is invisible to a viewer if one part of it invisible. So we do not need to check for invisibility for each and every part. So we check it for one part and then we simply say that other parts will also be invisible if that part is invisible.

(Refer Slide Time: 19:10)



Then third is edge coherence. Here, this property indicates visibility of an edge changes only when it crosses another edge. If one segment of a non-intersecting edge is visible, we determine without further calculation the entire edge is also visible. So edge coherence tells us that there will be a change in visibility only if the edge intersects another edge. In other words, if one segment of an edge is visible and the edge is not intersecting with any other edge that means we can say that entire edge is also visible.

(Refer Slide Time: 20:11)



Then comes scan line coherence, what it tells? It indicates or implies a line or surface segment that is visible in one scan line is also likely to be visible in the adjacent scan lines and we do not need to perform this visibility computations for every scan line. So we do it for one scan line and apply the result for adjacent scan lines.

(Refer Slide Time: 20:48)

Depth Coherence

- Tells us depth of adjacent parts of the same surface are **similar**
  - Very useful in determining visibility of adjacent parts

Next is depth coherence, which tells us that depth of adjacent parts of the same surface are similar. There is not much change in depth at the adjacent parts of a surface. This information, in turn, help us to determine visibility of adjacent parts of a surface without too much computation.

(Refer Slide Time: 21:26)

Frame Coherence

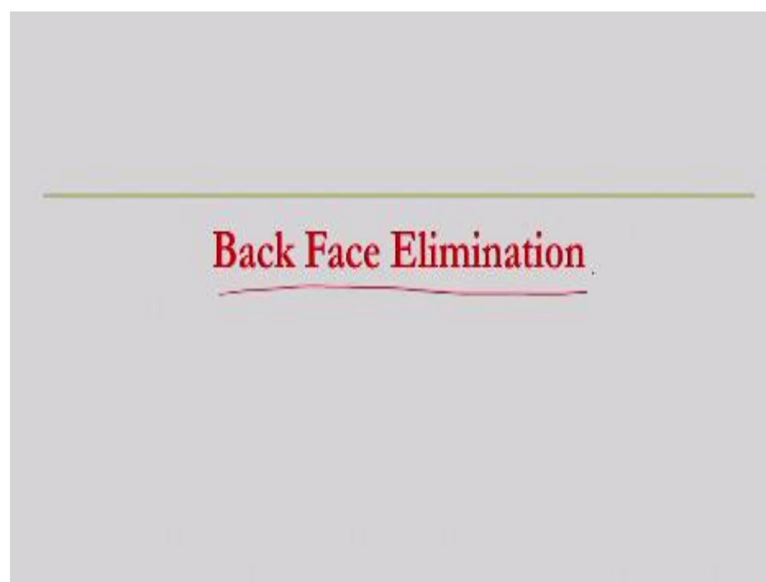
- Implies pictures of same scene at successive points in time likely to be similar, despite small changes in objects and viewpoint, except near the edges of moving objects
  - Visibility computations need not be performed for every scene rendered on the screen

Then frame coherence, which tells us that pictures of same scene at successive points in time are likely to be similar despite small changes in objects and viewpoint except near the edges of moving objects. That means visibility computations need not be performed for every scene rendered on the screen. So frame coherence is related to scene change. Earlier coherence

properties were related to static images here we are talking of dynamic change in images and based on this coherence property we can conclude that visibility can be determined without computing it again and again for every scene.

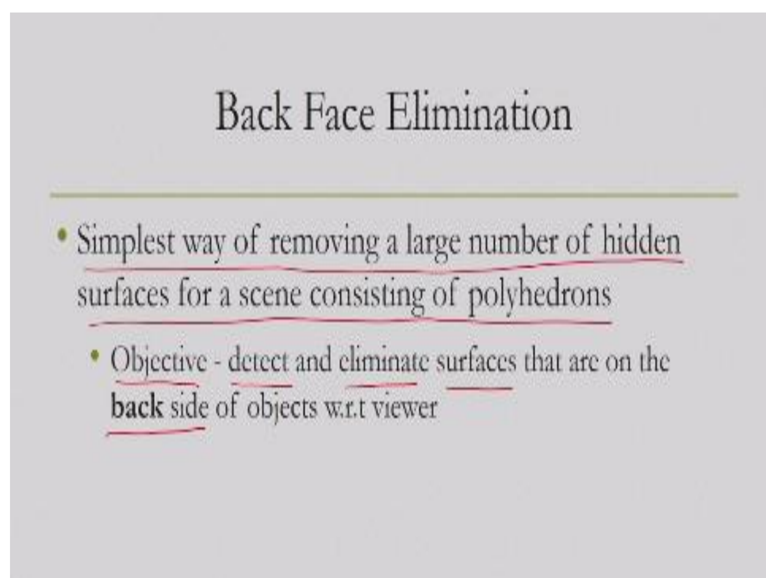
So that is in short the six coherence properties. First five properties are related to static images last property can be used for rendering animations, which anyway is not part of our lectures here and the hidden surface removal methods make use of this properties to reduce computations. Now let us go into the details of such methods.

(Refer Slide Time: 23:10)



So we start with a simple method that is called back face elimination method.

(Refer Slide Time: 23:19)



What is this method? This is actually the simplest way of removing a large number of hidden surfaces for a scene consisting of polyhedrons. So here, we are assuming each object is a polyhedron, and using back face elimination, we can remove a large number of hidden surfaces. The objective is to detect and eliminate surfaces that are on the backside of objects with respect to viewer.

So when a surface is on the backside of an object with respect to a particular viewer clearly, during rendering that back surface should not be shown. With back face elimination method, we can detect on those back surfaces and then remove them from further consideration during rendering.

(Refer Slide Time: 24:18)

Back Face Elimination

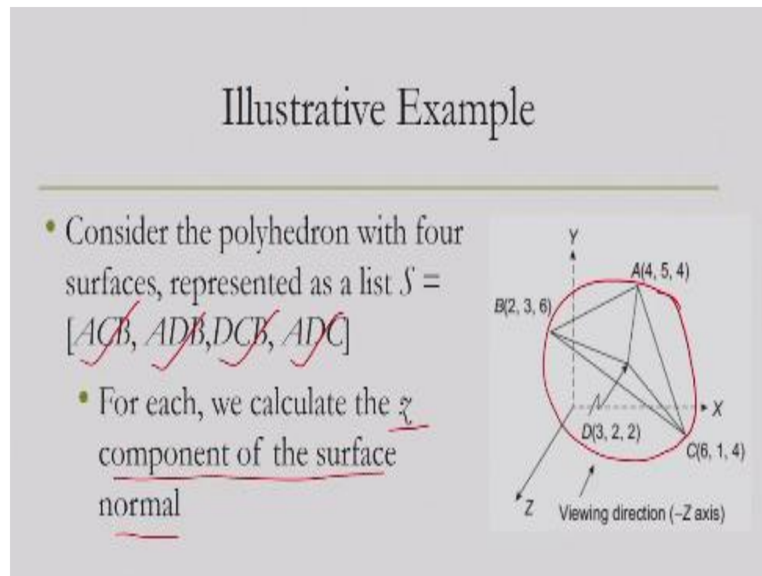
- Steps
  - Determine **normal** for each surface **N** (a,b,c)
  - If (c <= 0) **eliminate** surface [c < 0, surface is **back** face, c = 0, viewing vector **grazes** the surface]; Otherwise, retain surface (i.e., if c > 0)
  - Perform steps 1 and 2 for all surfaces

The steps are very simple for this particular method. So there are three steps. In the first step, we determine a normal vector for each surface N represented in terms of its scalar quantities a, b, c along the three-axis. I am assuming here that you all know how to calculate the normal vector for a given surface, if not then you may refer to any book on vector algebra, basic book; it is a very simple process and I will not discuss the details here.

Now, once the normal is decided we check for this Z component. If this Z component is less than equal to 0, the scalar component, then we eliminate that particular surface because when Z component is less than 0, the surface is back face, whereas when it is equal to 0 the viewing vector actually grazes the surface. In that case, also, we consider it to be a back face.

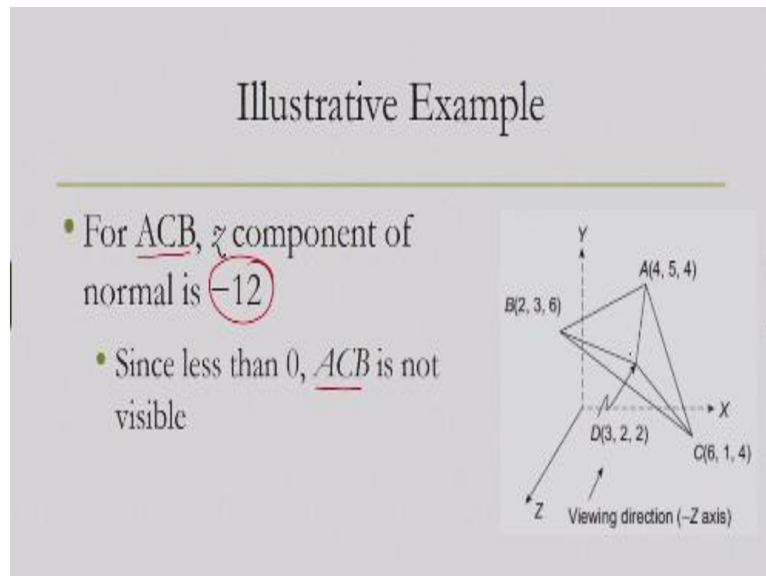
Now, if c is greater than 0 then we retain the surface it is not the back face and we perform these steps one and two for all the surfaces in a loop. So as you can see, it is a very simple method we simply take one surface at a time, compute its surface normal and check the Z component the scalar component of Z. If it is less than equal to 0 then we eliminate the surface otherwise we retain it and we do it for all the surfaces.

(Refer Slide Time: 26:24)



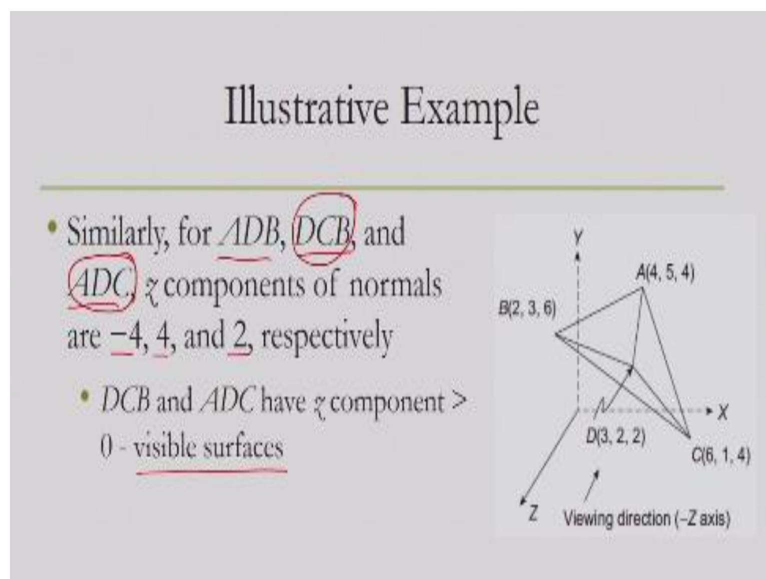
Let us consider one example. Suppose this is our object, it contain four surfaces ACB, ADB, DCB, and ADC; four surfaces are there. So for each of these surfaces, we perform the back face elimination method. For each of these surface, we calculate the Z component of the surface normal as mentioned in the previous steps.

(Refer Slide Time: 27:05)



Let us do it for the surfaces. For ACB, the z component of the normal is -12, so it is less than equal to 0. So ACB is not visible as you can see from this side ACB is on the backside of the surface.

(Refer Slide Time: 27:28)



For ADB, DCB, and ADC the z components of normal are -4, 4, and 2, respectively. So we can see that for DCB and ADC the z component is greater than 0, so these are visible surfaces. But for ADB, it is less than 0 again so it is not a visible surface. So that is the simple way of doing it.

(Refer Slide Time: 28:05)

Note

- Back face elimination works on surfaces - an object space method
  - Using this simple method, about half of the surfaces in a scene can be eliminated

And you should note here that we are dealing with 3D description of the objects in the view coordinate system, so it works on surfaces, therefore it is an object space method. In practice, using this very simple method, we can eliminate about half of all the surfaces in a scene without any further complicated calculations.

(Refer Slide Time: 28:45)

Note

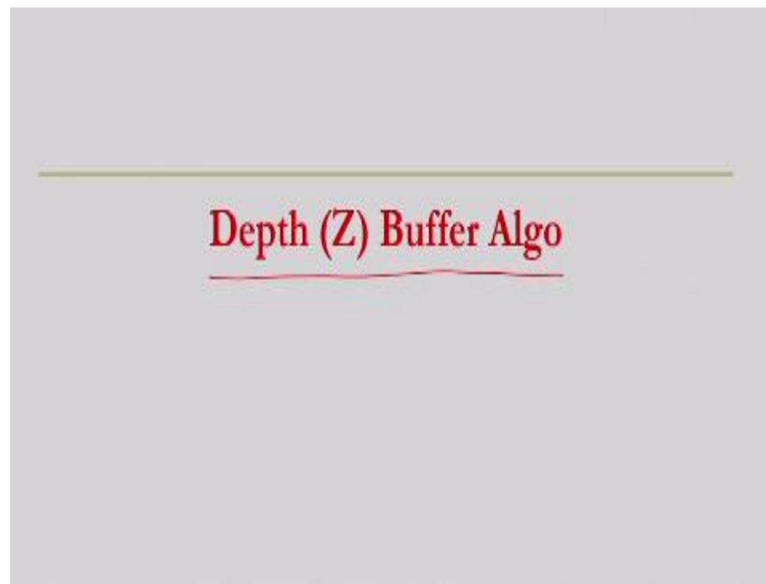
- However, the method does not consider obscuring of a surface by other objects in the scene
  - For such situations, we need to apply other algorithms (in conjunction with this method)

However, there is a problem with this method. This method does not consider obscuring of a surface by other objects in the scene. So what we did? We essentially eliminated back faces of an object. Now here, the back face are obscured by surface of the same object. If a surface is not a back face but it is obscured by surface of some other object then those surfaces

cannot be determined or detected using the back face elimination method and we require some other algorithms.

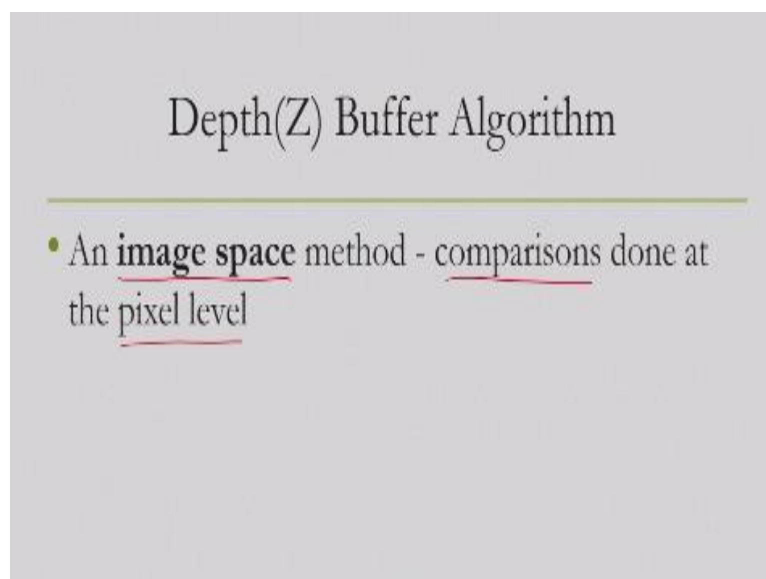
However, those other algorithms are useful for detection of a surface that is obscured by other object surfaces and we can use those algorithms in conjunction with this method.

(Refer Slide Time: 29:48)



Let us discuss one of those methods that is depth buffer algorithm or also known as Z-buffer algorithm.

(Refer Slide Time: 29:58)



Now, this algorithm is an image space method. That means here we perform comparisons at the pixel level. So we assume here that already the surfaces are projected on the pixel grid and then we are comparing the distance of the surface from the viewer position.

(Refer Slide Time: 30:24)

Depth(Z) Buffer Algorithm

- We assume an extra storage, **depth-buffer** (**z-buffer**) - size same as frame buffer (i.e., one storage for each pixel)

Earlier, we mentioned that after projection the depth information is lost. However, we require the depth information here to compare the distance of the surface from a viewer. So we store that depth information even after projection and we assume an extra storage, which has this depth information, which is called depth buffer or Z-buffer. Here, the size of this buffer is same as the frame buffer. That means there is one storage for each pixel in this buffer.

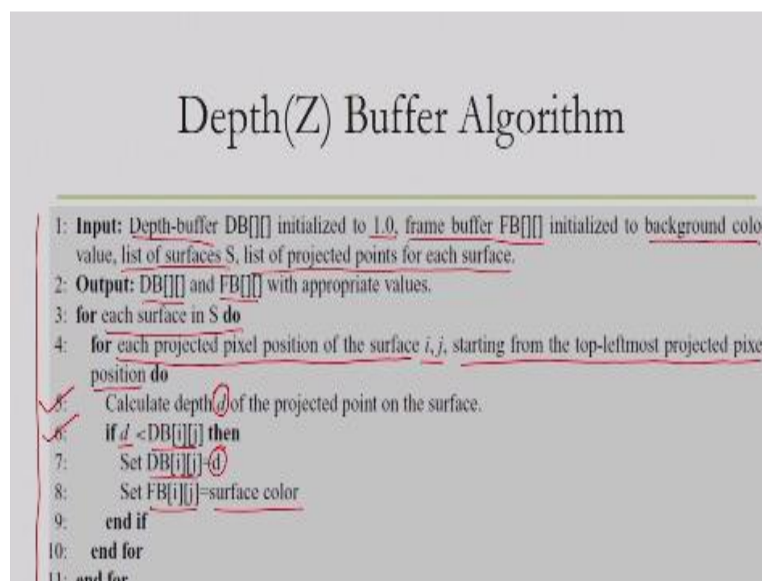
(Refer Slide Time: 31:12)

Depth(Z) Buffer Algorithm

- We also assume canonical volumes - depth of any point can't exceed normalized range
  - Can fix depth-buffer size (number of bits per pixel)

Another assumption we make that is we are dealing with canonical volumes. Depth of any point cannot exceed normalized range. So we already have a normalized range of the volume and the depth cannot exceed that range that we are assuming. If we assume that then we actually can fix the depth buffer size or number of bits per pixels. Otherwise, if we allow unrestricted depth then we do not know how many bits to keep and that may create implementation issues. So we go for some standardized considerations.

(Refer Slide Time: 32:02)



Now, this is the algorithm, the depth buffer algorithm shown here. Input is the depth buffer which is initialized to 1; then we have the frame buffer, which is initialized to the background color; list of surfaces and list of projected points for each surface; so all these are input. And output is this depth buffer and frame buffer with appropriate values. That means depth buffer value will keep on changing and frame buffer will contain the final values at the end of the algorithm.

So what we do? For each surface in this surface list, we perform some steps. Now, for each surface, we have the projected pixel positions. So for each projected pixel position of the surface i, j starting from the top-left most projected pixel position, what we do? We calculate the depth denoted by d of the projected point on the surface and then compare it with already stored depth of that point.

If d is less than what is already stored in the depth buffer at the corresponding location then we update this depth buffer information and then we update the frame buffer information with the color of the particular surface and this we continue for all the pixels for that

projected surface and we do it for all the surfaces. Now the crucial stage here is calculation of the depth, how do we do that?

(Refer Slide Time: 34:06)

Iterative Depth Calculation

- Step 5 – depth calculation
  - Can do iteratively

We can do that iteratively. Let us see how.

(Refer Slide Time: 34:14)

Iterative Depth Calculation

- Plane (surface) equation ax+by+cz+d=0
- Depth of any point on the surface

$$z = \frac{-ax - by - d}{c}$$

- CVV → all projections parallel
  - Point (x, y, z) projects to (x, y) on view plane

Consider this scenario here. This is an illustrative way to understand this iterative method. We are considering this triangular surface, which after projection look something like this. Now, the surface equation we know which we can represent as ax + by + cz + d = 0. Again, if you are not familiar with this, you may refer to basic textbooks on vectors and planes.

Now given this surface equation, we can find out this z values in terms of a b c as shown here. And z value is the depth of that particular point so this is the depth of any point on the surface. Now, we are assuming canonical view volume. That means all projections are parallel projections. So the projection is a simple one, if a point is x, y, z then after projection, it becomes x, y we drop the z component. So that is our assumption.

(Refer Slide Time: 35:53)

Iterative Depth Calculation

- Consider projected pixel (i, j) of surface
- Depth of original surface point

$$z = \frac{-ai - bj - d}{c}$$

Now let us consider one projected pixel i, j of the particular surface. So x is i, y is j. Now, depth of the original surface point z is then given by this expression where we replace x and y with i and j a, b, c, and d are constants.

(Refer Slide Time: 36:23)

Iterative Depth Calculation

- As we progress along same scan line, next pixel is at $(i + 1, j)$
- Depth of corresponding surface point is

$$z' = \frac{-a(i+1) - bj - d}{c} = \frac{-ai - bj - d}{c} - \frac{a}{c} = z - \frac{a}{c}$$

Now, as we progress along the same scan line, say consider this point and this point. The next pixel is at i+1, and j. Now, the depth of the corresponding surface point at the next pixel location is given by this expression where we replace i with i+1. After expanding and rearranging, we can get this form.

Now this part, we already know to be the depth of the point i, j. So we can simply say z′ depth is (z- a/c), and here you can see this is a constant term. So for successive points, we can compute depth by simply subtracting a constant term from the previous depth that is the iterative method.

(Refer Slide Time: 37:39)

Iterative Depth Calculation

- Similar iterations across scan lines
- Assume point (x, y) on an edge of projected surface
- Next scan line, x becomes $x - \frac{1}{m}$ (m is slope of edge line)
- y becomes y-1

Topmost scan line for the surface

yth scan line

(y - 1)th scan line

$z' = z + \frac{\frac{a}{m} + b}{c}$

$z'' = z' - \frac{a}{c}$

x x+1

So that is along a scan line. What happens across scan lines? A similar iterative method we can formulate. Now, let us assume the point x, y on an edge of a projected surface say, here. Now in the next scan line, x becomes (x- 1/m) in this case, and the y value becomes y minus 1; m is the slope of the edge line.

(Refer Slide Time: 38:22)

Iterative Depth Calculation

- Depth (z) becomes

$$z' = \frac{-a(x - \frac{1}{m}) - b(y - 1) - d}{c}$$

- Rearranging, we get

$$z' = z + \frac{\frac{a}{m} + b}{c}$$

Topmost scan line for the surface

yth scan line

(y − 1)th scan line

$x \quad x+1$

$z' = z + \frac{\frac{a}{m} + b}{c}$

$z'' = z' - \frac{a}{c}$

Then we can compute new depth at this new point as shown here and if we expand and rearrange what we will get? This new depth in terms of the previous depth and a constant term. So again, we see that across scan lines, we can compute depth at the edges by adding a constant term to the previous depth, and then along scan line, we can continue by subtracting a constant term from the previous depth.

So this is the iterative way of computing depth and this method we follow in the Z-buffer algorithm to compute depth at a given point. Let us try to understand this algorithm with an illustrative example.

(Refer Slide Time: 39:40)

Example

- Assume two triangular surfaces *s*1 and *s*2 in view volume. *S*1 vertices are [(0,0,6), (6,0,0), (0,6,0)] and *s*2 are [(2,0,6), (6,0,6), (4,4,0)]. Assuming parallel projection (due to CVV transformation), projected vertices of *s*1 on view plane are [(0,0), (6,0), (0,6)] (simply drop *z*). Similarly, for *s*2, the projected vertices are [(2,0), (6,0), (4,4)].

Y

X

6

(4, 4)

(0, 0)

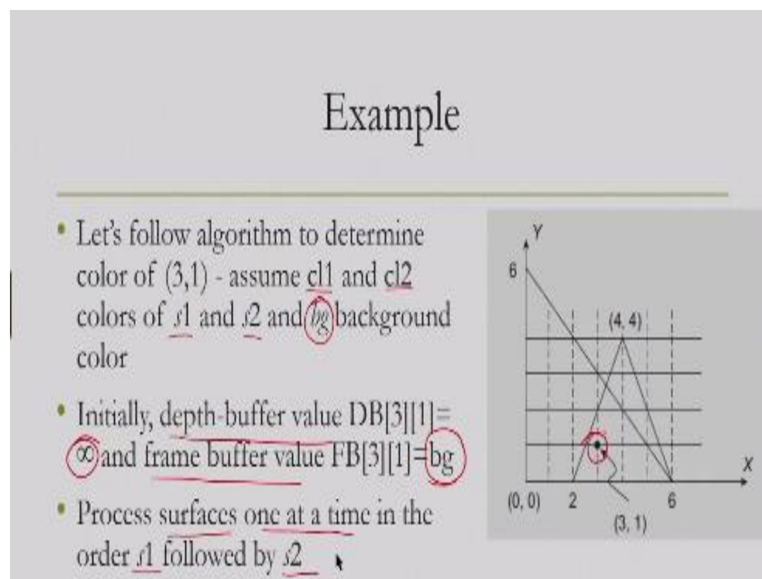
2

6

(3, 1)

Let us assume there are two triangular surfaces s1 and s2. Clearly, they are in the view volume. Now, vertices of s1 are given as these three vertices, s2 is also given as these three vertices. As before we are assuming parallel projection due to canonical view volume transformation and also, we can derive projection points or the projected vertices of s1 and s2 on the view plane to be denoted by this vertices shown here, which essentially, we can obtain by simply dropping the z component. Now, this is the situation shown in this figure.

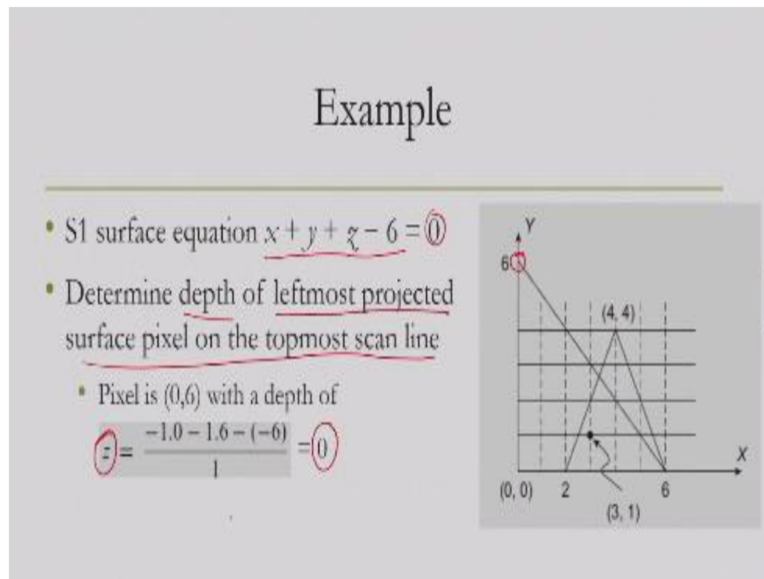
(Refer Slide Time: 40:48)



Now, we are given a point (3, 1) and we want to determine the color at this point. Note that this point is part of both the surfaces, so which surface color it should get, we can determine using the algorithm. Now let us assume that cl_1 and cl_2 are the colors of s_1 and s_2 and b_g is the background color.

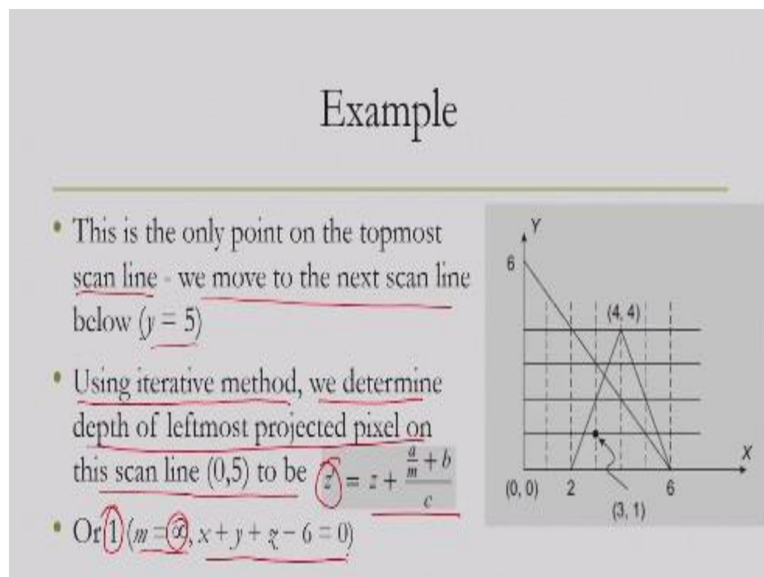
Now, initially, the depth buffer values are set at a very high value and frame buffer values are said to be background color, and then we follow the algorithm steps, we process the surfaces one at a time in the order s_1 followed by s_2.

(Refer Slide Time: 41:40)



Now let us start with s_1, what happens? From the given vertices, we can determine the s_1 surface equation to be x+y+z-6=0. Then we determine the depth of the left-most projected surface pixel on the topmost scan line that is pixel (0, 6) here, which is z to be 0.

(Refer Slide Time: 42:14)

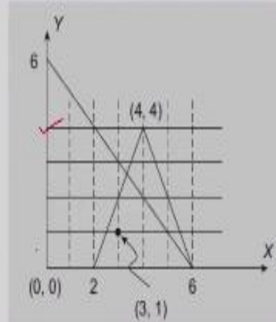


Now, this is the only point on the topmost scan line as you can see in the figure, then we move to the next scan line below that is y=5. And in this way using iterative method, we determine the depth of the left-most projected pixel on this scan line to be, using this expression to be 1 because here m is very high, infinity. And thus, surface equation is this.

(Refer Slide Time: 43:00)

Example

- The algorithm next computes depth and determines color **along** $y = 5$ till right edge
  - At that point, it goes to the next scan line down ($y = 4$)
- For brevity, we skip these steps and go to $y = 1$, as our point of interest is (3,1)

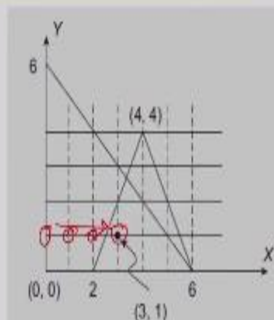


Then the algorithm proceeds to the computation of depth and color determination along y= 5 till the right edge. At that point, it goes to the next scan line down that is y=4 here. Now, we can skip all these steps and we can go directly to y=1, this line on which the point of interest lies.

(Refer Slide Time: 43:34)

Example

- Following iterative procedure across scan lines, we compute depth of left most point (0,1) as $z = 5$
- We now move along the line to (3,1) and calculate (iteratively) $z = 2$
- This depth value less than DB[3][1], which is ∞
- Hence, we set DB[3][1]=2 and FB[3][1]= c/1



Now, following this iterative procedure that we have outlined earlier across scan lines, we compute first the depth of the left-most point here as z=5. We skip those steps, you can do the calculations on your own and find out. Then we move along this scan line when this direction. So we go to the next point here, then here, and so on up to this point, (3, 1), and calculate that at this point z is 2.

Now, this depth value is less than already stored value which is infinity. So we set this value at the corresponding depth buffer location and then reset the frame buffer value from background to the color of surface 1.

(Refer Slide Time: 44:51)

Example

- We continue in this way for all the points for s1
- Next, we do similarly for s2 **(complete calculation yourself as exercise)**

Y
6
(4, 4)
X
(0, 0)
2
6
(3, 1)

Then our processing continuous for all points, but those are of not much relevance here because we are concerned only with this point, so we will skip those processing. So once the processing completes for s_1 for all the projected points, we go to s_2 and we perform similar iterative steps. And then, we find out the color at that particular point for s_2 and then perform comparison and assign the color.

So we skip here all the other calculations and it is left as an exercise for you to complete the calculations. So that is the idea behind depth buffer algorithm or the Z-buffer algorithm.

(Refer Slide Time: 45:54)

Note

- With depth-buffer, a pixel can have only one surface color
  - From any given viewing position, only one surface is visible
- Alright if we are dealing with opaque surfaces only

Now one point is there. With this particular algorithm, a pixel can have one surface color. So given multiple surfaces, a pixel at a time can have only one of those surface colors. From any given viewing position, that means only one surface is visible. So this situation is acceptable if we are dealing with opaque surfaces.

(Refer Slide Time: 46:28)

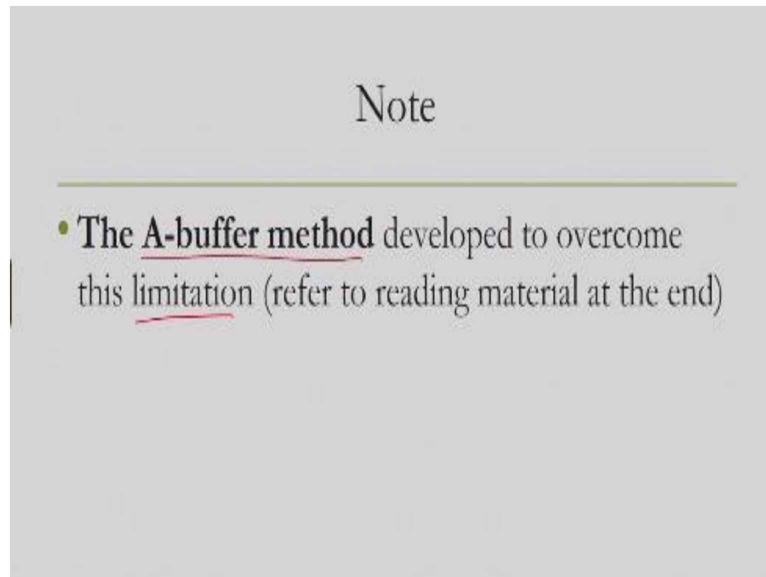
Note

- If scene contains transparent surfaces, pixel color is a **combination** of surface color + contributions from surfaces behind
- In depth-buffer method, we have only one location to store depth value for each pixel
  - Can't store all surfaces contributing to the color value

If the surfaces are not opaque, if they are transparent then definitely, we get to see multiple surfaces which is not possible with this particular depth buffer algorithm. In case of transparent surfaces, pixel color is a combination of the surface color plus contribution from surfaces behind, and our depth buffer will not work in that case because we have only one

location to store the depth value for each pixel. So we cannot store all surface contributions to the color value.

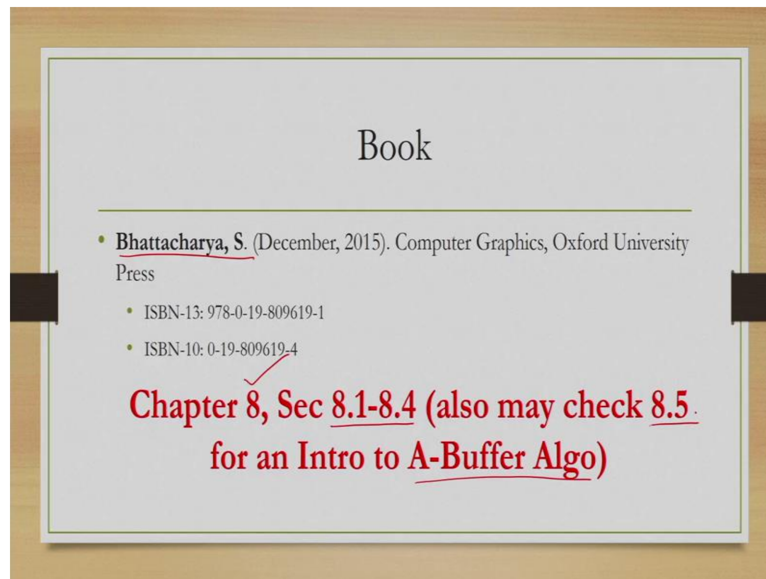
(Refer Slide Time: 47:08)



There is another method called A-buffer method which can be used to overcome this particular limitation. We will not go into the details of this method, you may refer to the material, reading material. So that is, in short, what we can do with depth buffer method.

So to recap, today we learned about basic idea of hidden surface removal. We learned about different properties that can be utilized to reduce computations. Then we learned about two broad class of hidden surface removal algorithms, one is the object space method, one is the image space method. We learned about one object space method that is the back face elimination method, and we also learned about one image space method that is the depth buffer algorithm or Z-buffer algorithm.

(Refer Slide Time: 48:14)



Whatever we have discussed today can be found in this book, you may refer to Chapter 8, sections 8.1 to 8.4. And if you want to learn more about A-buffer algorithm then you may also check section 8.5. That is all for today. Thank you and goodbye.