

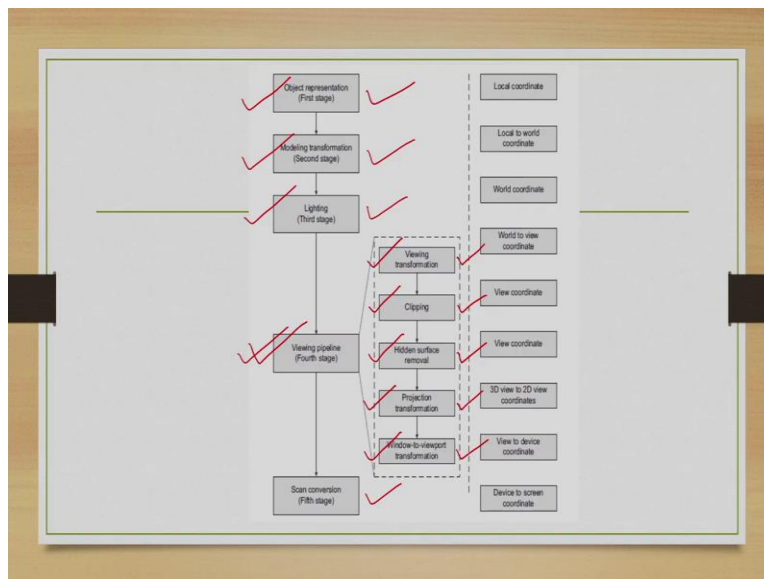
Computer Graphics
Professor. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Lecture 21

Clipping Introduction and 2D Point and Line Clipping

Hello and welcome to lecture number 21 in the course, Computer Graphics. We are currently discussing the graphics pipeline, that is how the 3D scene gets converted to a 2D image on the computer screen, what are the stages there to perform this task. Together these stages are known as pipeline, as we have already discussed. So, let us just quickly have a relook at the pipeline stages and then we will start our discussion on today's topic.

(Refer Slide Time: 01:09)

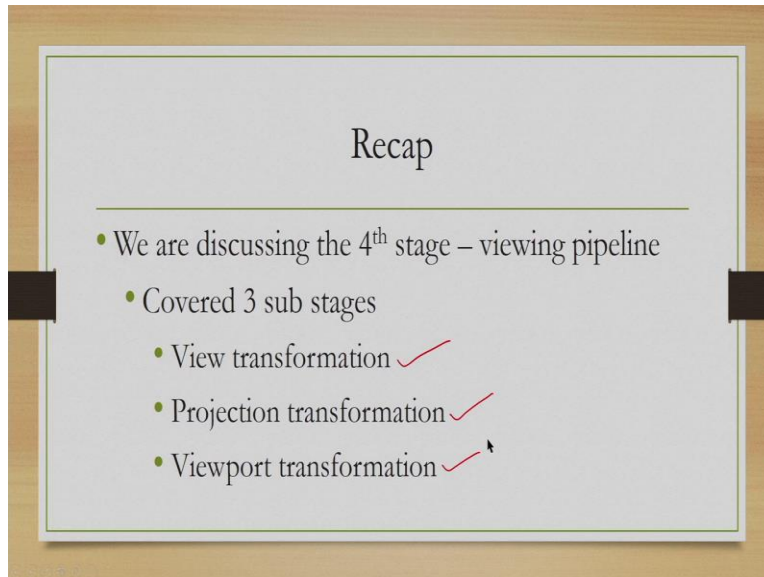


What are the pipeline stages? We have object representation as the first stage, then modeling transformation as the second stage, lighting or assigning color to objects as the third stage, viewing pipeline as the fourth stage, and scan conversion as the fifth stage. So, here I would like to reemphasize on the point that this sequence that is shown here need not be exactly followed during implementation of the pipeline, there the stages maybe in a slightly different sequence.

Now, among these stages, we have already discussed first stage, second stage, third stage, and currently we are discussing the fourth stage that is viewing pipeline. As you can see here, in the viewing pipeline there are sub stages. So, we have viewing transformation, clipping, hidden surface removal, projection transformation and window-to-viewport transformation.

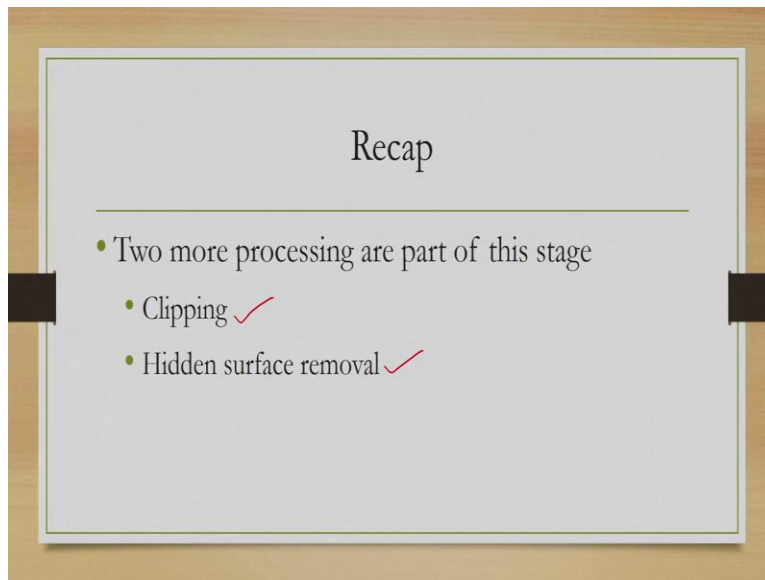
Among them, we have already discussed in the previous lectures, we have already discussed the viewing transformation, the projection transformation, and the window-to-viewport transformation. Two more operations in the fourth stage are remaining, namely clipping and hidden surface removal. So, these operations we are going to discuss in today's and subsequent lectures.

(Refer Slide Time: 02:56)



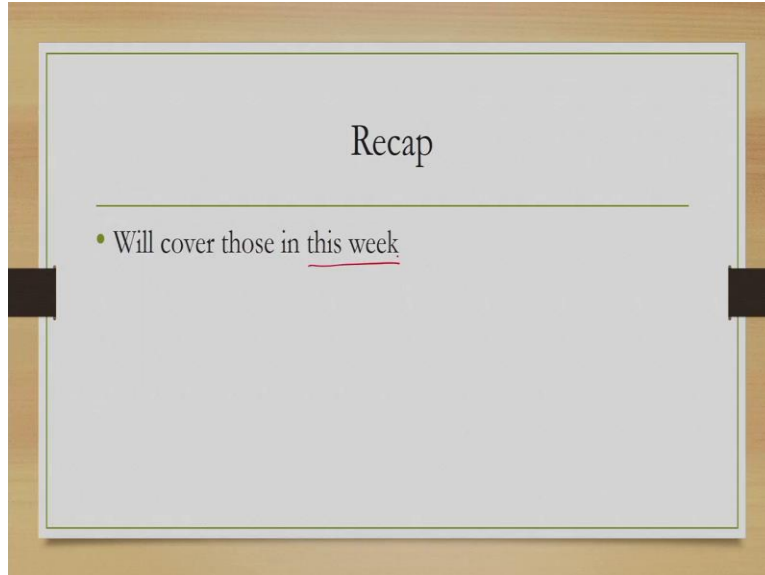
So, in the viewing pipelines stage we have covered these three transformations: view transformation, projection transformation, and viewport transformation already.

(Refer Slide Time: 03:07)



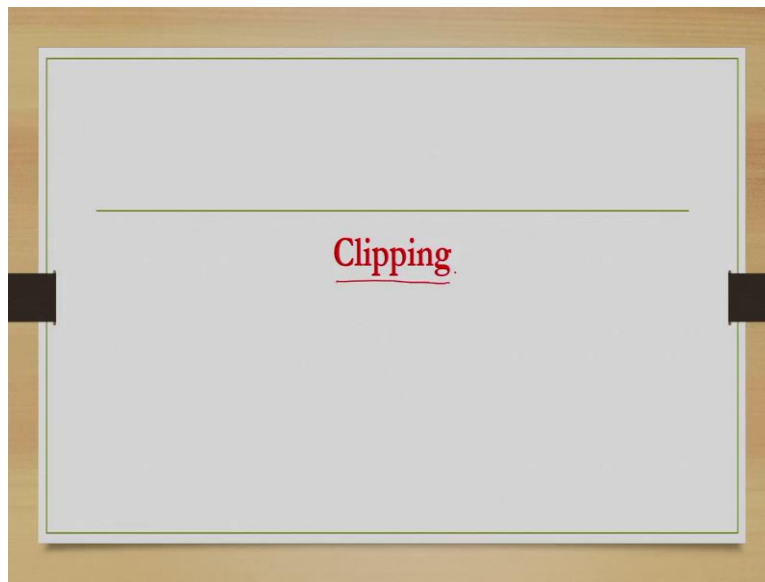
And there are two more operations: clipping and hidden surface removal which are part of the fourth stage.

(Refer Slide Time: 03:18)



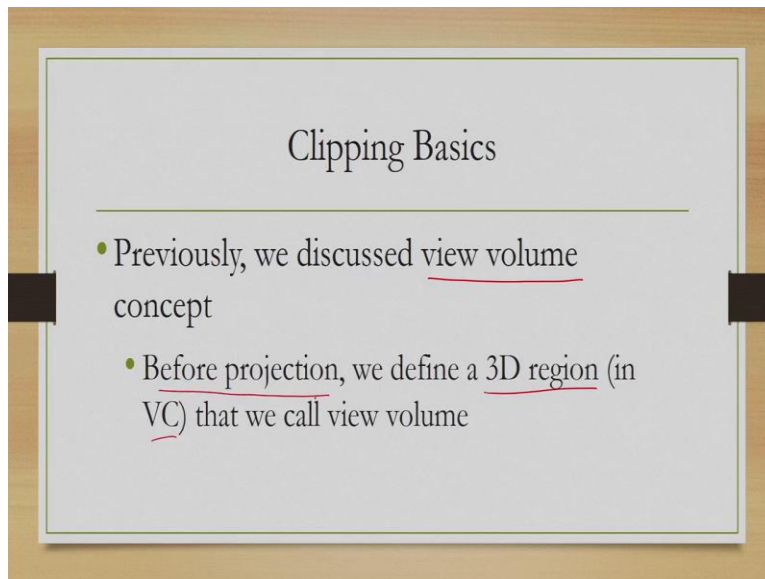
So, these operations we are going to cover in the lectures that we are going to have this week.

(Refer Slide Time: 03:29)



Let us start with clipping. What is this operation and how it is performed.

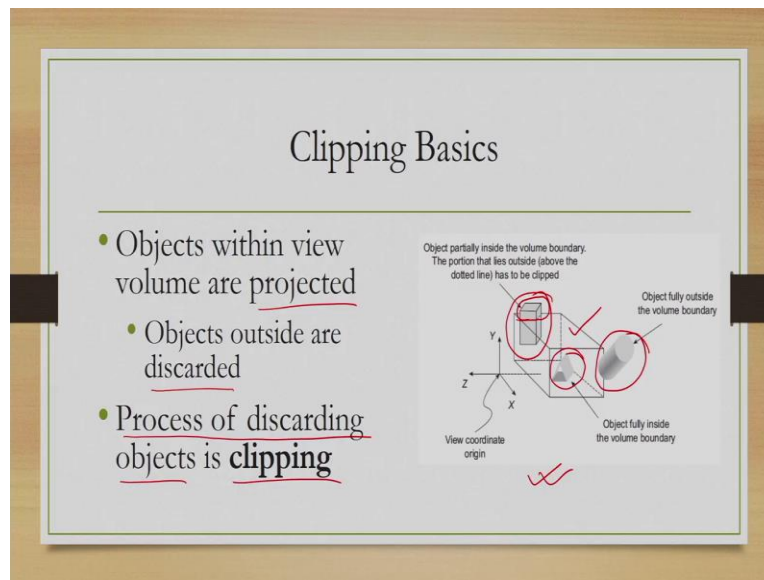
(Refer Slide Time: 03:37)



If you may recollect, earlier we talked about a concept called view volume. So, essentially what we discussed is that when we are trying to generate a 2D image, essentially this is analogous to taking a photo of a 3D scene. So, we first perform view transformation, to transfer that content from world coordinate system to view coordinate system, then we perform projection transformation to project the 3D view coordinate system description of the scene to a 2D view plane description. And finally, we perform window-to-viewport mapping.

However, in this process what we project on the view plane, we project only the objects that are present in the view volume that is a region in the 3D view coordinate space that we have decided to project. So, whatever objects are within the view volume should be projected. Thus, we need to define this view volume in this 3D region, in the view coordinate system before projection.

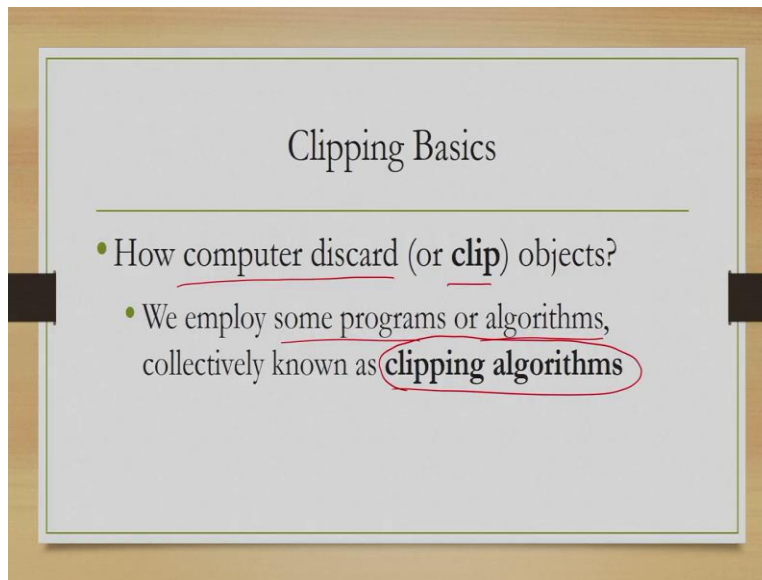
(Refer Slide Time: 05:19)



So whichever object lies within this volume are projected, whereas objects that are outside the volume are discarded. For example, let us have a look at this figure. Here the view volume is on this rectangular parallel pipe. Now, as you can see this object is entirely within the view volume, so it will be projected on the view plane. This object is partially within the view volume, so whichever part is within the view volume will be projected and the outside part, this one will be discarded. Whereas in this case, the entire object is outside the view volume so it will not be projected.

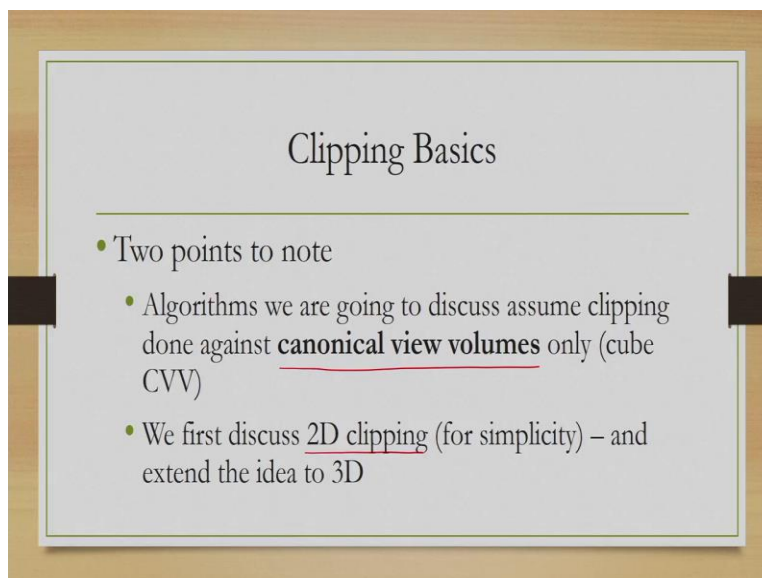
So, we have three situation. In one case, the entire object is within the volume, so entire object is projected. In the other case, object is partially within the volume. So which part is within the volume will be projected, whereas the outside part will be discarded. And in the third case, we have the entire object outside the volume which will be discarded. Now, the process of discarding objects is called clipping. So, before projection we would perform clipping and then whatever objects remain should be projected.

(Refer Slide Time: 06:46)



Now, the question is how a computer can discard or clip an object? That is done through some programs or algorithms, which collectively are known as clipping algorithms. So, we perform clipping with the help of some clipping algorithms. And in this lecture, and subsequent lectures we will go through few of those algorithms which are easier to understand.

(Refer Slide Time: 07:22)

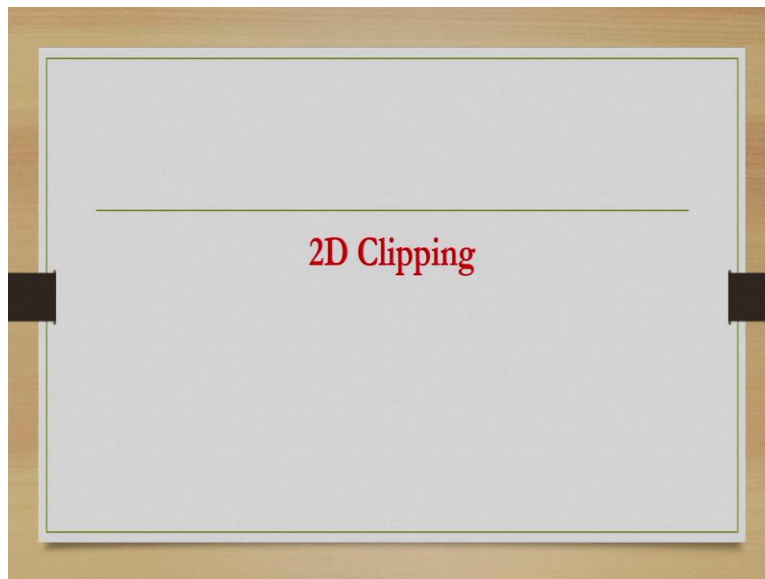


Before we start our discussion on the clipping algorithms, we should keep in mind two points. First thing is, the algorithms will be assuming that the clipping is done against canonical view volume. To recollect, a canonical view volume is a standardized view volume where the shape of

the view volume is rectangular parallel pipe and its bounding planes are within a fixed range. So, it is a standardized view volume.

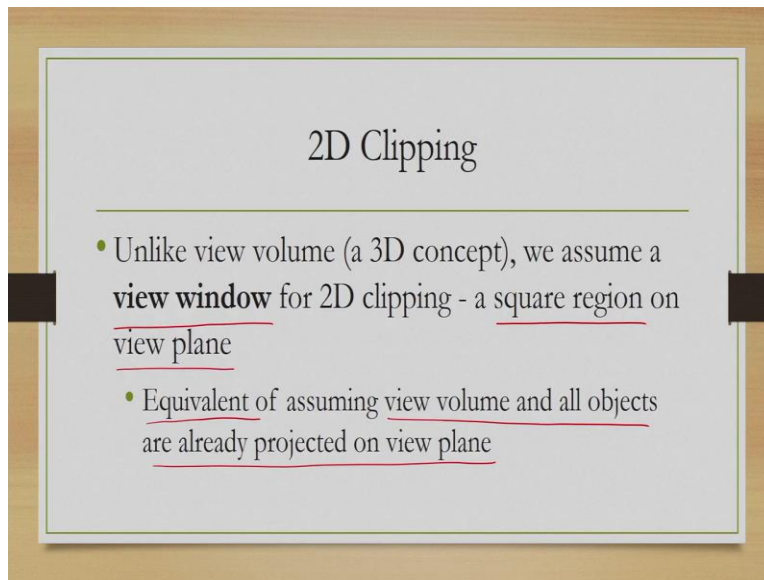
So, whenever we are talking of clipping, we assume that the scene is already transferred to the canonical view volume and then we are performing clipping. Secondly, we will first discuss 2D clipping algorithms for simplicity. It will be easier to understand clipping when we are dealing with 2D objects and then we will extend our discussion to 3D clipping.

(Refer Slide Time: 08:35)



So, let us see what are the algorithms that we can use to perform 2D clipping.

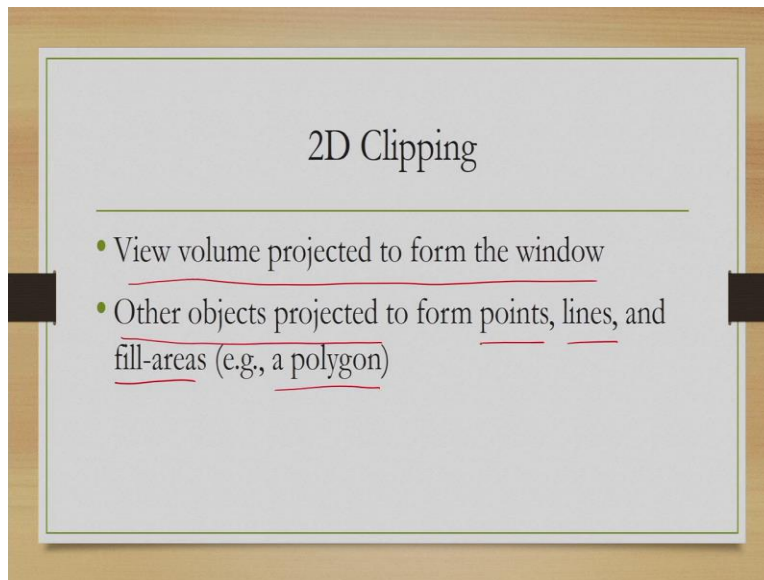
(Refer Slide Time: 08:44)



Now since we are discussing 2D clipping, we have to restrict ourselves to the 2D concepts. Now view volume that we mentioned earlier is a 3D concept but that is not relevant in 2D. So instead of view volume we now use the concept of view window which is a square region on the view plane. In earlier lectures, we already got introduced to idea that we mentioned that on the view plane there is a clipping window on which objects are projected, the concept is same as the view window.

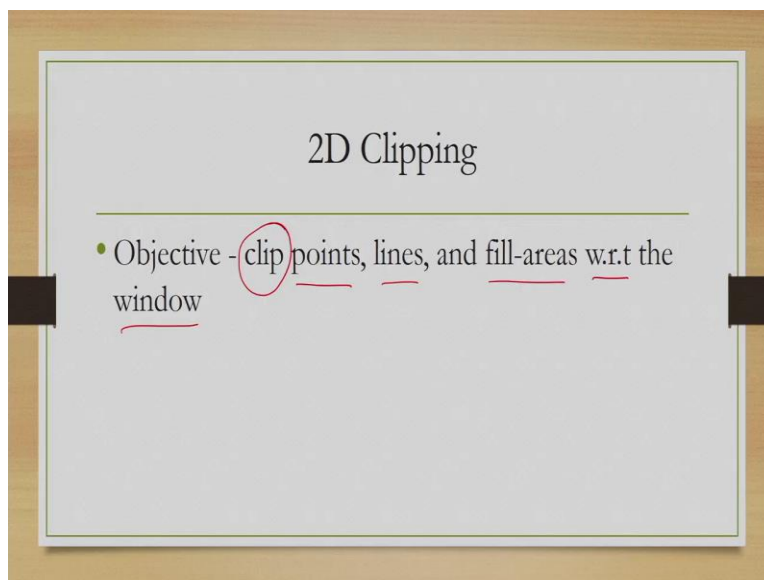
So it is equivalent actually, to assume that the view volume and all objects are already projected on the view plane, that is another way of looking at 2D clipping. So already projection is done and we have 2D clipping window now we want to perform clipping. So that is after projection we want to perform clipping.

(Refer Slide Time: 10:00)



So, this view volume is projected to form the window and other objects are projected to form points, lines and fill areas, such as a polygon. So, we have a window which is formed by projecting the view volume on the view plane and then other objects are projected to form points, lines as well as fill areas such as polygons.

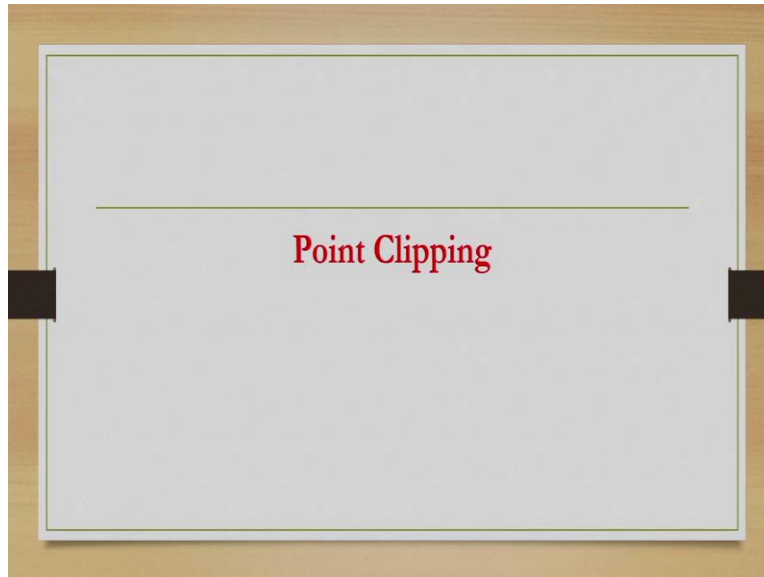
(Refer Slide Time: 10:34)



So, then our objective boils down to performing the clipping operation for points, lines, and fill areas with respect to the window. So, the scenario is that we have a clipping window or view window and we have objects that are projected on the view plane and we want to clip the

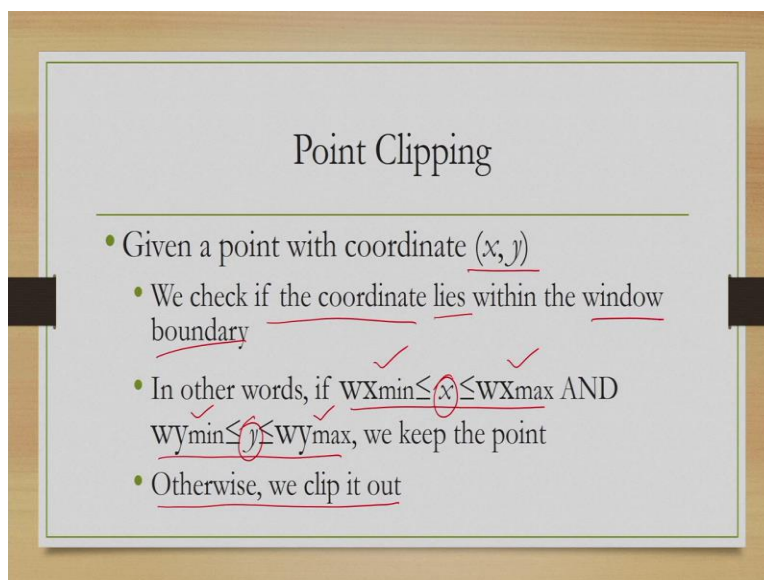
projected objects against this window where the projection takes place in the form of points, lines or fill areas.

(Refer Slide Time: 11:18)



Let us start with the simplest of all clipping, that is point clipping. How to clip a point against the view window.

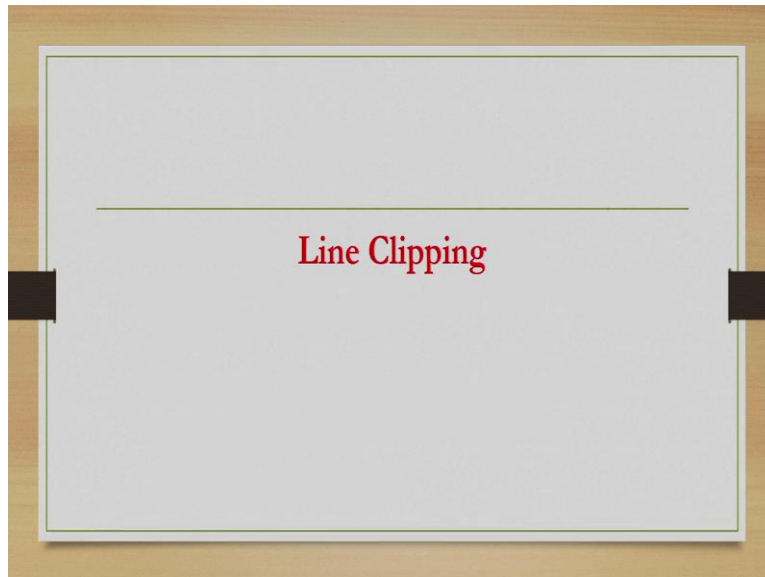
(Refer Slide Time: 11:31)



Let us assume that we are given a point with a coordinate x, y . Now the clipping idea is simple, we have to check whether the point is within the window or outside. So, what we have to do is

simply have to check if the coordinate values lie within the window boundary. So, we need to perform these check-ins. So, here we are checking the x value against the window boundaries and y value here we are checking against the window boundaries to determine whether they are inside the boundary or outside. So, if it is inside the boundary, we keep the point otherwise we clip it out. That is a very simple idea.

(Refer Slide Time: 12:35)



More complicated is line clipping. Here we do not have a single point, we have a large number of points to consider. So, the corresponding algorithm would be more complicated than what we have seen for point clipping.

(Refer Slide Time: 13:00)

Line Clipping

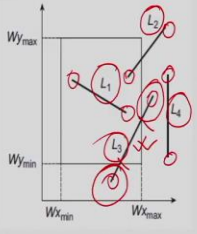
- More difficult
- Intuitive approach
 - Represent any line segment with its end points
 - Check end point positions to decide whether to clip

Let us first talk about a very simple intuitive approach. What we can do, we can represent any line segment with its end points and then we check the end point positions to decide whether to clip or not. So, we are representing a line with its end points. Then for each end point, we are applying this point clipping approach to check whether the endpoint is within the boundary or not. And after checking both the end points, we can come to a conclusion whether the line is within the window or not.

(Refer Slide Time: 13:49)

Line Clipping

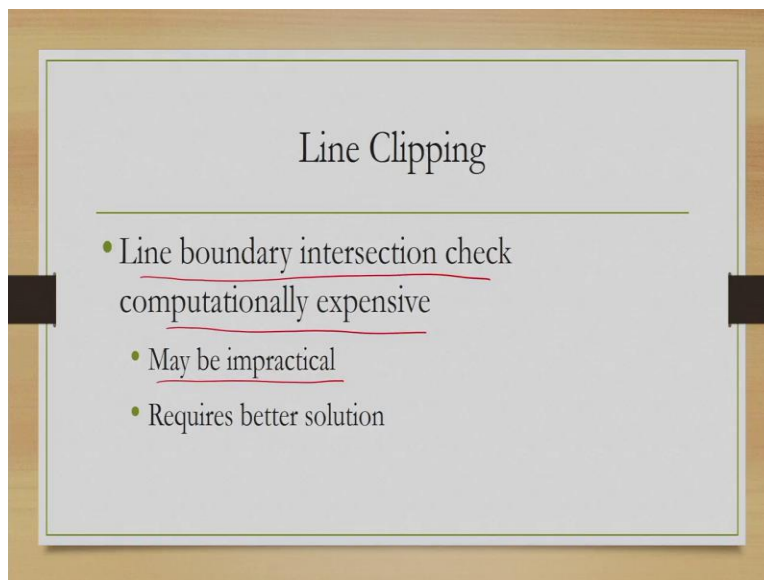
- If we follow this approach, either of three scenarios can occur
 - ✓ Both end points within window boundary - don't clip
 - ✓ One end point inside and the other outside - must be clipped
 - Both end points outside - can't say if whole line or part is outside; have to check line-boundary intersections



But there is a problem. If we follow this approach, there can be either of the three scenarios. In the first case, both end points may lie within the window boundary. Consider this line segment as L1. Here, both the end points are within the boundary. So we do not clip. In the second case, one end point is inside and the other end point is outside. Consider L2, here this endpoint is inside and this endpoint is outside. So, in that case it has to be clipped. And then there is a third scenario where both the endpoints are outside.

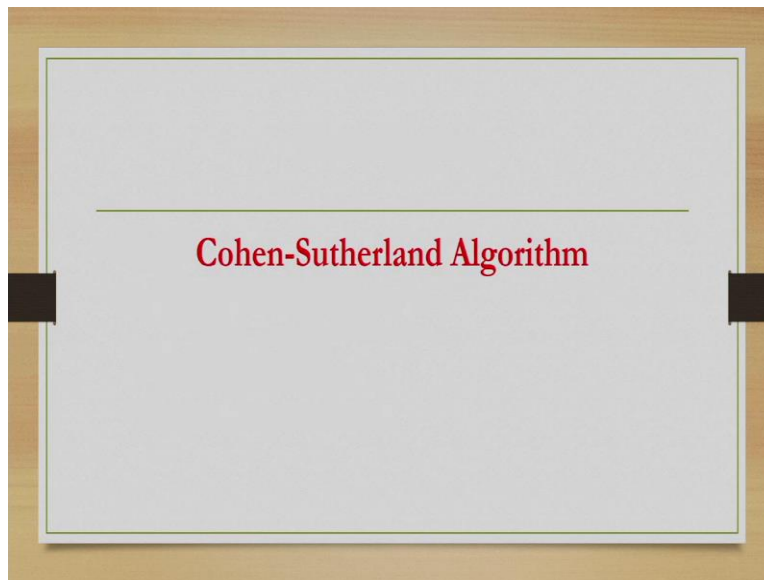
However, here we have a problem. Considered the line L4. In this case, both the end points are outside and the entire line is outside, so we can simply discard it. But in case of L3, here also both the end points are outside. However, a portion of it defined between these two intersection points is actually inside the view window. So, we cannot discard the line entirely. Instead, we have to clip out these outside parts and discard them whereas this inside part must be kept. So, in this case, then we have to check for line boundary intersections to determine whether some portion is inside the window or not.

(Refer Slide Time: 15:45)



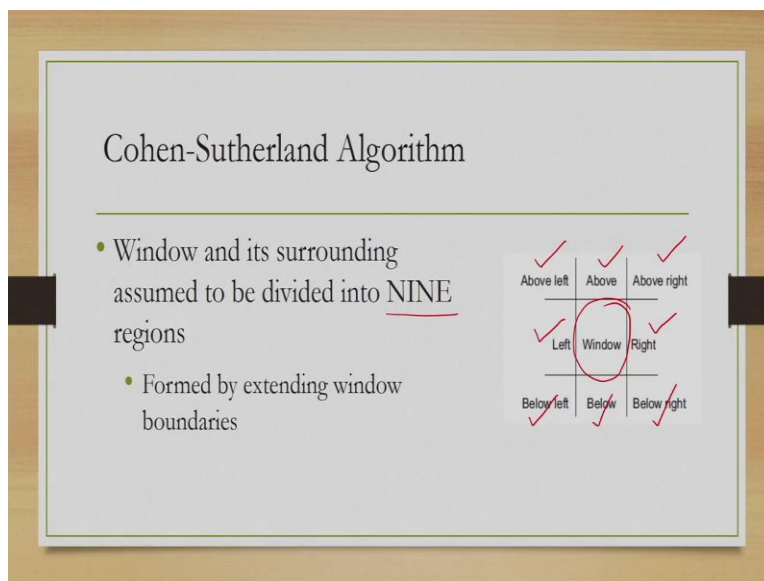
Now, this intersection checks are computationally expensive because they involve floating point operations. And in practical applications, every change of screen involves large number of such intersection checks, which may slow down the overall rendering process and it may even turn out to be impractical. So, we require some alternative, some better solution.

(Refer Slide Time: 16:25)



One such solution is provided by one algorithm called, Cohen-Sutherland Algorithm. Let us go through the steps of the algorithm.

(Refer Slide Time: 16:38)



So, in this algorithm, we assume a representation of the view plane. So here we assume that the window and its surrounding is divided into nine regions as shown here. So, this is the window, the central part, and then we have above left, above, above right with respect to the window, left, right, with respect to the window again, and below left, below, and below right with respect to the window position again. So, these nine regions are assumed to represent the view plane and

how we get the nine region, by extending the window boundaries as shown here. So, this is the first assumption.

(Refer Slide Time: 17:41)

Cohen-Sutherland Algorithm

- Each region has a 4-bit unique code - each bit indicates the position (above, below, right, or left, in that order from left to right) of the region w.r.t the window
- Ex - 1001 indicates corresponding region is above left of window

Above left	Above	Above right
Left	Window	Right
Below left	Below	Below right

1001	1000	1010
0001	0000	0010
0101	0100	0110

Above Below Right Left

1, 0

Then what we do, we assign a code to each of these regions. Now, there are nine regions. So, we require four bits to represent all the nine regions, which is very obvious and each region is given a unique code, a four bit unique code. Now, each bit indicates the position of the region with respect to the window. So, this is the nine region representation and this is the codes assigned to these nine regions. Note here that each code is unique.

So above left is represented by 1001, above is represented by 1000, above right is represented by 1010, left represented by 0001, right represented by 0010, below left represented by 0101, below represented by 0100, and below right represented by 0110. The central region or the window is represented by all zeros.

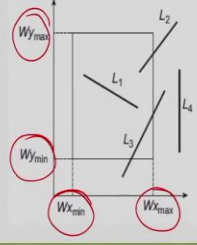
And the organization of the code looks something like this, where the leftmost bit indicates above location, next bit indicates below location, next bit indicates right, and the rightmost bit indicates left location. So, each location can take either 1 or 0, 1 means it is above, 0 means it is below. For example, consider 1001. So, we have above 1, below 0, right 0, and left 1, that means the region is above left as shown here because these two bits are 1, whereas below and right bits are 0. So that is how the codes are assigned to each region.

(Refer Slide Time: 20:26)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Step 1 - assign region codes to the end points
 - Let an end point be denoted by $P(x, y)$ and the window be specified by $(X_{min}, X_{max}, Y_{min}, Y_{max})$



Now that is the assumption and assignment of code. Then the algorithm starts working once those things are done. So, in step one of the algorithm we assign region codes to the end points of the line segment. So given a line segment, we first have to assign its endpoints, the corresponding region codes. How we can do that, assume that the end point, one end point is denoted by $P(x, y)$ and the window is specified by these boundary values $X_{min}, X_{max}, Y_{min}, Y_{max}$. So, these are the boundary values which specifies the window.

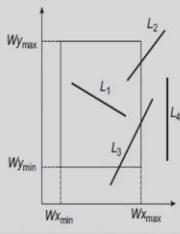
(Refer Slide Time: 21:30)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- We can determine region code of P through a simple check
 - Bit 3 = $\text{sign}(y - Y_{max})$
 - Bit 2 = $\text{sign}(Y_{min} - y)$
 - Bit 1 = $\text{sign}(x - X_{max})$
 - Bit 0 = $\text{sign}(X_{min} - x)$

$\text{sign}(a) = 1$ if a is positive, 0 otherwise



Then what we do, we perform a simple check to determine the region code of the endpoint P. What are those checks, so we check the sign of the difference between Y and Y_{max} . P has coordinates x and y and X_{max} , Y_{max} , X_{min} , Y_{min} are the window boundaries. So, we take the difference of Y and Y_{max} and check its sign. Now they sign up this quantity, will give us a result 1 if this quantity is positive, otherwise it is 0 in case of negative. So, we do that for each bit position. So, for bit 3, we checked the difference between Y and Y_{max} .

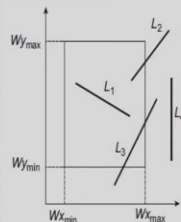
For bit 2, we check the difference between Y_{min} and Y. For bit 1 we checked the difference between X and X_{max} and for bit 0 we checked the difference between X_{min} and X and take their sign and then apply this rule to get the actual bit value.

(Refer Slide Time 22:54)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Step 2 – more checks and action
 - If both end point codes 0000, line **completely inside** - Retain line
 - If logical AND (i.e., bitwise AND) of end point codes **not equal to 0000**, line **completely outside** - Discard entire line



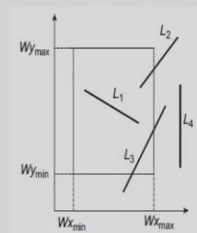
So, the first step is assignment of region codes to the end points. In step 2, we perform more checks on the region codes that are assigned to the endpoints and then we take action. So, if both endpoint codes turn out to be 0000, that means the line is completely inside. So, we retain the line. Now, if logical AND or the bitwise AND operation on the endpoint codes turn out to be not equal to 0000, then the line is completely outside and we discard the entire line. Note that, here we do not need to check for intersection.

(Refer Slide Time: 23:48)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- Step 3 – when none of these cases occur, the line is partially inside the window and we need to clip it



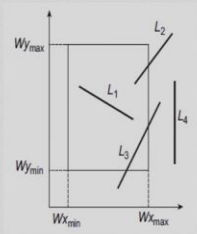
In the next step, when none of the cases that we just discussed in step 2 occur, we know that the line is partially inside the window and we need to clip it. So, in step 2 we perform checks to decide whether the line is completely inside or completely outside and accordingly we take step. Then if none of these conditions satisfy that means the line is partially inside and we need to clip it, that we do in step 3.

(Refer Slide Time: 24:26)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- For clipping, we need to calculate boundary - line intersection point
 - Done by taking one end point and following some order for checking, e.g., above, below, right, and left
- For each boundary, we compare corresponding bit values of end point region codes



So for clipping, we need to calculate the boundary and line intersection point. So, here we cannot avoid calculation of the intersections points. And that we can do in different ways, one possible

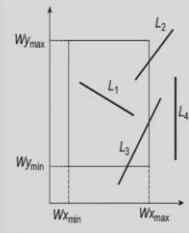
way can be take the boundaries in a particular order. For example, first above boundaries then below boundary, then right boundary, then left boundary and so on, any order is fine. And for each boundary compare the corresponding bit values of end point region codes to decide whether the line is crossing the boundary or not.

(Refer Slide Time: 25:25)

Cohen-Sutherland Algorithm

1001	1000	1010
0001	0000	0010
0101	0100	0110

- If they are **not same**, line intersects that particular boundary
 - Using line equation, determine intersection point and assign region code to the intersection point as before
 - In the process, we **discard** line segment outside window



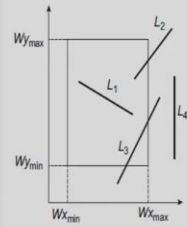
If the region codes are not same then the line intersects that particular boundary, then we form the line equation from the end points and determine the intersection point by solving the equations. And then we assign region code to the intersection point, as we have done earlier. And we do it for all boundaries and in this process, we discard the line segment that lies outside the window.

(Refer Slide Time: 26:06)

Cohen-Sutherland Algorithm

- Next, we compare the two new end points to see if they are completely inside the window
- If not, we take the other end point and repeat the process

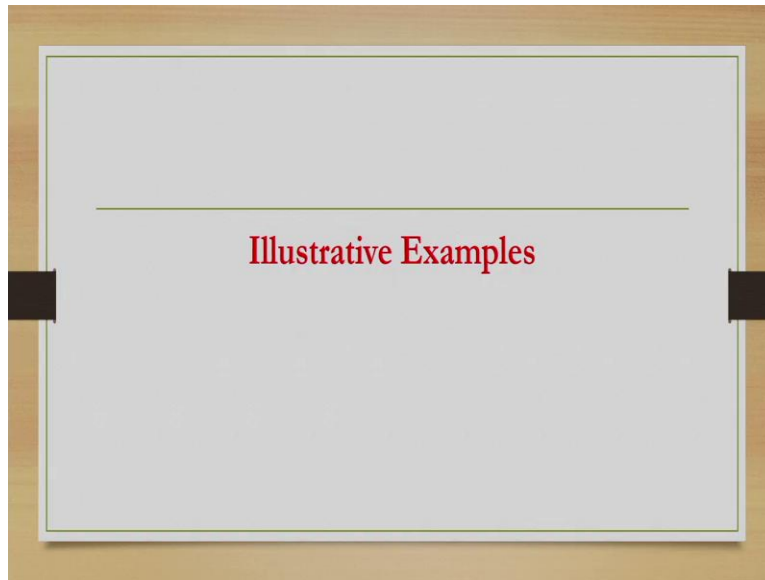
1001	1000	1010
0001	0000	0010
0101	0100	0110



Now we have a new region code that is the intersection point and the other end point with respect to a particular boundary. So, we compare the two new end points to see if they are completely inside the window. If they are, then of course we keep that, if not then we take the other end point and repeat the process. So, what we do in step 3, in step 3 we go for clipping the line. We start with one end point; assign a region code which is already there, we check the region code with respect to all the boundaries following a particular sequence and based on that check we determine if the line is intersecting a particular boundary.

If that is so, then we use the line equations and the boundary equations to solve for the intersection point. Then we assign a new code, the way we did before to this intersection point and the intersection point and the end point clearly lies outside the clipping window and we discard it. Then we have two new end points that is the intersection point and the original remaining endpoint. We check if it is completely inside the window if it is then we keep it, otherwise we repeat the process again for the other end point.

(Refer Slide Time: 27:55)



Let us try to understand whatever we have discussed so far in terms of some examples.

(Refer Slide Time: 28:03)

Example 1

- Consider the line segment AB
- Region codes of A

Bit 3 = $\text{sign}(3 - 4) = \text{sign}(-1) = 0$
Bit 2 = $\text{sign}(2 - 3) = \text{sign}(-1) = 0$
Bit 1 = $\text{sign}(5 - 4) = \text{sign}(1) = 1$
Bit 0 = $\text{sign}(2 - 5) = \text{sign}(-3) = 0$

The diagram shows a 2D coordinate system with a window defined by $X_{\min} = 2$, $X_{\max} = 4$, $Y_{\min} = 2$, and $Y_{\max} = 4$. The window is a square with vertices at (2,2), (4,2), (4,4), and (2,4). A line segment AB is drawn from point $A(5,3)$ to point $B(6,2)$. The window boundaries are labeled with their respective values: 2 and 4 on the x-axis, and 2 and 4 on the y-axis. The points $A(5,3)$ and $B(6,2)$ are also labeled. Red circles highlight the values 4, 2, 2, 4 on the axes and the points A and B.

So earlier we have shown how to assign the region codes, and based on that how to determine whether to clip a line or not. Let us consider this line segment defined between the end point A and B. And the window extension provided here, the X_{\min} is 2, X_{\max} is 4, Y_{\min} is 2, and Y_{\max} is 4. We check the sign and accordingly we assign region code to the end points. For example, for A, let us try to do this. Let us start with bit 3, here the corresponding expression is sign of 3 minus 4, which is sign of minus 1, since it is a negative quantity so bit 3 will be 0.

Similarly, bit 2 again negative quantity so it will be 0. Bit 1, this is a positive quantity 1 so it will yield 1. And bit 0 which is again a negative quantity and we are taking the sign of it, so it will result in 0. So, our region code will be given by these bits.

(Refer Slide Time: 29:35)

Example 1

- Consider the line segment AB
- Region codes of A
 - Bit 3 = $\text{sign}(3 - 4) = \text{sign}(-1) = 0$
 - Bit 2 = $\text{sign}(2 - 3) = \text{sign}(-1) = 0$
 - Bit 1 = $\text{sign}(5 - 4) = \text{sign}(1) = 1$
 - Bit 0 = $\text{sign}(2 - 5) = \text{sign}(-3) = 0$
- Code of $A = \underline{0010}$
- Similarly B code = 0010

Or 0010. In a similar way, we can check for B which is 0010, it will be the same.

(Refer Slide Time: 29:57)

Example 1

- Next step - series of checks
- First check fails - both end points not 0000
- Second check succeeds - logical AND of AB 0010 (i.e., $\neq 0000$)
- No further check required
 - Line totally outside window boundary
 - No clipping - discard

Now we go to step 2, series of checks. So, first check is whether both end points are equal to 0. Now they are not 0000, so first check fails. Second check is if they are logical AND is not equal

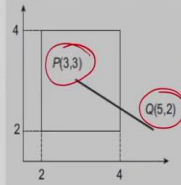
to 0000 which happens to be true in this case. So, we can be sure that this line is completely outside the window boundary. So, no further check is required and we can simply discard it. So, this particular line will be completely discarded just by checking the region codes, we managed to determine that it is completely outside.

(Refer Slide Time: 30:51)

Example 2

- Now consider line segment PQ
- First – determine end point codes
 - P code – 0000

Bit 3 =	$\text{sign}(3 - 4) = \text{sign}(-1) = 0$
Bit 2 =	$\text{sign}(2 - 3) = \text{sign}(-1) = 0$
Bit 1 =	$\text{sign}(3 - 4) = \text{sign}(-1) = 0$
Bit 0 =	$\text{sign}(2 - 3) = \text{sign}(-1) = 0$
 - Similarly Q code – 0010

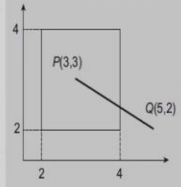


Now let us consider another line segment, that is given by the end points P and Q. As usual we will first determine the end point region codes. The P code will turn out to be 0000, as you can see here for bit 3 sign of a negative quantity 0. Again, bit 2 negative quantity 0, bit 1 negative quantity 0 and bit 0 sign of a negative quantity again 0. Similarly, we can determine the region code of Q, which turns out to be 0010. You can try yourself.

(Refer Slide Time: 31:43)

Example 2

- Next step - series of checks
- First check fails - both end points not 0000
- Second check also fails - logical AND of PQ is 0000
- Hence, need to determine line-boundary intersection

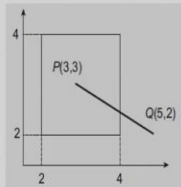


Then once the region codes are decided, we go for the series of checks. First check fails, the endpoints both endpoints are not 0000. Second check also fails, logical AND turns out to be 0000. So, it is not completely outside. So, we need to go for the third step and need to determine the intersections.

(Refer Slide Time: 32:15)

Example 2

- From the end points, we can derive the line equation as: $y = -\frac{1}{2}x + \frac{9}{2}$
- We check for intersection of this line with boundaries following the order: above, below, right, left
 - Bit 3 of P and Q same - line does not cross above boundary
 - Similarly, does not cross below boundary
 - Bit 1 are different - line crosses right boundary



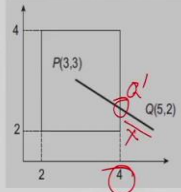
From the endpoints we can derive the line equation, as shown here. And then we check for intersection of this line with boundaries in the following order, above boundary first we will check with, then below boundary, then right boundary, and then left boundary. So, for above

boundary you can see that bit 3 which represents the above boundary of P and Q both the end points are same. So, the line does not cross above boundary. Similarly, you can see that that line does not cross below boundary. There is a difference of the bit values in case of bit 1, these values are different. So, we can conclude that the line crosses the right boundary.

(Refer Slide Time: 33:30)

Example 2

- Right boundary equation $x = 4$
- Putting this value in the line equation, we get intersection point as $Q'(4, 5/2)$
- We discard segment $Q'Q$ - since Q outside right boundary
- New line segment becomes PQ'



So, we have to find out the intersection point with the right boundary. Now, right boundary line equation is x equal to 4, as you can see here. And we put this value in the line equation to get the intersection point at 4, 5 by 2. This is the intersection point. Now, as you can see let us call this Q' . So, since the line crossed the boundary, so $Q'Q$ is outside the boundary because Q is outside. So, this segment is discarded and the new line segment becomes PQ' . These are the two new end points.

(Refer Slide Time: 34:26)

Example 2

- We determine region code of Q' - 0000
- Both P and Q' have region code 0000 - algorithm resets PQ by changing Q to Q'

Next, we determine the region code of Q' as we have seen earlier which turns out to be 0000. Now, as you can see both P and Q' have the same region code 0000. So, this segment is entirely within the window. And it is retained by changing Q to Q' .

(Refer Slide Time: 35:00)

Example 2

- Finally, we check left boundary
 - No intersection (bit 0 same for both end points)
- Algorithm returns PQ' (as clipped line) and stops

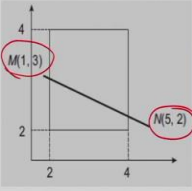
One thing was left, earlier we checked for above below right boundary, we did not check for left boundary. However, as we can see with this region codes of P and Q' because that is the new line segment bit 0 is same for both end point so there is no intersection. And so, at the end of the

algorithm, we get PQ' as the clipped line. So, that is how we can check for clipping following this algorithm.

(Refer Slide Time: 35:54)

Example 3

- Consider line segment MN
- Region code determination
 - $M - 0001$
 - Bit 3 = $\text{sign}(3 - 4) = \text{sign}(-1) = 0$
 - Bit 2 = $\text{sign}(2 - 3) = \text{sign}(-1) = 0$
 - Bit 1 = $\text{sign}(1 - 4) = \text{sign}(-3) = 0$
 - Bit 0 = $\text{sign}(2 - 1) = \text{sign}(1) = 1$
- Similarly $N - 0010$

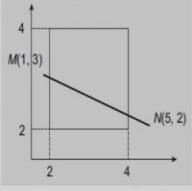


Let us see one more example. Now let us consider a slightly more complicated line segment given by M and N. As before we can determine the region code as shown here, for both the end points M and N which turns out to be for M it is 0001 and for N it is 0010.

(Refer Slide Time: 36:21)

Example 3

- Next step - series of checks
- First check fails - both end points not 0000
- Second check also fails - logical AND of MN is 0000
- Hence, need to determine line-boundary intersection points



Then in next step, we go for the checks, series of checks. Here also first check fails, both end points are not equal to 0000. Second check also fails. Logical AND results in 0000, so it is not 0000, so second check fails. Thus, we cannot completely keep it or completely discard it. So, there must be some intersection and we have to determine that intersection and clip the line. So, we need to determine line boundary intersection points and clip it.

(Refer Slide Time: 37:07)

Example 3

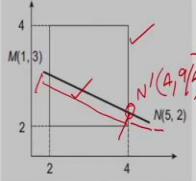
- Line equation (from end points) $y = -\frac{1}{4}x + \frac{13}{4}$
- we check for line intersection with boundaries in the order: above, below, right, left
 - Bit 3 of M and N same - line does not cross above boundary
 - Similarly, it does not cross below boundary (bit 2 same)
 - However, bit 1 are different - line crosses right boundary

For that we need to decide the line question from its end points, which is easy as shown here. Then we check for the line intersection with boundaries and following this order above, below, right, and left boundaries. So, bit 3 of M and N are same that means line does not cross above boundary. Similarly, it does not cross below boundary bit 2 is same. However, bit 1 are different and the line crosses right boundary.

(Refer Slide Time: 37:55)

Example 3

- Right boundary equation $x = 4$
- Putting this value in line equation, we get intersection point $N'(4, 9/4)$
- We discard segment $N'N$ since N outside right boundary
- Thus, new line segment becomes MN'
- We determine N' region code 0000

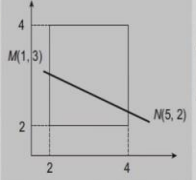


Then we check for intersection points, so the right boundary equation is x equal to 4. So, using this equation and the line equation we get the intersection point to be $N'(4, 9/4)$ by 4, that is here. Now this point is the intersection point between the right boundary and the line and since N is outside the right boundary, so as before we discard this segment $N'N$ and the new line segment becomes MN' , so this part. Then we decide the region code of the new end point N' which is 0000.

(Refer Slide Time: 39:05)

Example 3

- We now have two new end points M and N' with codes 0001 and 0000
- Boundary check now performed for left boundary (remaining one)
- Bit values are not same - we check for intersection of MN' with left boundary

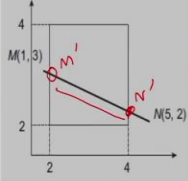


Thus, we now have two new endpoints M and N'. M has code 0001 and N' has code 0000. Now earlier we checked for above, below, and right boundary, left boundary check was pending, we will do that next. Here if we check, we can see that for the left boundary, the bit values are not same, that means again there is an intersection with the left boundary and we check for that intersection points between the left boundary and the new line segment given by the end points M and N'.

(Refer Slide Time: 39:59)

Example 3

- Left boundary equation $x = 2$
- Putting this value in line equation, we get intersection point $M'(2, 11/4)$
- We discard segment MM' . M outside left boundary
- New line segment becomes $M'N'$
- We determine M' code - 0000 .

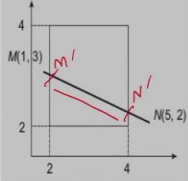


Now, we know left boundary equation is $x=2$. We use this equation and the line equation to get the intersection point M', which is 2 and 11/4, here. Now the point M is outside the left boundary that we already know, that means on the left side, so we discard this segment MM'. And new line segment becomes M'N' between these two points, this is M' and this one is N'. So, we now decide or determine the region code of this new endpoint M' which is 0000.

(Refer Slide Time: 40:52)

Example 3

- Both M' and N' have code 0000
 - Algorithm resets line segment to $M'N'$
- No more boundary remains to be checked - algorithm returns $M'N'$ (as clipped line) and stops



Then we go for checking the N point region codes again, step 2 of the algorithm and we find that M' and N' are same region code 0000, that means this entire line segment M' and N' is within the window. So, the algorithm resets the line segment to $M'N'$ and we have already checked for all the boundaries, so no more boundary remains to be checked. And the algorithm returns this line segment as the clipped line segment and stops. That is how the algorithm works.

(Refer Slide Time: 41:44)

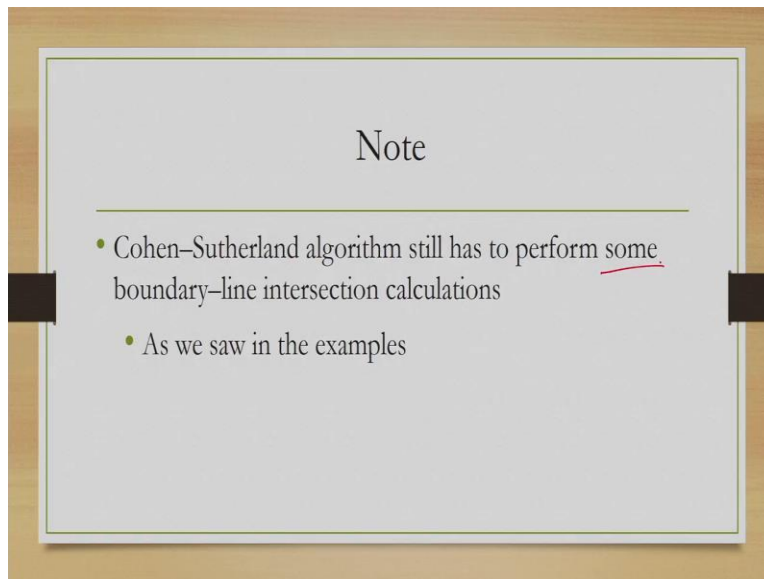
Note

So to summarize, the Cohen Sutherland algorithm is designed to reduce intersection calculations. If we go for intuitive method then when the line is completely inside or completely outside, it is

difficult to tell. So, for every line we have to go for intersection checks, that is not required in case of Cohen Sutherland method.

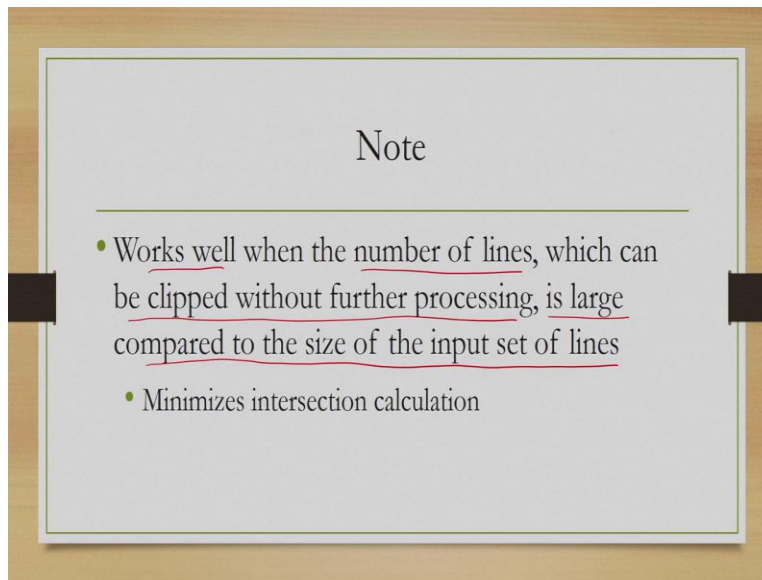
Here, we assign some region codes to the line endpoints and based on the region codes we can decide whether the line is completely inside or completely outside. In those cases, we do not need to go for intersection checking. However, if the line is not completely inside or completely outside, we know that there is some intersection and we need to clip, there we have to go for some intersection checks and find out the intersection points. So, some amount of intersection calculation still remains.

(Refer Slide Time: 42:48)



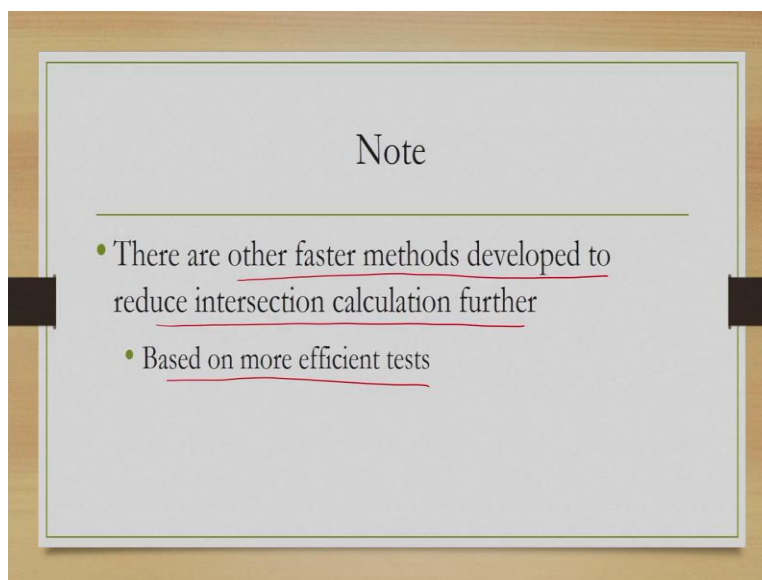
But it reduce the calculation to a great extent, which is beneficial to render complexions faster. But as I said, it still retains some intersection calculations.

(Refer Slide Time: 43:12)



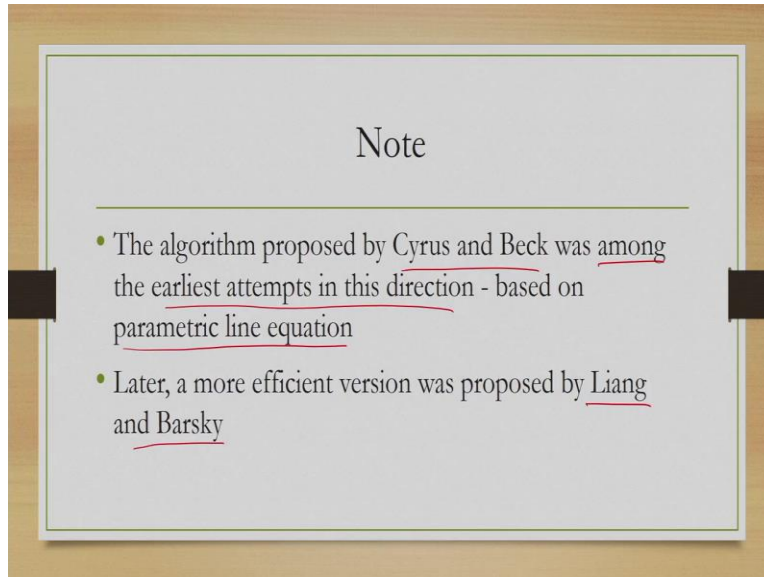
So, this particular algorithm works well when the number of lines which can be clipped without further processing is large compared to the size of the input set of lines. So, when the number of lines that can be clipped without further processing is large, then clearly Cohen Sutherland works much better because no intersection calculation is involved. But if that is not so, then it still has some problem.

(Refer Slide Time: 43:52)



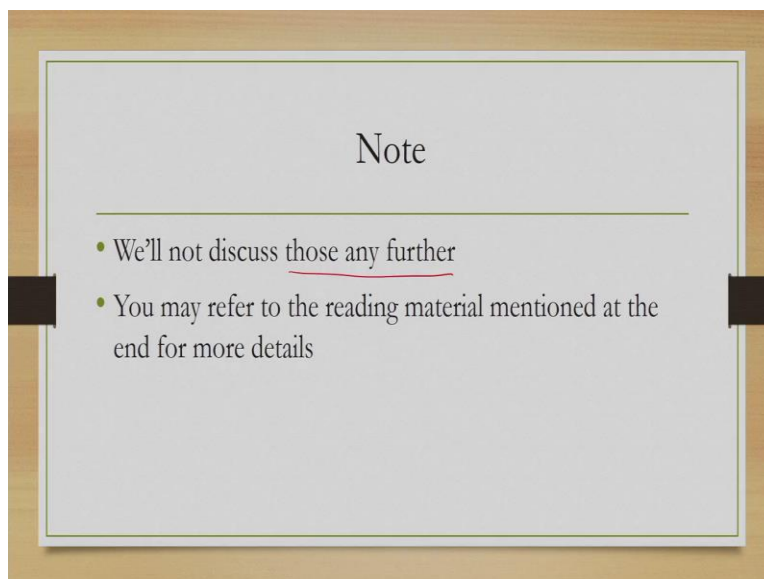
There are in fact other faster methods that are developed to reduce intersections calculation further and those methods are developed based on more efficient tests which do not require intersection or complex floating point operations.

(Refer Slide Time: 44:15)



There was one algorithm proposed by Cyrus and Beck, which was among the earliest attempts in this direction and which relied on parametric line equations. Later, a more efficient version was proposed by Liang and Barsky.

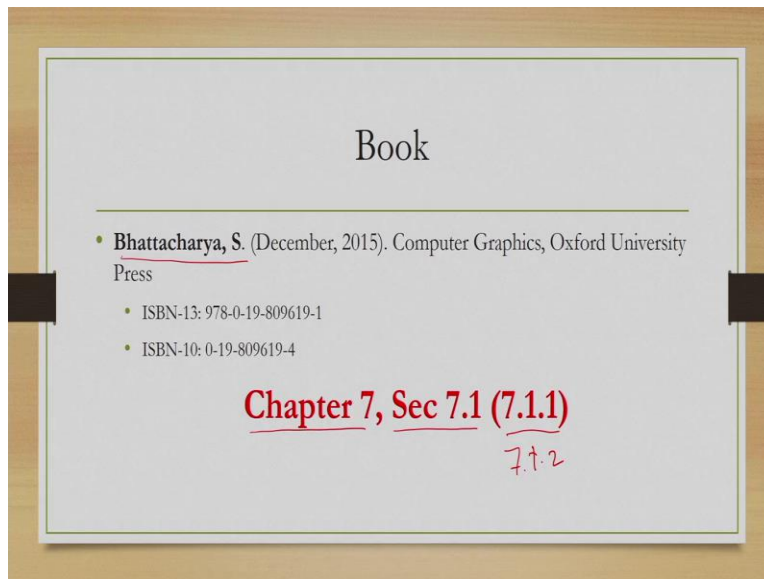
(Refer Slide Time: 44:44)



However, in this course we will not discuss those algorithms any further. If you are interested then you may refer to the reading material that will be mentioned next. So to summarize, today we have learned about clipping in 2D, clipping means discarding objects that are outside the view volume. In case of 2D clipping we want to discard lines or points that are outside the clipping window. Later on, we will see how to discard fill area also that are outside clipping window. And in order to do that, we rely on some algorithm to reduce extensive intersection point calculations.

One such algorithm we have learned today, that is Cohen Sutherland algorithm. It is quite efficient however it still retains some amount of complex calculations which can be alleviated with other more efficient algorithms, namely by Cyrus and Beck or by Liang and Barsky. So, those algorithms we have not discussed.

(Refer Slide Time: 46:07)



If you want to learn about those you may refer to the reading material that is mentioned in this slide. So, you may refer to this book. Go through chapter 7, section 7.1, today we discussed section 7.1.1. However, if you are interested to learn about Liang Barsky algorithm, then you can go through the next section as well, section 7.1.2 also. We will not discuss that algorithm but you may go through it, if you want more information. That is all for today. Thank you and goodbye.