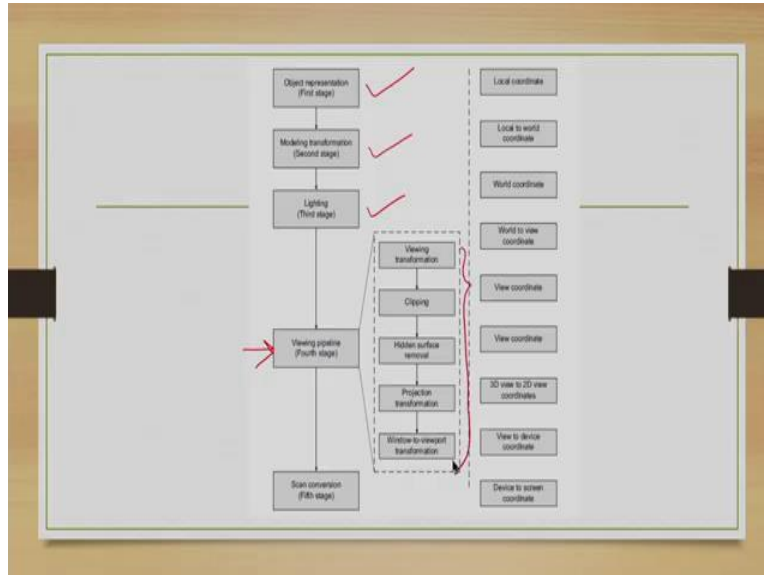


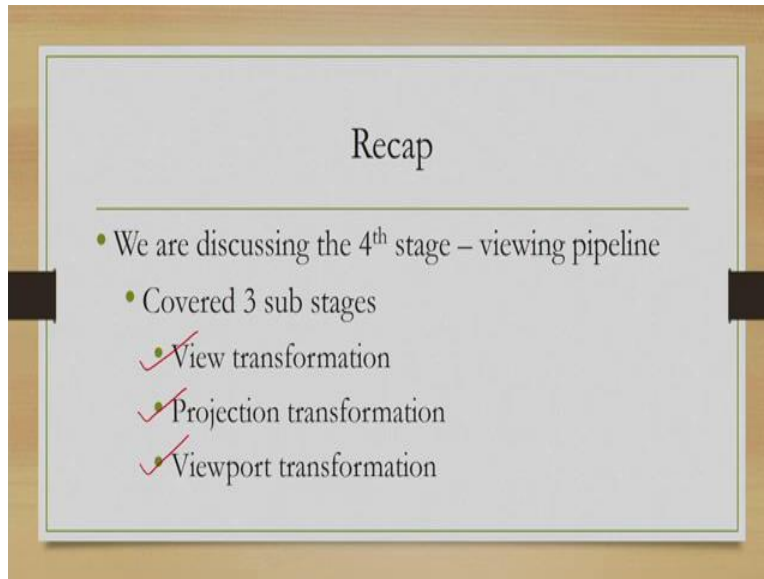
Computer Graphics
Professor Dr. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Lecture 22
2D Fill-area Clipping and 3D Clipping

(Refer Slide Time: 00:46)



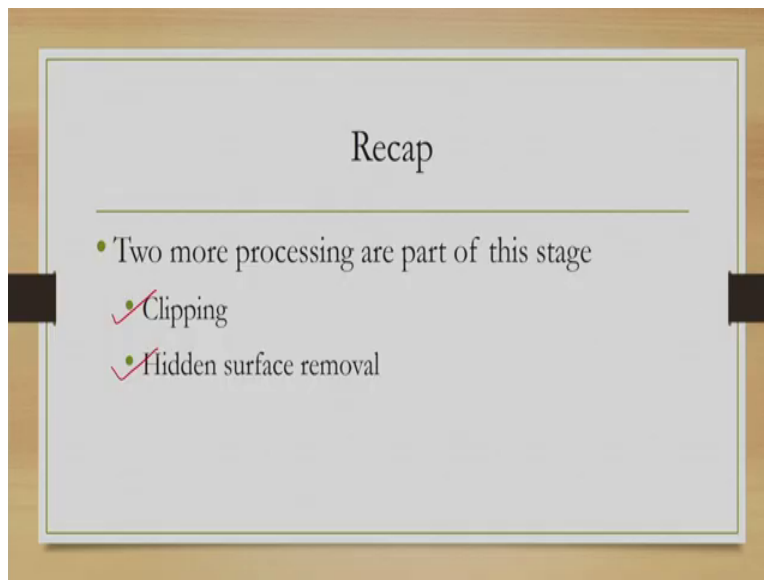
Hello and welcome to lecture number 22 in the course Computer Graphics. We are currently discussing the 3D graphics pipeline. And the pipeline has got 5 stages. We have already discussed object representation that is the first stage. Then modelling transformations - second stage. Lighting or assigning colour - third stage. Currently, we are in the fourth stage that is viewing pipeline. As you can see, it consists of 5 sub-stages. We have already discussed few of those and continuing our discussion on the remaining ones.

(Refer Slide Time: 01:20)



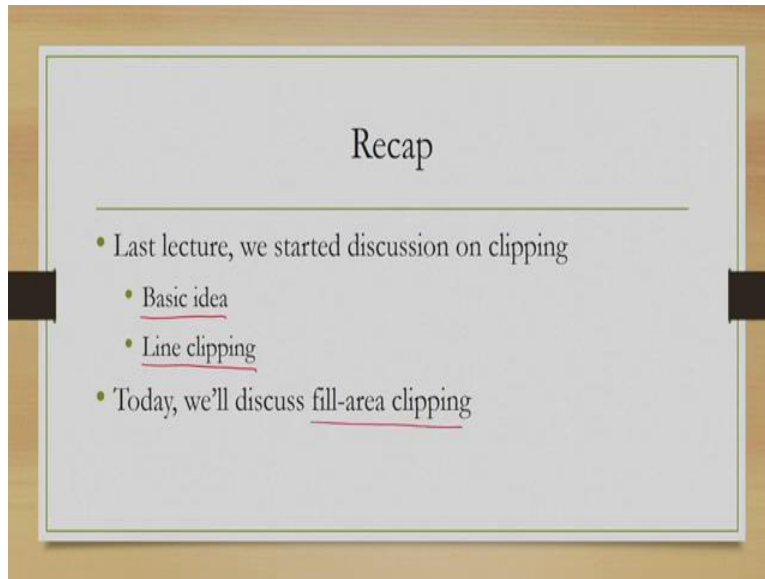
So, among those sub-stages we have already discussed earlier. View transformation, projection transformation and viewport transformation.

(Refer Slide Time: 01:34)



Two more operations are there, as we have seen in the pipeline; clipping and hidden surface removal. Among them currently we are discussing clipping.

(Refer Slide Time: 01:48)



So, in the last lecture, we introduced the basic idea of clipping and also discussed 2D line clipping. So, will continue our discussion on clipping. Today, we are going to discuss fill area clipping as well as 3D clipping.

(Refer Slide Time: 02:12)



Fill-Area Clipping

- Earlier, we discussed algorithms for clipping lines
- In many situations, we have to clip polygons against the window


So, what is this fill area clipping? So, as we mentioned, when we talk of clipping, there is a clipping window and earlier we have discussed how to clip points and lines against this window. However, when we project objects the projection maybe in the form of a fill area such as a polygon where there is a boundary.

Now clipping a filled area is different than clipping a point or a line, as we shall see in today's lecture. In fact, such situations are quite frequent in practice where we have to clip polygons against the clipping window. So, it requires some mechanism to do that.

(Refer Slide Time: 03:12)

Fill-Area Clipping

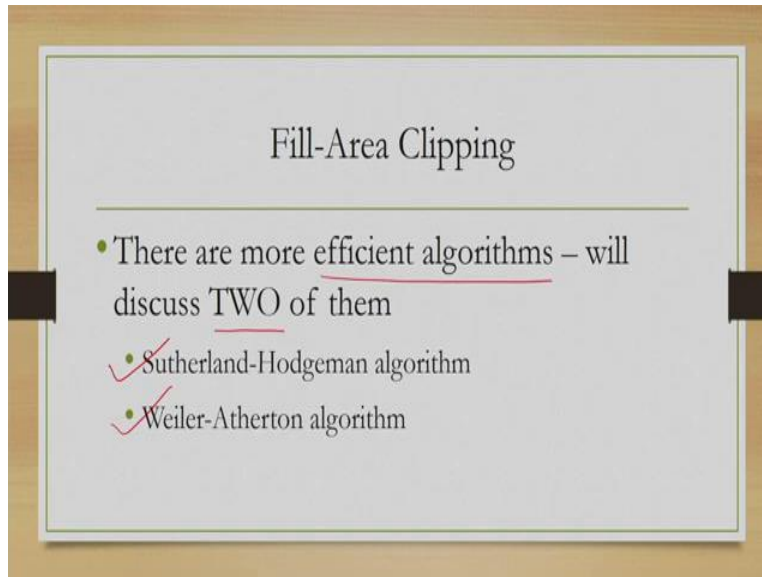
- Can use line clippers to the individual edges of the polygon (in principle)
- Not necessarily efficient and better



Now, what can be a very obvious and straightforward approach, let us try to understand the situation. Suppose this is our clipping window and we are given a polygon, something like this after projection say this triangle. So we have to keep this part which is inside the clipping window, which I am showing with shade and we have to clip out the, outside part. How we can do that?

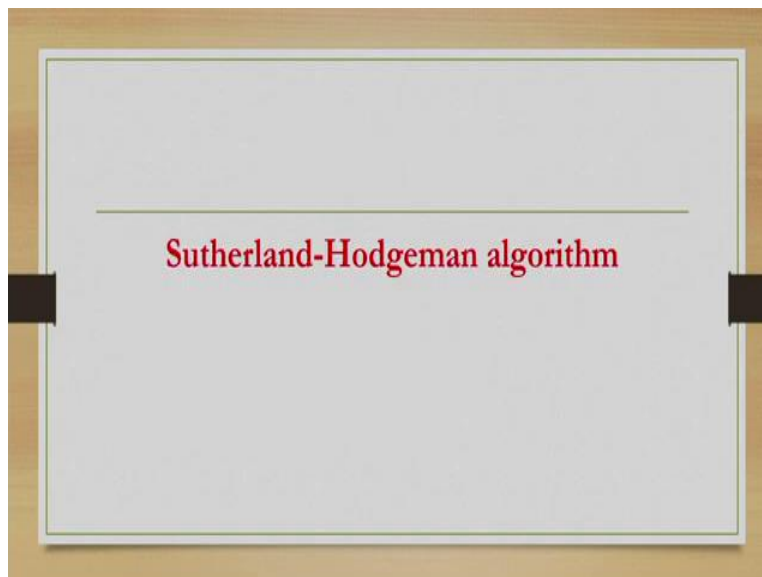
One way can be to use the line clippers that we discussed in earlier lecture for each of the edge, like here is one edge, one edge, one edge of the field area. And then perform clipping on the edges and decide on the clipped region. However, as you can see from this example, that is not necessarily easy, efficient and going to give us a good approach. Sometimes it is even difficult to understand how it works

(Refer Slide Time: 04:45)



Instead, we require better approaches. There are in fact, many efficient algorithms proposed for the purpose. In this lecture we are going to discuss two of those approaches. One is Sutherland-Hodgeman algorithm and the other one is Weiler-Atherton algorithm. Let us try to understand these algorithms.

(Refer Slide Time: 05:13)



Sutherland-Hodgeman Algorithm

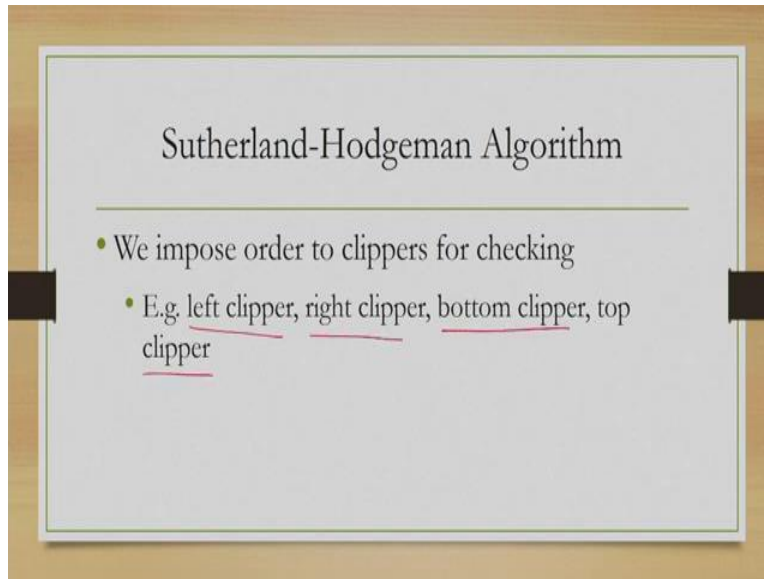
- We start with four **clippers** or lines that define window boundaries
- Each clipper takes as input a list of ordered pair of vertices (i.e., edges) and produces another list as output



We will start with the Sutherland-Hodgeman algorithm, what this algorithm does? Here in this algorithm we start with 4 clippers. Now, these clippers are essentially the lines that define the window boundary. For example, if this is my window boundary, then each of these lines defining the boundary is a clipper.

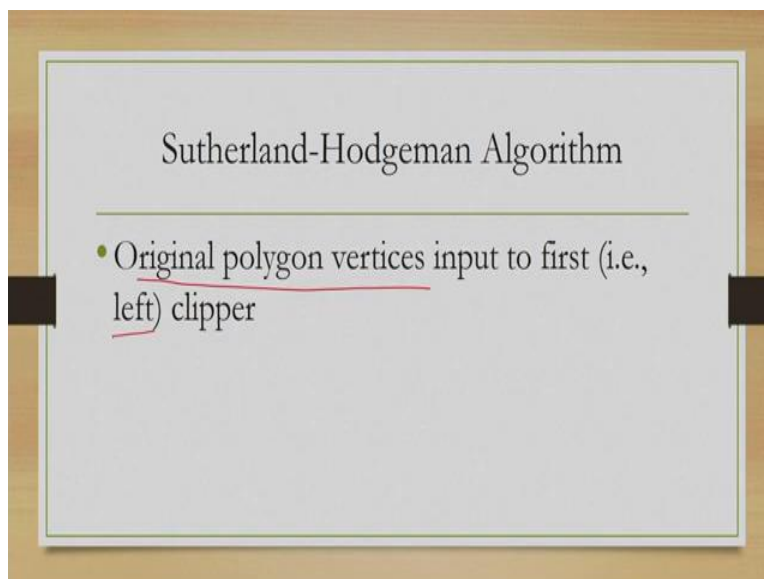
So, there are 4 clippers in 2D clipping that is right, left, above and below. Now, each clipper takes as input a list of ordered pair of vertices which essentially indicate the edges, each pair of vertex indicate the edge. And from that input list it produces another list of output vertices that is the basic idea. So, there are 4 clippers, each clipper takes as input a list of ordered pair of vertices where each pair of vertices represent an edge. And then it performs some operations to produce an output list of vertices.

(Refer Slide Time: 06:55)



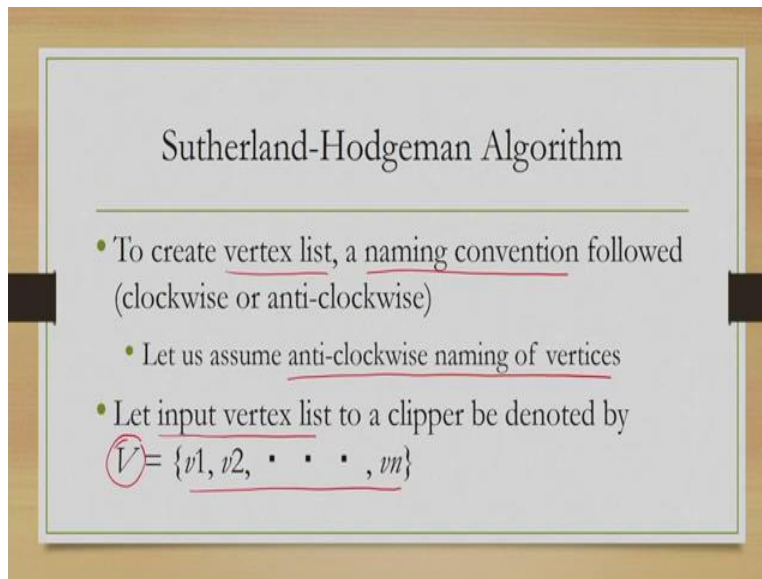
Now, when we perform these operations, we impose some order of checking against each clipper that can be any order. Here in this discussion will assume the order left clipper first, then right clipper, then bottom clipper, and at the end the top or above clipper.

(Refer Slide Time: 07:20)



Now, as we said we start with the left clipper. So, its input set is the original polygon vertices or in other words, the original polygon edges represented by the pair of vertices that is the input set to the first or the left clipper.

(Refer Slide Time: 07:44)

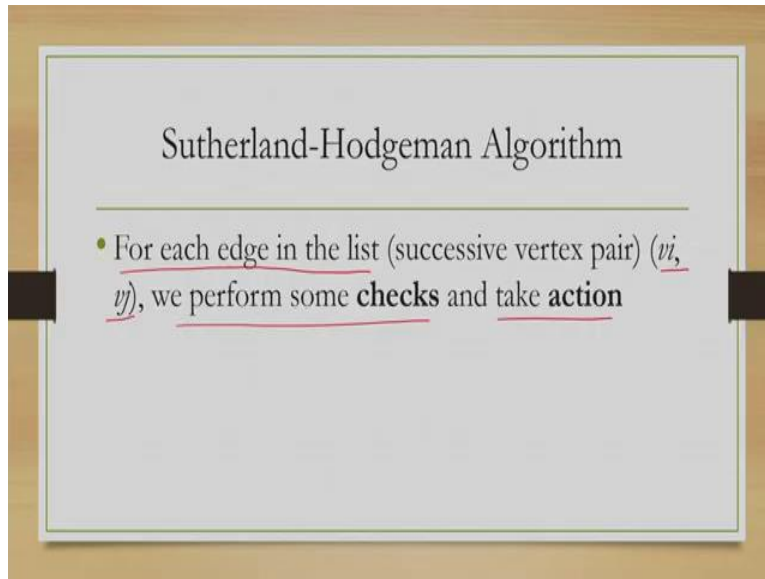


Sutherland-Hodgeman Algorithm

- To create vertex list, a naming convention followed (clockwise or anti-clockwise)
 - Let us assume anti-clockwise naming of vertices
- Let input vertex list to a clipper be denoted by $V = \{v_1, v_2, \dots, v_m\}$

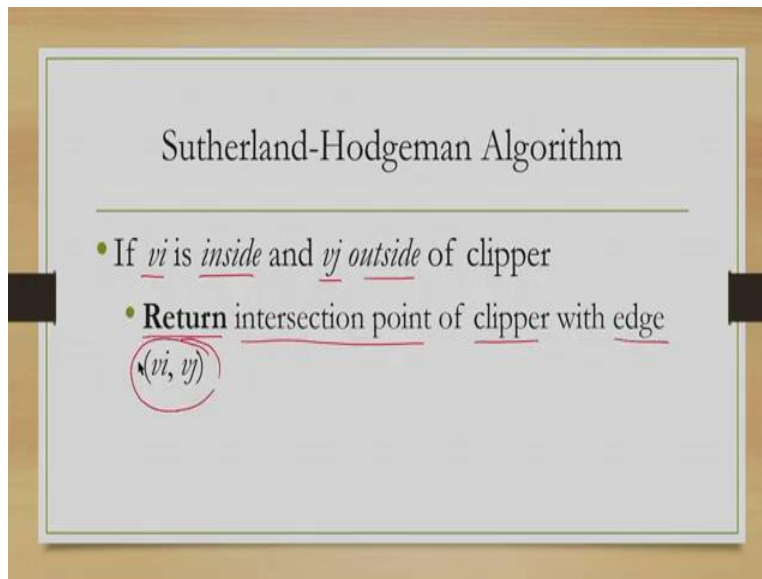
Now, to create a vertex list as output or also to provide the input vertex list, we need to follow a naming convention, whether to name the vertices in a clockwise manner or anticlockwise manner. Here again, we will assume that we will follow an anticlockwise naming of vertices. With these conventions, let us denote input vertex list to a clipper by the set V having these vertices.

(Refer Slide Time: 08:31)



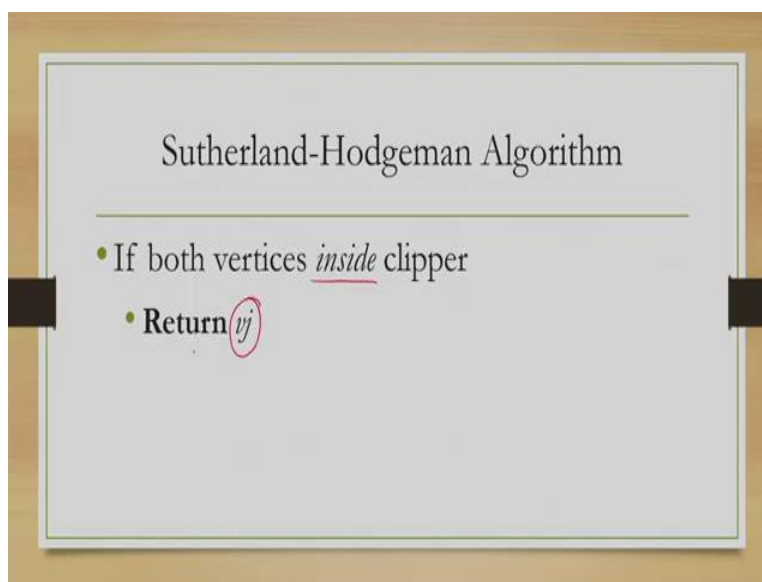
Now, for each edge or the pair of vertices in the list denoted by v_i, v_j . We perform some checks and based on the check results, we take some action. So, what are those checks?

(Refer Slide Time: 08:52)



Sutherland-Hodgeman Algorithm

- If v_i is inside and v_j outside of clipper
- **Return** intersection point of clipper with edge
 (v_i, v_j)



Sutherland-Hodgeman Algorithm

- If both vertices inside clipper
- **Return** v_j

If v_i is inside and v_j is outside of the clipper then we return the intersection point of the clipper with the edge represented by the vertex pair v_i, v_j . If both vertices are inside the clipper, then we return v_j .

(Refer Slide Time: 09:25)

Sutherland-Hodgeman Algorithm

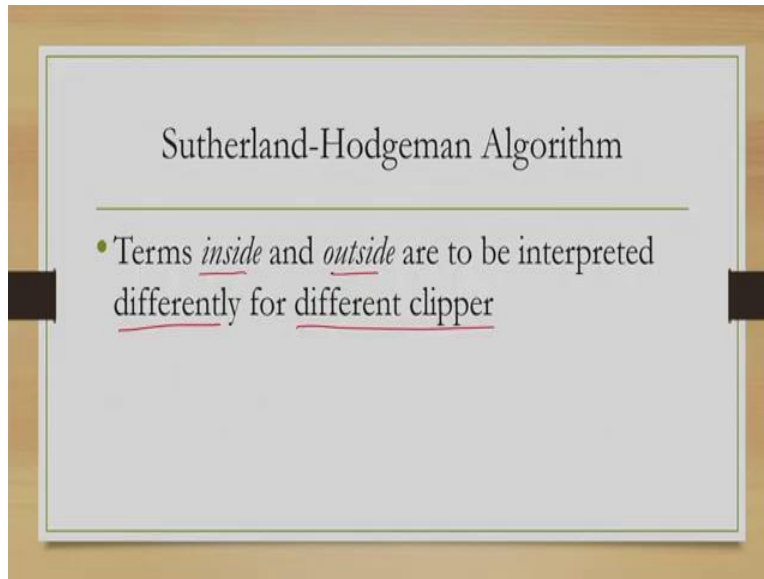
- If v_i *outside* and v_j *inside* of clipper
- **Return** intersection point of clipper with edge (v_i, v_j) and v_j

Sutherland-Hodgeman Algorithm

- If both vertices *outside* clipper
- **Return** NULL

If v_i is outside and v_j is inside of the clipper, then we return two things. One is the intersection point of the clipper with the edge represented by the pair v_i, v_j and also v_j . Both the things we return intersection point and v_j . And finally, if both vertices are outside the clipper then we do not return anything, we return null.

(Refer Slide Time: 10:05)



Now here we have use the terms inside and outside. So, how they are defined? In fact these terms are to be interpreted differently for different clipper. So, there is not a single meaning to these terms based on the clipper we define these terms.

(Refer Slide Time: 10:27)



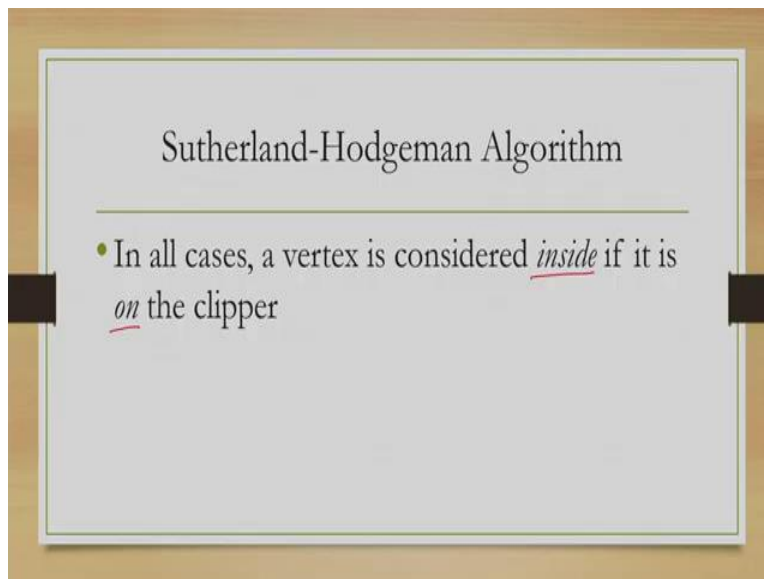
And let us now go through this definition for each of the 4 clippers. So, for the left clipper, when you talk of insight we mean that the vertex is on the right side of the clipper and when we talk of outside we mean that it is on the left side of the clipper. For right clipper it is just the opposite.

When the vertex is on the left side, we call it inside. Otherwise it is outside. For top clipper if a vertex is below the clipper that means it is inside.

Otherwise it is outside. And for a bottom clipper, it is again just the opposite of top clipper that means inside vertex means it is above the clipper, whereas outside means it is below. And how do we determine whether a vertex is on the right side or left side or above or below, just by considering the coordinates values, by comparing the coordinated values of the vertex with respect to the particular clipper.

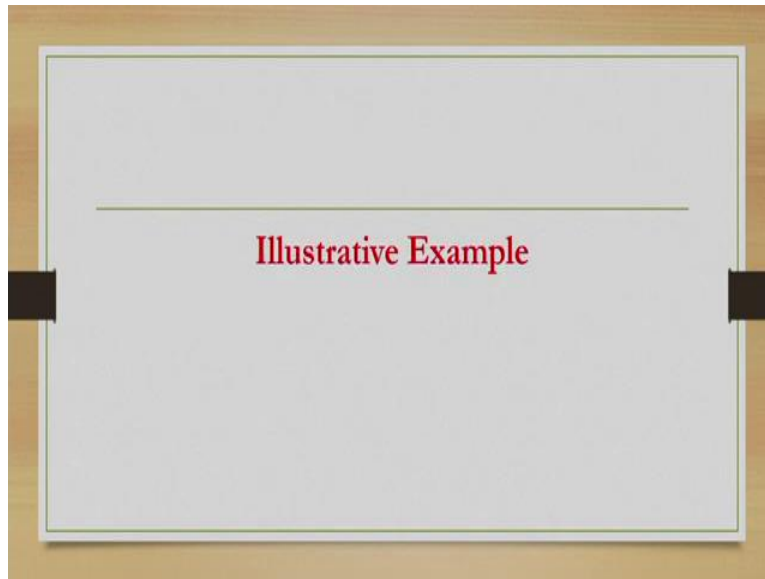
For example, suppose this is the top clipper, suppose it is equation is given by $x = y = 4$. Now suppose a point is denoted by (3, 5). Now we check here the y-value of the point that is 5, clearly 5 is greater than 4 which is the y-value of the boundary, top boundary. Then we can say that this point is outside because it is above the clipper. Similarly, we can determine the inside and outside based on comparing the x or y coordinate value of the vertex with respect to the clipper values.

(Refer Slide Time: 12:55)



If the vertex is on the clipper then it is considered inside in all the cases. So, for a left clipper inside means either it is on the right side or on the clipper, otherwise it is outside. And same is true for all other clippers.

(Refer Slide Time: 13:21)



Example

- Consider the polygon with vertices $\{1, 2, 3\}$ (named anti-clockwise). We wish to determine clipped polygon (i.e., polygon with vertices $\{2', 3', 3'', 1', 2\}$) following the Sutherland–Hodgeman algorithm

The diagram shows a 2D coordinate system with a red triangle representing the original polygon with vertices 1, 2, and 3. A red quadrilateral represents the clipped polygon with vertices 2', 3', 3'', and 1'. The clipping window is defined by a vertical line and a horizontal line. The vertices of the original polygon are labeled 1, 2, and 3. The vertices of the clipped polygon are labeled 2', 3', 3'', and 1'. A red checkmark is visible below the diagram.

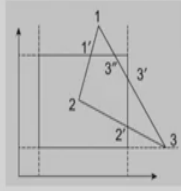
Now, let us try to understand this algorithm in terms of an illustrative example. Let us consider this situation here we have defined one clipping window and we have a fill area. Now this fill area is defined by the vertices $\{1, 2, 3\}$ as you can see here, we followed a counter clockwise or anticlockwise naming convention to list the vertices.

Our objective is to determine the clipped polygon. That is this polygon denoted by the vertices $\{2', 3', 3'', 1', 2\}$. And we use to do that by following the Sutherland-Hodgeman algorithm.

(Refer Slide Time: 14:44)

Example

- Check vertex list against each clipper
 - In the order left, right, top



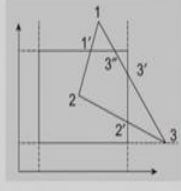
The diagram shows a triangle with vertices labeled 1, 2, and 3. A coordinate system is overlaid with horizontal and vertical dashed lines representing clipper boundaries. The vertices are projected onto these lines, with the projections labeled 1', 2', and 3'.

So, at the beginning we start with the left clipper. Then we check against right clipper, then top clipper and then bottom clipper.

(Refer Slide Time: 15:06)

Example

- **Left clipper** - input vertex list $V_{in} = \{1, 2, 3\}$, pair of vertices to be checked $\{1,2\}$, $\{2,3\}$, and $\{3,1\}$
- For each pair, we perform check
 - $\{1,2\}$ - both vertices on right side of left clipper (i.e., both inside); $V_{out} = \{2\}$
 - Similarly, after checking $\{2,3\}$, $V_{out} = \{2, 3\}$
 - After checking $\{3,1\}$, final list $V_{out} = \{2, 3, 1\}$



The diagram shows a triangle with vertices labeled 1, 2, and 3. A coordinate system is overlaid with horizontal and vertical dashed lines representing clipper boundaries. The vertices are projected onto these lines, with the projections labeled 1', 2', and 3'.

Let us see what happens after checking against the left clipper. So here the input vertex list is the original vertices that is $\{1, 2, 3\}$ which indicates the three edges represented by the vertex pair; $\{1, 2\}$; $\{2, 3\}$ and $\{3, 1\}$. So, for each pair we perform the check. For pair $\{1, 2\}$ we can see that

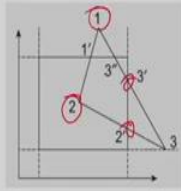
both the vertices are on the right side of the left clipper that means both are inside. So, V_{out} is 2 as per the algorithm.

Similarly, after checking for {2, 3} against the left clipper, we can see that the final V_{out} becomes {2, 3} taking into account 2. And after checking {3, 1} the final list becomes {2, 3, 1}. In all the cases against the left clipper all the vertices are inside.

(Refer Slide Time: 16:29)

Example

- **Right clipper** - $V_{in} = V_{out} = \{1, 2, 3\}$ and $V_{out} = \text{NULL}$; pair of vertices to be checked $\{1,2\}$, $\{2,3\}$, and $\{3,1\}$
- $\{1,2\}$ - both inside (i.e., left side of right clipper); $V_{out} = \{2\}$
- $\{2,3\}$ - vertex 2 is inside while vertex 3 outside
 - Compute intersection point $2'$ and set $V_{out} = \{2, 2'\}$
- $\{3,1\}$ - vertex 3 outside (right side) and vertex 1 inside (left side)
 - Calculate intersection point $3'$ and set $V_{out} = \{2, 2', 3', 1\}$



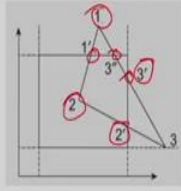
Now, let us check against right clipper. So, now the input vertex list is {1, 2, 3} same and initially V_{out} is NULL. So, pair of vertices has to be checked {1, 2; 2, 3} and {3, 1}, all the three edges we need to check. For 1, 2 both are inside the right clipper. We can check by comparing their coordinative values because both of them are on the left side of the right clipper. So, V_{out} is now 2. Then we check the pair {2, 3} here we can see that 2 is inside whereas 3 is outside. So, in that case we compute the intersection point $2'$ this point and then set V_{out} to be {2, $2'$ }.

Then we check {3, 1} here vertex 3 is outside because it is on the right side of the clipper, whereas 1 is inside because it is on the left side. So, here we calculate the intersection point $3'$ and then finalize the output vertex list as {2, $2'$, $3'$ (and) 1}; because in this case we return 1 also. So, then after checking against the right clipper, we get this output list; {2, $2'$, $3'$ (that means this point and), 1}.

(Refer Slide Time: 18:40)

Example

- **Top clipper** - $V_{in} = V_{out} = \{2, 2', 3', 1\}$, $V_{out} = \text{NULL}$;
pair of vertices to check $\{2, 2'\}$, $\{2', 3'\}$, $\{3', 1\}$, $\{1, 2\}$
- $\{2, 2'\}$ - both are inside (below clipper), $V_{out} = \{2'\}$
- $\{2', 3'\}$ - both inside; $V_{out} = \{2', 3'\}$
- $\{3', 1\}$ - $3'$ inside whereas 1 outside
 - Calculate intersection point $3''$, $V_{out} = \{2', 3', 3''\}$
- $\{1, 2\}$, 1 outside while 2 inside
 - Calculate intersection point $1'$, $V_{out} = \{2', 3', 3'', 1', 2\}$



Then we check against the top clipper. Now in this case, the V_{in} or the input vertex list is the output vertex list after checking against the right clipper. So that is $2, 2', 3', 1$. So, initially V_{out} is NULL. And the pair of vertices we need to check are $\{2, 2'\}$, $\{2', 3'\}$, $\{3', 1\}$ and $\{1, 2\}$.

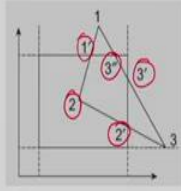
So, first we check $\{2, 2'\}$ against the top clipper and we find that both 2 and $2'$ are inside because both of them are below the clipper. So, output list becomes $2'$. Then we check the next vertex pair $\{2', 3'\}$ again $\{2', 3'\}$ both are below, so inside then V_{out} becomes $2'$ and $3'$.

Then we check $3', 1$ in this case, we see that $3'$ is inside, whereas 1 is outside. Then we calculate the intersection point $3''$ here and modify our output list to be $\{2', 3', (\text{and}) 3''\}$. Finally, we check $\{1, 2\}$. Here we see that 1 is outside whereas 2 is inside. Then again we calculate the intersection point $1'$ and modify V_{out} to be $\{2', 3', 3'', 1' (\text{and}) 2\}$. So, this is our output list after checking against top clipper, and this serves as the input list to the remaining clipper to be checked that is bottom clipper.

(Refer Slide Time: 20:43)

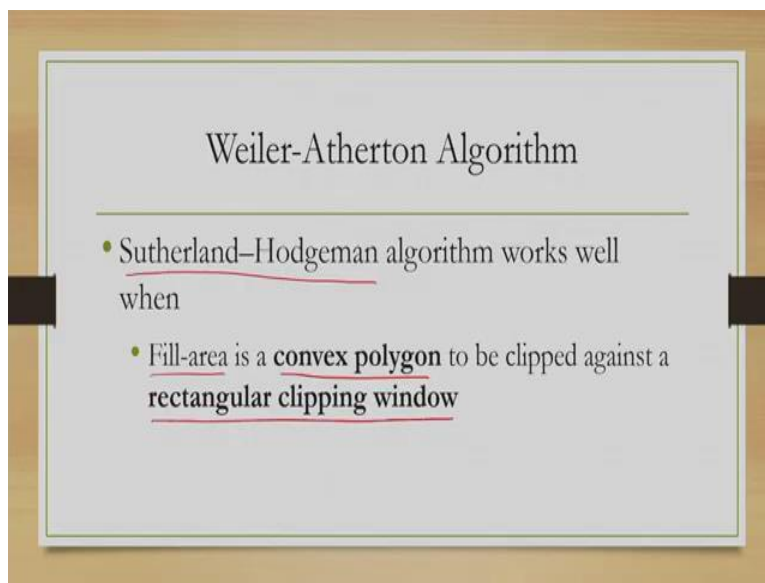
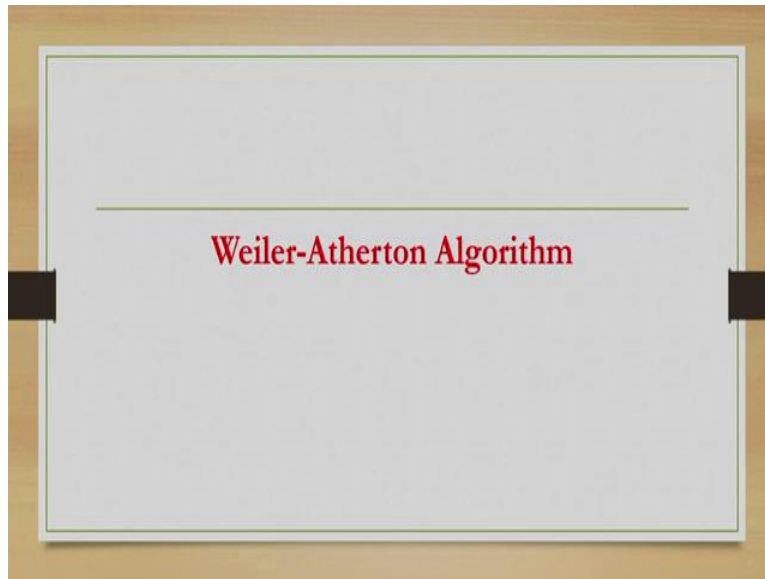
Example

- **Bottom clipper** - $V_{in} = V_{out} = \{2', 3', 3'', 1', 2\}$ and $V_{out} = \text{NULL}$.
- All vertices are inside (above the clipper) - output list becomes $V_{out} = \{2', 3', 3'', 1', 2\}$
- Also the output of the algorithm



This is the input list for the bottom clipper and output list is initially null and as we can see all these vertices 2, 2', 3', 3'' and 1' are inside because they are above the bottom clipper. So, the output list becomes the same that is $\{2', 3', 3'', 1', 2\}$. This is also the output of the algorithm, because here no more clipper are there to check against and the algorithm stops. So, at the end of the algorithm, we get this vertex list which represents the clipped region. That is how the algorithm works.

(Refer Slide Time: 21:43)



Now let us move to our next algorithm that is Weiler-Atherton algorithm. Now, the Sutherland-Hodgeman algorithm that we just discussed works well when the fill area is a convex polygon and it is to be clipped against a rectangular clipping window. So, if this condition satisfy then Sutherland-Hodgeman algorithm works well.

(Refer Slide Time: 22:16)

Weiler-Atherton Algorithm

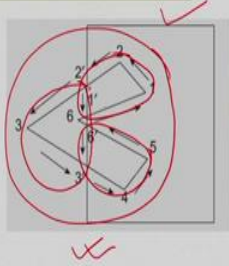
- Weiler-Atherton algorithm provides more general procedure
- Can be used for any polygon fill-area (concave or convex) against any polygonal clipping window

However that need not be the case always and that Weiler-Atherton algorithm provides a more general solution. This algorithm can be used for any polygon, either concave or convex against any polygonal clipping window. Need not be only a rectangle. Let us see how it works.

(Refer Slide Time: 22:44)

Weiler-Atherton Algorithm

- We start with processing **fill-area edges** in a particular order (typically anti-clockwise)



So, we will try to understand the algorithm in terms of an example rather than formal steps. Let us consider this scenario here we have a rectangular clipping window and a fill area. So, we will try to understand how the algorithm helps us identify the parts to be discarded that is this region

and the parts to be kept after clipping that is these two regions this one and this one. So, here we start with processing the fill area edges in a particular order, which is typically anticlockwise order. So, here we start with processing the fill area edges in a particular order which typically is anticlockwise order.

(Refer Slide Time: 24:00)

Weiler-Atherton Algorithm

- Continue along the edges till we encounter an edge that crosses to the **outside** of the clip window boundary
- For edges crossing into inside, we just record the intersection point

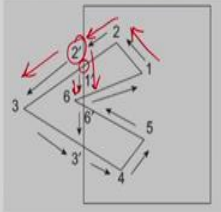
So, what we do in the processing, we check the edges one by one, continue along the edges till we encounter an edge that crosses to the outside of the clip window boundary. Let us start with this edge (1, 2) this edge. So, we check it whether it crosses the window boundary or not, that is our processing. It does not cross so we continue to the next stage, that is {2, 3} represented by the vertex pair {2, 3}.

Now this edge crosses to the outside of the window boundary. Note that here we are following anticlockwise order. If the edge does not cross to the outside instead if the edge is crossing into inside of the window then we just record by intersection point, whereas if the edge is crossing to the outside boundary, then we stop and perform some different action, what we do in that case.

(Refer Slide Time: 25:24)

Weiler-Atherton Algorithm

- At the intersection point, we make a detour
- We now follow the edges of the clip window (along the same direction maintaining the traversal order)



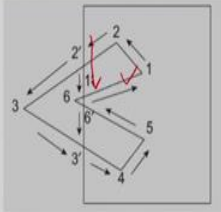
At the intersection point, we make a detour. So, here the intersection point is 2' this point. So, then we make a detour. We no longer continue along this direction. Instead what we do we now follow the edge of the clip window along the same direction, maintaining the traversal order.

So, now in this example so we will follow this anticlockwise direction and make a detour from here now along the window boundary, so here we will follow this order. So, essentially, how you are traversing, we initially traversed in this way then while traversing in this way found that this edge is crossing to the outside. So, then we traverse in this way instead of continuing along the edge.

(Refer Slide Time: 26:25)

Weiler-Atherton Algorithm

- We continue our traversal along the edges of the clip boundary
- Till we encounter another fill-area edge that crosses to the inside of the clip window

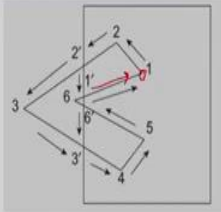


Now, this along the boundary traversal, we continue till we encounter another fill area edge that crosses to the inside of the clip window. So, here as you can see, if we follow a anticlockwise traversal, then this edge is actually crosses to the inside. So, the edge is 6, 1 denoted by the vertex pair 6, 1 which crosses to the inside of the window and we encountered it while traversing along the window boundary. At that point what we do?

(Refer Slide Time: 27:17)

Weiler-Atherton Algorithm

- The process continues till we encounter a previously processed intersection point



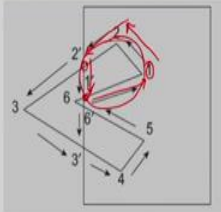
The diagram shows a polygon with vertices labeled 1 through 6. A vertical line intersects the polygon at two points, labeled 1' and 3'. Arrows indicate the direction of traversal along the polygon edges. A red arrow points to point 1', indicating the current position in the traversal process.

At that point, we resume the polygon edge traversal again along the same direction. So, we stop here and then again continue along the same direction till we encounter previously processed intersection point. So, here we continue up to point 1 because point one is already processed. So, we stop here.

(Refer Slide Time: 27:53)

Weiler-Atherton Algorithm

- TWO rules of traversal
 - From an intersection point due to outside-to-inside fill-area edge, follow fill-area polygon edges
 - From an intersection point due to inside-to-outside fill-area edge, follow window boundaries



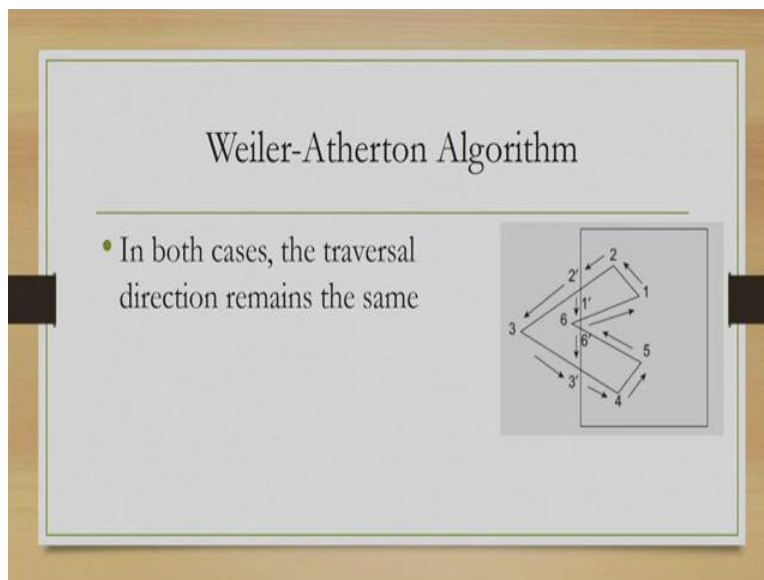
The diagram shows a polygon with vertices labeled 1 through 6. A vertical line intersects the polygon at two points, labeled 1' and 3'. Arrows indicate the direction of traversal along the polygon edges. A red arrow points to point 1', indicating the current position in the traversal process.

So, then after this part, we see that we started from here, then traversed up to this point, determined this intersection point, then traversed along this line up to this intersection point,

traversed back up to the originating point. So, there are two rules of traversal from an intersection point due to outside to inside fill area edge we should follow the polygon edges from an intersection point due to inside to outside fill area edge we should follow the window boundaries.

So, these are the rules we applied while performing the traversal. But this gives us one part of the clipped area that is this part and apparently here it stopped. So, how to get to the other part? Actually, here the algorithm does not stop. What happens next?

(Refer Slide Time: 29:00)



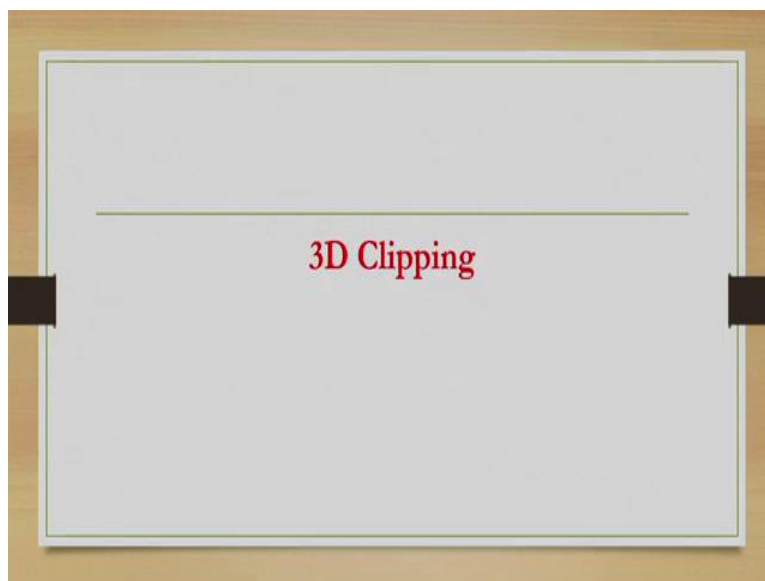
Before we go into that, also, you should remember that whenever we are traversing the traversal direction remains the same, irrespective of whether you are traversing along the edge or along the windows boundary. So, if you are following an anticlockwise direction, it should be anticlockwise always.

an inside outside polygon edge. So, our last intersection point of an inside outside polygon edge is $2'$ here. Remember that this $1'$ is outside inside edge. So, it is not applicable. So, what is applicable is $2'$. So, from there we resume our traversal till we cover the remaining vertices.

And this traversal is in a similar way that we have done before. So, here what we do, we traverse along this anticlockwise direction to the vertex here. So, we traverse this edge, then this edge. But here, as you can see, there is an outside to inside crossing. So, we do not do anything, we keep on traversing this way, this way. Now at this point we can see that one inside to outside crossing is there. In the earlier case, it was outside to inside.

Here it is, inside to outside at $6'$. So, now we traverse along the edge. Then we encountered this intersection point again. This is from outside to inside. So, now we resume our traversal along edge. So, finally what we did, we traversed this direction, this direction, this direction, then this direction, then this direction, this direction. Now since already we have encountered 4 before so we stop our traversal here when we encounter 4. Then we get this remaining portion of the clipped area also just like the way we got it earlier. So, that is how Weiler-Atherton works.

(Refer Slide Time: 32:19)

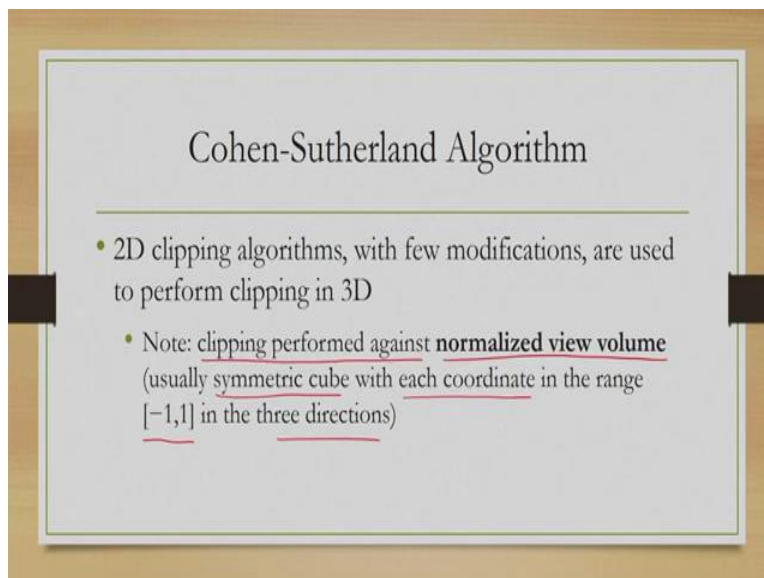


So, we encountered or we discussed two algorithms; one is Sutherland-Hodgeman, one is Weiler-Atherton. Sutherland Hodgeman is simpler but it has restrictive use. It is applicable when we have a convex polygon which is clipped against a rectangular window. Whereas Weiler-

Atherton is more generic it is applicable for any fill area, polygonal fill area, either concave or convex against any polygonal clipping window.

So, so far we have discussed clipping in 2D. So, we have learned how to clip a point line and fill area. Now let us try to understand clipping in 3D, because here our main focus is 3D graphic pipeline. So, we will try to understand clipping in 3D which is essentially extension of the ideas that we have already discussed that is clipping in 2D. Let us see how these extensions are done.

(Refer Slide Time: 33:42)



Only thing we have to keep in mind is that here we are talking about clipping against normalized view volume which is usually a symmetric cube with each coordinate in the range minus 1 to 1 in the 3 directions. That is a normalized view volume we assume while developing the or performing the clipping. Now, Cohen-Sutherland we can extend the basic 2D version to 3D with some modification.

(Refer Slide Time: 34:22)

Point Clipping

- Given a point with coordinate (x,y,z) , we simply check if the coordinate lies within the view volume
 - If $-1 \leq x \leq 1$ AND $-1 \leq y \leq 1$ AND $-1 \leq z \leq 1$, we keep the point
 - Otherwise, we clip it out

Point clipping also, we can extend, so let us first talk about point clipping. Here we check for x, y and z earlier we are checking only for x and y whether these values are within the range of the canonical volume. If that is so, then the point is to be kept. Otherwise it is to be clipped out.

(Refer Slide Time: 34:53)

Cohen-Sutherland Line Clipping Algorithm (3D)

- Core idea same - divide view coordinate space into regions
- However, TWO major differences
 - We have 27 regions (against 9 earlier)
 - 6 bits region coding for 6 planes (far, near, top, bottom, right, left), unlike 4 bits earlier

The diagram illustrates the 3D clipping algorithm with a central 3D box representing the view volume. The planes are labeled: Far (top), Near (bottom), Top (left), Bottom (right), Left (top), and Right (bottom). Three 3x3 grids of region codes are shown, each circled in red. The top grid is labeled 'Region codes behind far plane' and contains the following codes: 101001, 101000, 101010; 100001, 100000, 100010; 100101, 100100, 100110. The middle grid is labeled 'Region codes between near and far planes' and contains: 011001, 011000, 011010; 010001, 010000, 010010; 010101, 010100, 010110. The bottom grid is labeled 'Region codes in front of near plane' and contains: 001001, 001000, 001010; 000001, 000000, 000010; 000101, 000100, 000110. A legend at the bottom shows six bits: Bit 6 (Far), Bit 5 (Near), Bit 4 (Top), Bit 3 (Bottom), Bit 2 (Right), and Bit 1 (Left), each in a red circle.

In case of Cohen Sutherland line clipping algorithm, it can be easily extended to 3D clipping. However, with some modifications, core idea remains the same. That is, we divide view coordinate space into regions. Now, earlier we had 9 regions. Now since we are dealing with 3D

we have 27 regions, 3 times. Now since we have 27 regions. So, each region needs to be represented with 6 bits. Each bit for the 6 planes that define the canonical view volume. Far, near, top, bottom, right, left this is in contrast with the 4 bits earlier used to denote the 4 sides of the window.

Now for each plane, we have this 9 regions defined so there are 9 regions behind the far plane. There are 9 regions between near and far plane and there are 9 regions in front of the near plane. Together there are 27 regions and each region is represented with this 6 bit code, where bit 6 represent the far region, bit 5 is the near region, bit 4 is the top region, bit 3 is the bottom region, bit 2 represents the right region and bit 1 represents the left region. The idea remains the same with 2D only the size changes because we are now dealing with 3D. The other steps remained the same.

(Refer Slide Time: 37:21)

Slide 1: Fill-Area Clipping Algorithm (3D)

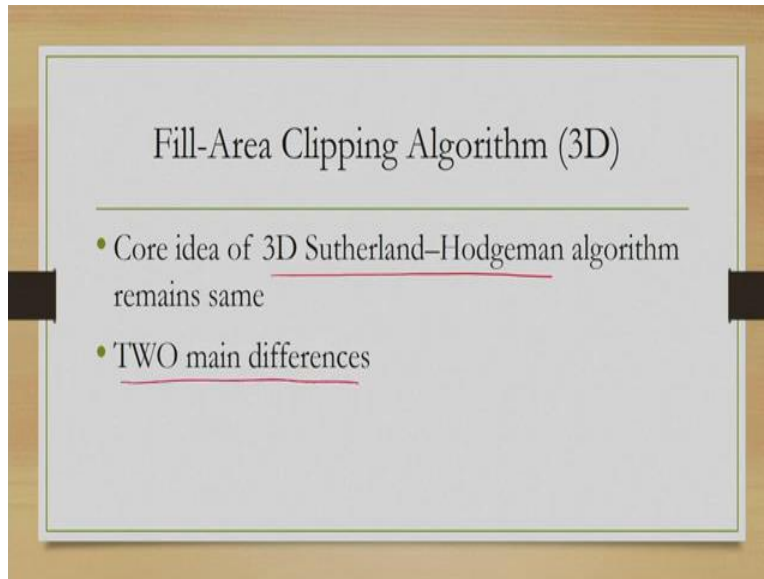
- We first check if the bounding volume of the polyhedron is outside the view volume (by comparing their maximum and minimum coordinates in each of the x, y, z directions)
 - If yes, clip out entirely

Slide 2: Fill-Area Clipping Algorithm (3D)

- Otherwise, we can apply 3D extension of Sutherland–Hodgeman algorithm for clipping

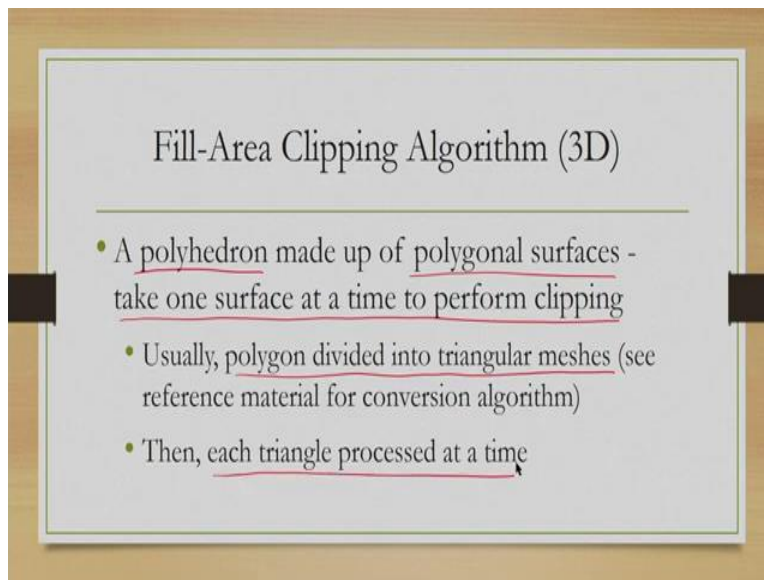
Now let us try to understand the extension of the algorithms for fill area clipping. So, here what we do. We first check if the bounding volume of the polyhedron that is the fill area is outside the view volume simply by comparing their maximum and minimum coordinate values in each of the x, y and z directions. If the bounding volume is outside then we clip it out and entirely. Otherwise we apply 3D extension of the Sutherland Hodgeman algorithm for clipping.

(Refer Slide Time: 38:16)



Here also the core idea of 3D Sutherland Hodgeman algorithm remains the same with 2D version with two main differences. What are those differences?

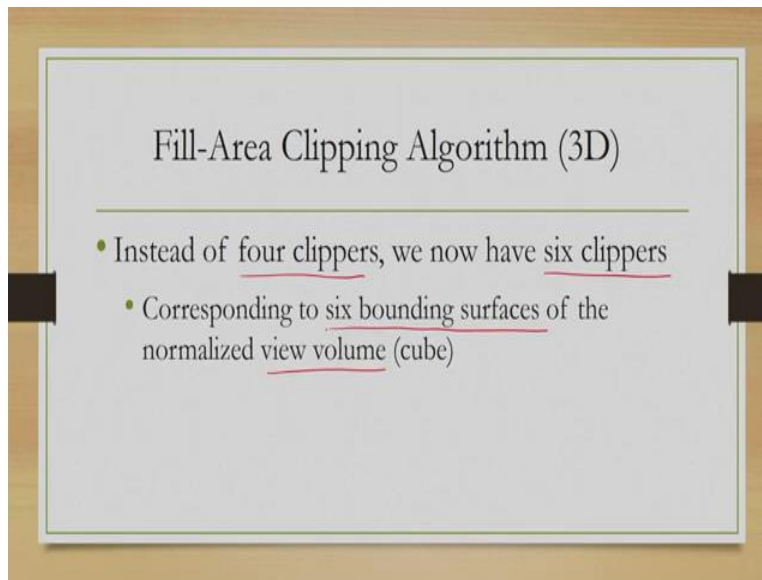
(Refer Slide Time: 38:31)



A polyhedron is made up of polygonal surfaces. So, here we take one surface at a time to perform clipping. Earlier what we were doing, we took one line at a time. Here we are taking one surface at a time. Now, usually polygons divided into triangular meshes and there are algorithms to do so which you can refer to in the reference material at the end of this lecture. So, using those

algorithms, we can divide a polygon into a triangular mesh and then each triangle is processed at a time.

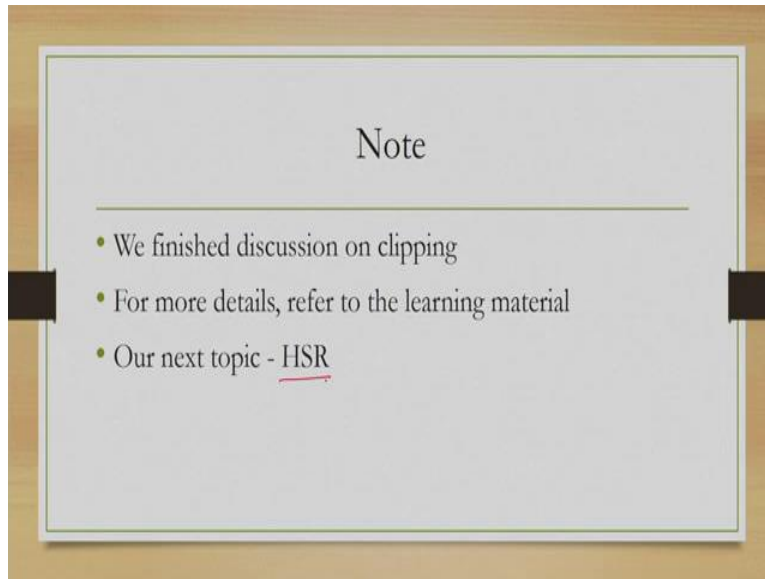
(Refer Slide Time: 39:17)



And the second difference is, instead of the 4 clippers that we had earlier, we now have 6 clippers. Which correspond to the 6 bounding surfaces of the normalized view volume which is a cube. So, these are the differences between the 2D version of the algorithm and the 3D version that earlier we are considering line at a time for clipping. Now we are considering a surface at a time. Now these surfaces are polygonal surfaces and we can convert these surfaces into triangular meshes.

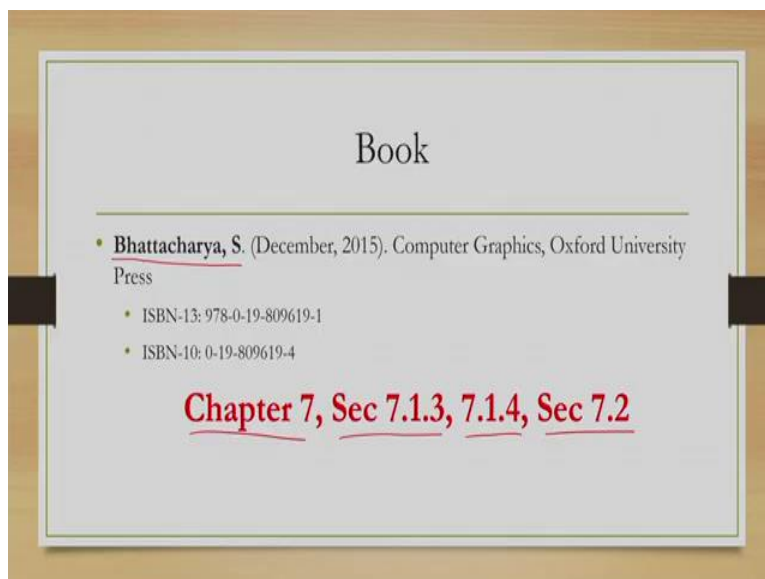
And then we perform clipping for each triangle at a time that is one difference. Other difference is earlier we are dealing with 4 clippers, now we have 6 clippers representing the 6 bounding planes of the view volume which is a cube. So, that is in summary the major differences between 2D clipping and 3D clipping. Core ideas remain the same some minor changes are there. So, with that we come to the end of our discussion on clipping.

(Refer Slide Time: 40:43)



And our next topic will be hidden surface removal. So, here few things omitted during the discussion. For example the triangular mesh creation from given polygon. So, for these details you may refer to the material that will be mentioned in the next slide.

(Refer Slide Time: 41:16)



So, whatever I have covered today can be found in this book. You can go through chapter 7, section 7.1.3, 7.1.4 and section 7.2. For the topics that I have covered however outside this topics

also there are few interesting things that I did not discuss but you can find that in the book. So, you may like to go through those material as well. That is all for today. Thank you and goodbye.