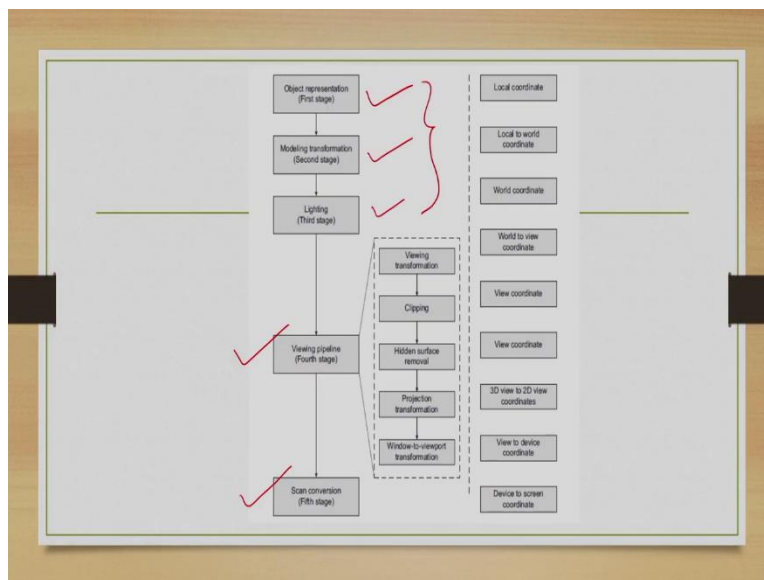


Computer Graphics
Professor Dr Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Lecture 20
Windows to Viewport Transformation

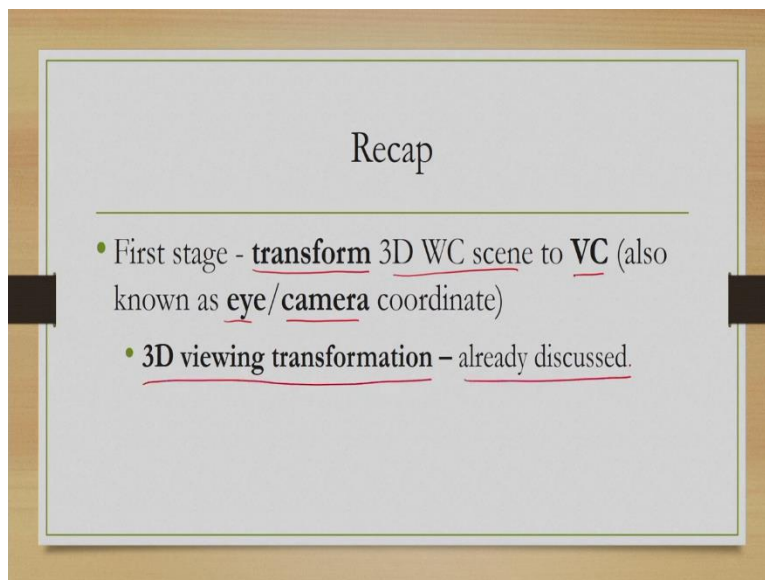
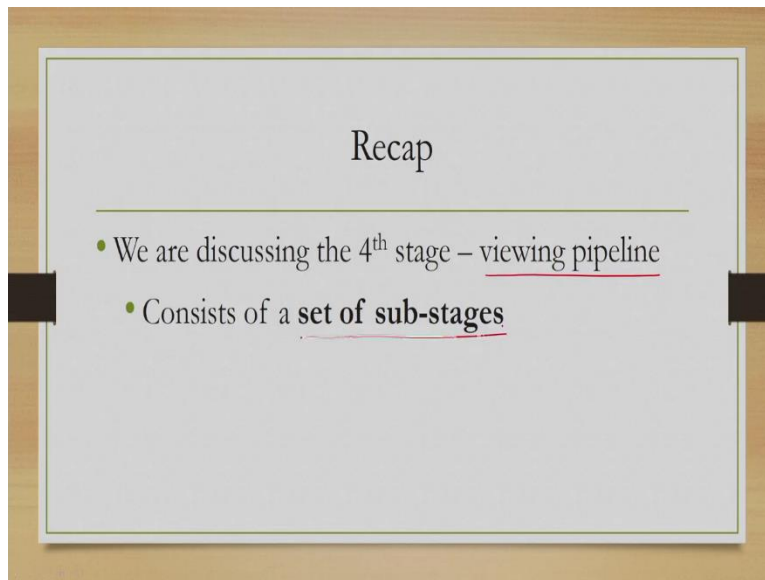
Hello and welcome to lecture number 20 in the course Computer Graphics. We are currently discussing the graphics pipeline, that is the series of stages or steps that has to be performed to convert a 3D description of a scene to a 2D image on a computer screen or on that display that we get to see.

(Refer Slide Time: 0:58)



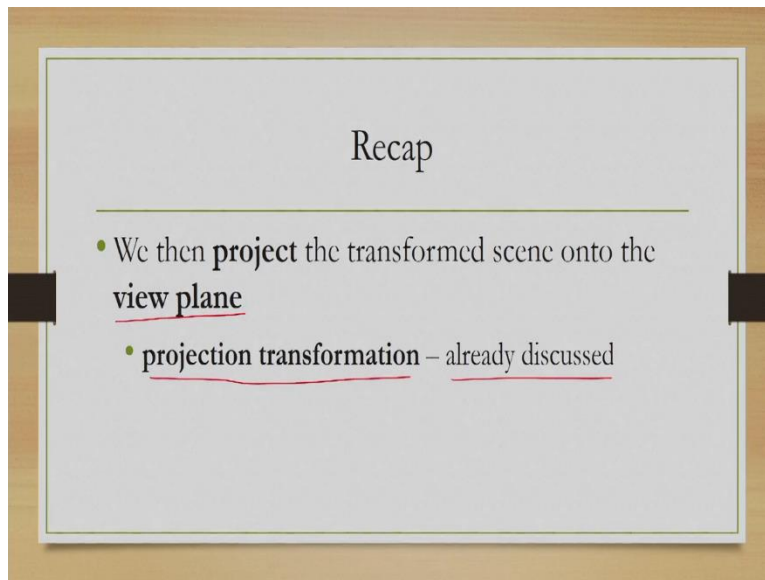
So, there are five stages, as we have already mentioned object representation first stage, modelling transformations second stage, lighting third stage, these three stages we have already discussed completely. Currently, we are in the fourth stage viewing pipeline, and there will be one more stage fifth stage that is scan conversion.

(Refer Slide Time: 1:24)



Now, the fourth stage, the viewing pipeline, contains a set of sub-stages. The first sub-stage is a transformation from a 3D world coordinated scene description to a 3D view coordinate scene description. Now, this view coordinate is also called an eye or camera coordinate system. And this transformation is generally called 3D viewing transformation, which we have already discussed in our earlier lectures.

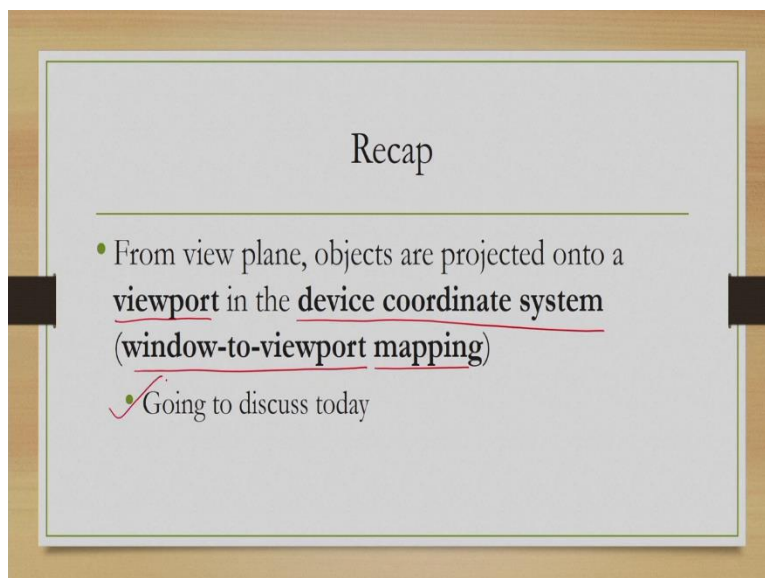
(Refer Slide Time: 2:11)



The second stage is projection, so we project the 3D view coordinate description onto the view plane.

And this projection is performed through a transformation which is generally called projection transformation. This also we have discussed in our earlier lectures.

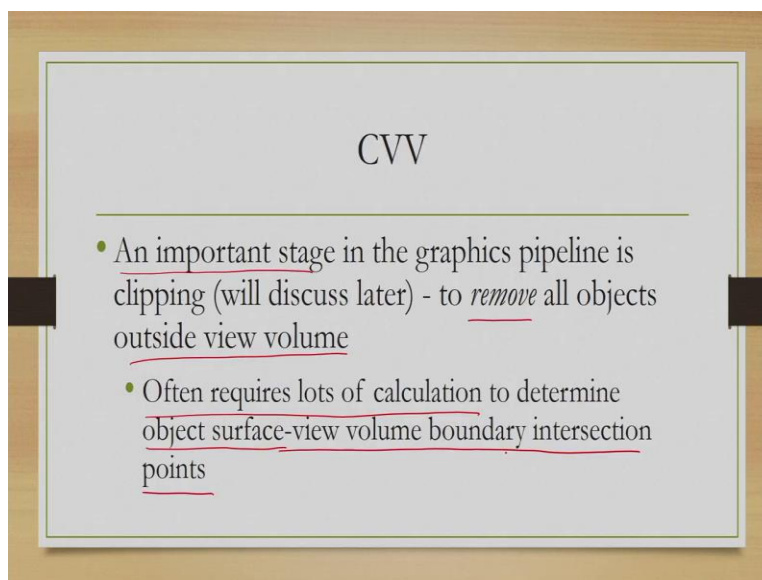
(Refer Slide Time: 2:43)



There is a third stage in which we perform a mapping or transformation that is from the view plane we map this to a viewport which is defined on the device coordinate system. This is called the window to viewport mapping where the window is on the view plane, and viewport is on the device coordinate system. And this third stage we are going to discuss today.

Now, before we discuss the mapping, we would discuss one important aspect of projection transformation that we did not discuss in our last lecture, that is the idea of the canonical view volume. Let us see what this volume means.

(Refer Slide Time: 3:42)



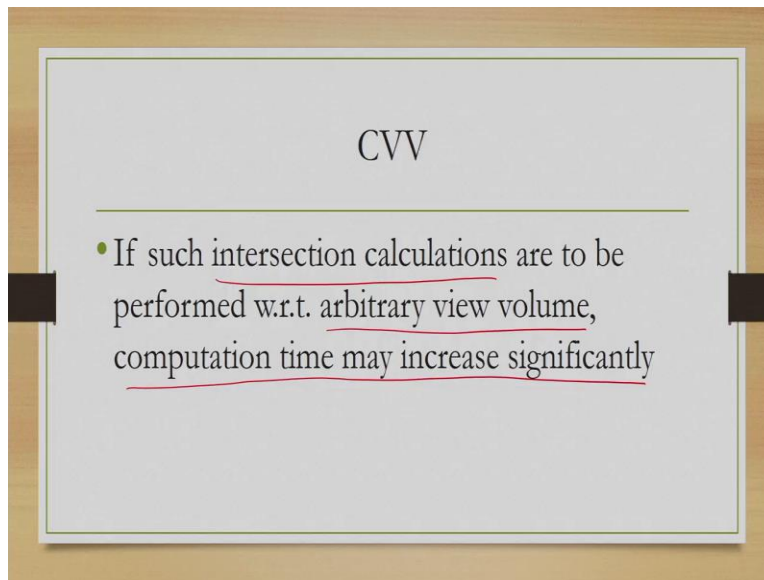
As we mentioned earlier, there is an important stage in the graphics pipeline. In fact, this is part of the fourth stage that we are currently discussing that is viewing pipeline. Here, what we do is whatever objects are outside the view volume are clipped out. Now, that stage is called clipping. And we do it to remove all the objects that are outside the view volume.

We already know what a view volume is, that is a region in the 3D space which we want to project. Now, if it involves lots of objects which are partly outside of view volume and partly inside or lots of objects that are outside the view volume, then we require a large number of calculations to determine which want to clip out.

And this involves object surface, view volume boundary intersection point calculations. So where the object surfaces are intersecting with the view volume boundaries. So, if the number is

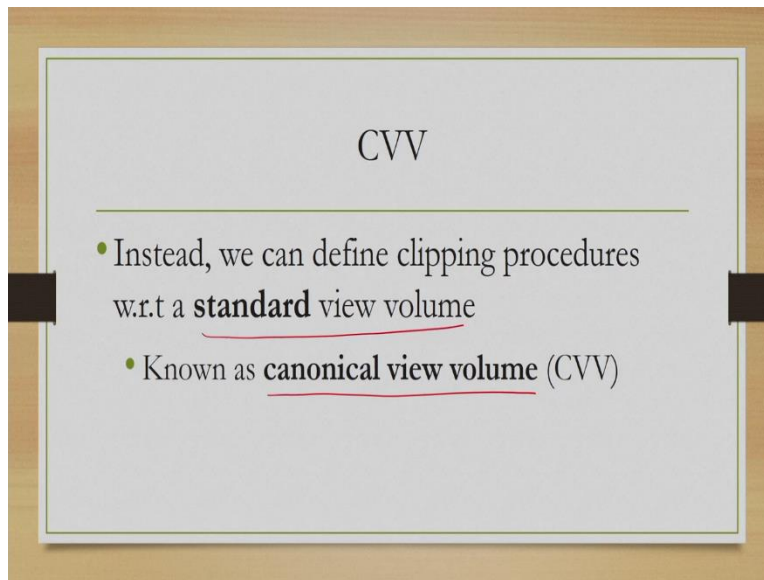
large, then such boundary calculations will be large, and these boundary calculations are not easy. They involve a lot of floating-point operations, and accordingly, the complexity is high.

(Refer Slide Time: 5:21)



Now, if we have to perform such intersection calculations with respect to arbitrary view volume where we have no control on the boundary planes of the view volume, then this complexity is only to be more. So, we can expect a large number of computations, which is likely to take a large amount of time reducing the quality of the image as we will get to see flicker.

(Refer Slide Time: 6:05)



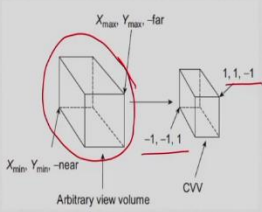
In order to avoid that, we can use one simple idea, that is we can come up with a standardized definition of view volume irrespective of how the actual view volume looks. We can always convert it to a standardized view volume or a standard view volume. This is called a canonical view volume or CVV in short.

Essentially it is a standard representation of view volume irrespective of the actual nature of the volume. Remember that there are two types of view volume. One is for parallel projection, that is a rectangular parallel pipe, and the other one is for perspective projection that is a frustum. Now both these can be converted to a standard form which we call canonical view volume, which makes the intersection calculations standard and easier to implement.

(Refer Slide Time: 7:10)

Parallel Projection CVV

- A cube within the range $[-1,1]$ along the x , y , and z directions
- Any arbitrary view volume can be **transformed** to the CVV by **scaling**

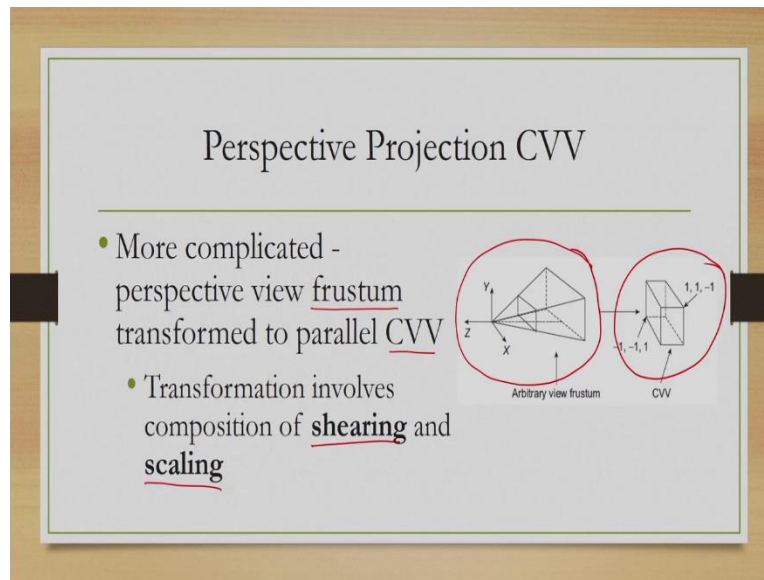


The diagram illustrates the transformation of an arbitrary view volume into a Canonical View Volume (CVV). On the left, an 'Arbitrary view volume' is shown as a 3D box with its bounding planes labeled as X_{min} , Y_{min} , $-near$, X_{max} , Y_{max} , and $-far$. An arrow points from this volume to the 'CVV' on the right. The CVV is a cube with its bounding planes labeled as -1 , -1 , 1 , 1 , 1 , and -1 .

So, for both parallel and perspective projection, the standardized view volume looks the same. However, the way to arrive at the standardized or canonical volume for both the projections for the two types of projections are different. So, let us start with the parallel projection. So, for parallel projection, the canonical view volume that we define is a cube within a specified range, which is -1 to 1 along with all the three axis X, Y and Z. And as I already mentioned, any arbitrary view volume can be transformed to the CVV simply by the scaling operation.

So, suppose this is an arbitrary view volume defined in terms of its bounding planes, six planes, so we can always map it by scaling within this range along the X Y Z direction and correspondingly, we get the canonical view volume.

(Refer Slide Time: 8:28)



In case of perspective projection, this transformation is slightly more complicated because here we are dealing with a view frustum and we need to convert it to a canonical view volume for parallel production that is the rectangular parallel pipe where the X, Y and Z extent of the bounding planes are within a specified range.

So, what we can do here, we will just talk about the idea rather than going into the details, we can convert this arbitrary view frustum to the canonical view volume here, by applying shearing and scaling in sequence. As we can guess from the figures, that shearing is required to change the shape and scaling is required to change the size. So, when we apply this to transformations on the original view frustum, we get the canonical view volume, of course, here will not go into any further details than this basic idea.

So, what we have learned? That we define a view volume and this view volume, we transform to a canonical view volume so that in later stages when we perform clipping, the calculations are easier to implement because we are dealing with a standardized definition of the view volume.

(Refer Slide Time: 10:14)

Projection Transformation - Revisited

- Sequence of transformations to project P (world coordinate) on view plane
 - ✓ First, transformed to VC
 - ✓ Next, view volume in which the point lies transformed to CVV
 - ✓ Finally, point in the CVV is projected (projection transformation)

Projection Transformation - Revisited

- In matrix notation, we can write this series of steps as:
$$P' = T_{proj} T_{cvv} T_{vc} P$$
- T_{proj} projection transformation matrix
- T_{cvv} matrix for transformation to CVV
- T_{vc} matrix for transformation to view coordinate

Let us revisit the sequence of transformations that we perform to project a point p in the world coordinates scene to a point on the view plane. Now, we mentioned that this is akin to taking a photograph that is we transfer it to view coordinate system, then take a projection. However, earlier, we mentioned these two steps.

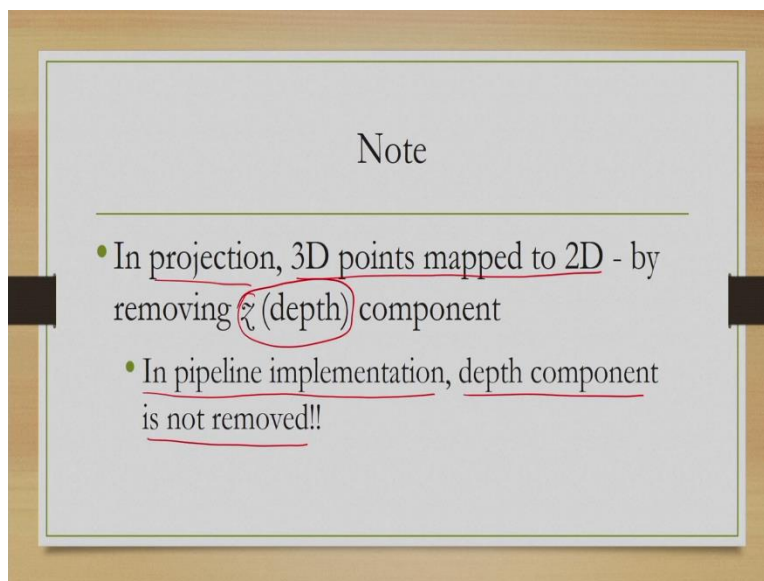
Now one third step is added. So first, we transform the world, coordinate point on the view coordinate system, as we have discussed in earlier lecture. And the next step is not projection. Instead, what we do is in this view coordinate description, we define a view volume, and this

view volume is transformed to a canonical view volume. And accordingly, the point is also transformed by applying the same set of transformations. So, the next stage is to transform the point in the view volume to a point in the canonical view volume; then the final stage is to perform the projection transformation that is, project the point in the canonical view volume on the view plane.

So, these three steps, a transformation to view coordinate then transformation to canonical view volume and then projection transformation constitute the sequence through which we project a point in the world coordinate scene into a point on the view plane. Mathematically or in matrix notation that we are following, we can write this series of steps as shown here in this expression where this transformation represents the transformation to view volume.

This one represents the transformation to canonical view volume, and this one represents the projection transformation. Since we are applying them in sequence, so we are following the right to left rule. So, the first transformation to view coordinate system, then transformation to canonical view volume and then transformation to view plane through projection.

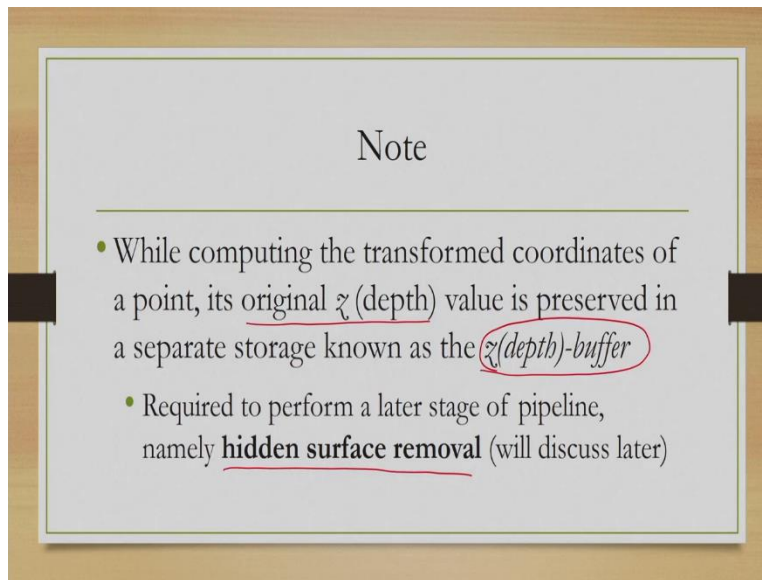
(Refer Slide Time: 13:11)



So, that is the idea of performing projection on the view plane. There is one more point to be noted. So far, what we mentioned that in projection 3D points are mapped to 2D. The implication is that we are removing the Z or depth component. However, it may be noted here at this point

that while we implement the pipeline, this depth component is actually not removed, and why that is so?

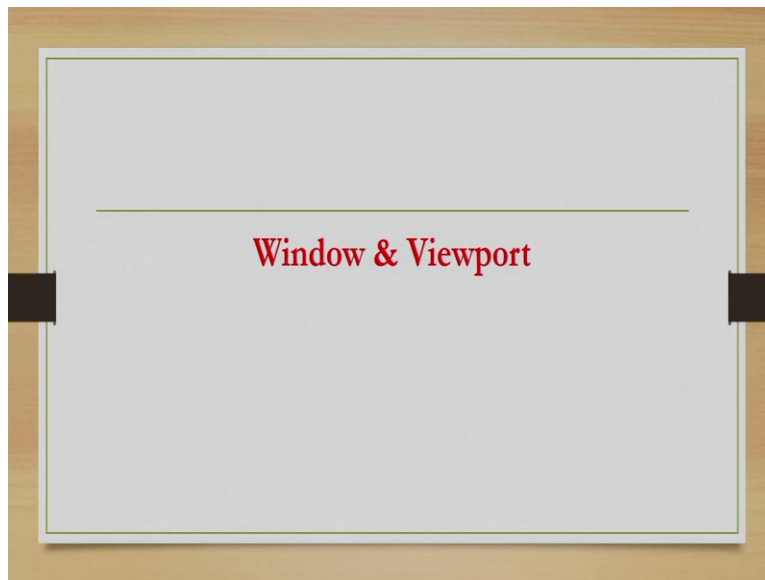
(Refer Slide Time: 13:53)



One operation that we perform in this fourth stage is called hidden surface removal. We will talk about this operation in details in a later lecture. The point to be noted here is that this operation requires depth information. So, the depth information after projection is actually not removed. Instead, this original depth information is stored in separate storage, which is called the Z-buffer or the depth buffer.

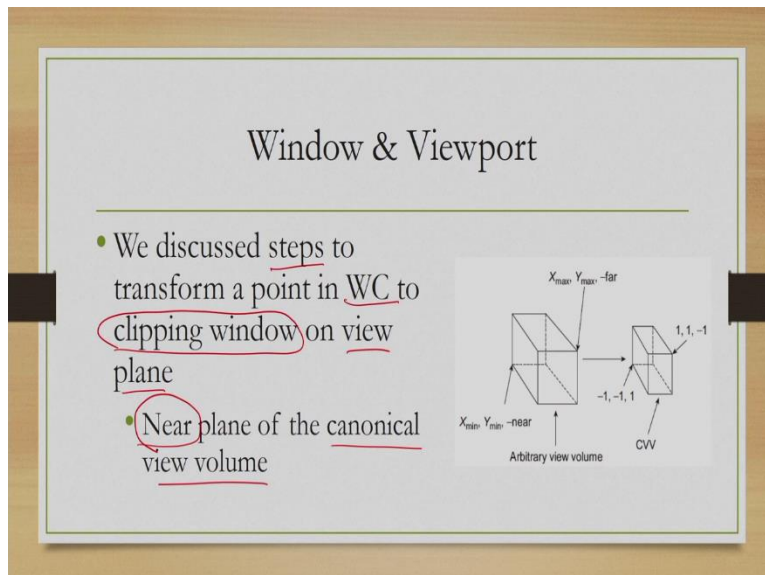
So, we are actually not removing the depth information, although we are performing a projection instead, we are keeping it stored separately in the Z-buffer or the depth buffer. And this information is required to perform a later operation called hidden surface removal, which gives us a realistic effect in an image.

(Refer Slide Time: 14:56)



So, that is in short what we do during projection and how we project from a world coordinated scene to a view plane. Now there is one more stage. Let us go to that stage that is mapping from this view plane to a viewport on the device space.

(Refer Slide Time: 15:21)



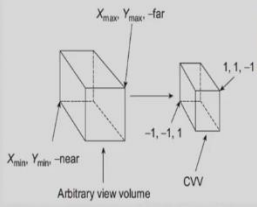
So, far what we have discussed? We discussed steps to transform a point in world coordinate to a clipping window on the view plane. That means a region on the view plane on which we are projecting the objects that are part of the view volume.

Now, also, you have shown that this is typically the near plane of the canonical view volume. So, this is our window or clipping window.

(Refer Slide Time: 15:59)

Window & Viewport

- For simplicity, window is assumed to be at zero depth (i.e., $z = 0$)

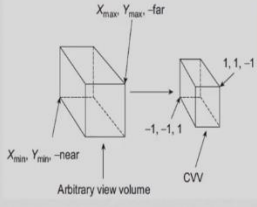


We can assume that for simplicity that the window is at 0 depth or Z equal to 0, that is just for simplicity, although in general, that is not an absolute requirement.

(Refer Slide Time: 16:17)

Window & Viewport

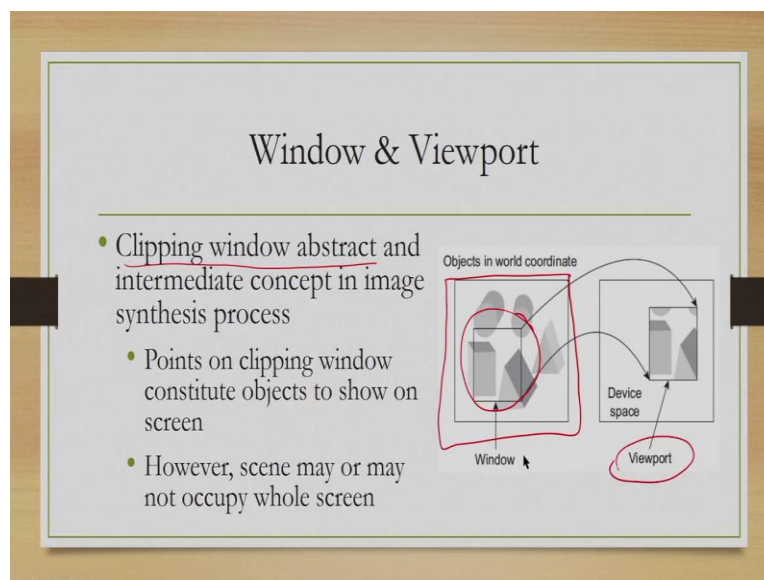
- Also, since x and y extents are fixed $[-1, 1]$, points in the window have a fixed range, irrespective of their actual position in WC
- For this reason, clipping window on the CVV is often called normalized window



It may also be noted that we are talking of canonical view volume that is X and Y extents must be within a fixed range irrespective of their actual position in the world coordinates scene, and because of this region where we are restricting everything within a fixed range, this canonical view volumes are standardized, and the clipping window that is defined on the near plane of the canonical view volume is often called a normalized window.

So, here we are dealing with a normalized window where the extent of values are to be within a predefined range.

(Refer Slide Time: 17:19)



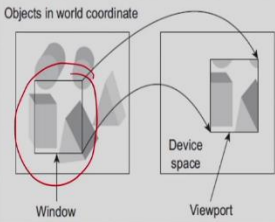
Now, this view plane is actually an abstract concept, so accordingly, the clipping window is also an abstract and intermediate concept. We cannot see it; what we get to see on the screen is something different. The points that are there on the clipping window are to be shown on the screen. But the scene that is there in the window need not occupy the whole screen, for example, here.

Suppose this outer rectangle defines a whole scene out of which we have projected this part defined within the clipping window. Now, this part can be displayed on any region of the screen and can be in any size now the region on which this part is displayed on the screen it is called the viewport.

(Refer Slide Time: 18:41)

Window & Viewport

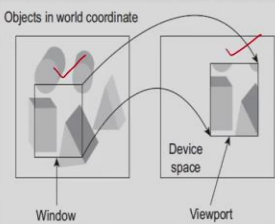
- **Window** - same as (normalized) clipping window
- WC objects projected on this window



The diagram shows 'Objects in world coordinate' at the top. Below it, a 'Window' is shown as a rectangle containing a portion of the objects. A red circle highlights the objects within the window. An arrow points from the window to a 'Viewport' in 'Device space', which is a smaller rectangle containing the projected image of the objects.

Window & Viewport

- Viewport defined in **device space** - w.r.t screen origin and dimensions
- One more transformation required to transfer points from window (VC) to viewport (DC)

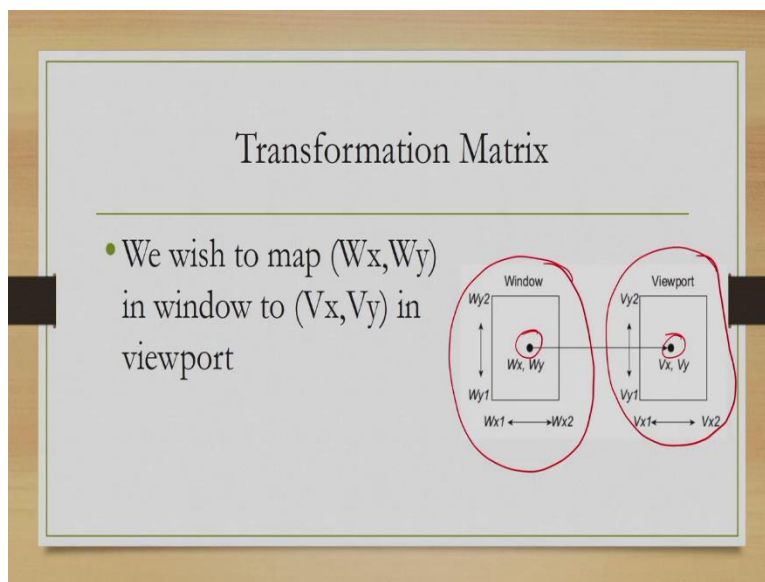
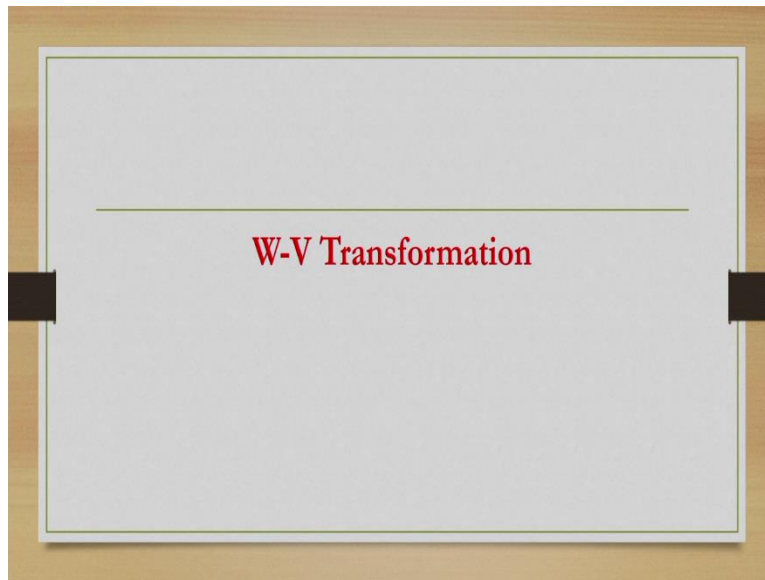


The diagram is identical to the one above, but with a red checkmark in the 'Device space' area, indicating that the transformation from the window to the viewport is complete.

So, we have two concepts here, window, which is same as the clipping window, which is normalized. And objects are projected on this window, and we have the other concept viewport, which is defined in the device space with respect to the screen origin and dimensions. So, this viewport refers to a region on the device space where this projected image needs to be shown. Now, this region can be at any location on the space device space and can be of any size, irrespective of the size of the clipping window. So, what we need? We need to map from this window to the viewport.

So, it requires one more transformation to transfer the points from the window to the viewport.

(Refer Slide Time: 19:59)



So, let us see how we can perform this transformation. So what we want, suppose this is our window, and this is our viewport, note that here we are not using this normalized range. We are formulating the problem in a very generic scene where this W_x and W_y can take any value. So, W_x, W_y is a point on the window, and we want to map it to a point on the viewport V_x, V_y .

(Refer Slide Time: 20:44)

Transformation Matrix

- To maintain relative position of the point in the viewport, we must have

$$\frac{W_x - W_{x1}}{W_{x2} - W_{x1}} = \frac{V_x - V_{x1}}{V_{x2} - V_{x1}}$$

The diagram illustrates the mapping from a Window to a Viewport. The Window is a rectangle with width $W_{x2} - W_{x1}$ and height $W_{y2} - W_{y1}$. A point (W_x, W_y) is located inside the window. The Viewport is a smaller rectangle with width $V_{x2} - V_{x1}$ and height $V_{y2} - V_{y1}$. A point (V_x, V_y) is located inside the viewport. A horizontal line connects the two points, and a vertical line connects the bottom edges of the two rectangles, showing the relative position of the point within the window and viewport.

So, how we can do that? The first thing is that we have to maintain the relative position of this point with respect to its boundaries, so the same relative position has to be maintained in the viewport, so if we want to maintain that, then we get relationships like the one shown here between the window dimensions and the viewport dimensions.

(Refer Slide Time: 21:17)

Transformation Matrix

- Or

$$V_x = s_x \cdot W_x + t_x$$
$$s_x = \frac{V_{x2} - V_{x1}}{W_{x2} - W_{x1}}$$
$$t_x = s_x \cdot (-W_{x1}) + V_{x1}$$

The diagram illustrates the mapping from a Window to a Viewport. The Window is a rectangle with width $W_{x2} - W_{x1}$ and height $W_{y2} - W_{y1}$. A point (W_x, W_y) is located inside the window. The Viewport is a smaller rectangle with width $V_{x2} - V_{x1}$ and height $V_{y2} - V_{y1}$. A point (V_x, V_y) is located inside the viewport. A horizontal line connects the two points, and a vertical line connects the bottom edges of the two rectangles, showing the relative position of the point within the window and viewport.

Now, this expression can be simplified in this form. So, we can represent the X coordinate of the point in the viewport in terms of the X coordinate of the point in window and these two constants, which are defined here in terms of the window and viewport sizes.

(Refer Slide Time: 22:04)

Transformation Matrix

- Similarly, we must have

$$\frac{W_y - W_{y1}}{W_{y2} - W_{y1}} = \frac{V_y - V_{y1}}{V_{y2} - V_{y1}}$$

Transformation Matrix

- Or

$$V_y = s_y \cdot W_y + t_y$$

$$s_y = \frac{V_{y2} - V_{y1}}{W_{y2} - W_{y1}}$$

$$t_y = s_y \cdot (-W_{y1}) + V_{y1}$$

Similar relationship we can form between the Y coordinate of the point in the viewport and the Y coordinate of the same point in the window by again, forming, the relationships first between the y coordinates and then simplifying and rearranging to get this relationship where V_y is the Y coordinate of the point in the viewport, W_y is the Y coordinate of the point in the window.

And these two are constants defined here in terms of, again, the window and viewport sizes.

(Refer Slide Time: 23:05)

Transformation Matrix

- Using the expressions, we can get transformation matrix

$$T_{vp} = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix}$$

So, using those expressions, we can actually form that transformation metrics as shown here. So, this is the metrics to transform this window point to the viewport point.

(Refer Slide Time: 23:23)

Transformation Matrix

- to get transformed point

$$\begin{bmatrix} x'' \\ y'' \\ w \end{bmatrix} = T_{vp} P_w = \begin{bmatrix} sx & 0 & tx \\ 0 & sy & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = \frac{x''}{w} \quad y' = \frac{y''}{w}$$

And we will follow the same rule that is to get the transform point will multiply the original point with the transformation metrics as shown here. Note that here again, we are dealing with

the homogeneous coordinate system since these are two-dimensional points, so we have three-element vectors and three by three matrices.

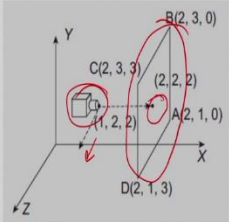
And at the end, we need to divide the obtained coordinates here with the homogeneous factor as shown here, to get the transformed points. The approach is similar to what we have seen earlier. So, that is the basic idea of how to transform from a point in the window or the clipping window to a point in the viewport, which can be anywhere on the device space.

Now, let us try to understand the concepts that we have gone through so far in terms of illustrative examples.

(Refer Slide Time: 24:38)

Example

- Recollect the example we discussed earlier
- We are looking at the square object with vertices $A(2,1,0)$, $B(2,3,0)$, $C(2,3,3)$ and $D(2,1,3)$. The camera is located at the point $(1,2,2)$ and the look-at point is the center of the object $(2,2,2)$. The up direction is parallel to the positive z direction.



So, in our earlier lecture, we have come across this example where we mentioned one object, shown here and a camera position view up direction, everything has been mentioned, and we computed the transform centre point of the object in the view coordinate system.

(Refer Slide Time: 25:11)

Example

- Earlier, we computed coordinate of the center of the object, after its transformation to the viewing coordinate system
- Which is $(0,0,-1)$

So, we will not go into the details of how we calculated that again, let us just mention the transform point. That is $0, 0, -1$ that we got after applying this viewing transformation.

(Refer Slide Time: 25:29)

Example 1

- What would be the coordinates of the object center on a view plane $z = -0.5$, if we want to synthesize images of the scene for parallel projection?
- Assume view volume sufficiently large to encompass whole transformed object

Now let us assume that the view plane is located at Z equal to -0.5 . And we want parallel projection. So, what would be the coordinate of the object centre after the projection? Assuming that the view volume is sufficiently large to encompass the whole transformed object.

(Refer Slide Time: 26:08)

Example 1

- Transformation matrix for parallel projection

$$T_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$T_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

So, our parallel projection transformation matrix is given here, and we know D is 0.5. So, this is our parallel projection matrix.

(Refer Slide Time: 26:27)

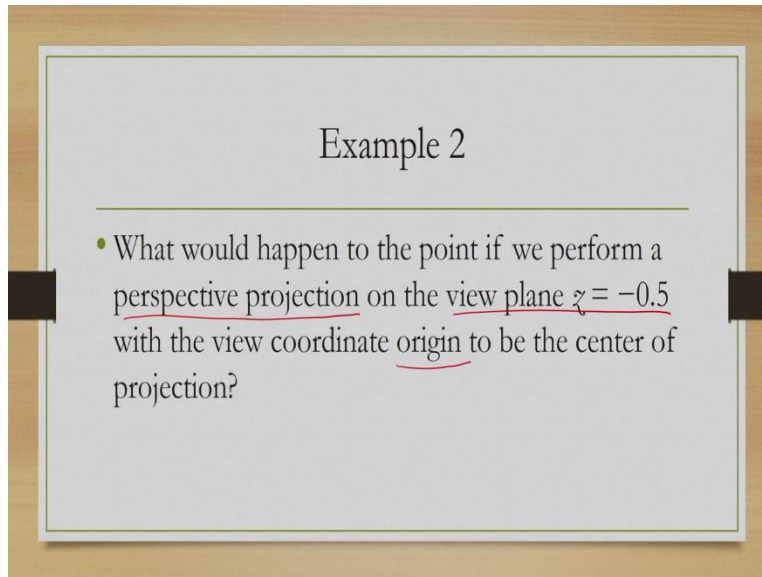
Example 1

- Point after projection

$$T_{par}P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -0.5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -0.5 \\ 1 \end{bmatrix}$$

So, if we use these matrix and perform the matrix multiplication here. The projection matrix and the point vector. Then we get this point as the projected point on the view plane. Now here, since the homogeneous factor is 1, so our point is directly obtained.

(Refer Slide Time: 26:56)

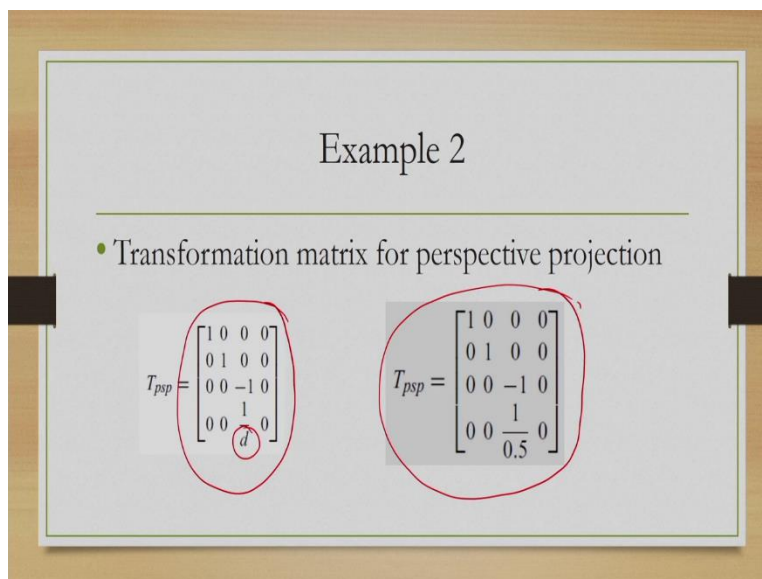


Example 2

- What would happen to the point if we perform a perspective projection on the view plane $z = -0.5$ with the view coordinate origin to be the center of projection?

Now, let us consider perspective projection. Earlier, we considered parallel projection, what will happen if we now consider the perspective projection with the same view plane? So, what would be the new point after projection?

(Refer Slide Time: 27:29)



Example 2

- Transformation matrix for perspective projection

$$T_{psp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & d \end{bmatrix}$$
$$T_{psp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{0.5} & 0 \end{bmatrix}$$

So, the transformation metrics for perspective projection is shown here. We know the value of d replacing d in this, we get our projection metrics. And with this matrix, what we do?

(Refer Slide Time: 27:58)

Example 2

- Point after projection

$$T_{\text{psp}}P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & \frac{1}{0.5} & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ -2 \end{bmatrix}$$

Example 2

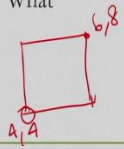
- Note here $h = -2$
- Hence, the projected point is $\left(\frac{0}{-2}, \frac{0}{-2}, \frac{1}{-2}\right)$
- Or $(0, 0, -0.5)$

We multiply it with the point vector as before, as shown here, so after multiplication, we get this transform point in a homogeneous coordinate system. Now note here that the homogeneous factor is not 1, earlier I mentioned that in projection, particularly perspective projection, we get homogeneous factors that are not 1, so there we need to be careful in concluding the final transformed point we have to divide whatever we got with the homogeneous factor. So, after division, we will get this point, or this is the final point that we get after perspective projection applied on the central point.

(Refer Slide Time: 29:01)

Example 3

- Let us assume point is projected on a normalized clipping window (as we saw, projected point in either parallel or perspective projection is $(0,0,-0.5)$, which lies at the center of the normalized window). We want to show the scene on a viewport having lower left and top right corners at $(4,4)$ and $(6,8)$ respectively. What would the position of the point be in the viewport?



So, we have performed projection. Now let us try to see what happens if we want to perform this window to viewport transformation. Now, let us assume that we projected the point on a normalized clipping window. And this projected point is at the centre of the normalized window. Now we are defining a viewport with a lower-left corner at $(4, 4)$ and the top right corner at $(6, 8)$.

So, that means if this is our viewport, then the lower-left corner is this 1. This is $(4, 4)$ and top right corner is $(6, 8)$. So, if we perform a window to viewport transformation, then what would be the position of the point, the same central point in the viewport? Let us try to derive that.

(Refer Slide Time: 30:18)

Example 3

- Since clipping window is normalized, we have $W_{x1} = -1, W_{x2} = 1, W_{y1} = -1$ and $W_{y2} = 1$
- Also, from viewport specification, we have $V_{x1} = 4, V_{x2} = 6, V_{y1} = 4$ and $V_{y2} = 8$

Now, we already mentioned that the clipping window is normalized, so the values or the extent of the window are fixed. And we get these values. So, this is between - 1 to 1, and again, this is also between - 1 to 1. So, we get these values, and from viewport specification, we can see that this is (4, 4). So, this 4 and this is so this point is (6, 8) then this must be 6. This must be 8. So, we get these values. So, next, we simply replace these values in the transformation matrices that we have seen earlier.

(Refer Slide Time: 31:27)

Example 3

- Therefore, $s_x = \frac{6 - 4}{1 - (-1)} = 1$
- $s_y = \frac{8 - 4}{1 - (-1)} = 2$
- $t_x = 1 \cdot (-1) + 4 = 5$ $t_y = 2 \cdot (-1) + 4 = 6$

We first compute the constant values s_x , s_y , t_x , t_y by using those values that we have seen earlier to get these results. S_x is 1, s_y is 2, t_x is 5 and t_y is 6.

(Refer Slide Time: 31:53)

Example 3

• So, transformation matrix (W-V)

$$T_{vp} = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{vp} = \begin{bmatrix} 1 & 0 & 5 \\ 0 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

So, the transformation matrix can be obtained by replacing the s_x , s_y , t_x , t_y values in this original transformation matrix which gives us this matrix. So, this will be our window to viewport transformation matrix.

(Refer Slide Time: 32:16)

Example 3

• Transformed point (homogeneous coordinate)

$$\begin{bmatrix} 1 & 0 & 5 \\ 0 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -2.5 \\ -3 \\ -0.5 \end{bmatrix}$$

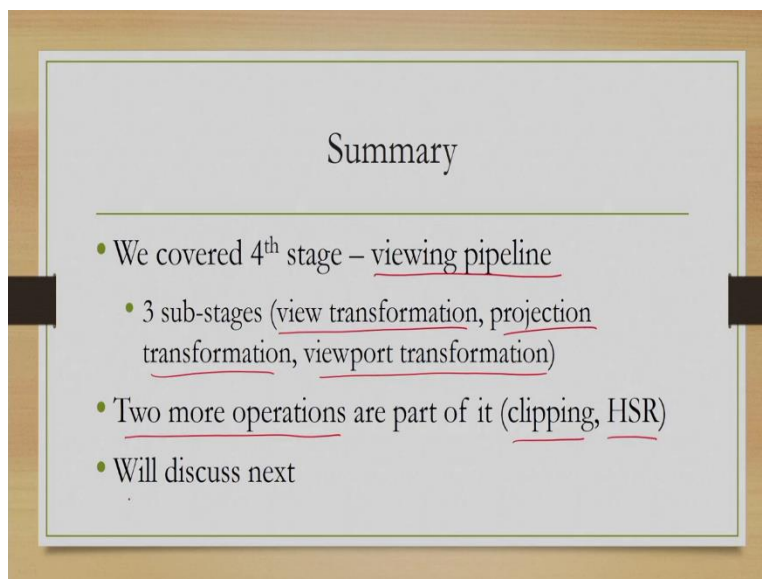
• The point is $\begin{pmatrix} -2.5 \\ -3 \end{pmatrix}$ or $(5, 6)$

Now, once we obtain the transformation matrix, then it is easier to obtain the transformed point in the viewport by multiplying the transformation metrics with the point vector to obtain the transform point in homogeneous coordinate. Now, here again, the coordinate factor is not 1. So, we have to divide these values with the homogeneous factor as shown here and here, which eventually gives us the point (5, 6).

So, this will be our transformed point after we apply the window to viewport transformation. So, this is how we get a transformed point in the viewport. Now in this example, you probably have noted that we have defined viewport irrespective of the window description, we can define it anywhere in the device space.

What we need is basically a transformation also, you probably have noted that the viewport size has nothing to do with the window size, the window is normalized, whereas the viewport is not normalized. So, we can define any size by specifying its coordinate extents, and through mapping, we get that transformed point. So, this gives us the flexibility of placing the projected image anywhere on the screen with any size.

(Refer Slide Time: 34:18)



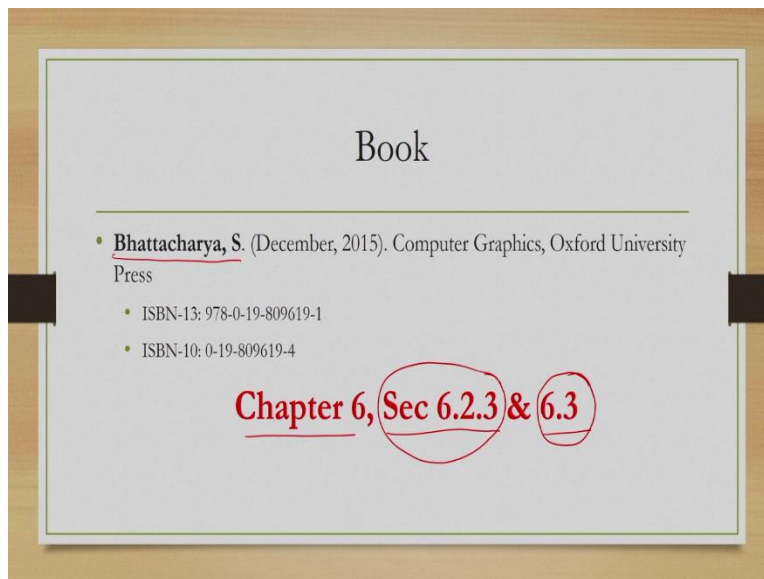
So, in summary, what we have discussed so far are three sub-stages of the viewing pipeline stage. So, these three sub-stages are view transformation, projection transformation and viewport transformation. Just to recap, so these three sub-stages are used to simulate the effect of taking a

photograph. So, when you take a photograph, we look at the scene through the mechanism provided in the camera so that we mimic by performing the viewing transformation, we transform the world coordinate scene to a 3D view coordinate system, which is actually equivalent to watching the scene through the viewing mechanism of the camera. Then, we take a photo that means we project it on the view plane that is done through projection transformation.

And finally, we display it on the screen, which is of course not part of the photograph analogy, but we do it in computer graphics so that stages mimicked with the use of windows to viewport transformation. This transformation is required to have the flexibility of displaying the projected image anywhere on the screen and with any size, irrespective of the size of the clipping window. In the fourth stage, apart from these three sub-stages, which are related to three types of transformations, there are two more operations that are done.

We already mentioned it in this lecture. One is clipping one is hidden surface removal. So, these two operations we will discuss in our subsequent lectures.

(Refer Slide Time: 36:11)



Whatever I have discussed today can be found from this book, Computer Graphics. You can go through Chapter 6, the section 6.2.3 and 6.3 this section is on the topic of canonical view volume, and this section discusses in detail the window to viewport transformation. That is all for today. Thank you and goodbye.