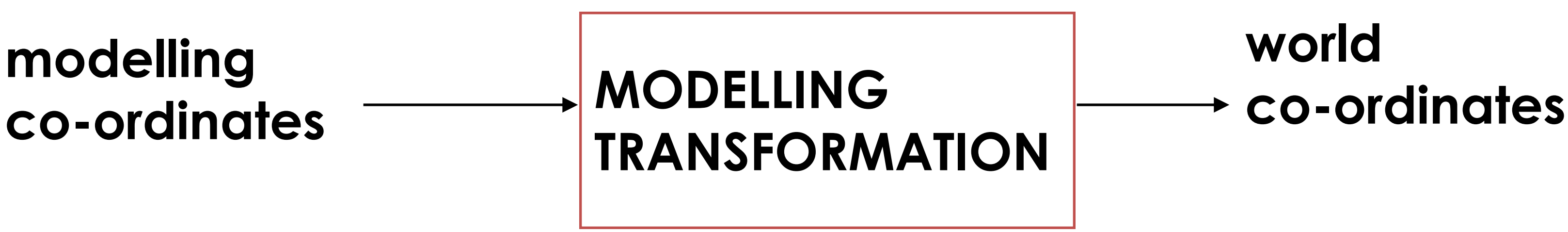
Viewing pipeline

Unit 2 – Lecture 5

The Story So Far ...Lecture 2

- We have seen how we can model objects, by transforming them from their local co-ordinate representation into a world co-ordinate system



The Story So Far...Lecture 3

- And we have seen how we can transform from a special viewing co-ordinate system (camera on z-axis pointing along the axis) into a projection co-ordinate system

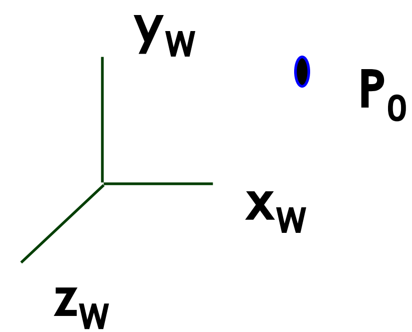
Completing the Pipeline - Lecture 4

- We now need to fill in the missing part

to get

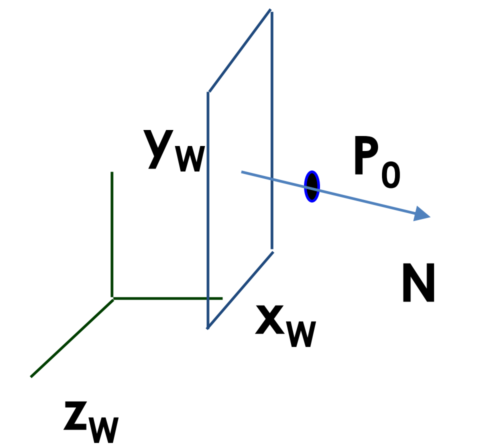
Viewing Coordinate System - View Reference Point

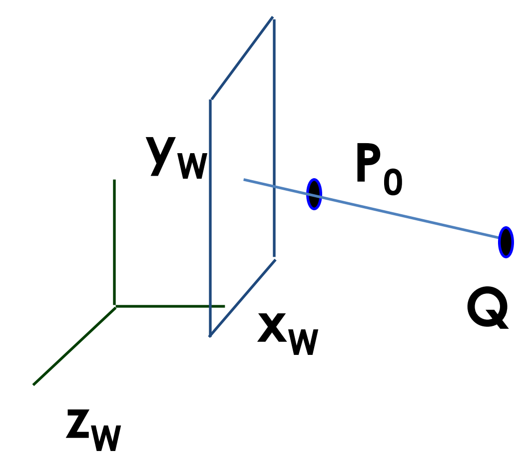
- In our world co-ordinate system, we need to specify a view reference point - this will become the origin of the view co-ordinate system
- This can be any convenient point, along the camera direction from the camera position
  - indeed one possibility is the camera position itself



Viewing Coordinate System - View Plane Normal

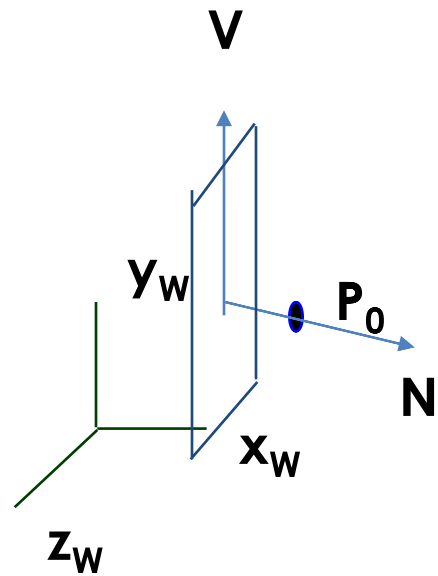
- Next we need to specify the view plane normal, N - this will give the camera direction, or z-axis direction
- Some graphics systems require you to specify N ...
- ... others (including OpenGL) allow you to specify a 'look at' point, Q, from which N is calculated as direction to the 'look at' point from the view reference point





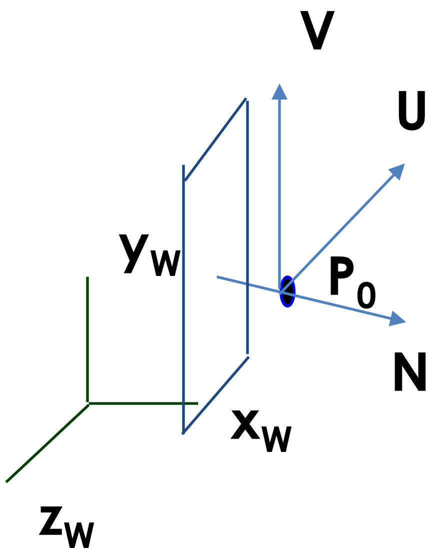
Viewing Coordinate System - View Up Direction

- Finally we need to specify the view-up direction, V - this will give the y-axis direction



Viewing Co-ordinate System

- This gives us a view reference point P_0,and vectors N (corresponding to z_V) and V (corresponding to y_V)
- We can construct a vector U perpendicular to both V and N, and this will correspond to the x_V axis
- How?

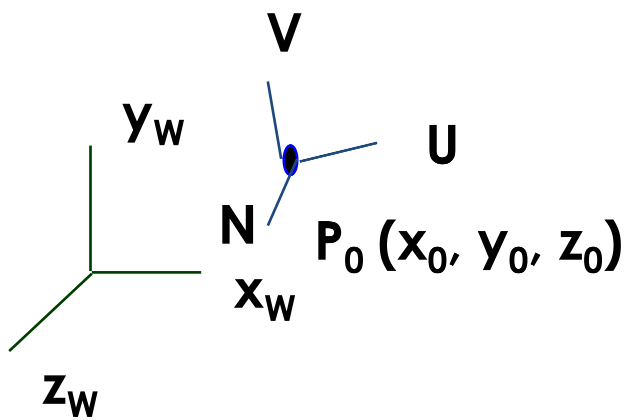


Transformation from World to Viewing Co-ordinates

- Given an object with positions defined in world co-ordinates, we need to calculate the transformation to viewing co-ordinates
- The view reference point must be transformed to the origin, and lines along the U, V, N directions must be transformed to lie along the x, y, z directions

Transformation from World to Viewing Co-ordinates

- Translate so that P_0 lies at the origin



- apply translation by $(-x_0, -y_0, -z_0)$

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformation from World to Viewing Co-ordinates

- Apply rotations so that the U, V and N axes are aligned with the x_W, y_W and z_W directions
- This involves three rotations Rx, then Ry, then Rz
  - first rotate around x_W to bring N into the x_W-z_W plane
  - second, rotate around y_W to align N with z_W
  - third, rotate around z_W to align V with y_W
- Composite rotation R = Rz. Ry. Rx

Rotation Matrix

- Fortunately there is an easy way to calculate R, from U, V and N:

$$R = \begin{bmatrix} u_1 & u_2 & u_3 & 0 \\ v_1 & v_2 & v_3 & 0 \\ n_1 & n_2 & n_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $U = (u_1 \ \ u_2 \ \ u_3)^T$ **etc**

Viewing Transformation

- Thus the viewing transformation is:

$$M = R \, . \, T$$

- This transforms object positions in world co-ordinates to positions in the viewing co-ordinate system..

.. with camera pointing along negative z-axis at a view plane parallel to x-y plane

- We can then apply the projection transformation

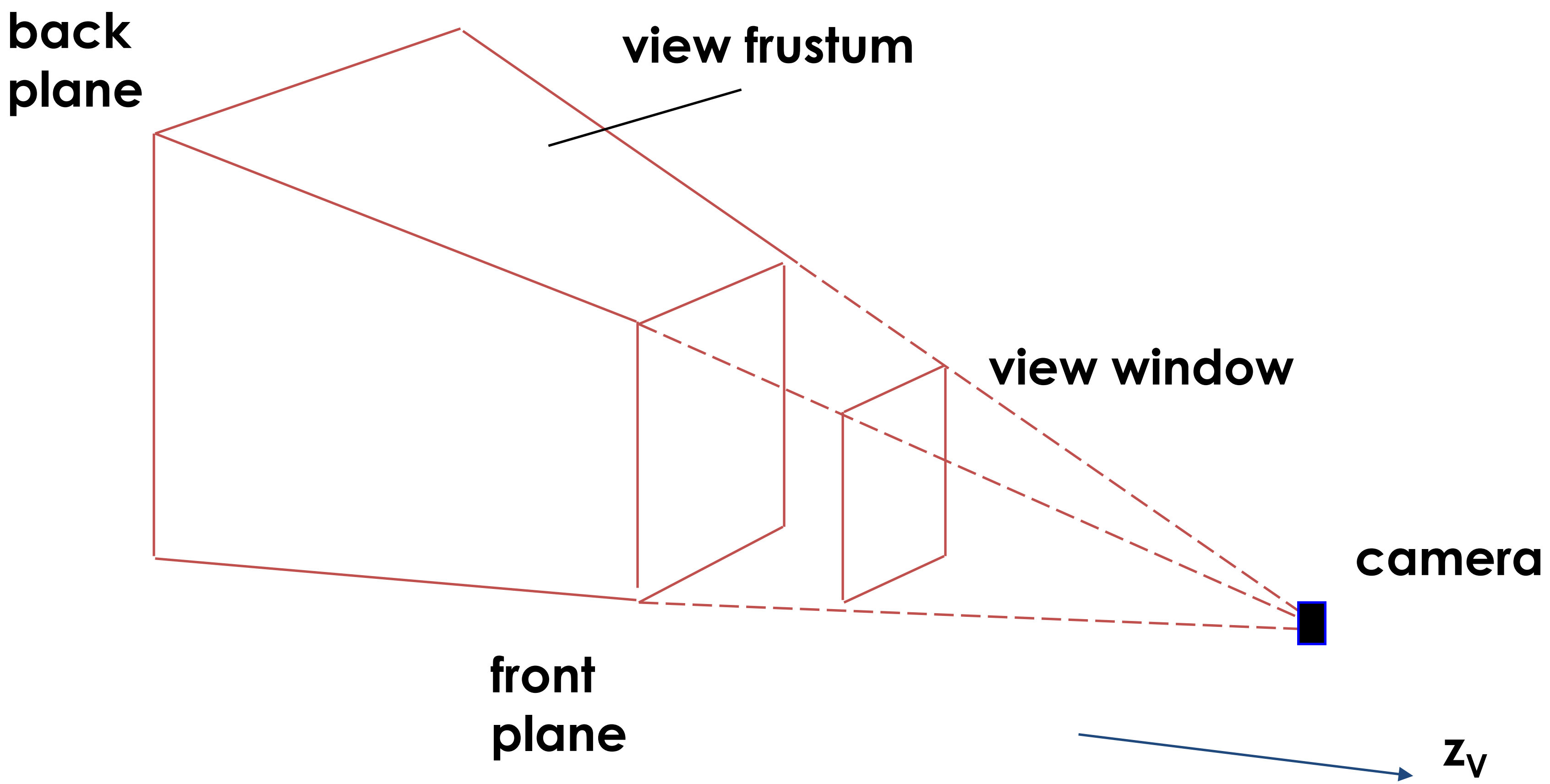
Viewing Pipeline So Far

- We now should understand this viewing pipeline

Clipping

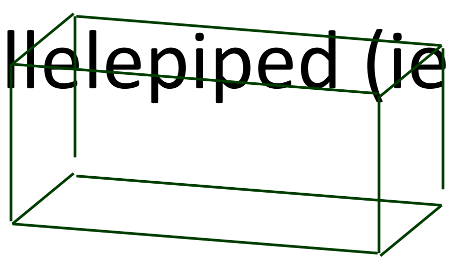
- Next we need to understand how the clipping to the view volume is performed
- Recall that with perspective projection we defined a view frustum outside of which we wanted to clip points and lines, etc
- The next slide is from lecture 3 ...

View Frustum - Perspective Projection



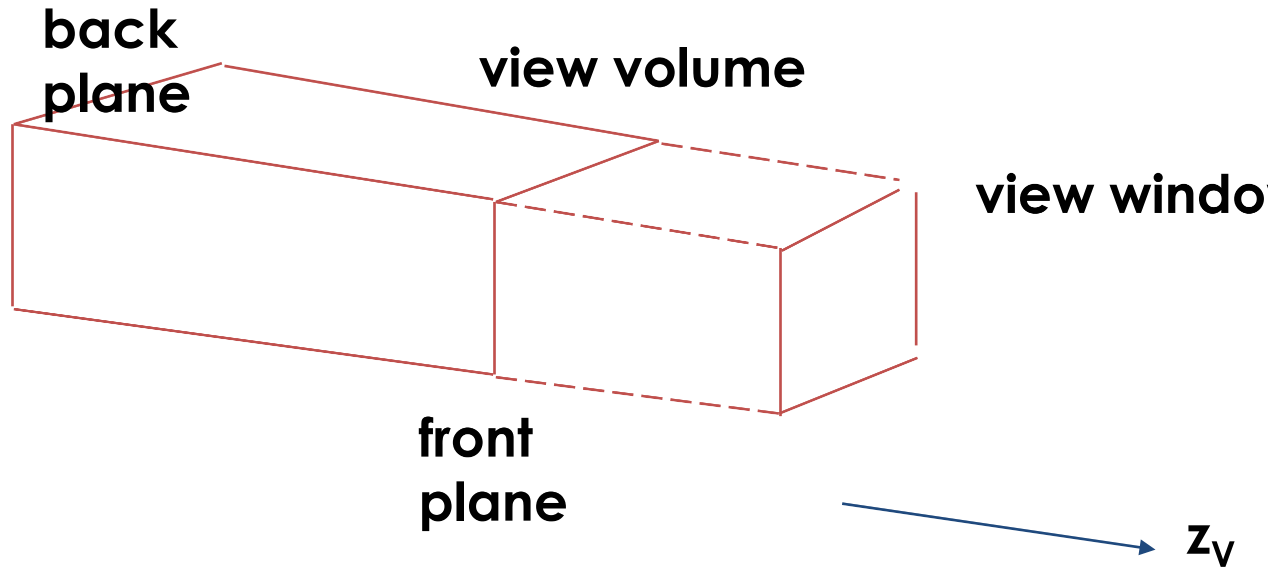
Clipping to View Frustum

- It is quite easy to clip lines to the front and back planes (just clip in z)..
- .. but it is difficult to clip to the sides because they are ‘sloping’ planes
- Instead we carry out the projection first which converts the frustum to a rectangular parallelepiped (ie a cuboid)



Clipping for Parallel Projection

- In the parallel projection case, the viewing volume is already a rectangular parallelepiped

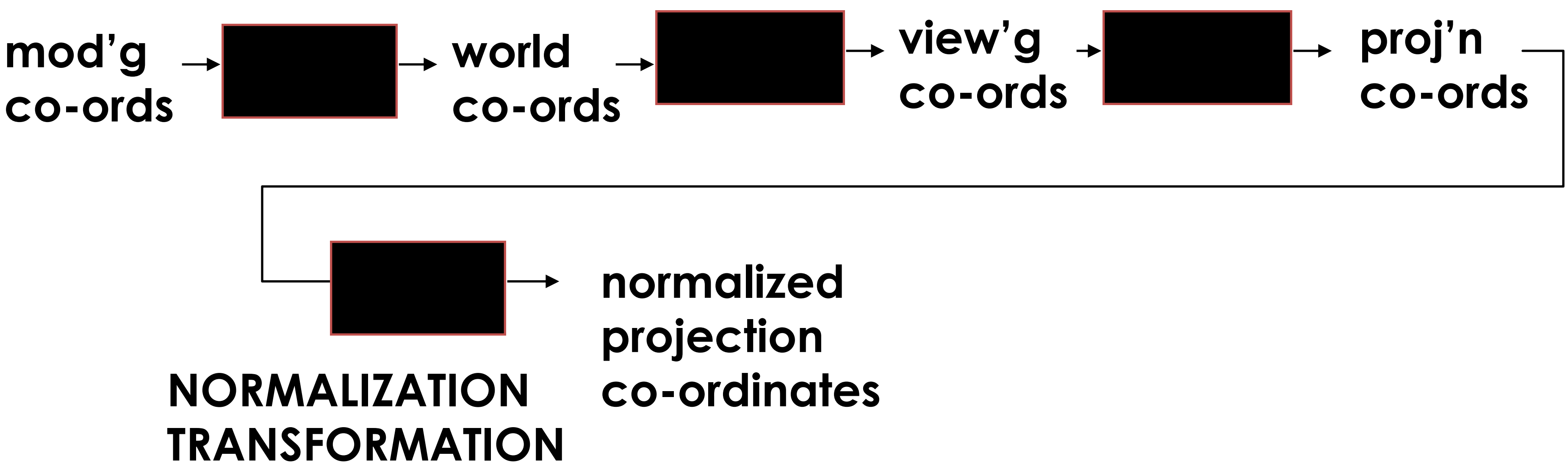


Normalized Projection Co-ordinates

- Final step before clipping is to normalize the co-ordinates of the rectangular parallelepiped to some standard shape
  - for example, in some systems, it is the cube with limits +1 and -1 in each direction
- This is just a scale transformation
- Clipping is then carried out against this standard shape

Viewing Pipeline So Far

- Our pipeline now looks like:



And finally...

- The last step is to position the picture on the display surface
- This is done by a viewport transformation where the normalized projection co-ordinates are transformed to display co-ordinates, ie pixels on the screen

Viewing Pipeline - The End

- A final viewing pipeline is therefore: