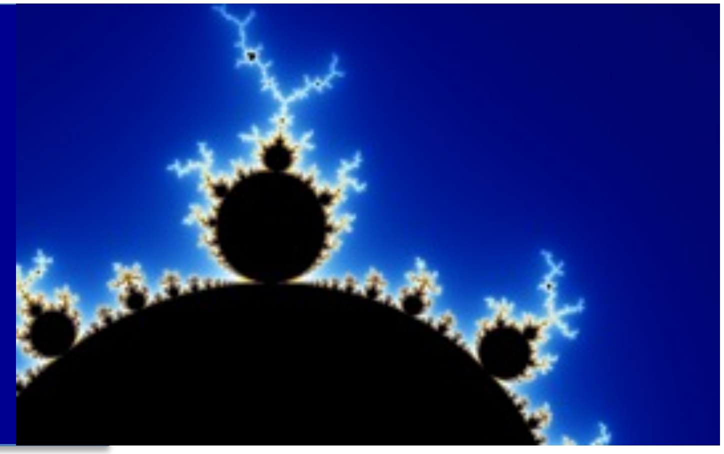
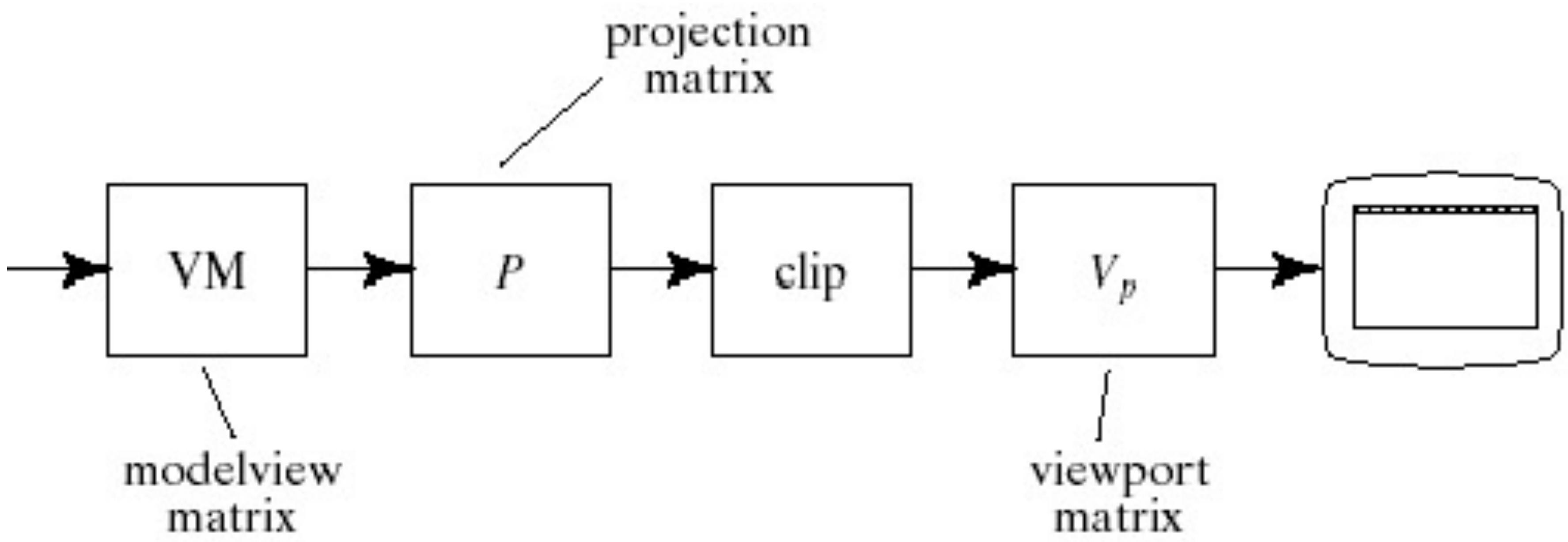
Computer Graphics



View Matrix (V)

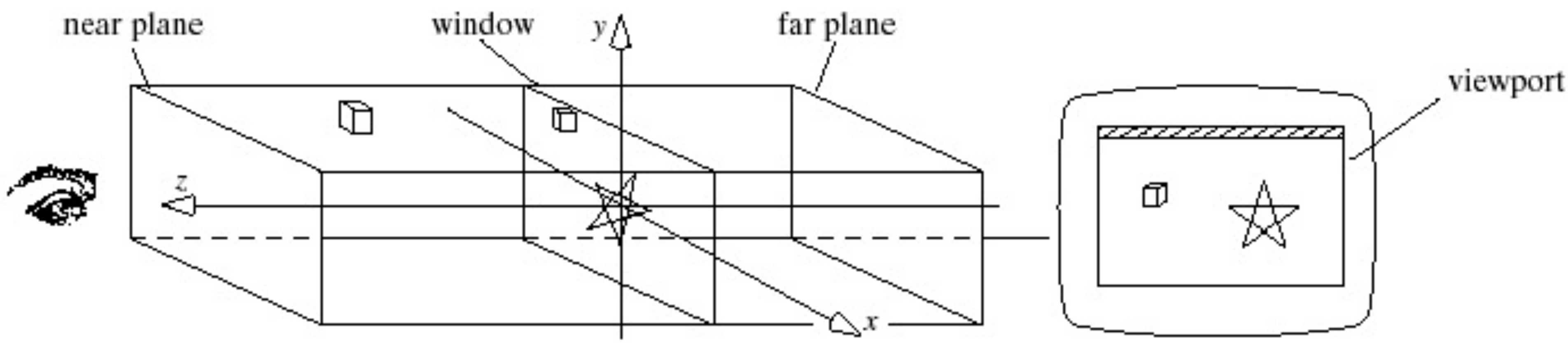
The Graphics Pipeline

- OpenGL provides functions for defining the view volume and its position in the scene, using matrices in the graphics pipeline.



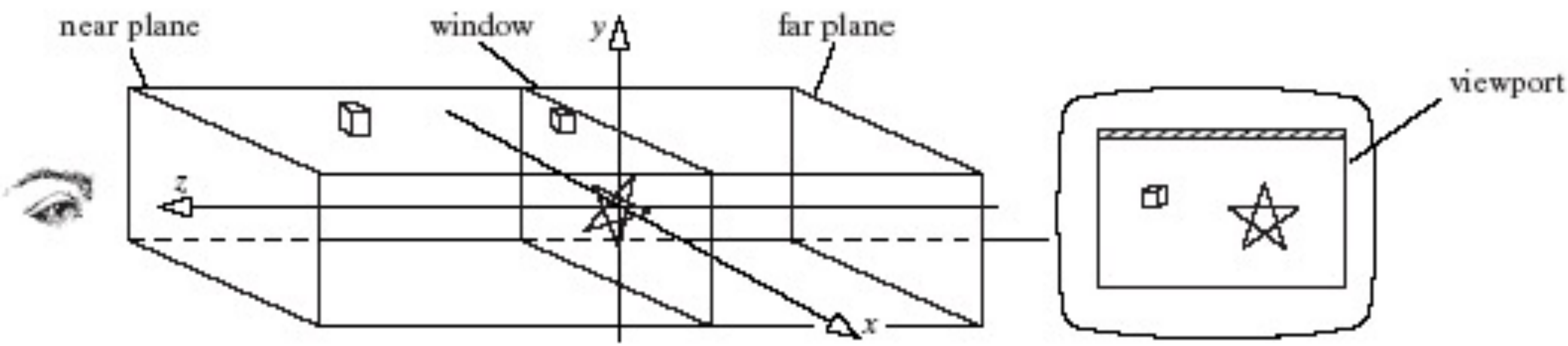
The Viewing Process

- The 2D drawing so far is a special case of 3D viewing, based on a simple parallel projection.
- The eye is looking along the z-axis at the world window, a rectangle in the *xy*-plane.



The Viewing Volume

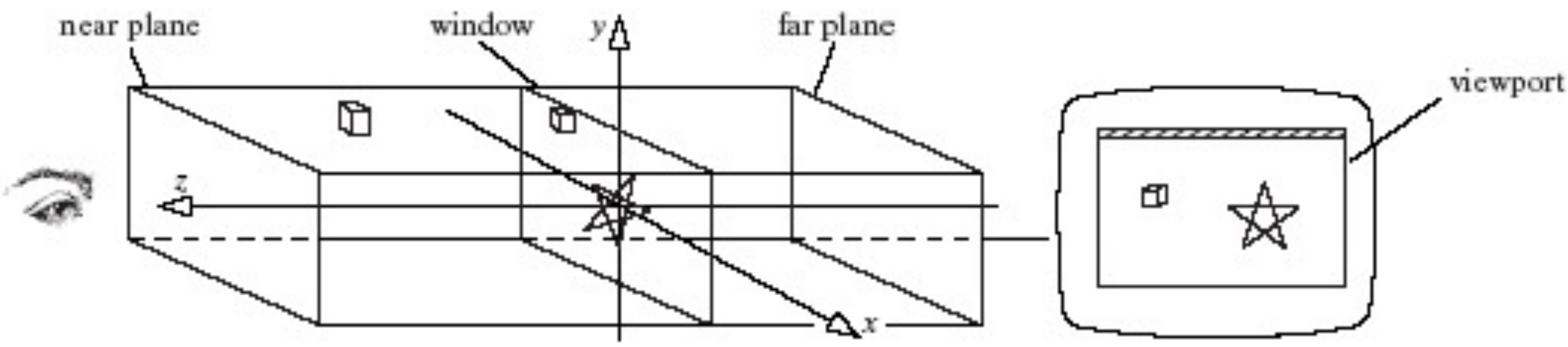
- *Eye* is simply a point in 3D space.
- The "orientation" of the eye ensures that the view volume is in front of the eye.
- Objects closer than *near* or farther than *far* are too blurred to see.



Different Images from different views

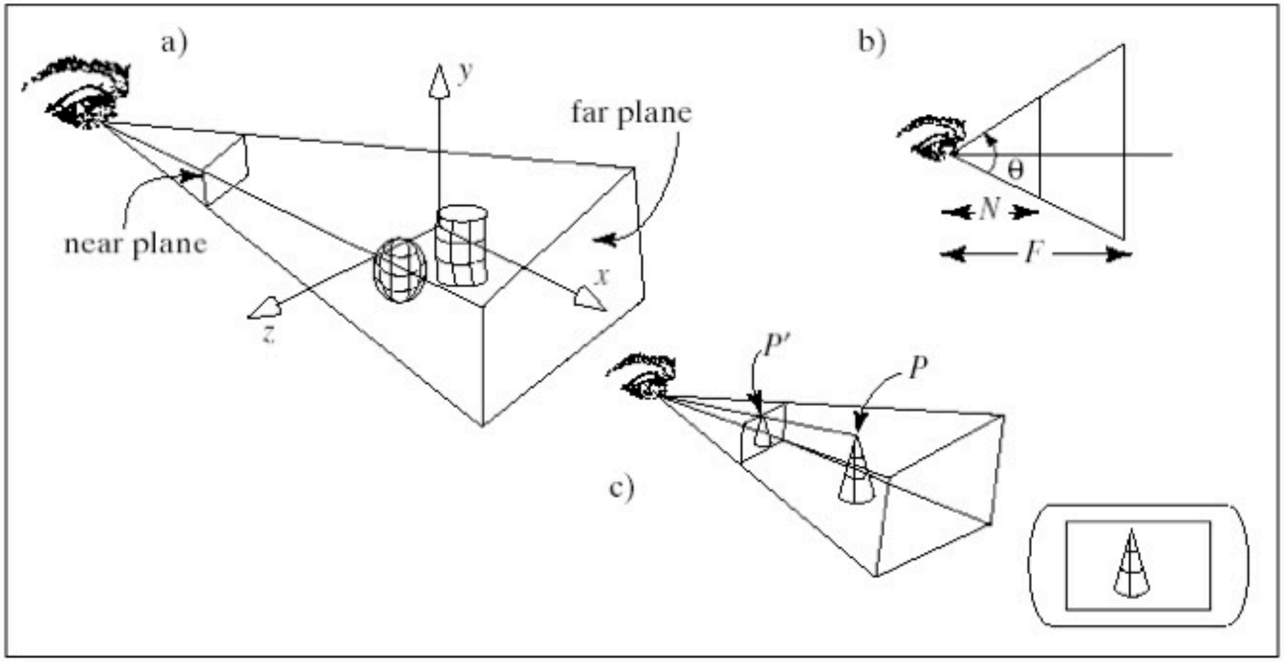
The Viewing Volume in Orthographic Projection

- The **view volume** of the camera is a rectangular parallelepiped (Orthographic Projection)
  - projectionMatrix=ortho(left, right, bottom, top, near, far)
- Its side walls are fixed by the window edges; its other two walls are fixed by a **near plane** and a **far plane**.



The Viewing Volume in Perspective Projection

- The **view volume** of the camera using Perspective Projection (to be discussed later)
- Its side walls are fixed by the window edges; its other two walls are fixed by a **near plane** and a **far plane**.

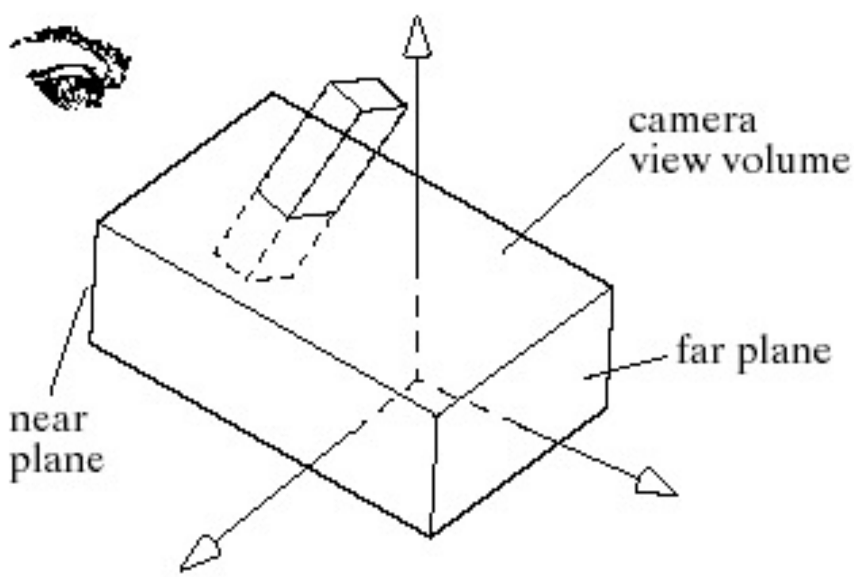


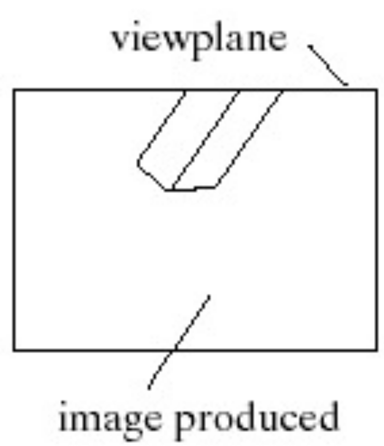
The Viewing Process in Orthographic Projection

- Points inside the view volume are projected onto the window along lines parallel to the z-axis.
- We ignore their z-component, so that the 3D point (x_1 y_1, z_1) projects to (x_1, y_1, 0).
- Points lying outside the view volume are clipped off.
- Everything inside it is projected along lines parallel to the axes onto the window plane (parallel projection).

The Viewing Process in Orthographic Projection

- In 3D, the only change we make is to allow the camera (eye) to have a more general position and orientation in the scene in order to produce better views of the scene.



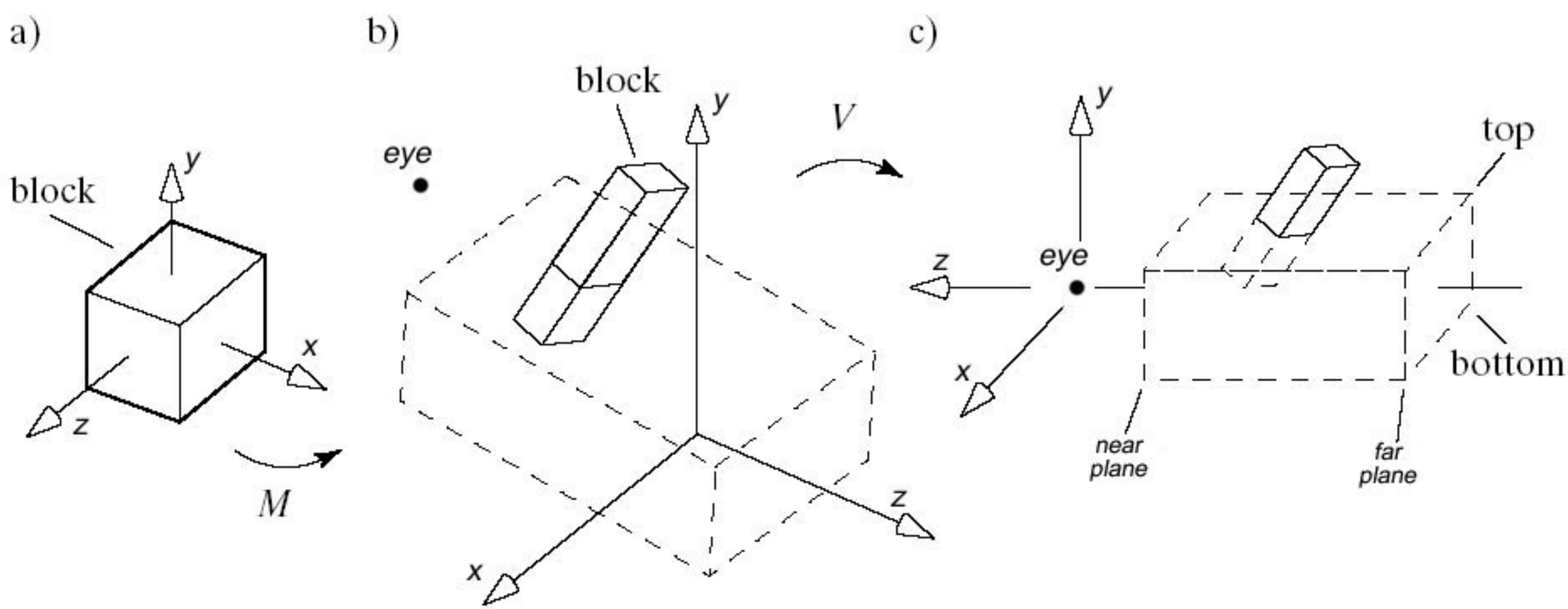


The Viewing Process and the Graphics Pipeline (9)

- Each vertex of an object is passed through this pipeline through Vertex Shader.
- The vertex is multiplied by the various matrices, clipped if necessary, and if it survives, it is mapped onto the viewport.
- Each vertex encounters three matrices:
  – The **modelview matrix;**
  – The **projection matrix;**
  – The **viewport matrix;**

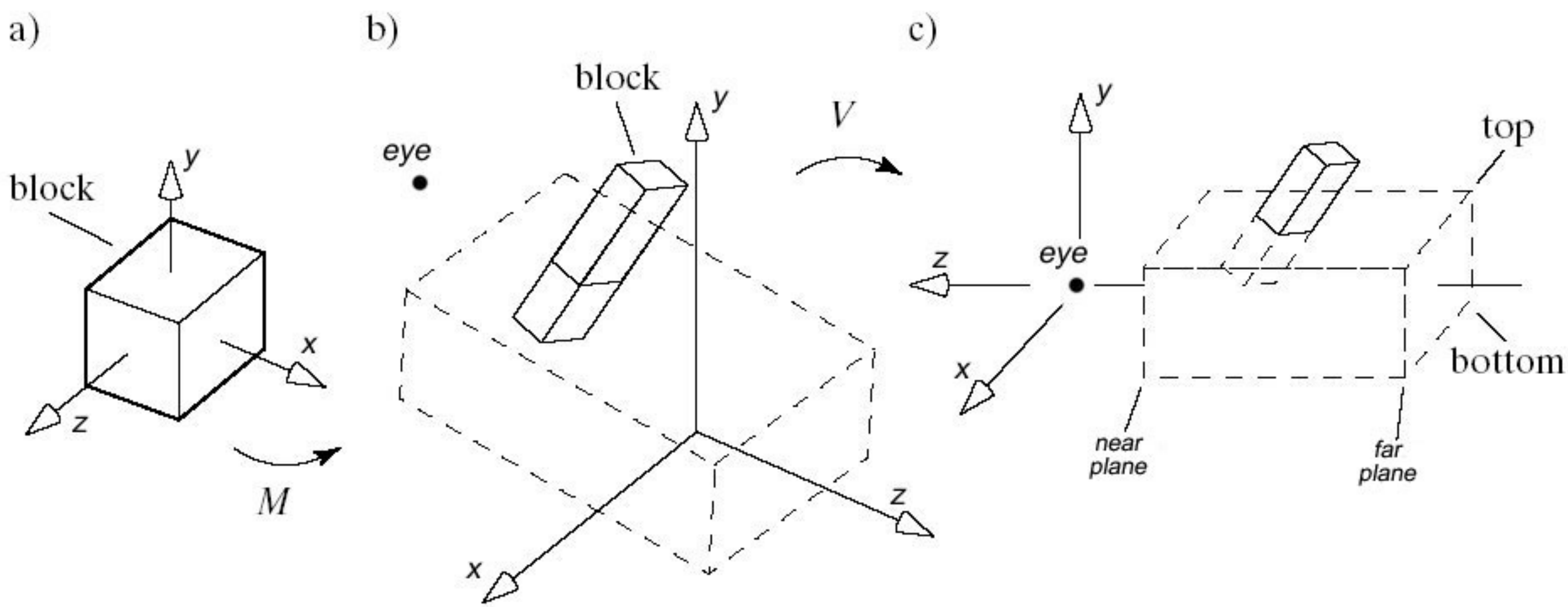
The Modelview Matrix

- A modeling transformation *M* scales, rotates, and translates the cube into the block.

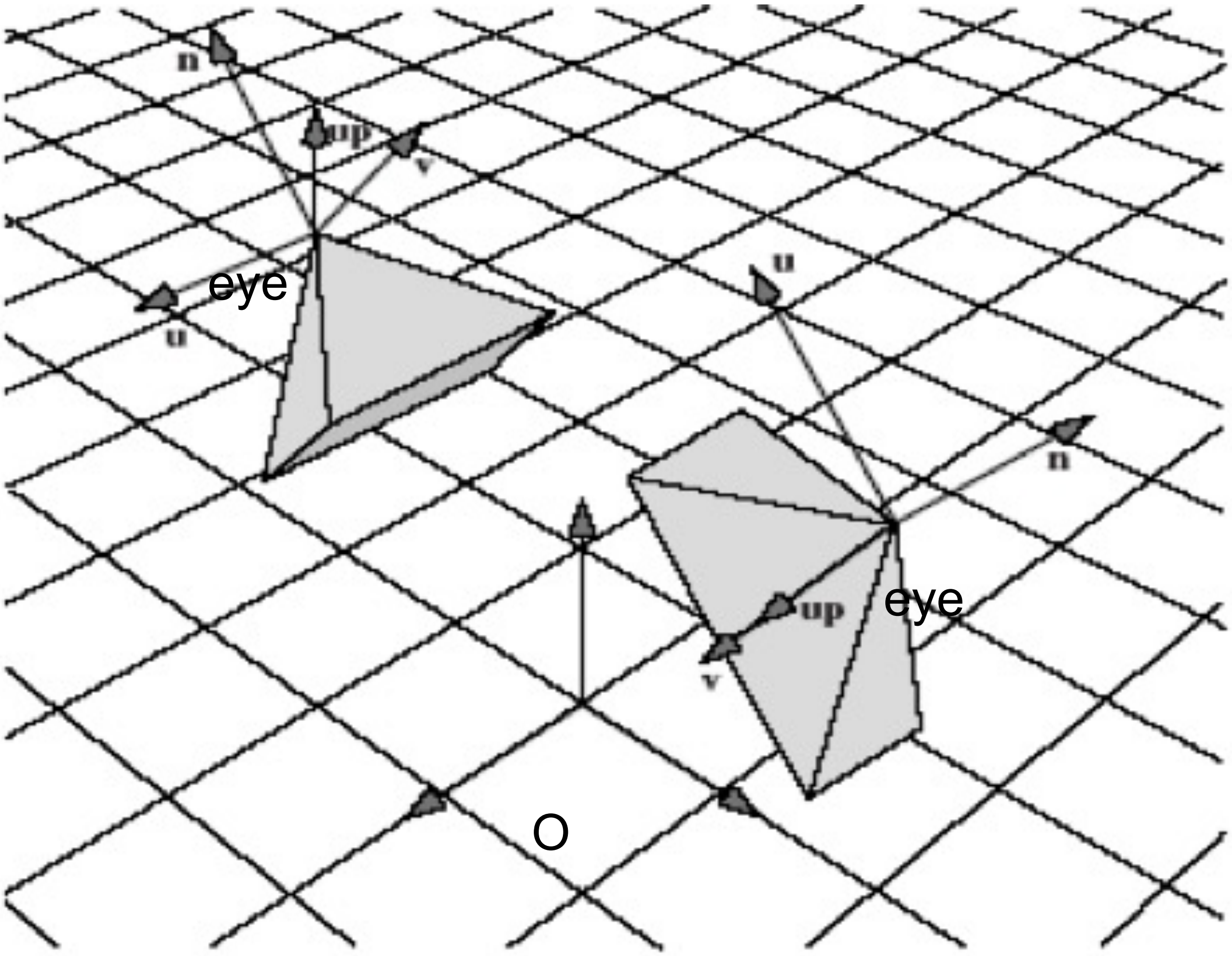


The Modelview Matrix (V)

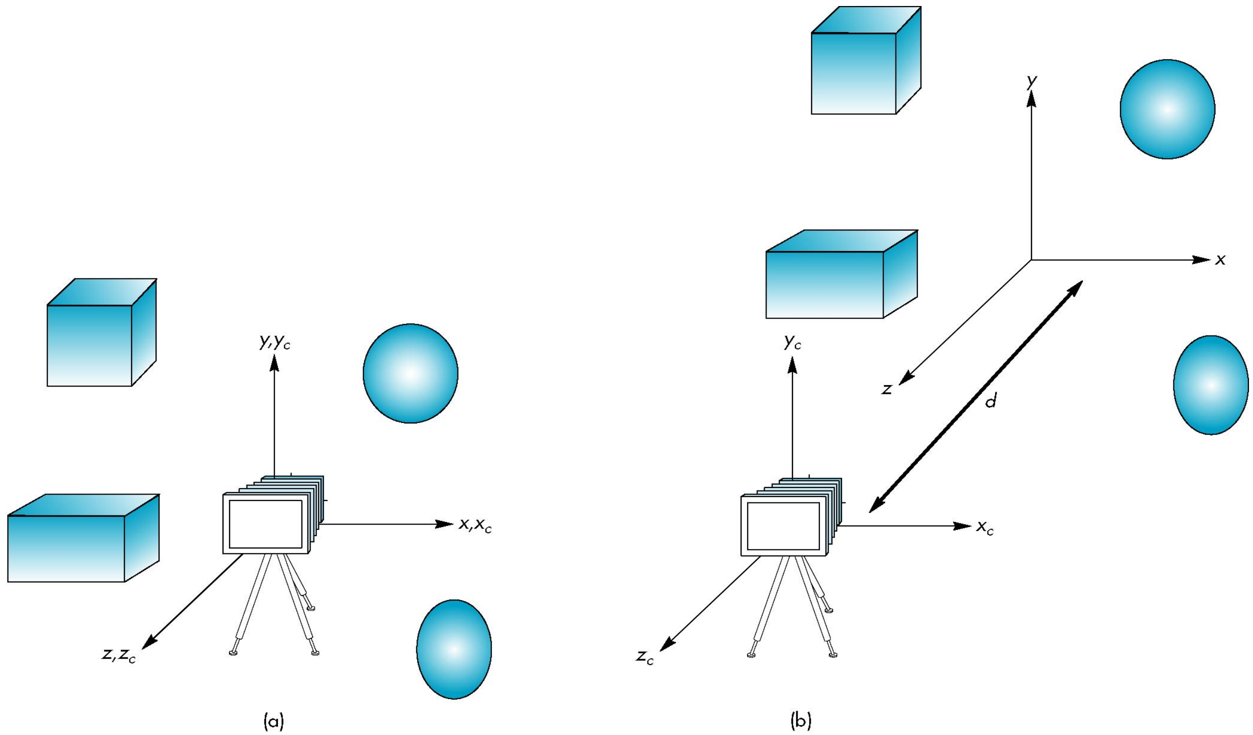
- The camera moves from its position in the scene to its generic position (eye at the origin and the view volume aligned with the *z*-axis).
- The coordinates of the block's vertices are changed so that projecting them onto a plane (e.g., the near plane) displays the projected image properly.



The effect of lookAt



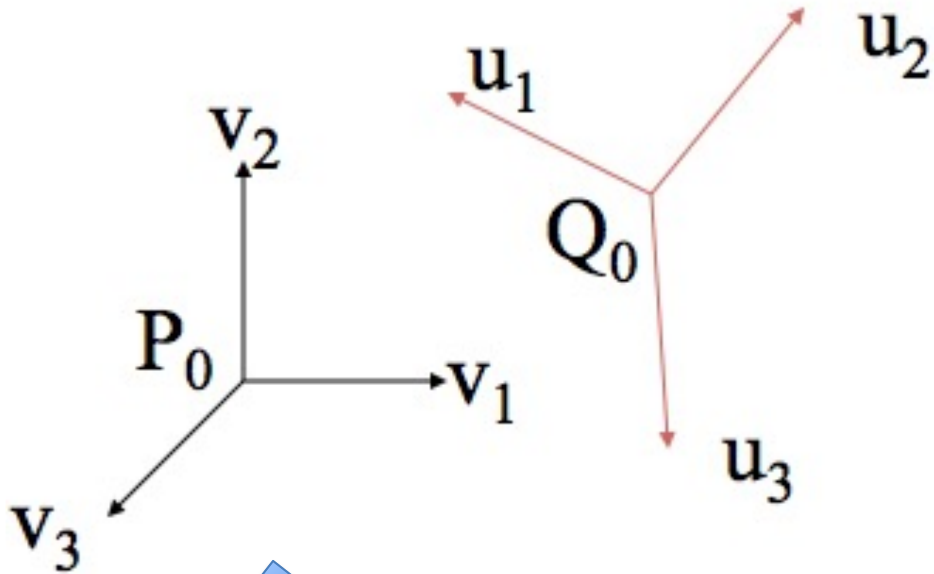
Change of Frame



Moving the camera (Change from the world frame to the camera frame)

- Initially these frames overlaps (V=Identity)
- Since objects are on both sides of z=0 plane, we must move the camera
- Afterwards, the objects need to be transformed into the camera frame by changing their locations in the world frame to the camera frame using the view matrix

How to change from one frame to a new frame in general?



$$
\begin{aligned}
u_1 &= \gamma_{11}v_1+\gamma_{12}v_2+\gamma_{13}v_3 \\
u_2 &= \gamma_{21}v_1+\gamma_{22}v_2+\gamma_{23}v_3 \\
u_3 &= \gamma_{31}v_1+\gamma_{32}v_2+\gamma_{33}v_3 \\
Q_0 &= \gamma_{41}v_1+\gamma_{42}v_2+\gamma_{43}v_3+\gamma_{44}P_0
\end{aligned}
$$

$$
T=\begin{bmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{bmatrix}
$$

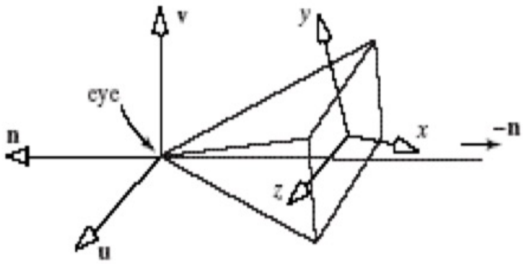
The Modelview Matrix (V)

- The matrix *V* changes the coordinates of the scene vertices into the **camera's coordinate system,** or into **eye coordinates**.

To inform WebGL that we wish it to operate on the ModelView matrix we call

```
viewMatrix = lookAt (eye,   // eye position
                     at,    // the "look at" point
                     up)    // approximation to true up direction
```

// Then do the modeling transformations



Setting Up the Camera (2)

- What lookAt function does is create a camera coordinate system of three mutually orthogonal unit vectors: **u**, **v**, and **n.**
  - **n** = eye - look;
  - **u** = **up** x **n**;
  - **v** = **n** x **u**
- Normalize **n**, **u**, **v** (in the camera system) and
- let **e** = eye - $\mathcal{O}$ in the camera system, where $\mathcal{O}$ is the origin.

Setting Up the Camera (3)

- Then lookAt () sets up the view matrix

$$V = \begin{pmatrix} u_x & u_y & u_z & d_x \\ v_x & v_y & v_z & d_y \\ n_x & n_y & n_z & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  where **d** = (-**e**·**u**, -**e**·**v**, -**e**·**n**)

- **up** is usually (0, 1, 0) (along the y-axis), look is frequently the middle of the window, and *eye* frequently looks down on the scene.

Practice Question

- Given: lookAt (4, 4, 4, 0, 1, 0, 0, 1, 0);

What is the View matrix V?

Steps:

1. Compute vectors n, u, v
2. Normalize n, u, v
3. Compute vector: **e** = eye – $\mathcal{O}$

   d = (-**e**·**u**, -**e**·**v**, -**e**·**n**)
4. Put the view matrix together

Practice Question

$$\mathbf{n} = eye - look = (4, 3, 4)$$

$$\mathbf{u} = \mathbf{up} \times \mathbf{n} = \begin{vmatrix} \mathbf{i} & \mathbf{j} & \mathbf{k} \\ 0 & 1 & 0 \\ 4 & 3 & 4 \end{vmatrix} = (4, 0, -4)$$

$$\mathbf{v} = \mathbf{n} \times \mathbf{u} = (-12, 32, -12)$$

Normalize **u**, **v**, and **n**,

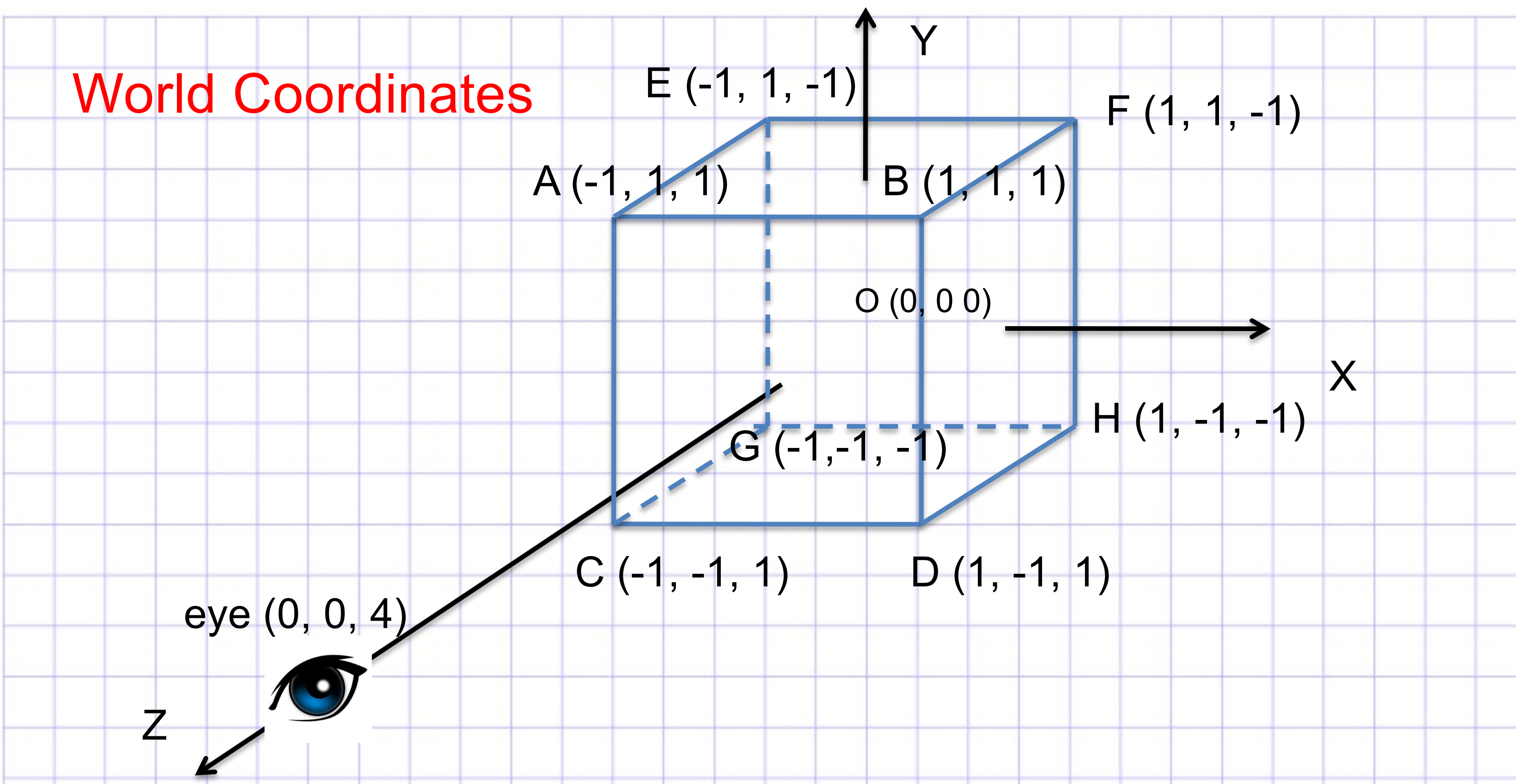
$$\mathbf{n} = (0.6247, 0.4685, 0.6247)$$
$$\mathbf{u} = (0.70711, 0, -0.70711)$$
$$\mathbf{v} = (-0.3313, 0.8834, -0.3313)$$

dx =**-e**.**u** =(-4, -4, -4) **.** (0.70711, 0, -0.70711) = 0
dy =**-e**.**v** =(-4, -4, -4) **.** (-0.3313, 0.8834, -0.3313) = -0.88345
dz =**-e**.**n** =(-4, -4, -4) **.** (0.6247, 0.4685, 0.6247) = -6.872

e = eye – O – (4, 4, 4) -e = (-4, -4, -4)
So that,

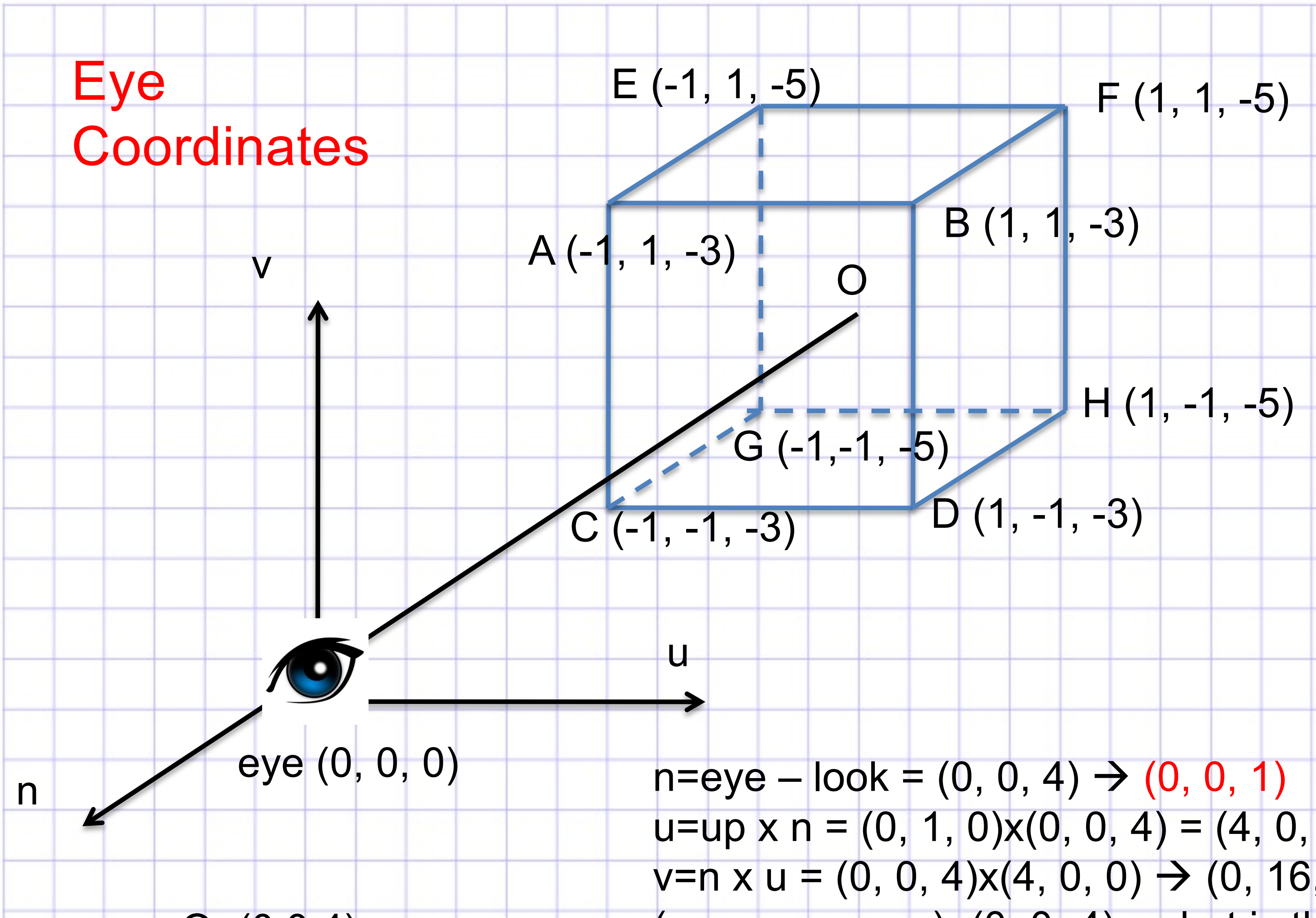
$$V = \begin{pmatrix} .70711 & 0 & -.70711 & 0 \\ -.3313 & .88345 & -.3313 & -.88345 \\ .6247 & .4685 & .6247 & -6.872 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

World vs. Eye Coordinates



ortho(-3, 3, -3, 3, 2, 6);
lookAt(0, 0, 4, 0, 0, 0, 0, 1, 0);

Transform into Eye Frame



n=eye – look = (0, 0, 4) → (0, 0, 1)

u=up x n = (0, 1, 0)x(0, 0, 4) = (4, 0, 0) → (1, 0, 0)

v=n x u = (0, 0, 4)x(4, 0, 0) → (0, 16, 0)→ (0, 1, 0)

e=eye - O=(0,0,4)

(-**e**·**u**, -**e**·**v**, -**e**·**n**)=(0, 0,-4)   what is the view matrix?

Transform into Eye Frame

n=eye – look = (0, 0, 4) → (0, 0, 1)
u=up x n = (0, 1, 0)x(0, 0, 4) = (4, 0, 0) → (1, 0, 0)
v=n x u = (0, 0, 4)x(4, 0, 0) → (0, 16, 0)→ (0, 1, 0)
(-**e**·**u**, -**e**·**v**, -**e**·**n**)=(0, 0,-4)

what is the view matrix?

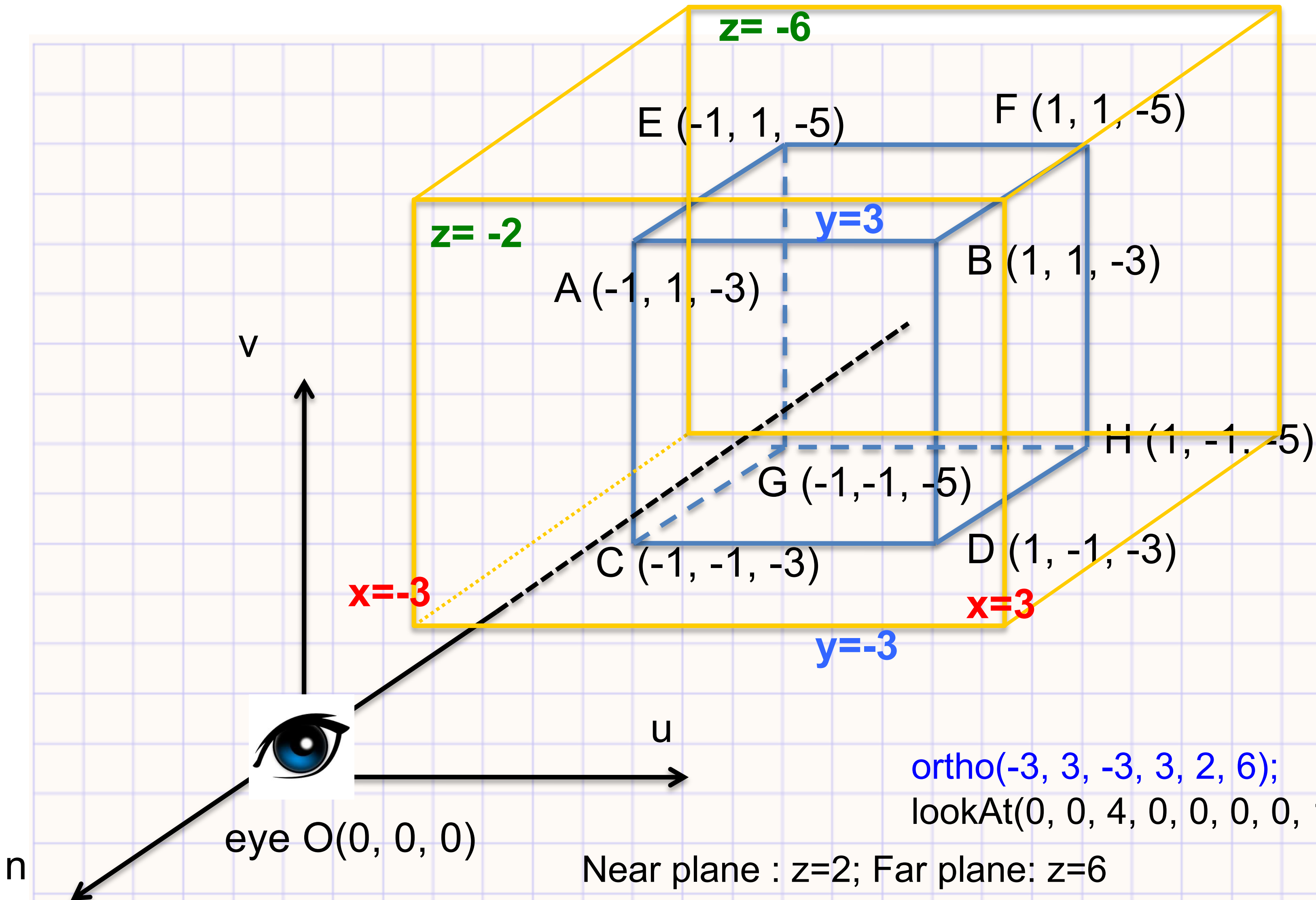
$$V=\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & -4\\0 & 0 & 0 & 0\end{bmatrix}$$

What are values of the points A, B, C and D in the Eye frame?

$$A' = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 1 & -4\\0 & 0 & 0 & 0\end{bmatrix} * \begin{bmatrix}-1\\1\\1\\1\end{bmatrix} = \begin{bmatrix}-1\\1\\1-4\\1\end{bmatrix} = \begin{bmatrix}-1\\1\\-3\\1\end{bmatrix}$$

How about B', C' and D'?

Adding the View Volume

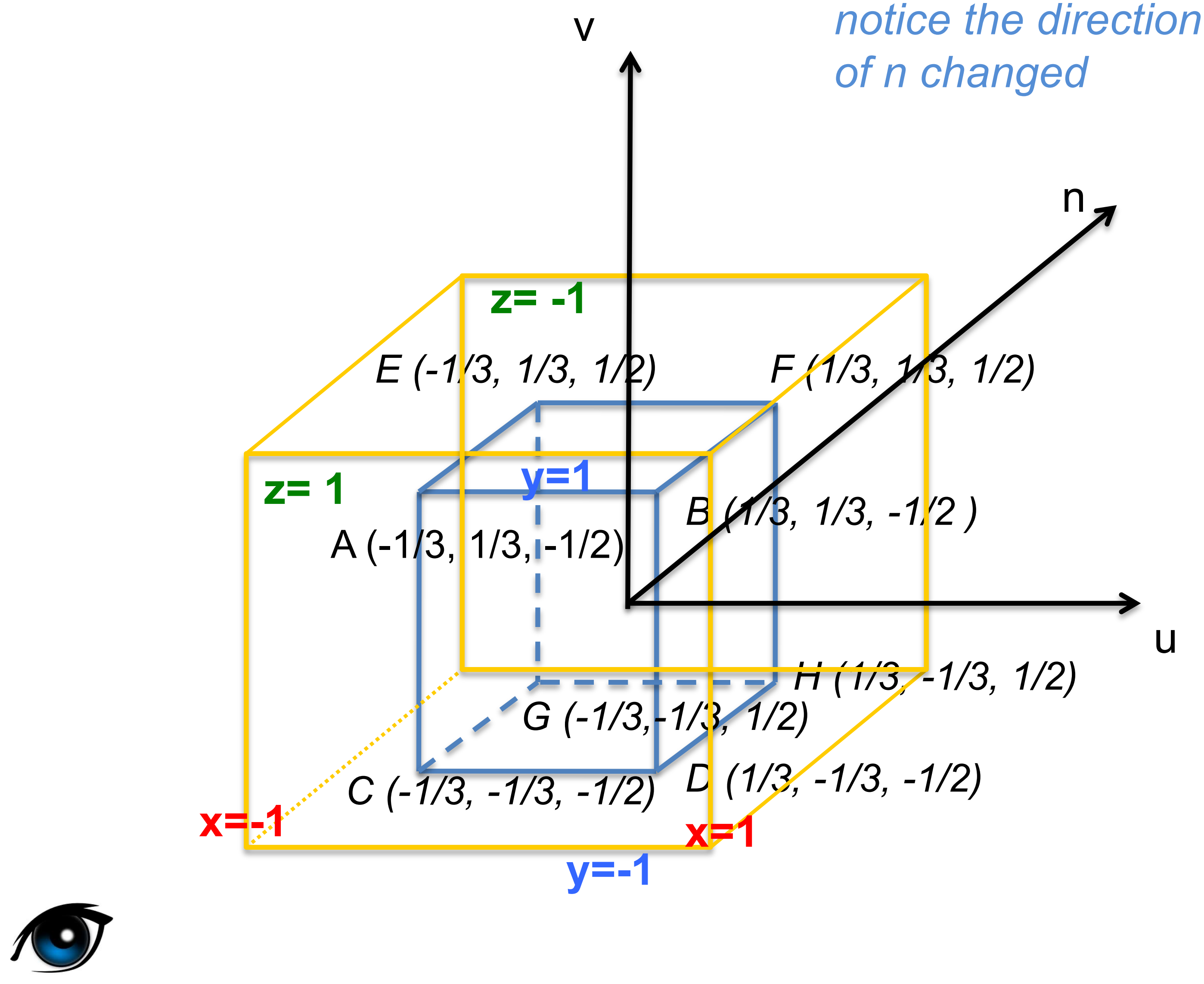


ortho(-3, 3, -3, 3, 2, 6);
lookAt(0, 0, 4, 0, 0, 0, 0, 1, 0);

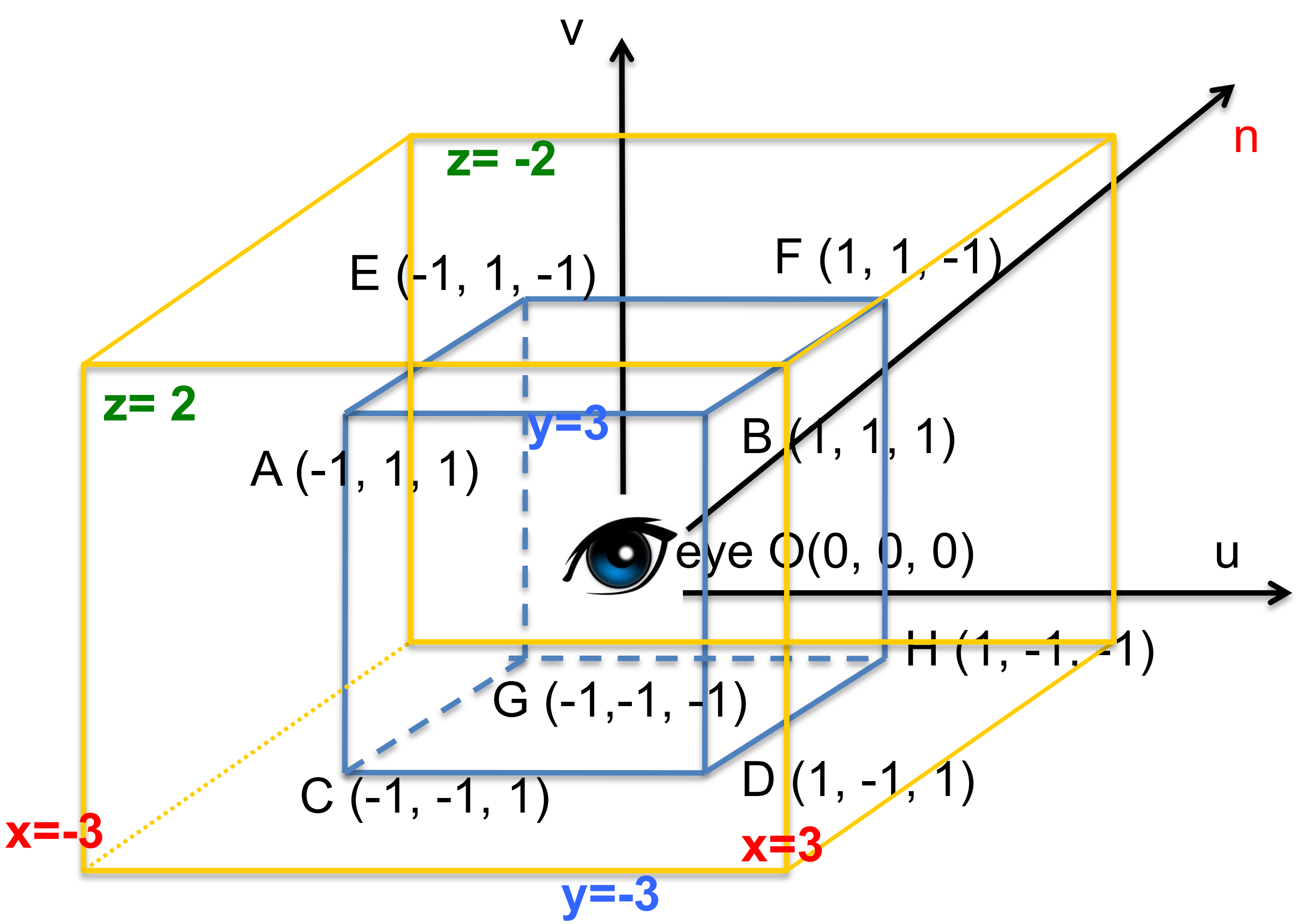
Near plane : z=2; Far plane: z=6
Near plane: z=-2, Far plane: z=-6 (eye coordinates)

Eye Coordinates

Projection (step 1: scale the view volume to CCV)

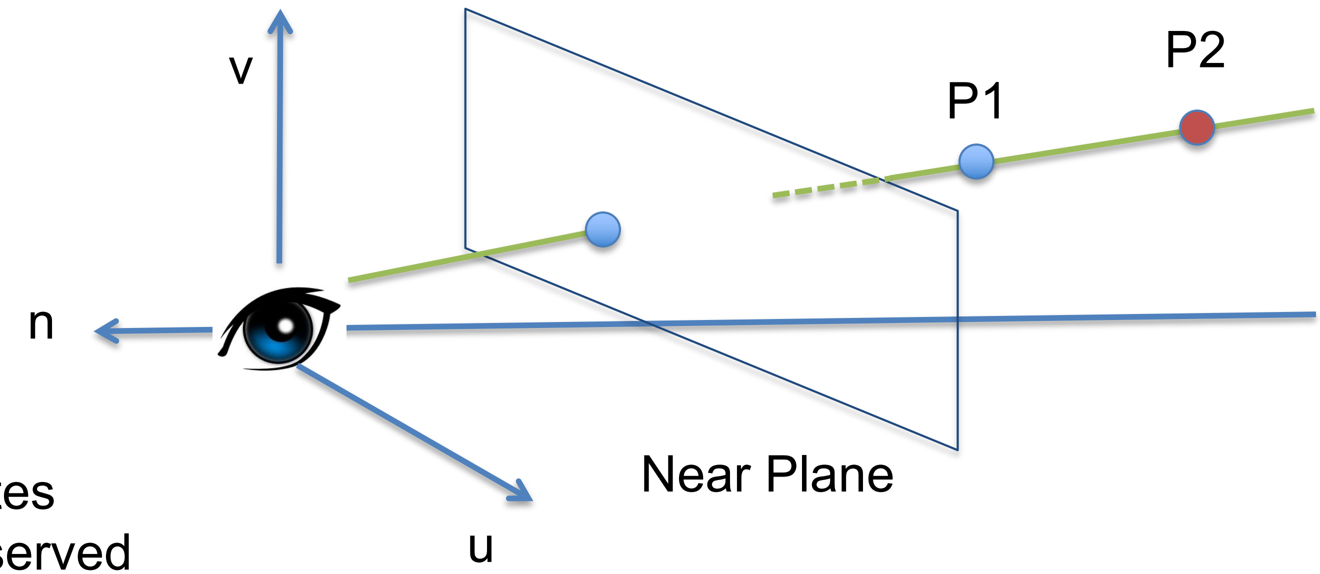


Projection (step 2: translate the center of the view volume to eye location)



Depth info → Depth buffer

- We need to add depth information
- Depth information tells which point/surface is in front of other point/surface, for hidden surface removal.



The x and y coordinates of each point are preserved using the orthographic projection

Incorporating Perspective in the Graphics Pipeline (2)

- We use a projection point

  $(x^*, y^*, z^*) = [N/(-P_z)](P_x, P_y, (a + b/P_z)),$

  Pseudo-depth

  and choose a and b so that

  $P_z^* = -1$ when $P_z = -N$ and 1 when $P_z = -F$.

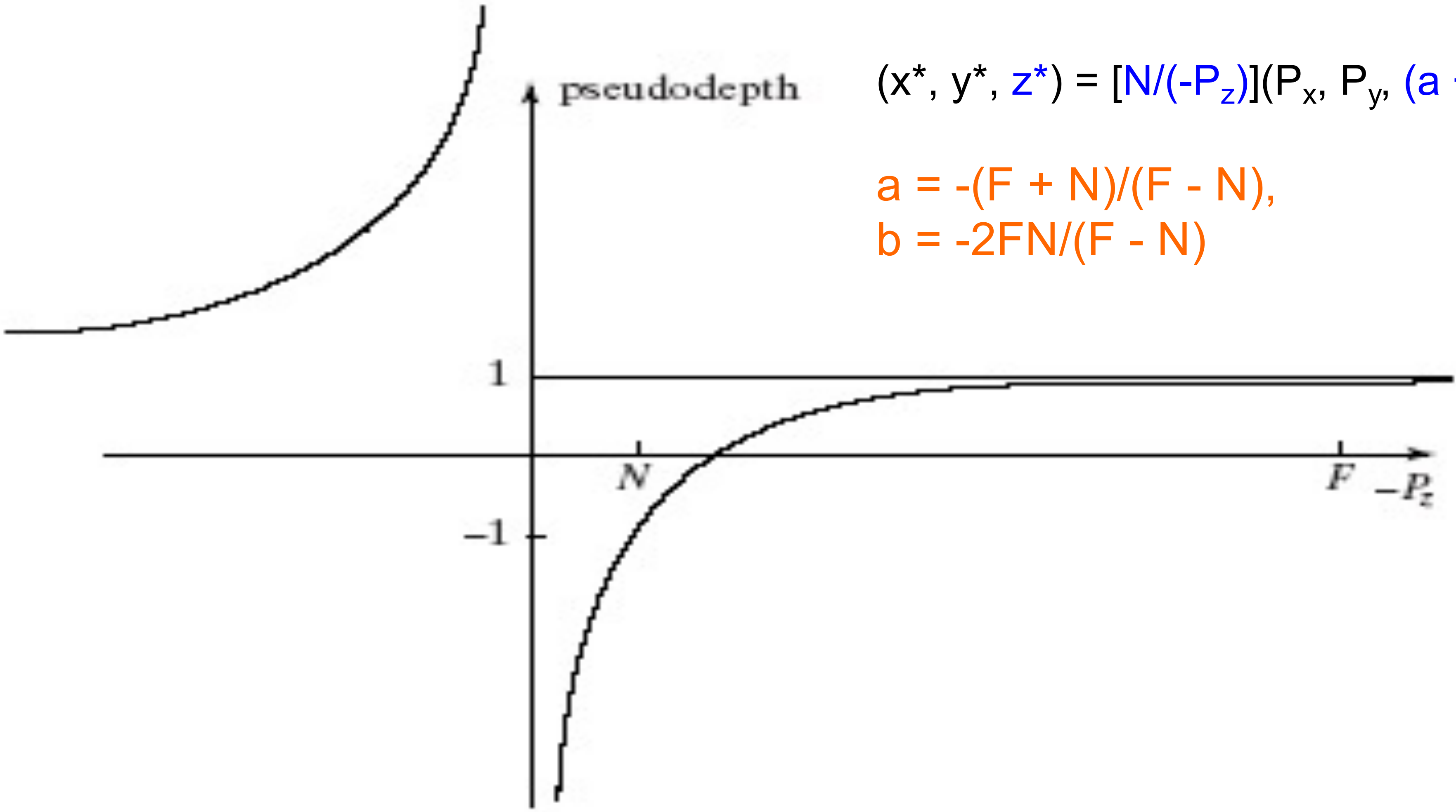
- Result:

  $a = -(F + N)/(F - N),$

  $b = -2FN/(F - N).$

- P_z^* increases (becomes more positive) as P_z decreases (becomes more negative, moves further away).

Illustration of Pseudo-depth Values



$(x^*, y^*, z^*) = [N/(-P_z)](P_x, P_y, (a + b/P_z))$

$a = -(F + N)/(F - N)$,
$b = -2FN/(F - N)$