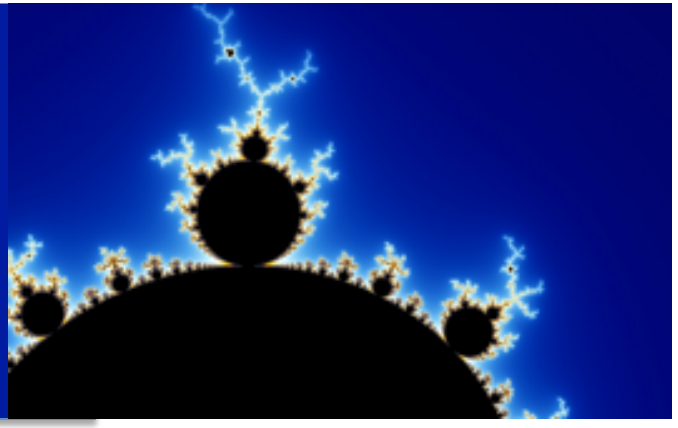
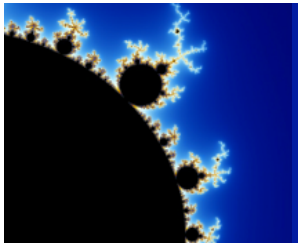
Computer Graphics

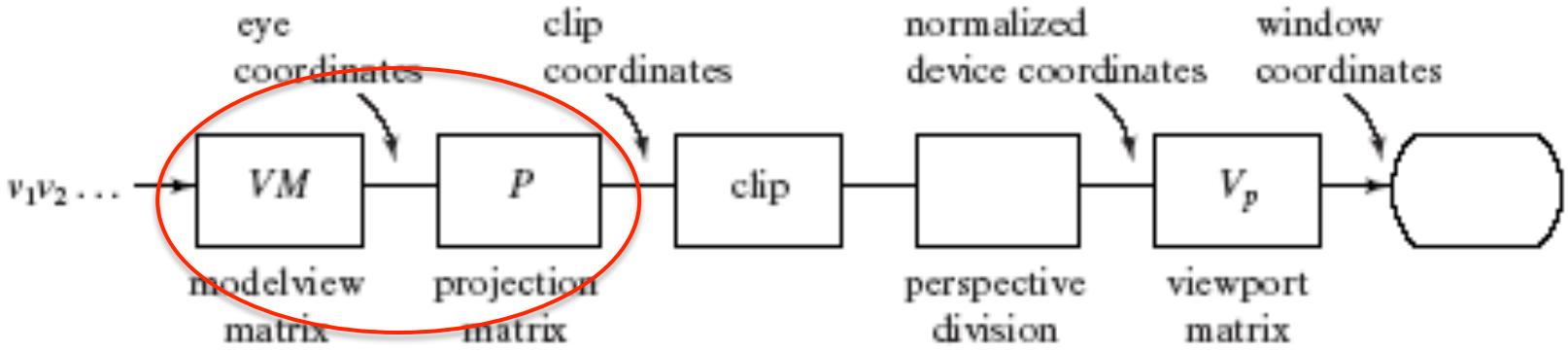


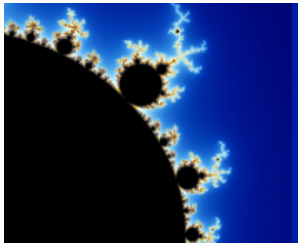
Orthographic vs. Perspective Projection



Projections of 3-D Objects

- The graphics pipeline: vertices start in world coordinates; after MV, in eye coordinates, after P, in clip coordinates; after perspective division, in normalized device coordinates; after V, in screen coordinates.

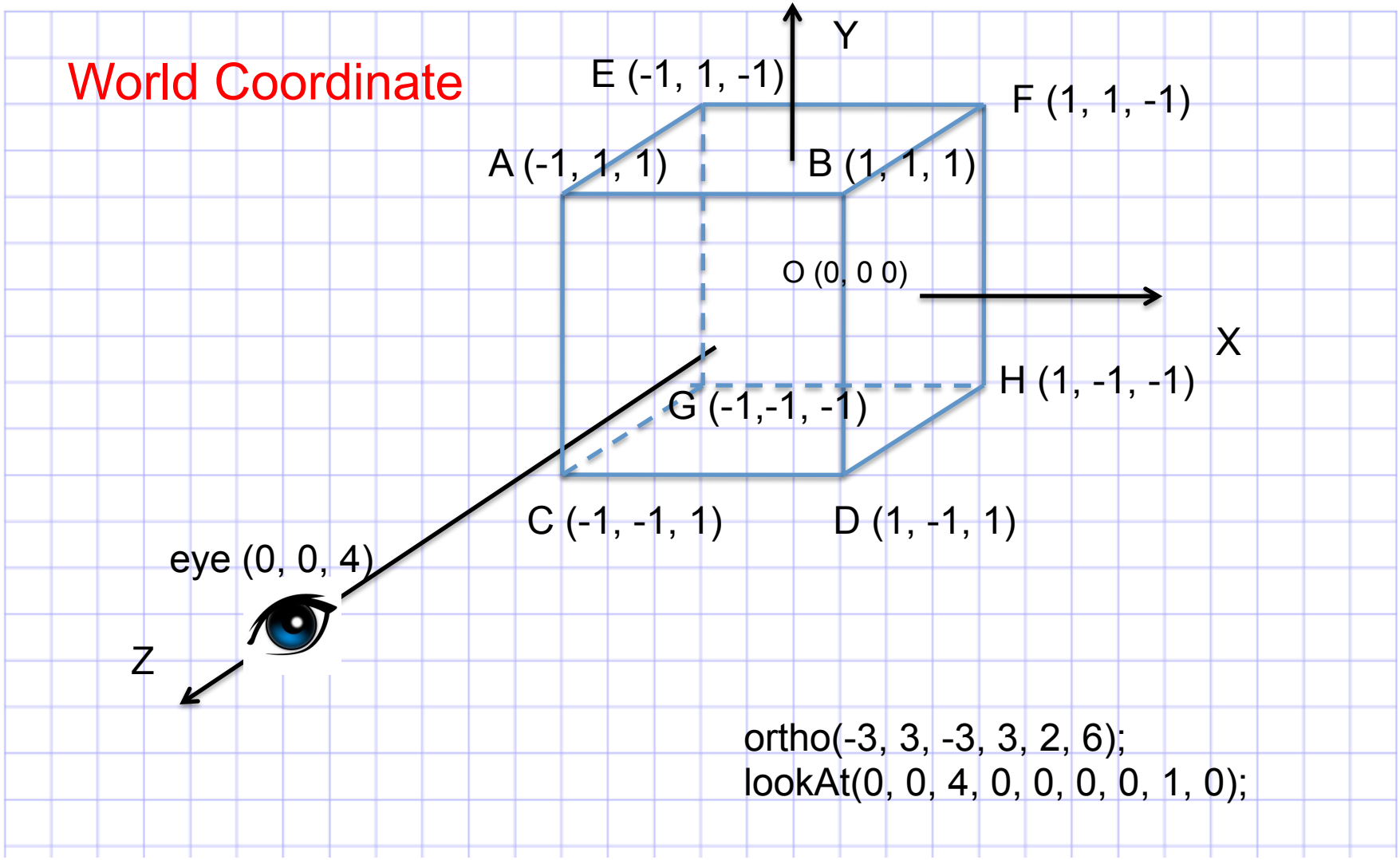


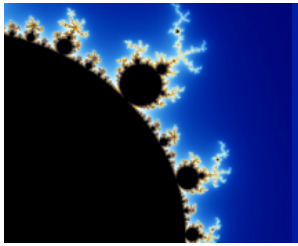


Projections of 3-D Objects (2)

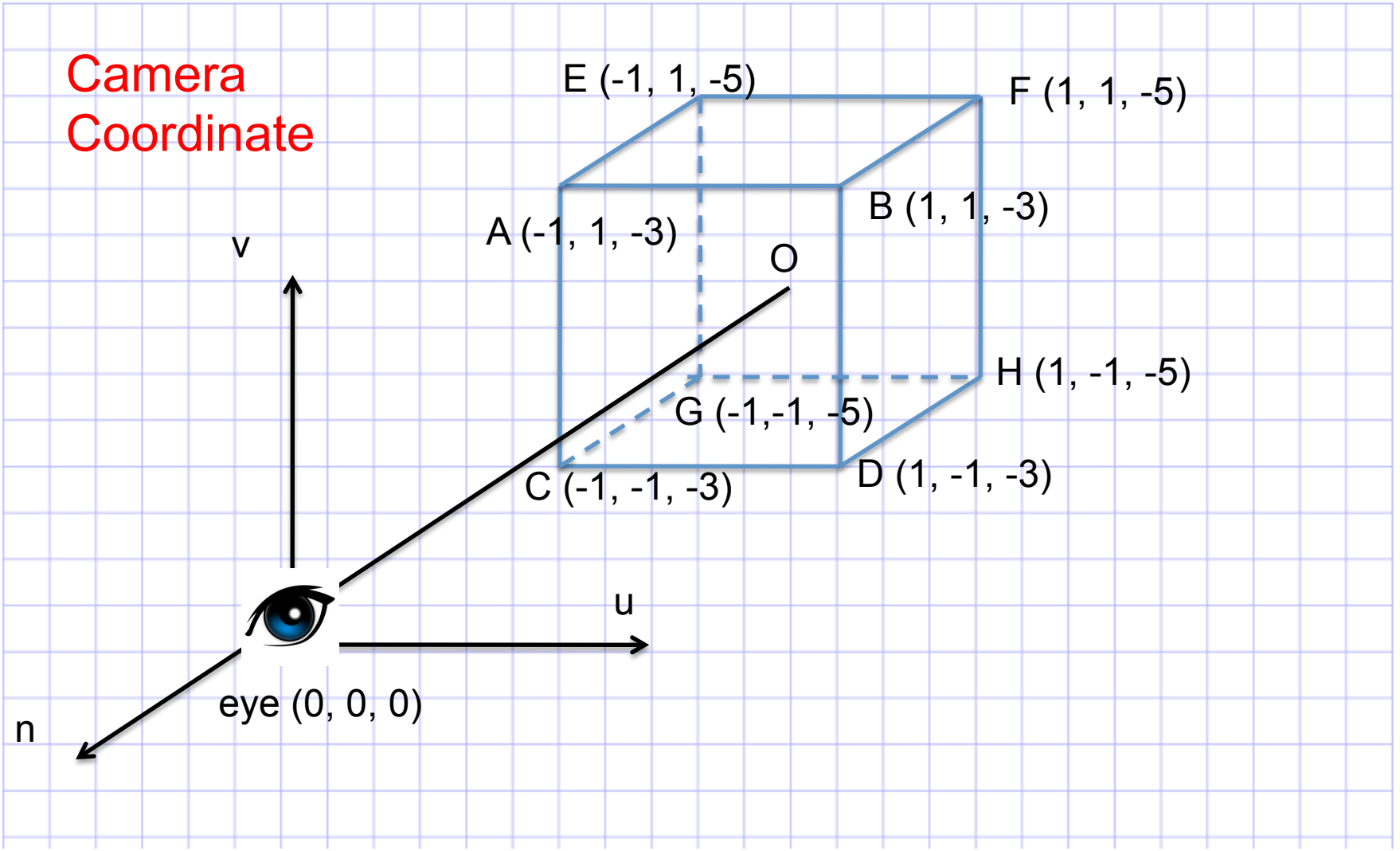
- Each vertex *v* is multiplied by the modelview matrix (*VM*), containing all of the modeling transformations for the object; the viewing part (*V*) accounts for the transformation set by the camera's position and orientation. When a vertex emerges from this matrix it is in **eye coordinates**, that is, in the coordinate system of the eye.
- The figure shows this system: the eye is at the origin, and the near plane is perpendicular to the *z*-axis, located at $z = -N$.

World vs. Camera Coordinates

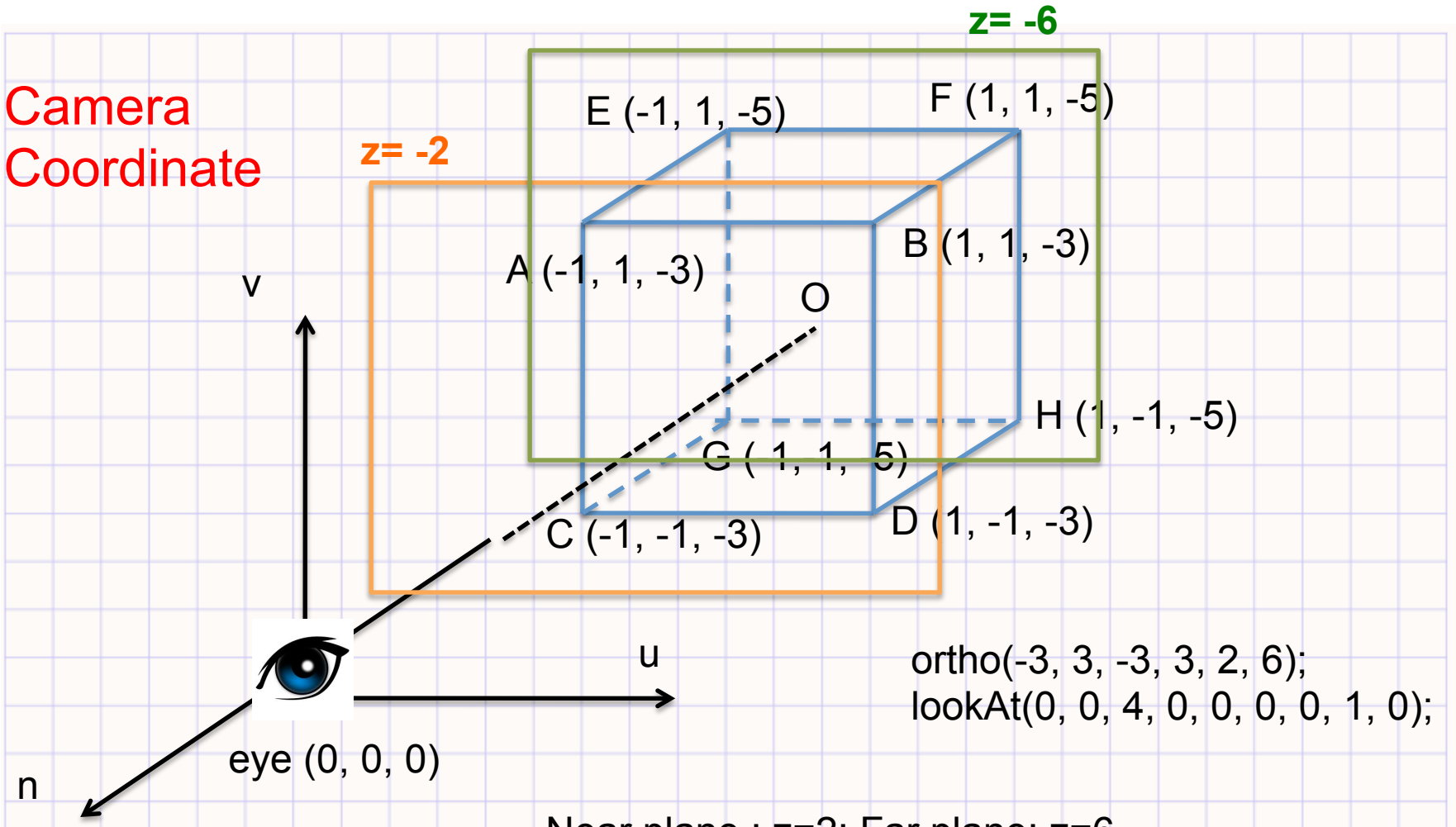


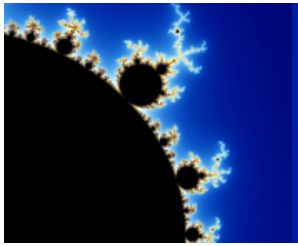


World vs. Camera Coordinates(2)



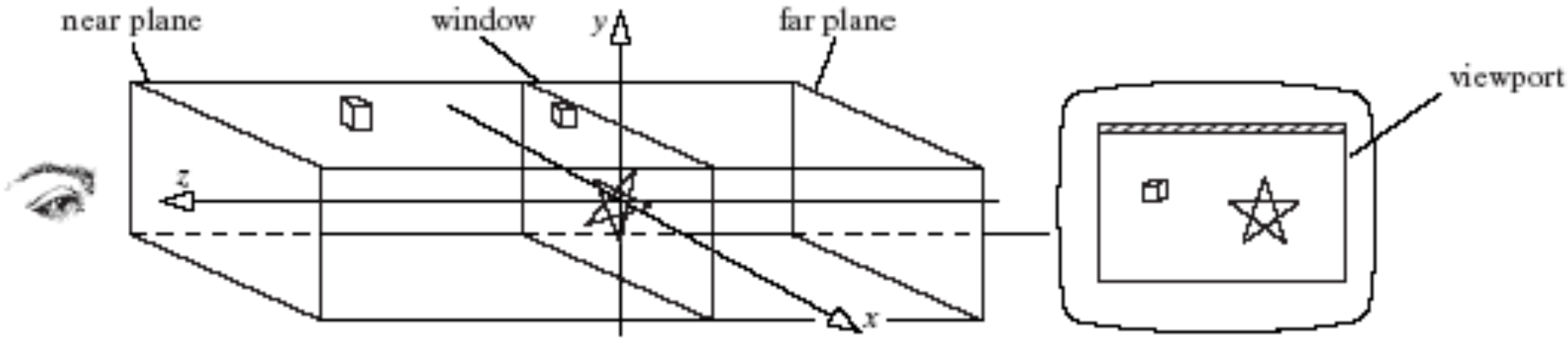
World vs. Camera Coordinates(3)



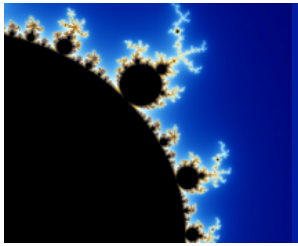


Orthographic Projection Matrix P

- The **view volume** of the camera is a rectangular parallelepiped (Orthographic Projection)

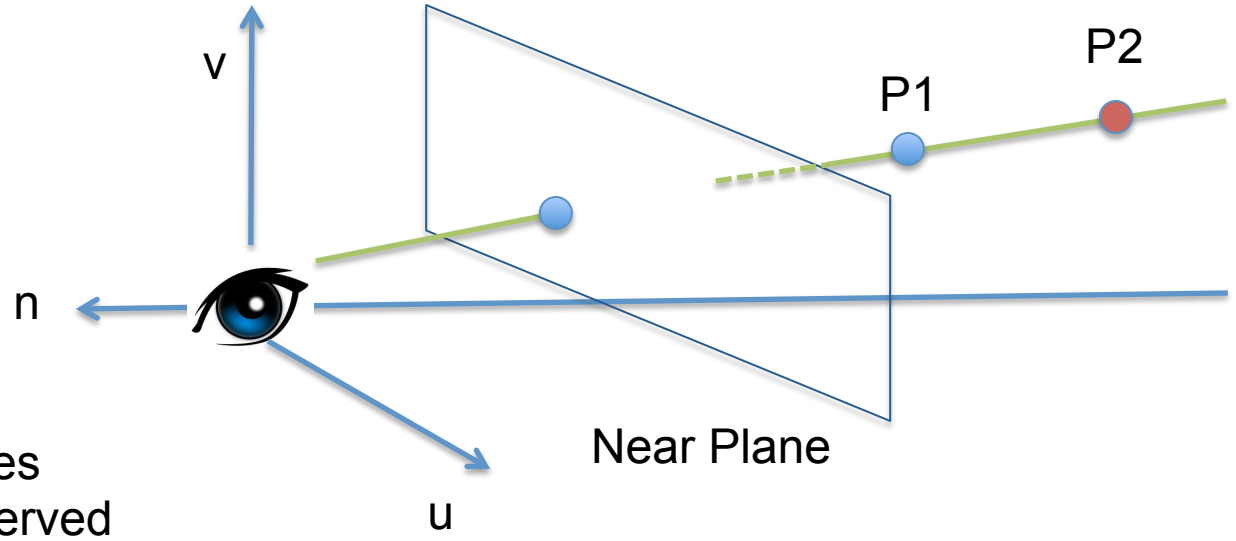


The x and y coordinates of each point are preserved using the orthographic projection



Orthographic Project Matrix P (2)

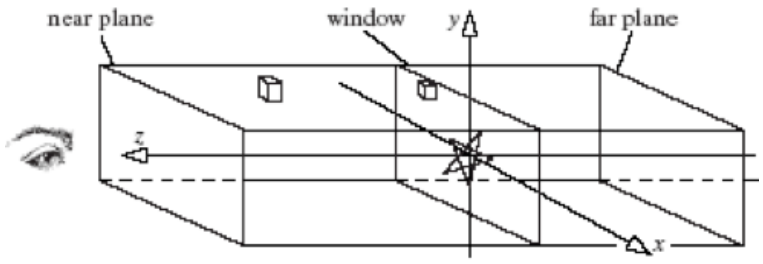
- We need to add depth information
- Depth information tells which surfaces are in front of other surfaces, for hidden surface removal.



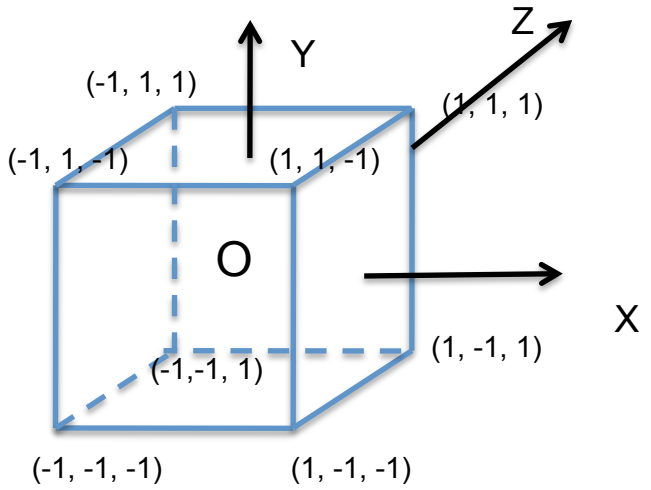
The x and y coordinates of each point are preserved using the orthographic projection

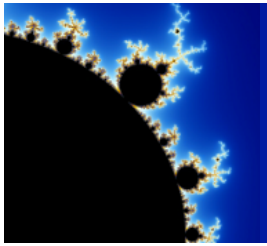
Orthographic Project Matrix P (3)

- Instead of Euclidean distance, a pseudo-depth, $-1 \le P_z' \le 1$ for $-N > P_z > -F$ is used. This quantity is faster to compute than the Euclidean distance.
- The view volume is transformed into a cube of size 2, centered at origin.



Orthographic Projection Matrix P





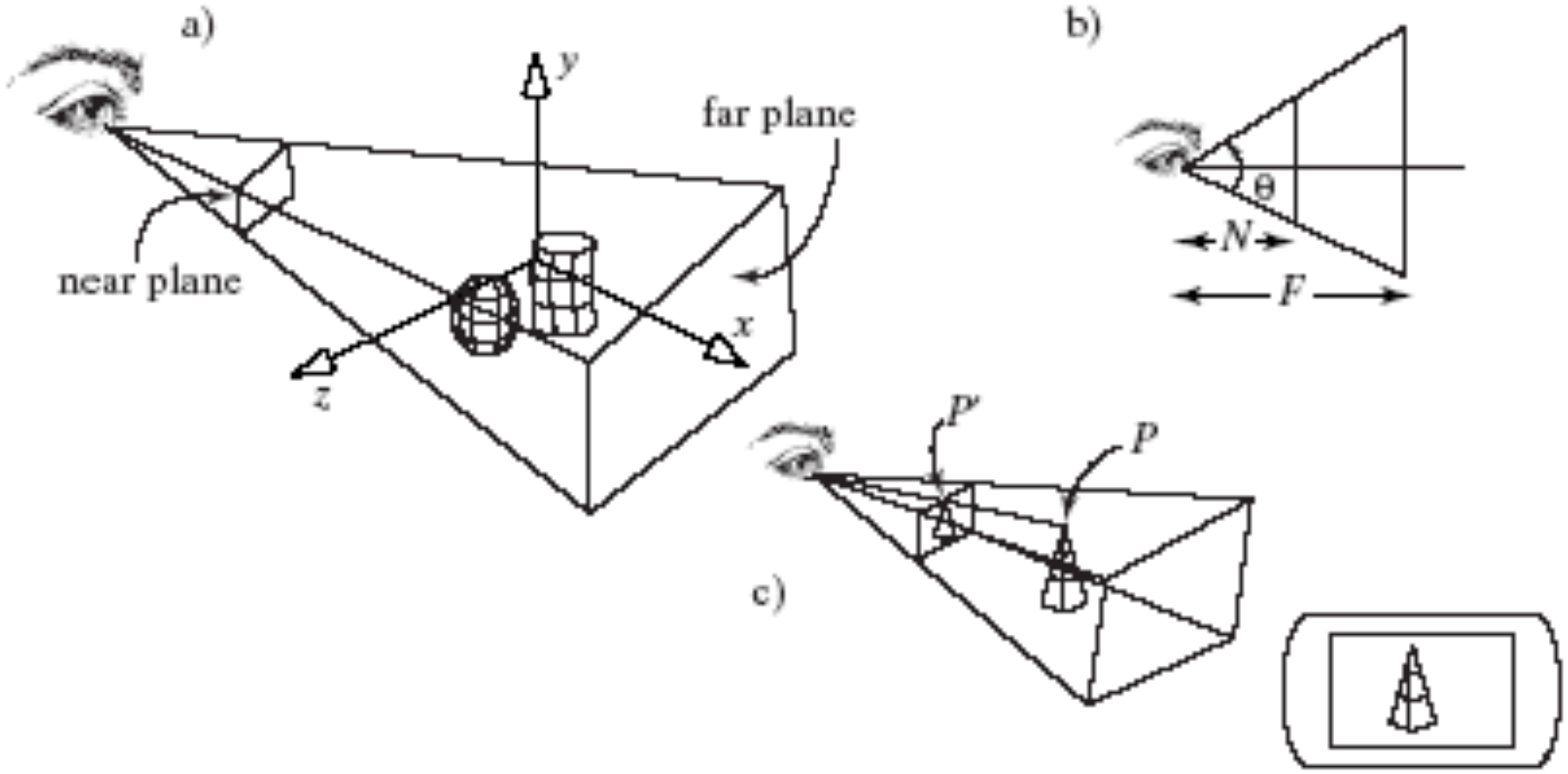
Orthographic Project Matrix P (4)

- Translate the view volume to center at the origin
- Scale the view volume to x:[-1, 1], y:[-1, 1], z: [-1:1]

$$P = ST = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{-2}{far-near} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\frac{left+right}{2} \\ 0 & 1 & 0 & -\frac{top+bottom}{2} \\ 0 & 0 & 1 & -\frac{far+near}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

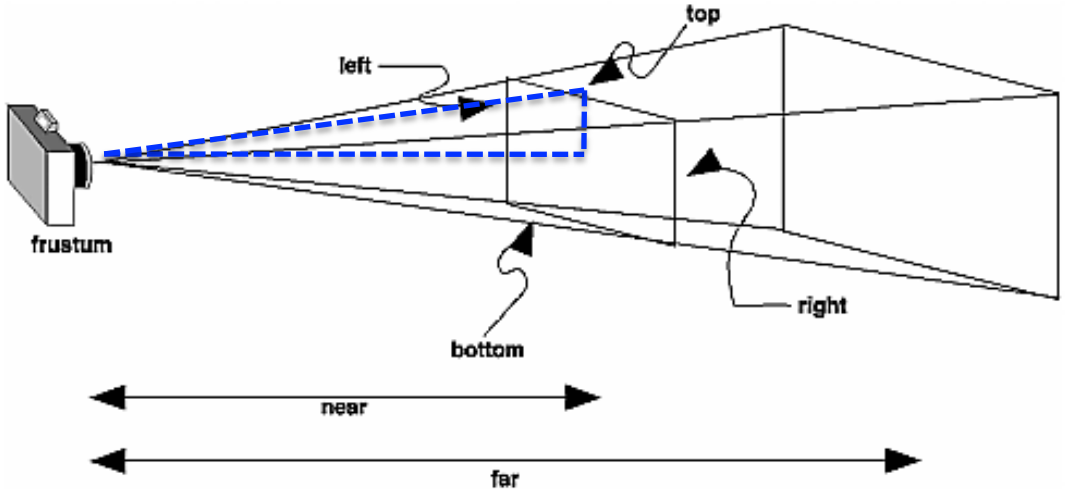
Perspective Projection



Perspective projection in webGL

frustum(left, right, bottom, top, N, F); or

perspective(viewAngle, aspect, N, F);



frustum➔perspective:

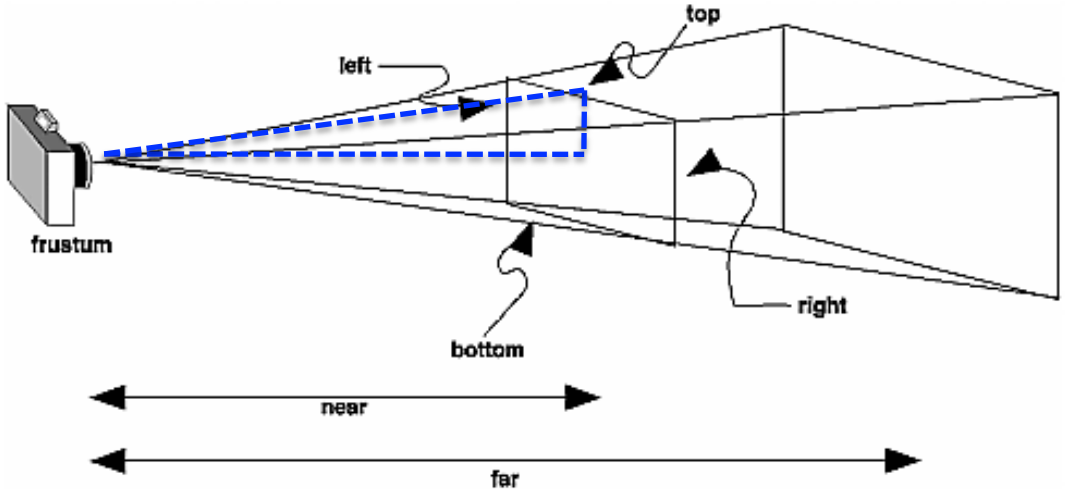
aspect = (right-left)/(top-bottom)

viewAngle=2* arctan(1/2*(top-bottom)/N)

Perspective projection

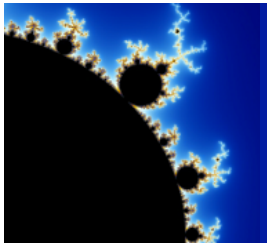
frustum(left, right, bottom, top, N, F); or

perspective(viewAngle, aspect, N, F);



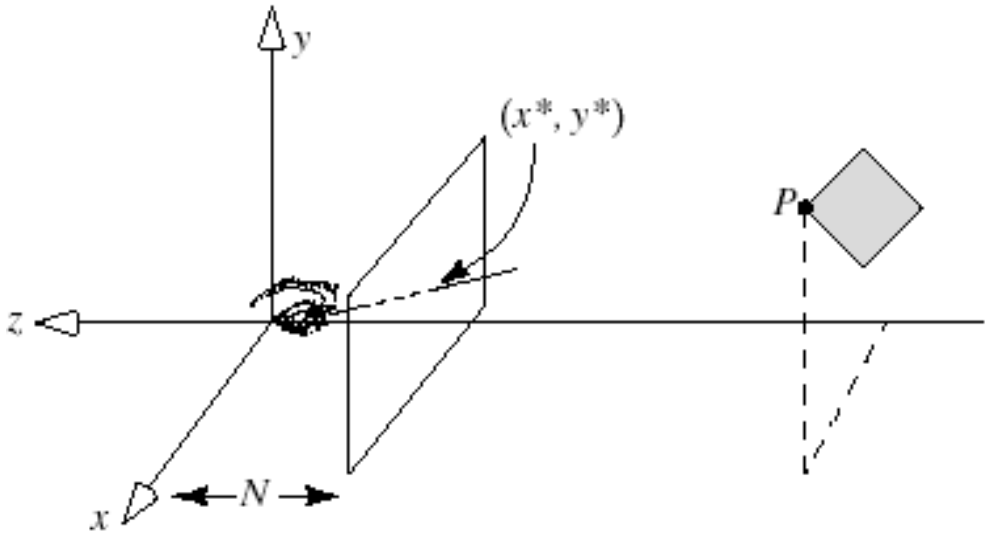
perspective ➔frustum:

top = N*tan(1/2* viewAngle*PI/180*)
bottom = -top
right = top*aspect
left = -right



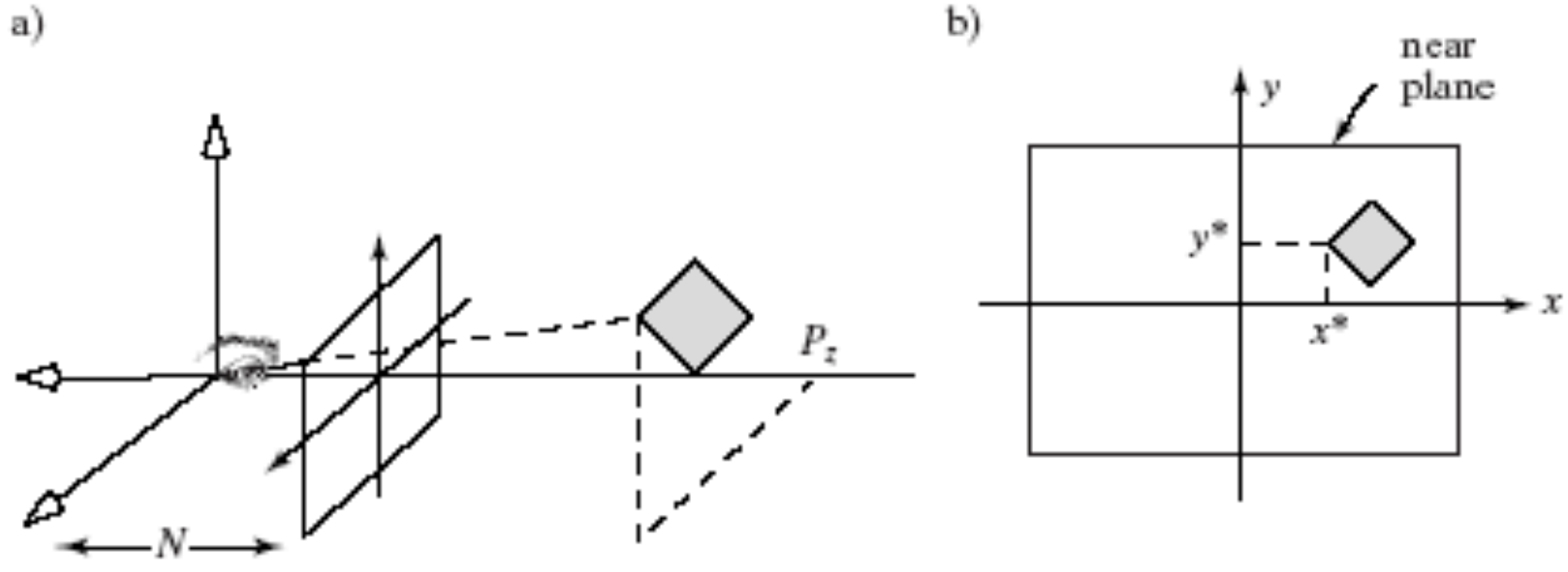
Perspective Projections of 3-D Objects

- A vertex located at *P* in eye coordinates is passed through the next stages of the pipeline where it is projected to a certain point (*x**, *y**) on the near plane, clipping is carried out, and finally the surviving vertices are mapped to the viewport on the display.

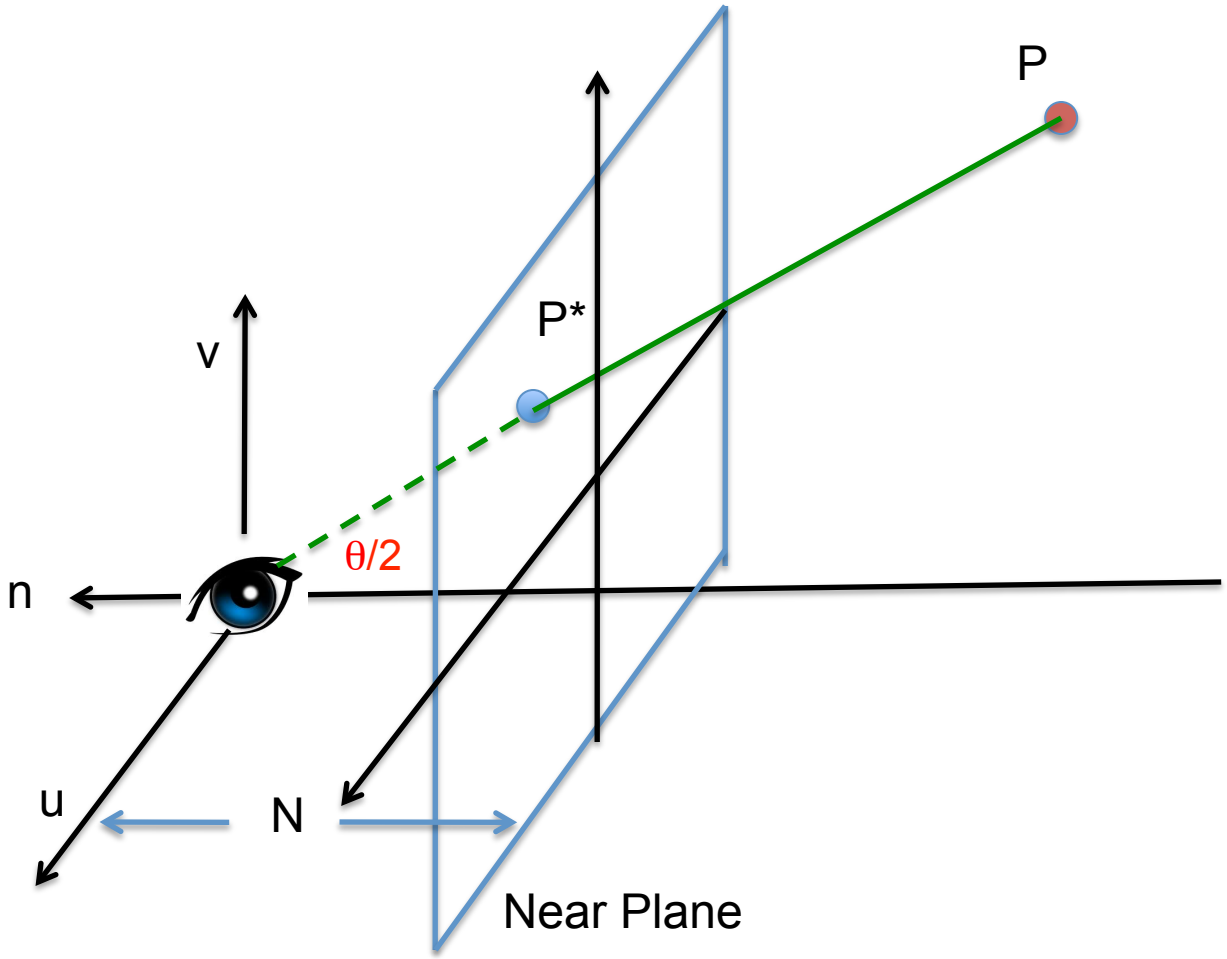


Perspective Projections of 3-D Objects (4)

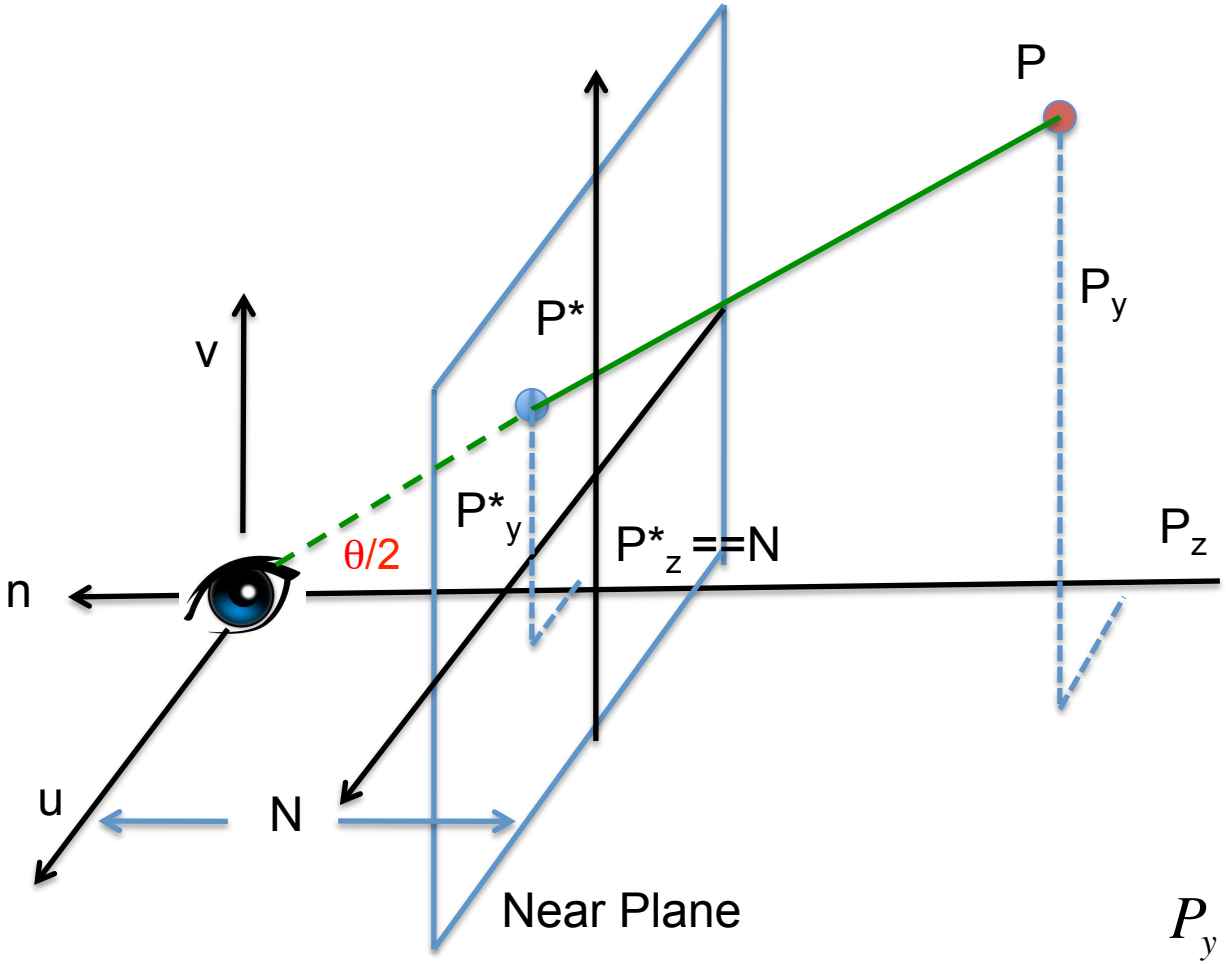
- We erect a local coordinate system on the near plane, with its origin on the camera's *z*-axis. Then it makes sense to talk about the point x* units right of the origin, and y* units above the origin.
- (P_x, P_y, P_z) projects to (P^*_x, P^*_y).



Perspective projection

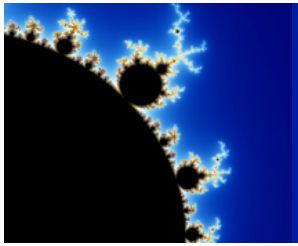


Perspective projection in OpenGL



By similar triangles

$$\frac{P_y^*}{P_y} = \frac{N}{-P_z} \Rightarrow\Rightarrow P_y^* = \frac{NP_y}{-P_z}$$



Perspective Projection of 3-D Objects (5)

- Same goes for P_x, we get:

$$\frac{P^*_x}{P_x} = \frac{N}{-P_z} \Rightarrow\Rightarrow P^*_x = \frac{NP_x}{-P_z}$$

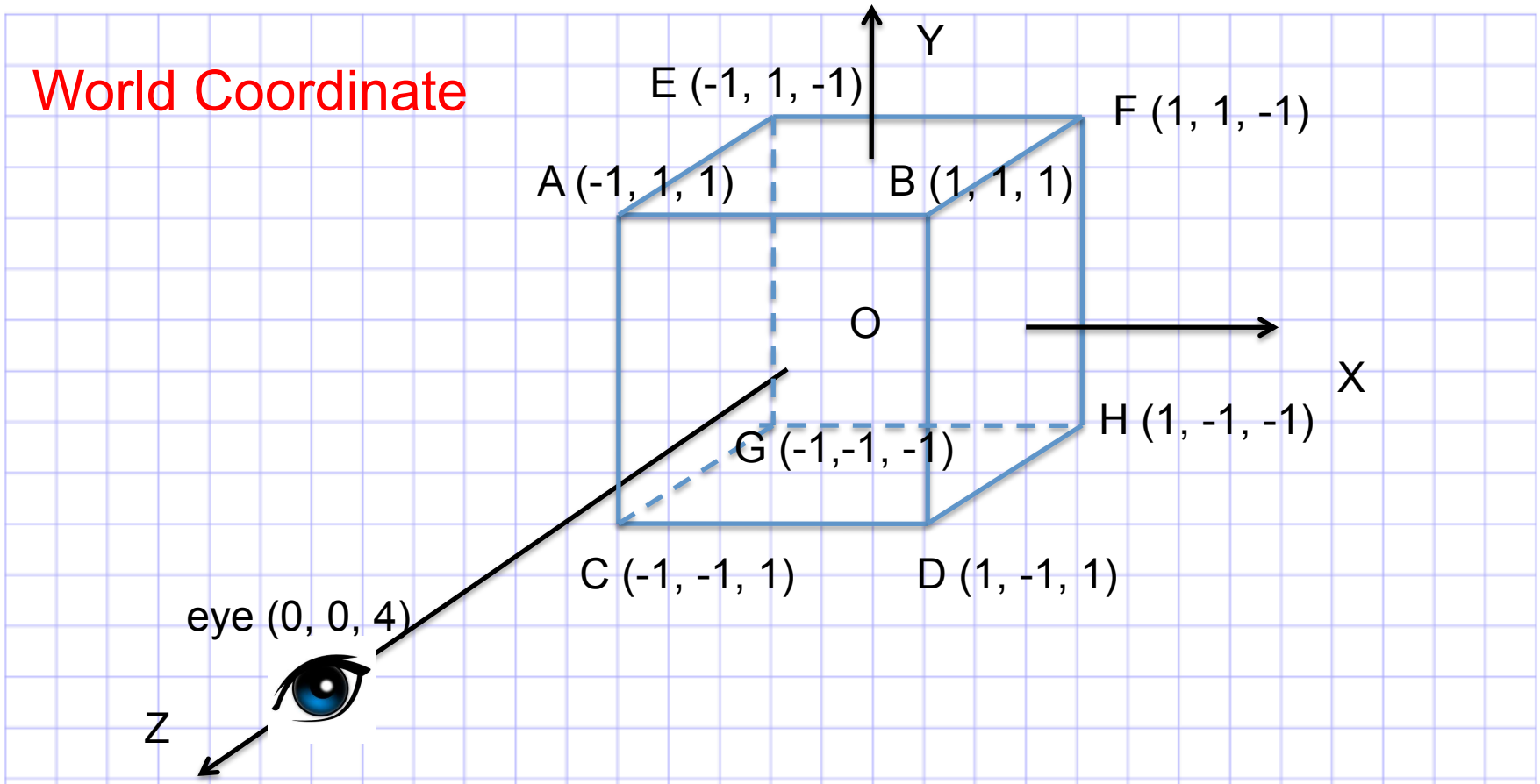
$$\frac{P_y^*}{P_y} = \frac{N}{-P_z} \Rightarrow\Rightarrow P_y^* = \frac{NP_y}{-P_z}$$

The further away the point, the larger $-P_z$ value, The smaller the project P_x and P_y values

- Thus:

$$P^* = \begin{pmatrix} x^* \\ y^* \end{pmatrix} = \begin{pmatrix} \frac{NP_x}{-P_z} \\ \frac{NP_y}{-P_z} \end{pmatrix}$$

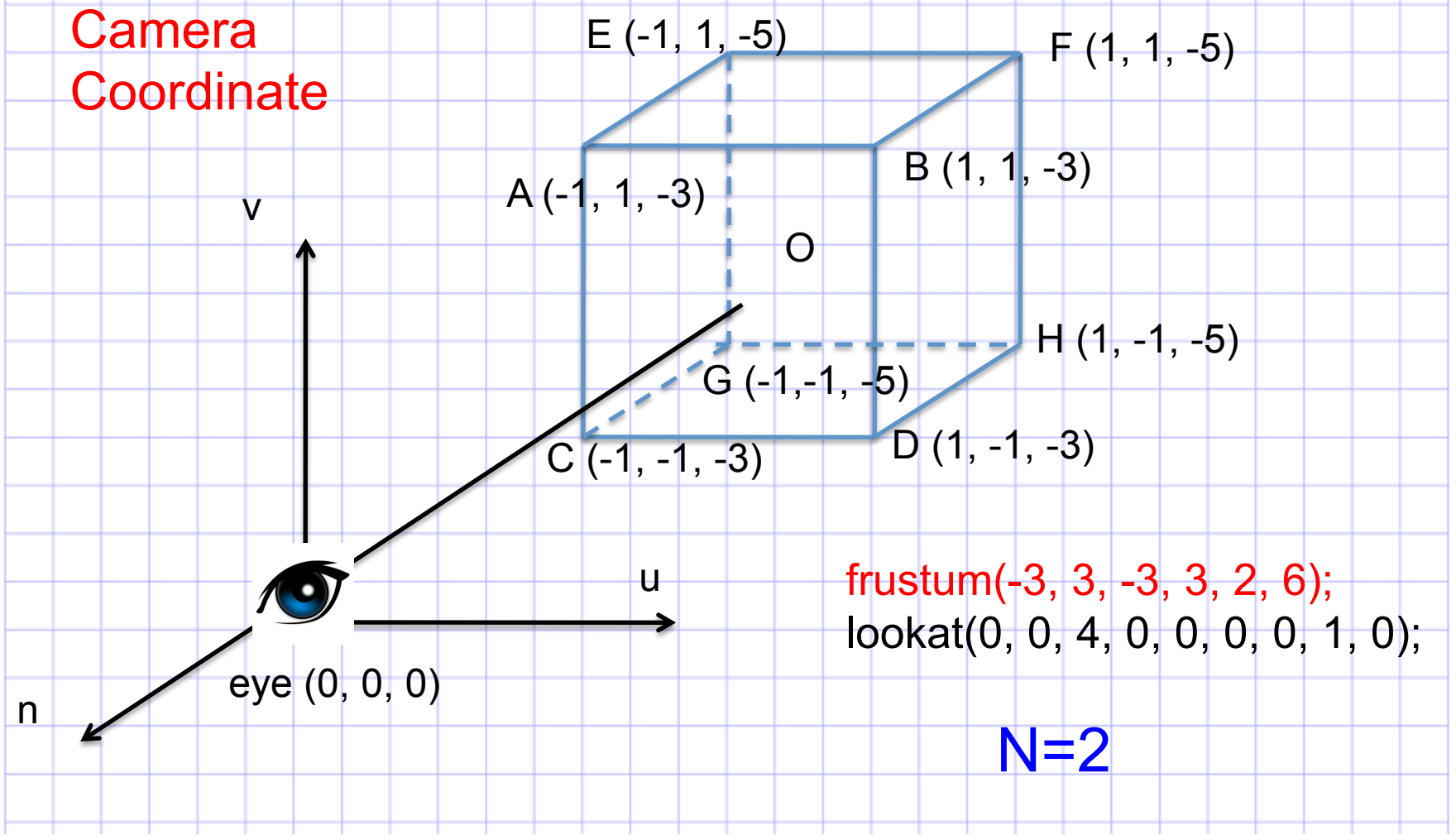
What is the projected image of the cube on the viewplane?

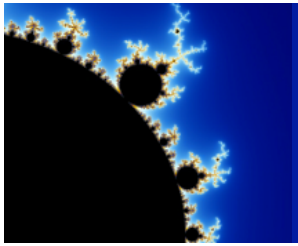


frustum(-3, 3, -3, 3, 2, 6);

lookat(0, 0, 4, 0, 0, 0, 0, 1, 0);

Practice Question





Perspective Projection Properties

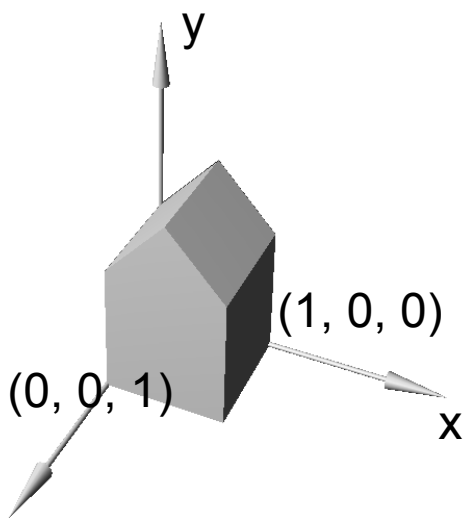
- $|P_z|$ is larger for points further away from the eye, and, because we divide by it, causes objects further away to appear smaller (perspective foreshortening).
- We do not want $P_z \geq 0$; generally these points (at or behind eye) are clipped.
- Projection to a plane other than N simply scales P*; since the viewport matrix will scale anyway, we might as well project to N.

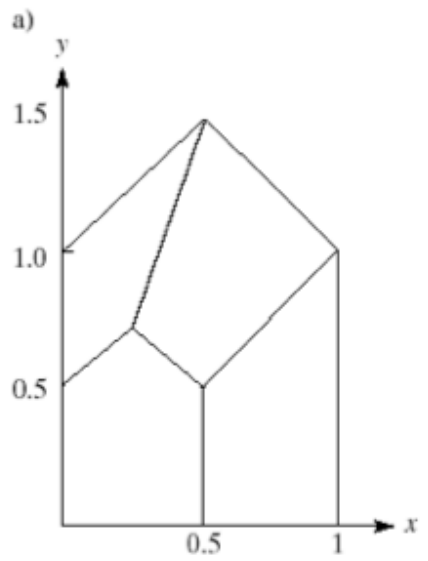
Barn Example

- **View #1**: perspective(30, 1.5, 1, 2);
  lookat(0, 0, 2, 0, 0, 0, 0, 1, 0);

The near plane coincides w/ the front of the barn.

- In camera coordinates all points on the front wall of the barn have P_z = -1 and those on the back wall have P_z = -2. So any point (P_x, P_y, P_z) on the front wall projects to P' = (P_x, P_y) and any point on the back wall projects to P' = (P_x /2, P_y / 2).
- The foreshortening factor is two for points on the back wall. Note that edges on the rear wall project at half their true length. Also note that edges of the barn that are actually parallel in 3D *need not* project as parallel.



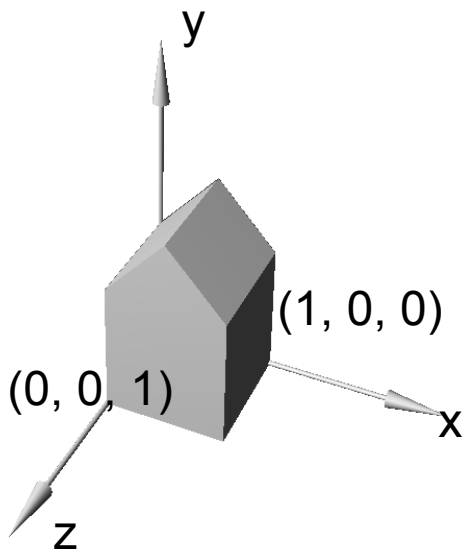


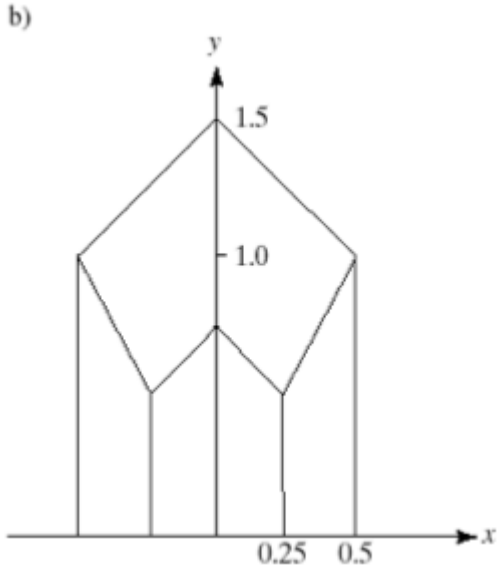
Barn Example (2)

- In view #2, the camera has been moved right, but everything else is the same:

  perspective(30, 1.5, 1, 2);

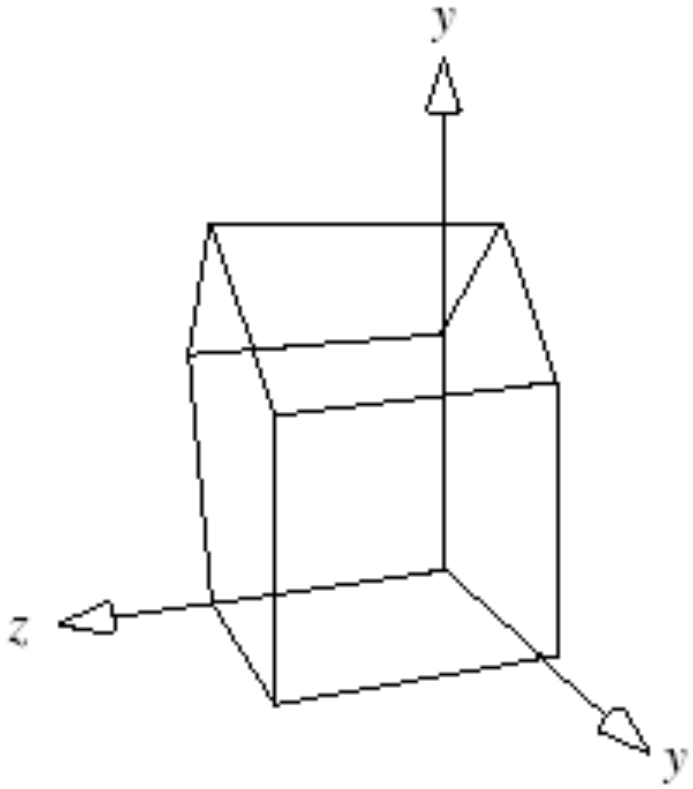
  lookAt(0.5, 0, 2, 0, 0, 0, 0, 1, 0);

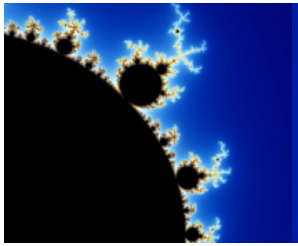




Example (3)

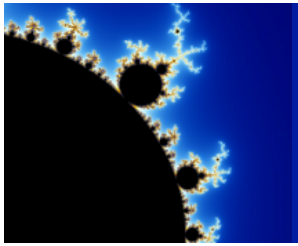
- In part c, we look down from above and right on the barn.





Perspective Projection of Lines

- Straight lines are transformed to straight lines.
- Lines that are parallel in 3D project to lines, but not necessarily parallel lines. If not parallel, they meet at some vanishing point.
- If $P_z \geq 0$, lines that pass through the camera undergo a catastrophic "passage through infinity"; such lines must be clipped.
- Perspective projections usually produce geometrically realistic pictures. But realism is strained for very long lines parallel to the viewplane.



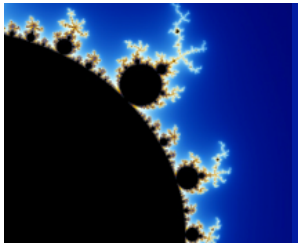
Projection of Straight Lines (2)

- For any line in 3D: P = A + **c**t, its projected line on viewplane:

$$P_x'(t) = \frac{N * P_x(t)}{-P_z(t)} = -\frac{N * (A_x + c_x * t)}{A_z + c_z * t}$$

$$P_y'(t) = \frac{N * P_y(t)}{-P_z(t)} = -\frac{N * (A_y + c_y * t)}{A_z + c_z * t}$$

The projected Point A is P’(0) = - $N/A_z * (A_x, A_y)$.



Projection of Straight Lines (3)

- Effect of projection → on parallel lines:
- If the line is parallel to the near plane N, $c_z = 0$:
- Thus:

$$P_x'(t) = \frac{N * P_x(t)}{-P_z(t)} = -\frac{N * (A_x + c_x * t)}{A_z + c_z * t} = -\frac{N}{A_z}(A_x + c_x * t)$$

$$P_y'(t) = \frac{N * P_y(t)}{-P_z(t)} = -\frac{N * (A_y + c_y * t)}{A_z + c_z * t} = -\frac{N}{A_z}(A_y + c_y * t)$$

- This is a line with slope c_y/c_x and all lines with direction **c**

  → a line with this slope.
- **Thus if two lines in 3D are parallel to each other *and* to the viewplane, they project to two parallel lines.**

Projection of Straight Lines (3)

- If the line is not parallel to plane N (near plane), look at limit as t becomes ∞ for

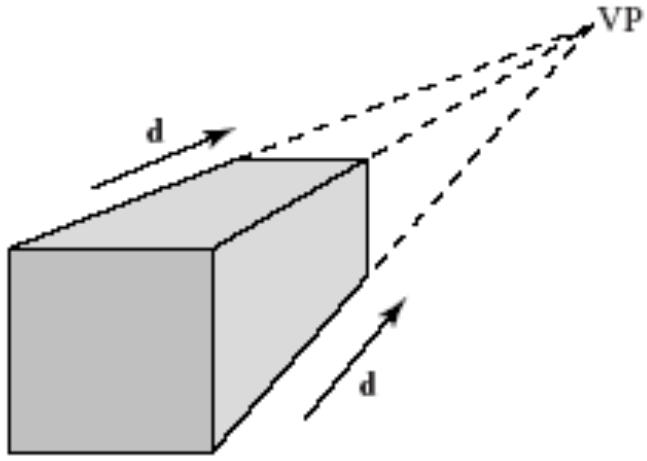
$$P_x'(t) = \frac{N * P_x(t)}{-P_z(t)} = -\frac{N * (A_x + c_x * t)}{A_z + c_z * t} = -\frac{N * c_x}{c_z}$$

$$P_y'(t) = \frac{N * P_y(t)}{-P_z(t)} = -\frac{N * (A_y + c_y * t)}{A_z + c_z * t} = -\frac{N * c_y}{c_z}$$

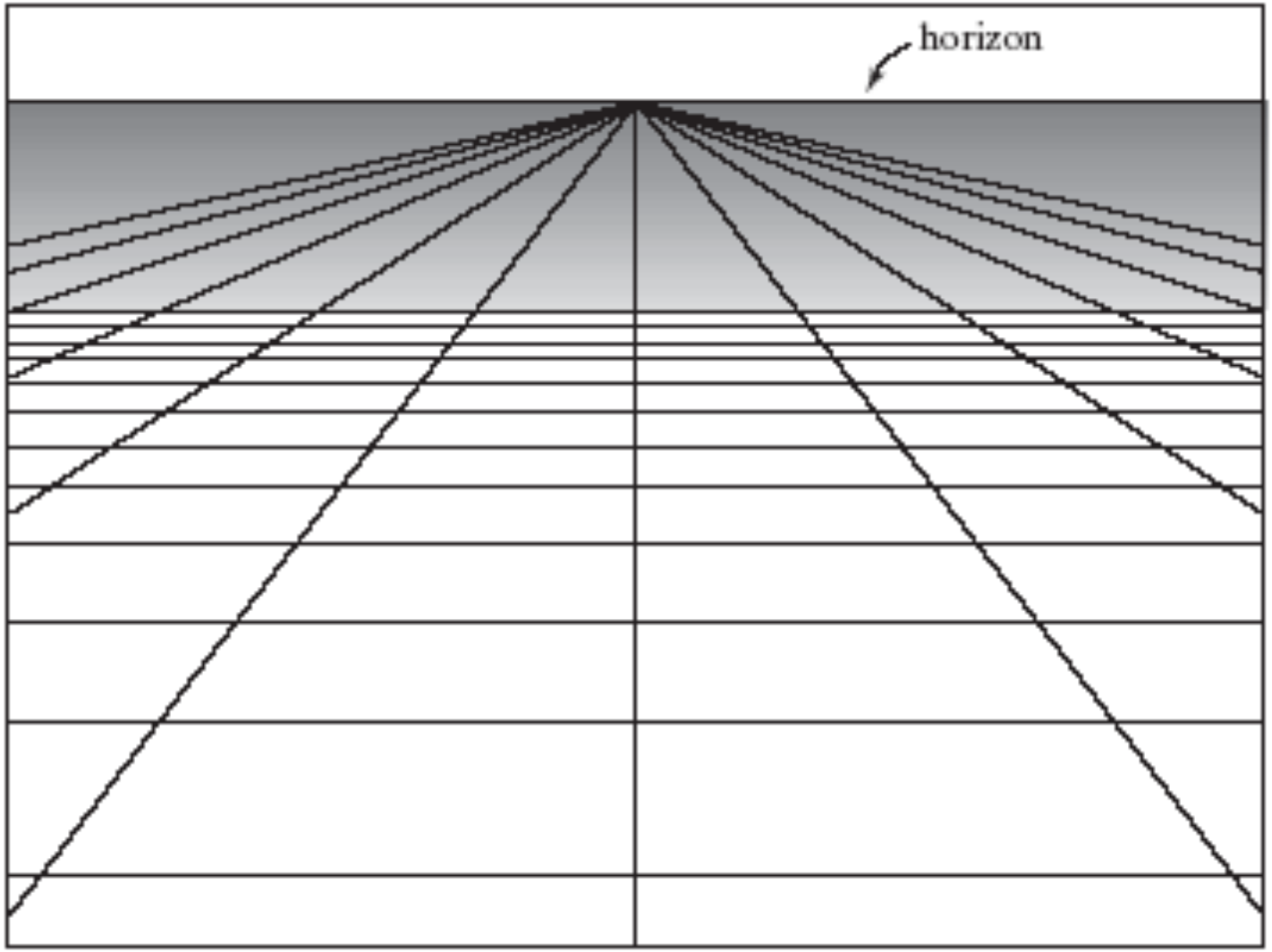
➔ $p(t)=-N/c_z\ (c_x, c_y)$, a constant.

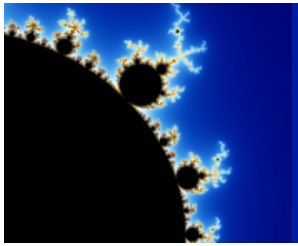
  – All lines with direction **c** reach this point as t becomes ∞; it is called the vanishing point.

- Thus all parallel lines share the same vanishing point.
- In particular, these lines project to lines that are *not* parallel.



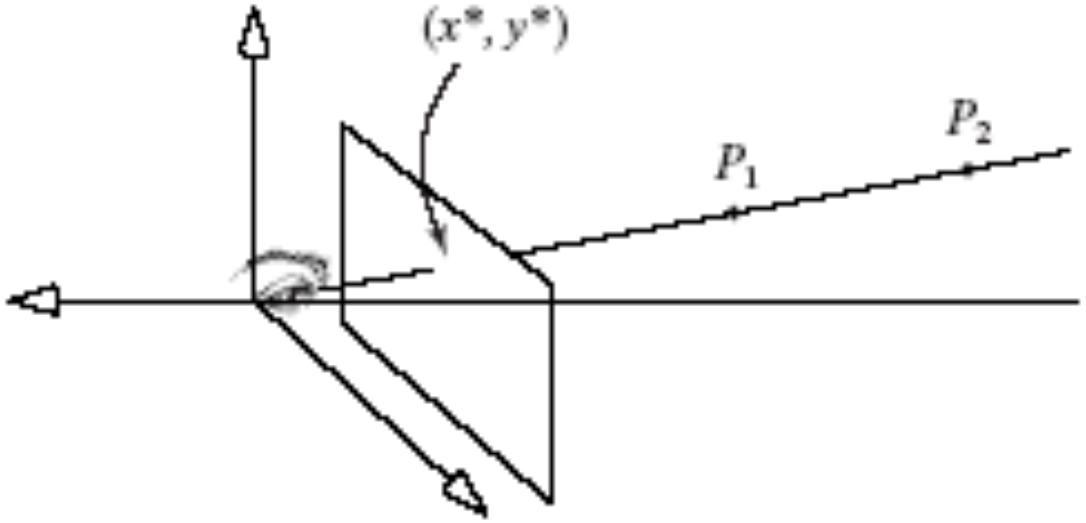
Example: horizontal grid in perspective

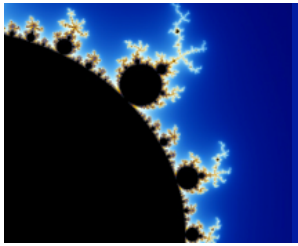




Incorporating Perspective in the Graphics Pipeline

- We need to add depth information (destroyed by projection).
- Depth information tells which surfaces are in front of other surfaces, for hidden surface removal.





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 $P_z = -F$.

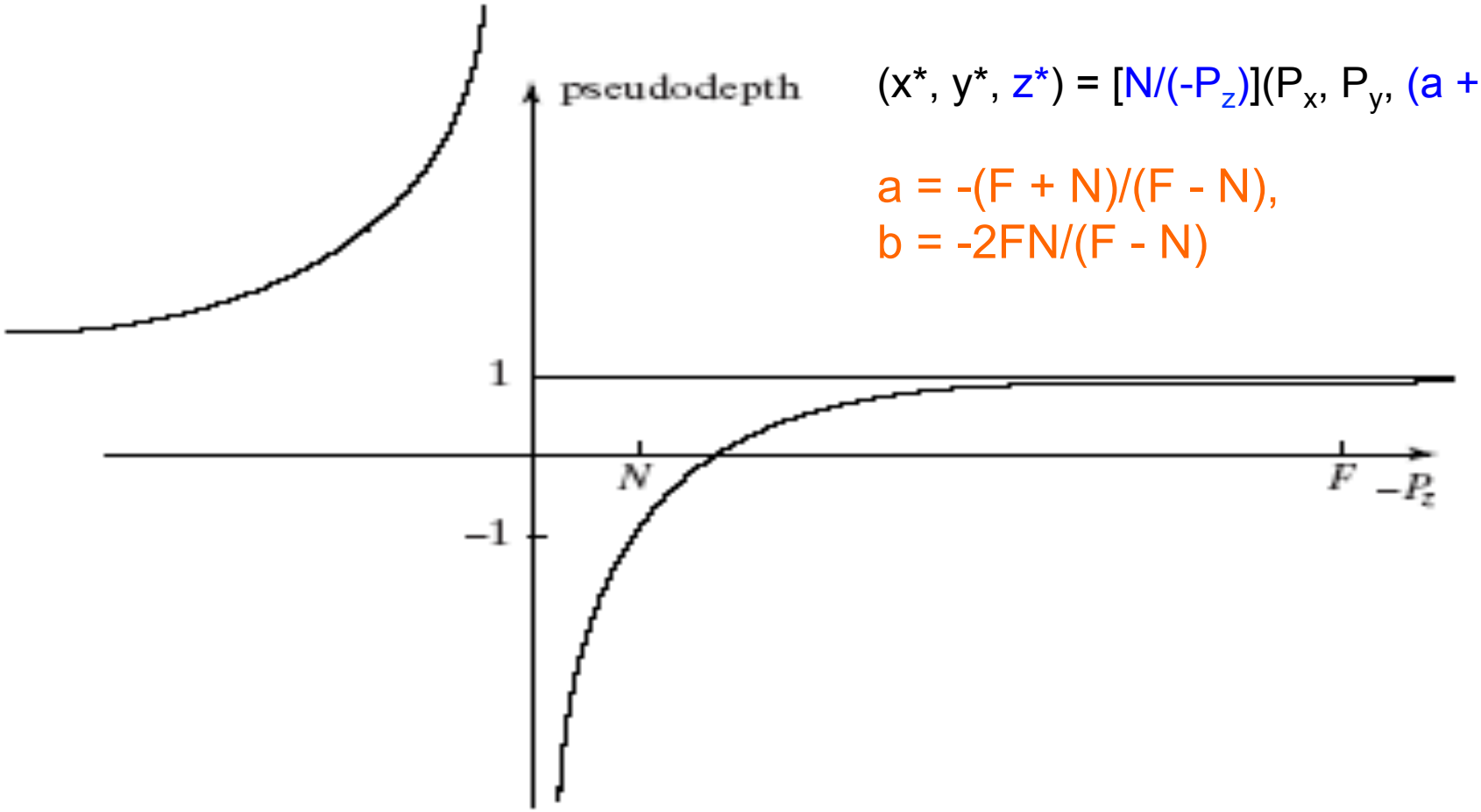
- Result:

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

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

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

Illustration of Pseudo-depth Values



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

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