Lecture #8

Normalization Transformation

Computer Graphics

Winter Term 2016/17

Marc Stamminger / Roberto Grosso

Last Lecture

- Matrix to set camera position and view direction:

  → **Viewing Transformation**

  rigid transformation, i.e. affine

- Matrix to generate parallel perspective

  → Orthogonal Projection

  simple affine transformation

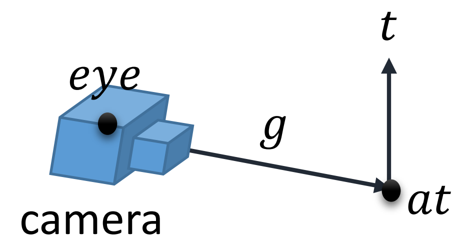
- Matrix to generate real perspective

  → Perspective Projection

  projective transformation → interesting new properties

Viewing Transformation

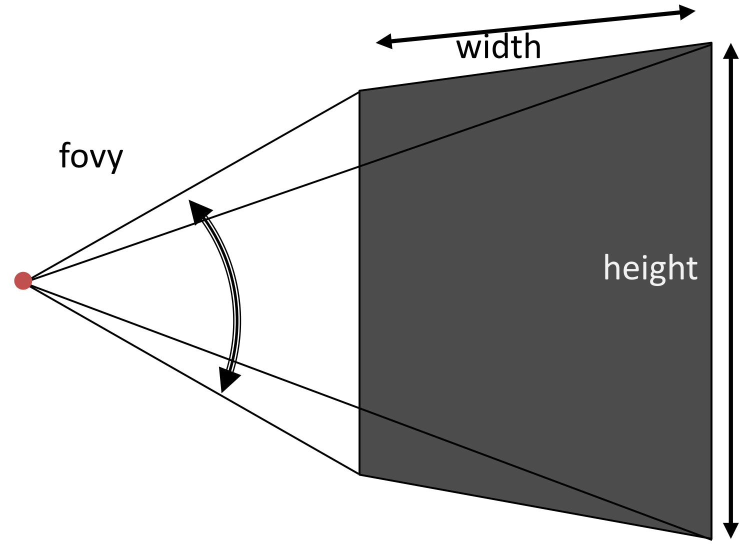
- Defined using vectors
  - eye (camera position),
  - g (gaze or viewing direction)
  - t (up vector)



- Signature in most libraries:
  `lookAt(eye_x,eye_y,eye_z,at_x,at_y,at_z,up_x,up_y,up_z);`
  - at is "look at" point
  - this point gets centered in the final image
  - $g = at - eye$

Perspective Transformation

- Defined by
  - $near$ and far plane
  - $aspect$ ratio (width over height)
  - field of view $fovy$: opening angle
    → large $fovy$ = wide angle lens, e.g. 75°
    → small $fovy$ = tele lens, e.g. 30°

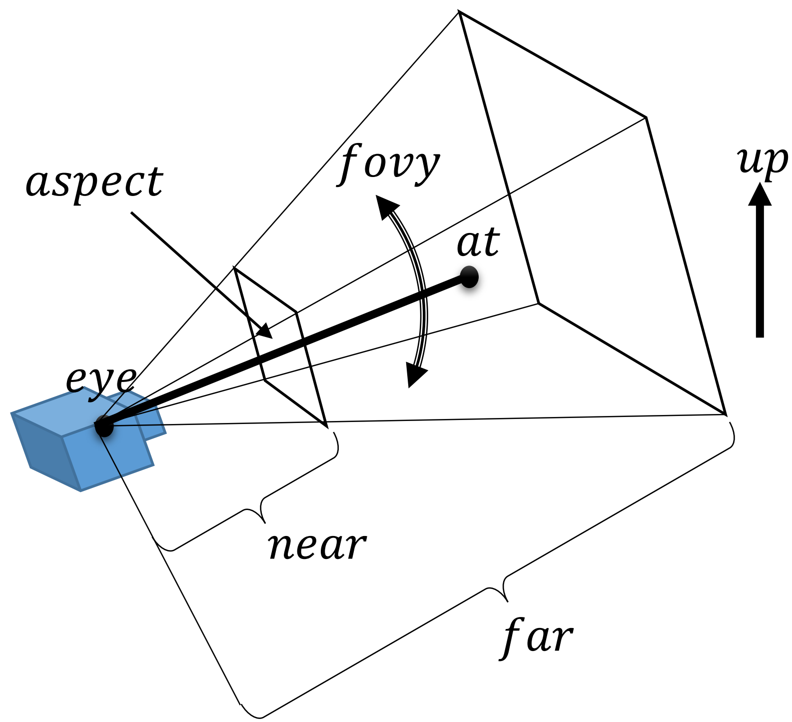


- Signature in most libraries:
  `perspective(fovy,aspect,near,far);`

All together

- eye, at, up:
  viewing paramters
  = **extrinsic parameters**

- $fovy, aspect, near, far$:
  perspective parameters
  = intrinsic parameters



In OpenGL / WebGL

- In old OpenGL versions, matrices where handled by OpenGL:
  - there is one matrix PROJECTION
    →orthogonal projection matrix set by:
    `glOrtho(left,right,bottom,top,near,far);`
    →perspective matrix set by
    `glFrustum(left,right,bottom,top,near,far);`
    → or by
    `gluPerspective(fovy,aspect,near,far);`

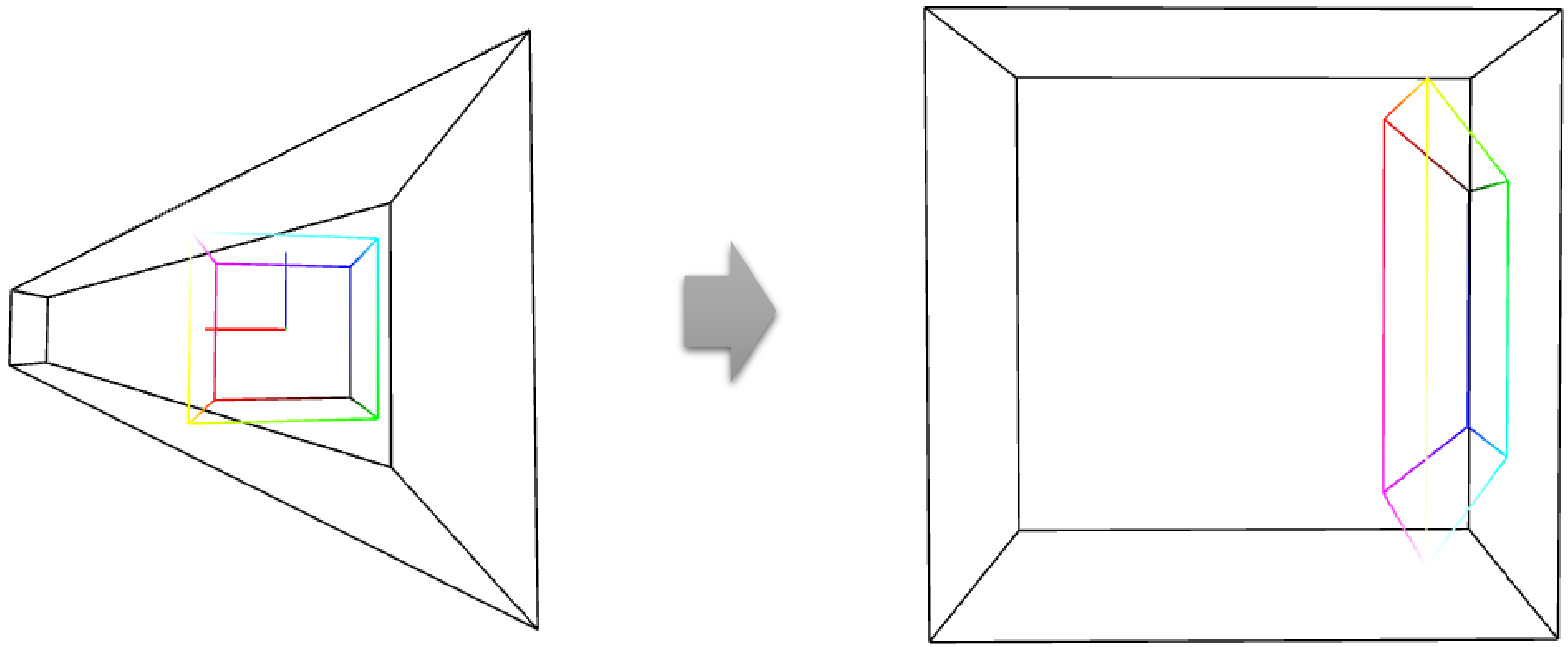
  - Viewing matrix and model matrix are stored as one MODELVIEW matrix
    → first, viewing is set using
    `gluLookAt(eyex,eyey,eyez,atx,aty,atz,upx,upy,upz);`
    where the view direction is set using a lookat point: $g = at - eye$
    → then modeling transformations can be appended, e.g. using
    `glTranslate(…), glRotate(…), glMultMatrix(…)`
- To every vertex, first the MODELVIEW and then the PROJECTION matrix is applied before rasterization

In OpenGL / WebGL

- New OpenGL and WebGL have to do all this in the vertex shader
- So the matrix stuff must happen by the application
- In javascript: libraries, e.g. **`gl-Matrix.js`**
- and then upload the matrices as uniforms

Normalizing Transformation

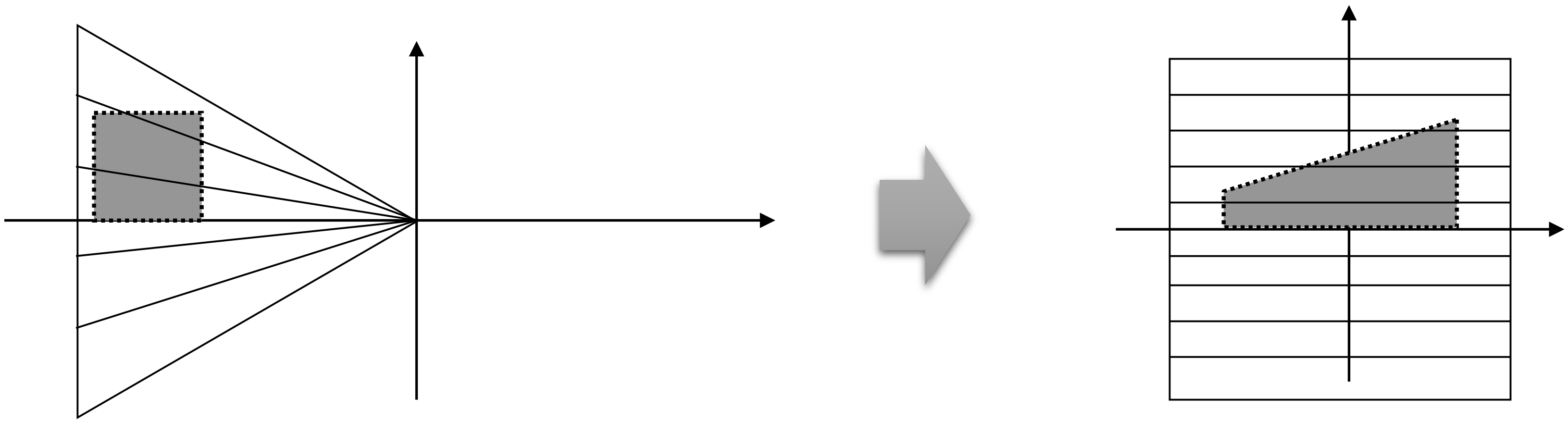
- The perspective matrix transforms the view frustum to the unit cube
- Let’s play 7:



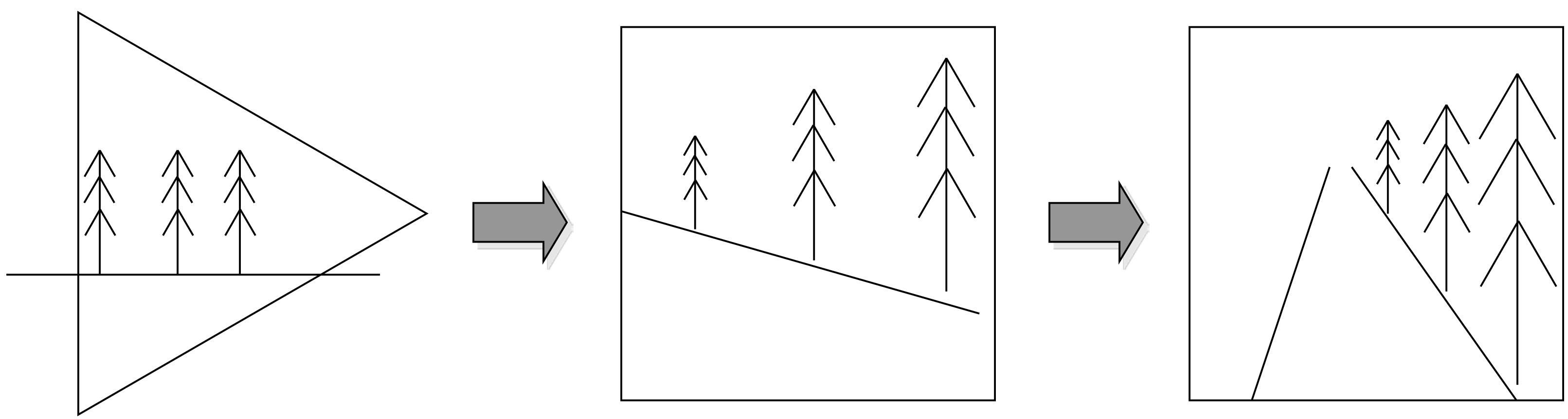
- We also call this the **Normalizing Transformation**
- It belongs to the class of **Projective Transformations**

Perspective Transformation

- The perspective matrix transform the view frustum to the unit cube

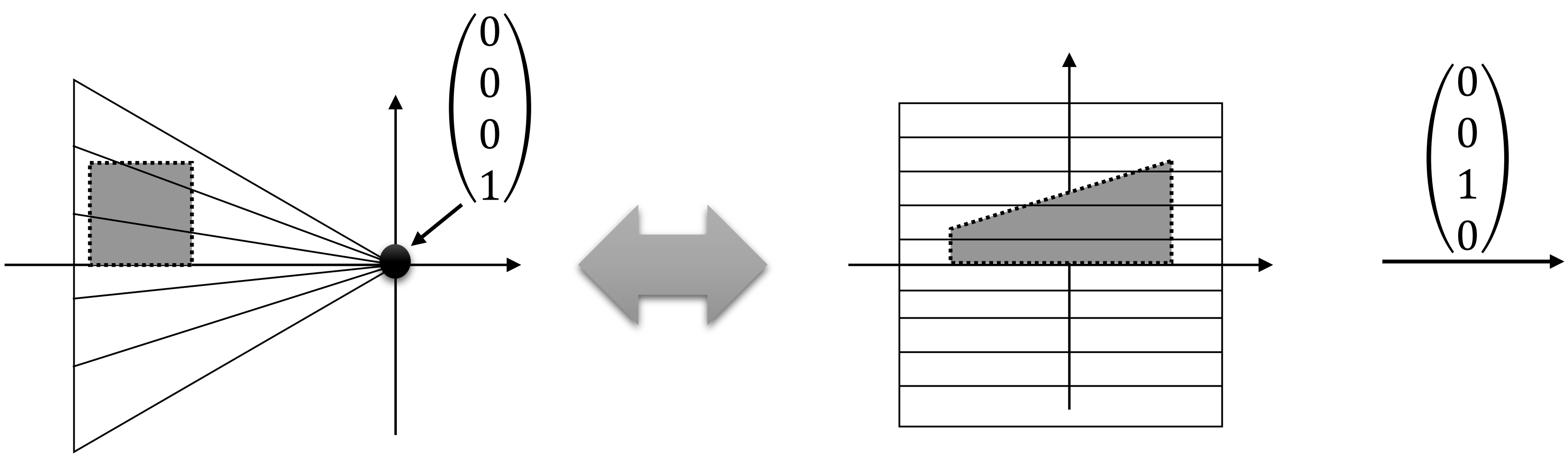


- Regions close to observer are enlarged, distant regions are shrunk
  $\Rightarrow$ perspective distortion



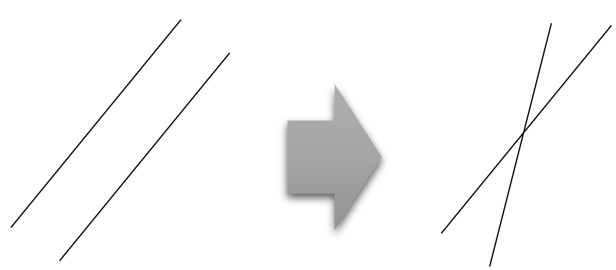
Projective Transformations

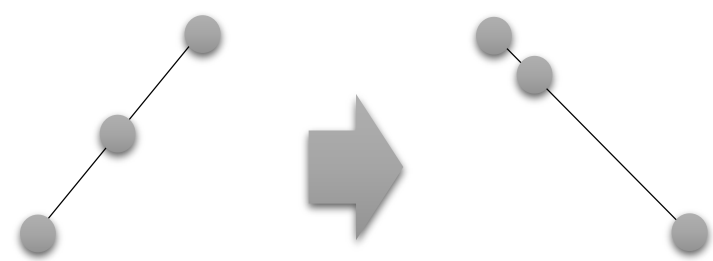
- homogenous coordinates allows us to represent points at infinity: $\lim_{w \to 0}(\mathrm{x}, \mathrm{y}, \mathrm{z}, \mathrm{w}) =$ point at infinity in direction (x, y, z)
  → points at infinity = directions = $(x, y, z, 0)$
- A projective matrix can map such infinity points $(x, y, z, 0)$ to finite points (x, y, z, w), $w \neq 0$ and vice versa !
- Intersection of parallel lines = point at infinity = direction of these lines gets mapped to finite point and vice versa



Projective Transformations

- Properties:
    - lines remain lines
    - parallel lines don’t remain parallel
    - ratios are not preserved





Projective Transformations

- Direction = Vector ?
  - a direction is the vector between two points:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{pmatrix} - \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 0 \end{pmatrix}$$

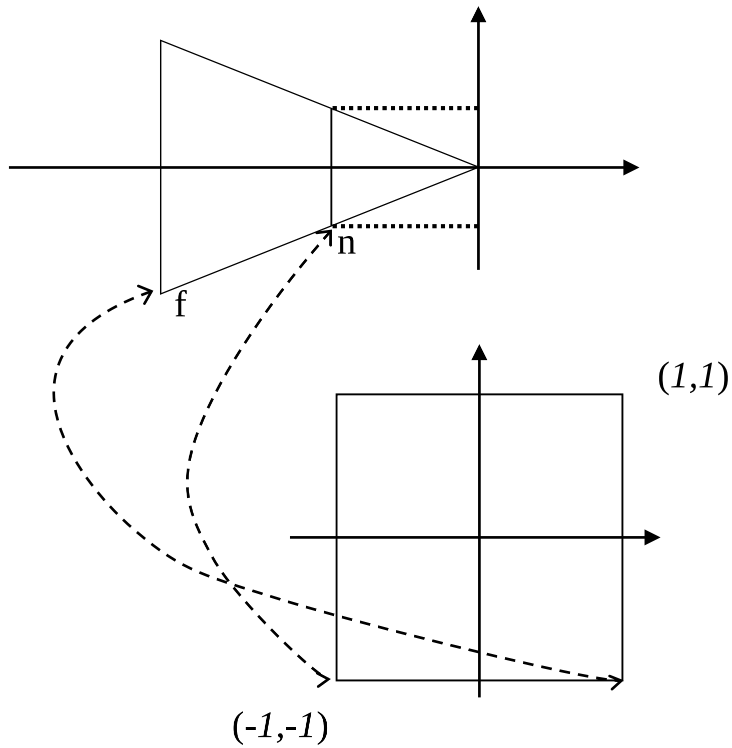
  - Points: w=1, Vectors: w=0
  - also look at matrix-point and matrix-vector multiplication !

Normalizing Transformation

- How to choose near and far planes: Nonlinear mapping of z:

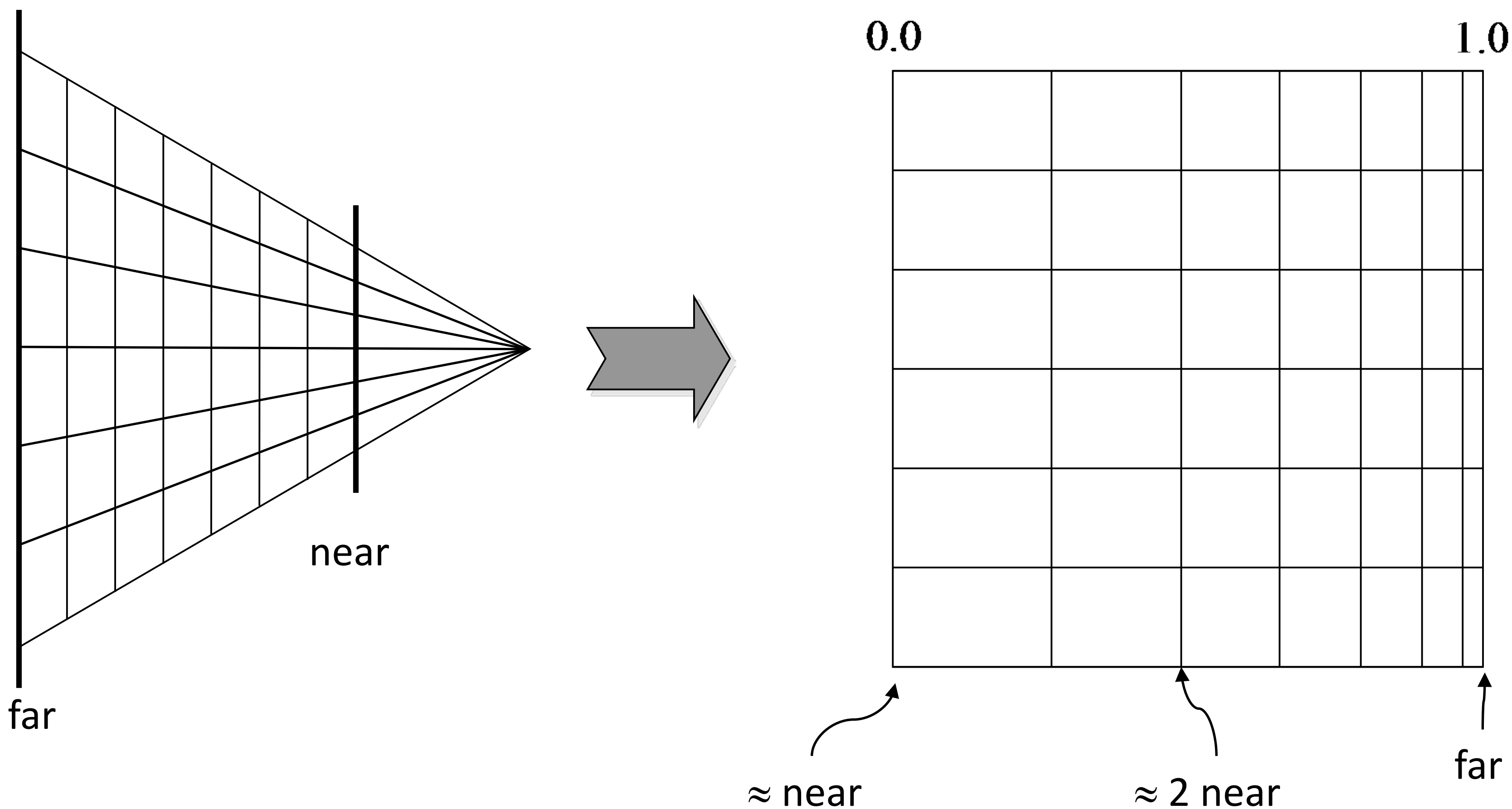
$$z \rightarrow \frac{n+f}{f-n} - \frac{2nf}{(f-n)z}$$

  - z-buffer with low resolution
  - Objects at far distance collapse
  - Drawing will fail sorting far objects due to insufficient resolution, z-fighting.



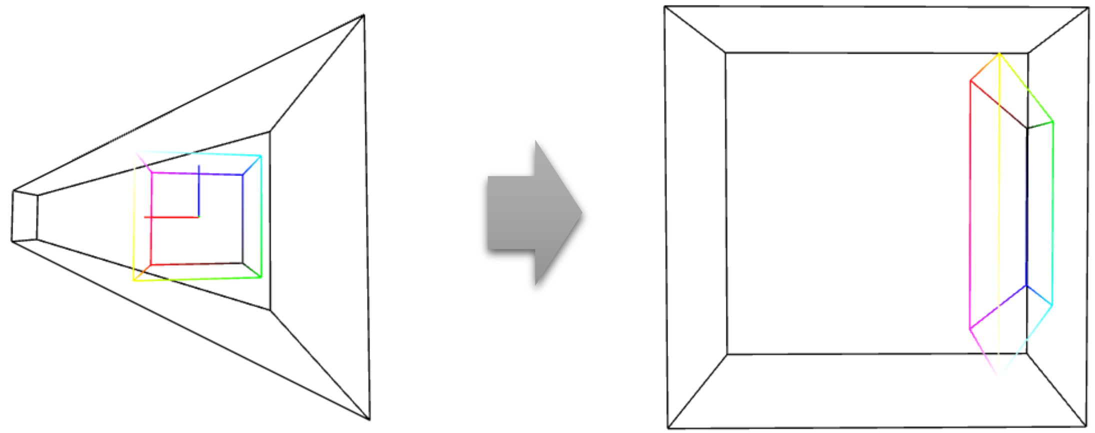
Normalizing Transformation

- Objects at far distance collapse

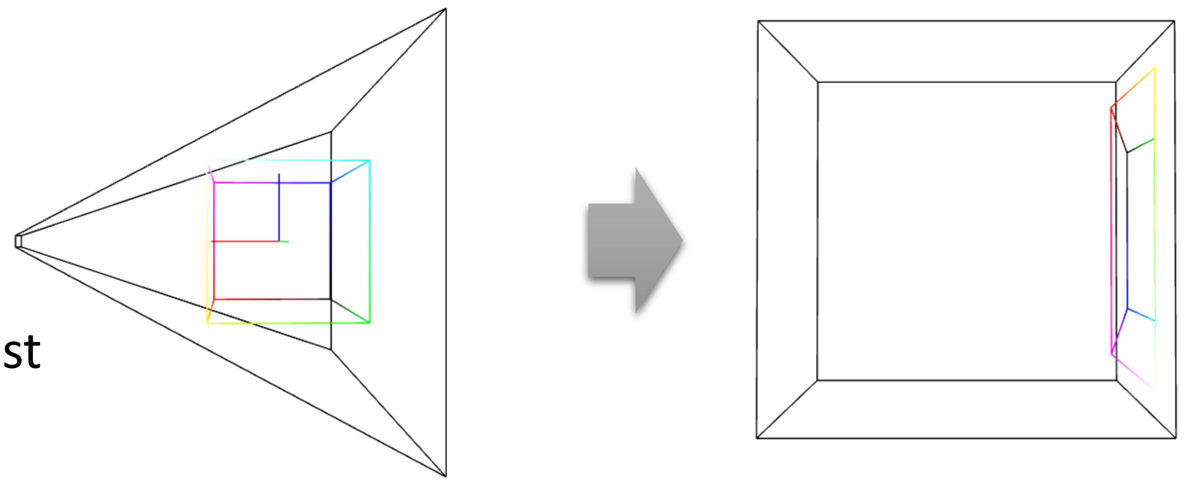


Normalizing Transformation

- choose reasonable near!
- resonable near

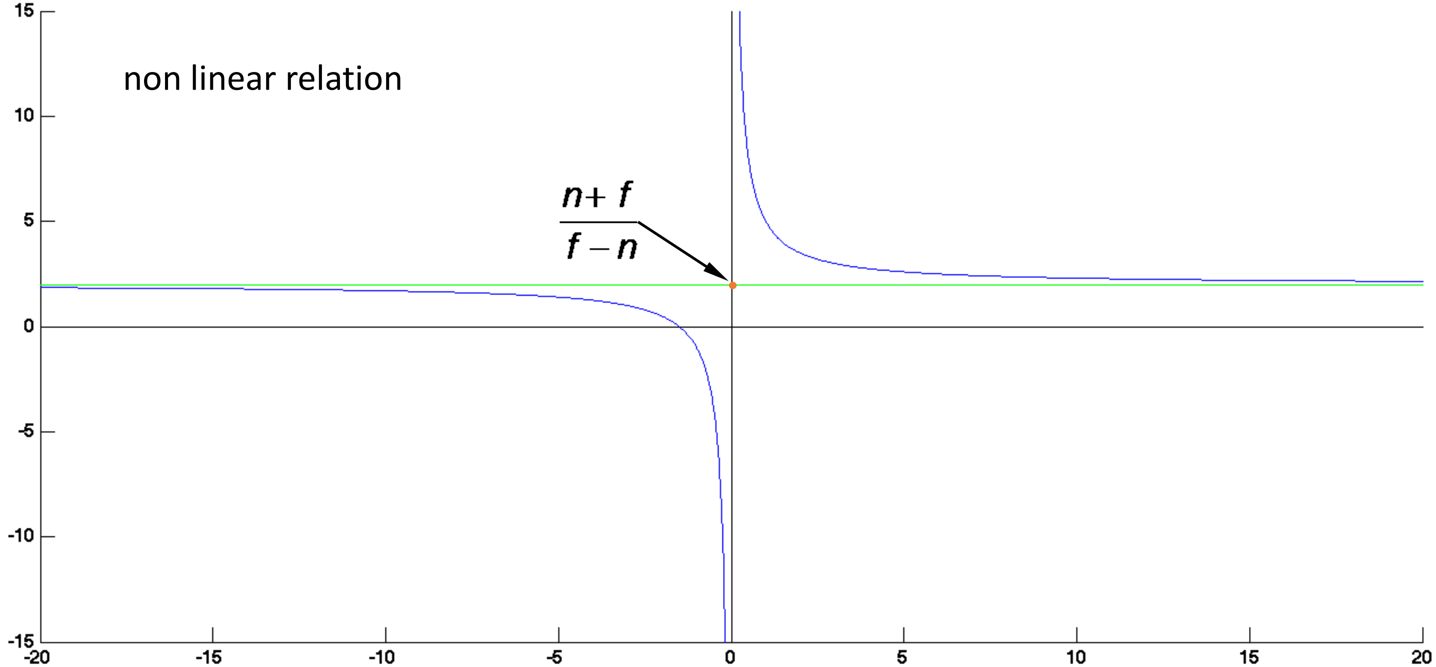


- too small near
  → z-values very close
  → depth order can get lost

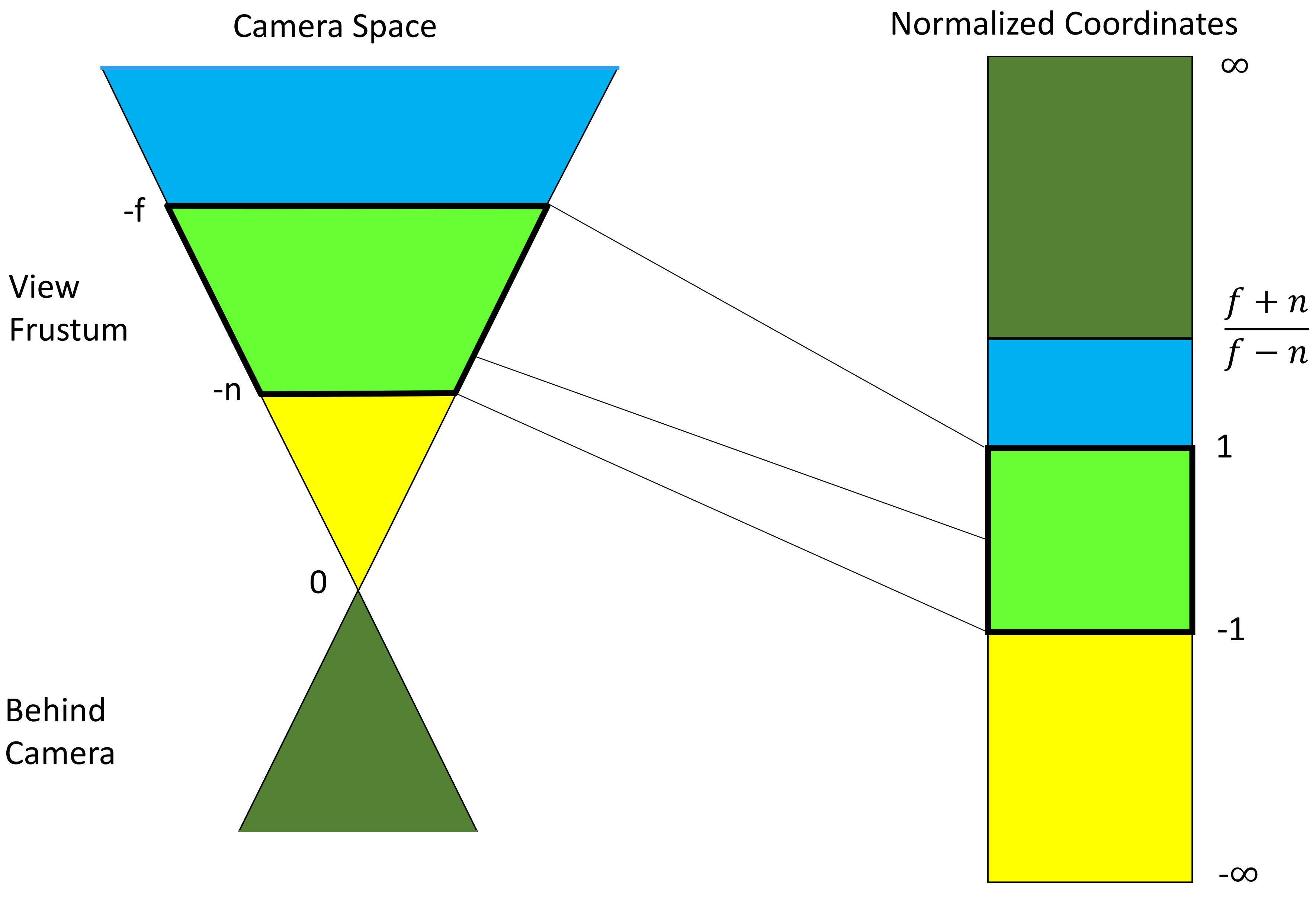


Normalizing Transformation

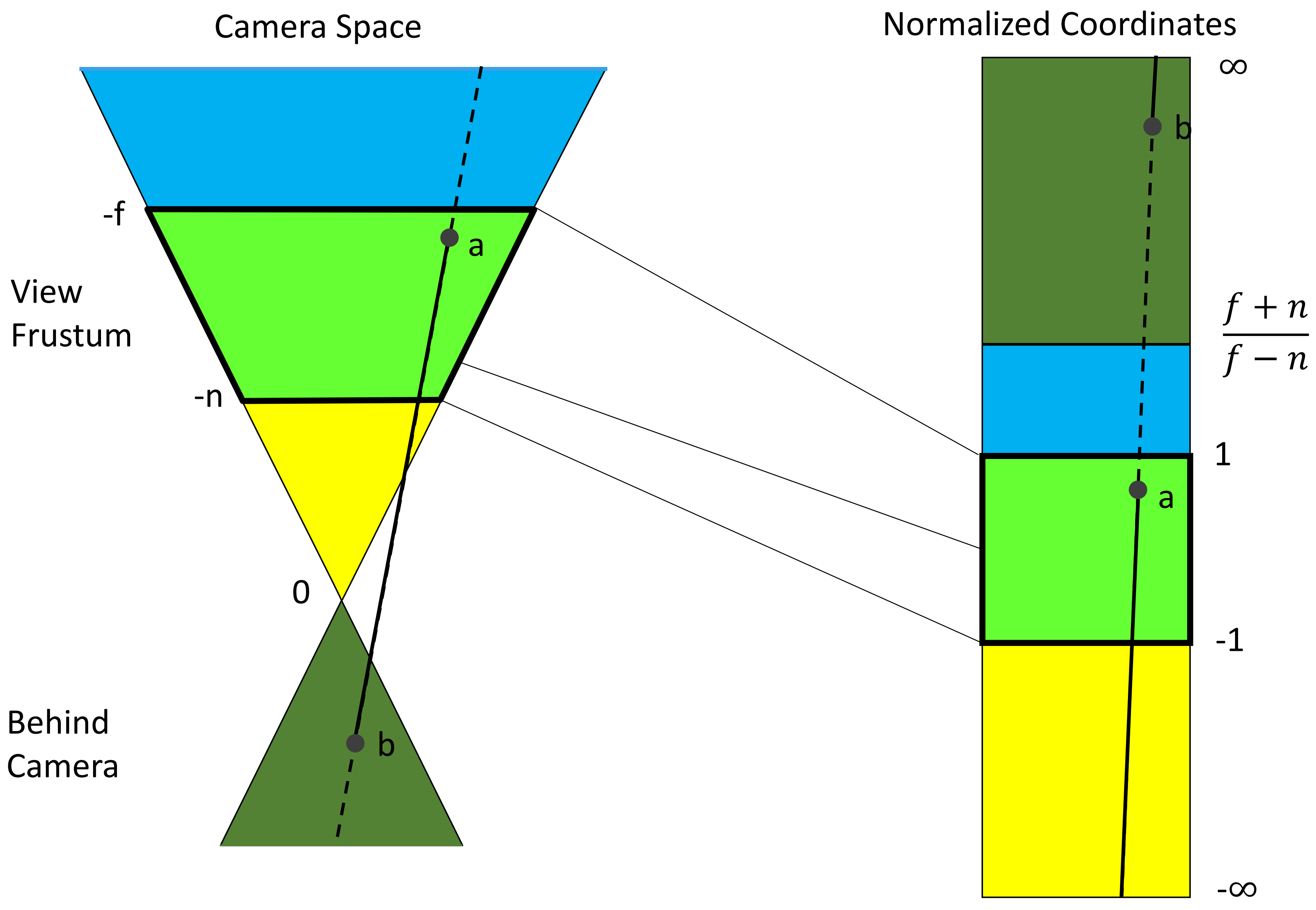
- $z \rightarrow \frac{n+f}{f-n} - \frac{2nf}{(f-n)z}$
  - If z is between planes n and f, it is mapped to $[-1,1]$
  - If z is between 0 and n, i.e. between camera and near plane, it is mapped to the interval $[-\infty, -1]$
  - If z is positive, i.e. behind the camera, then it remains positive after mapping.



Normalizing Transformation

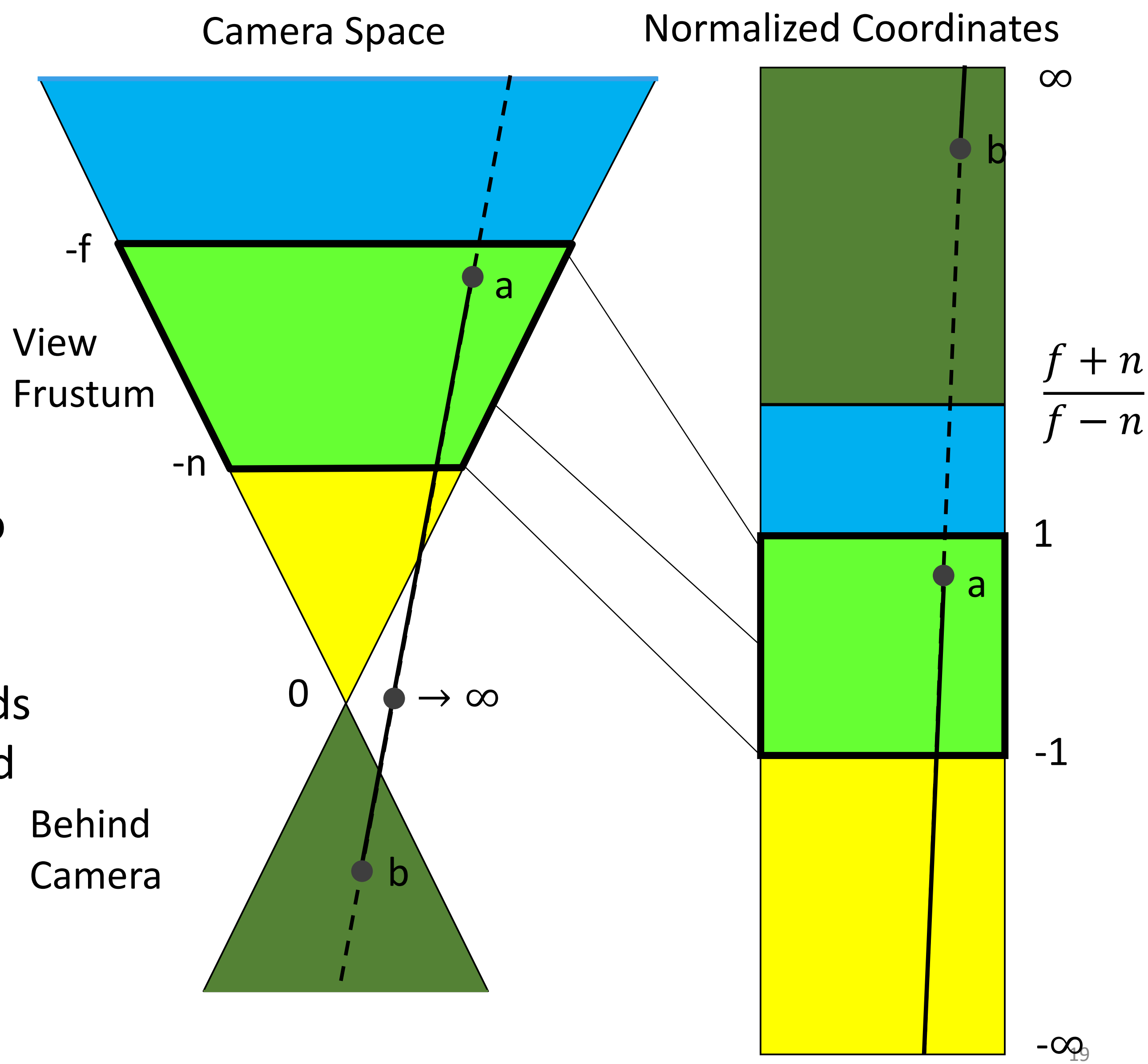


Normalizing Transformation



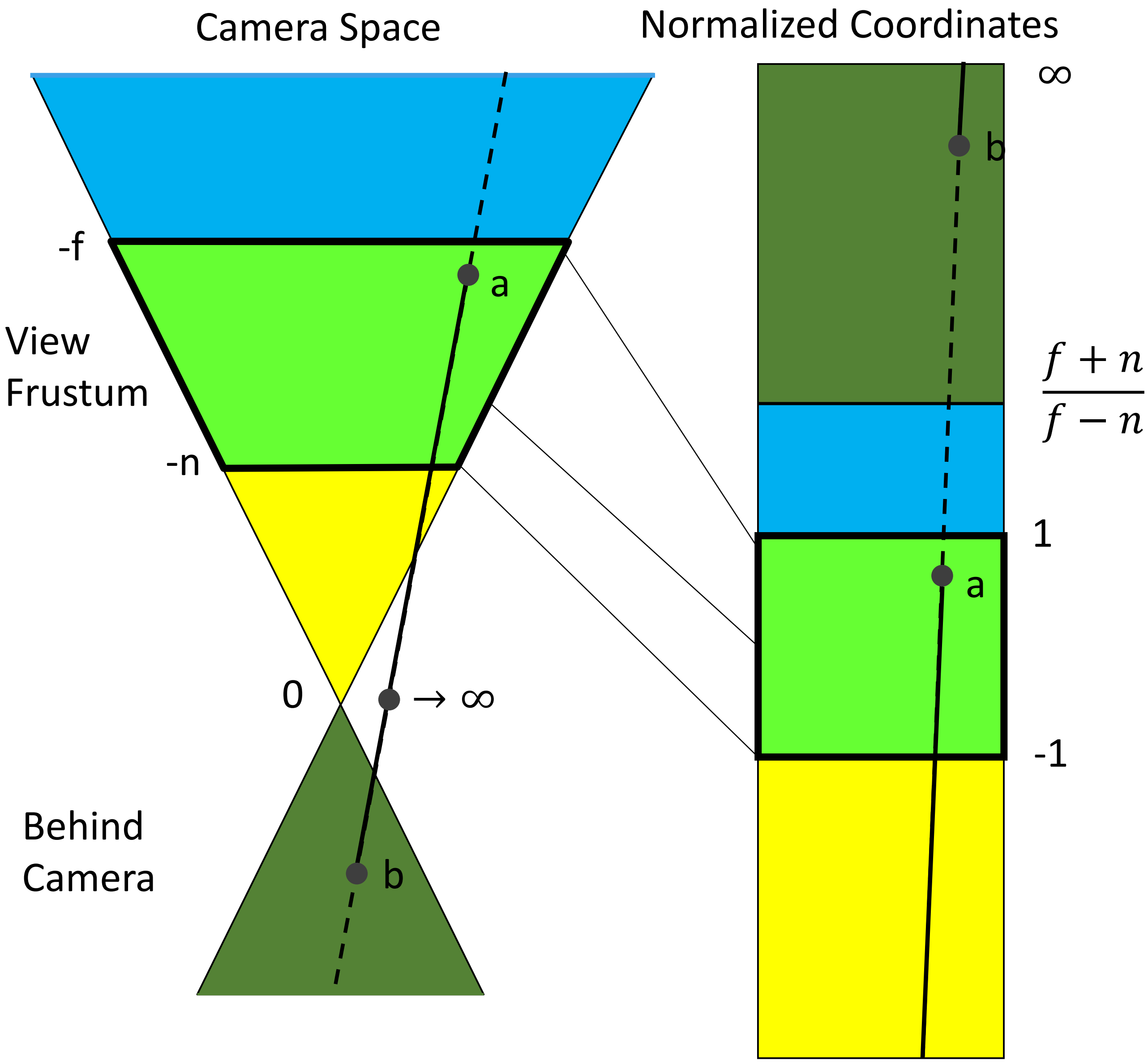
Normalizing Transformation

- The line through a and b is mapped to the line through the image of a and b
- The point at $z = 0$ is mapped to infinity
- So if we walk on the line segment from a to b in camera space, in image space, we walk from a downwards to infinity, wrap around to the other end and finally arrive at b



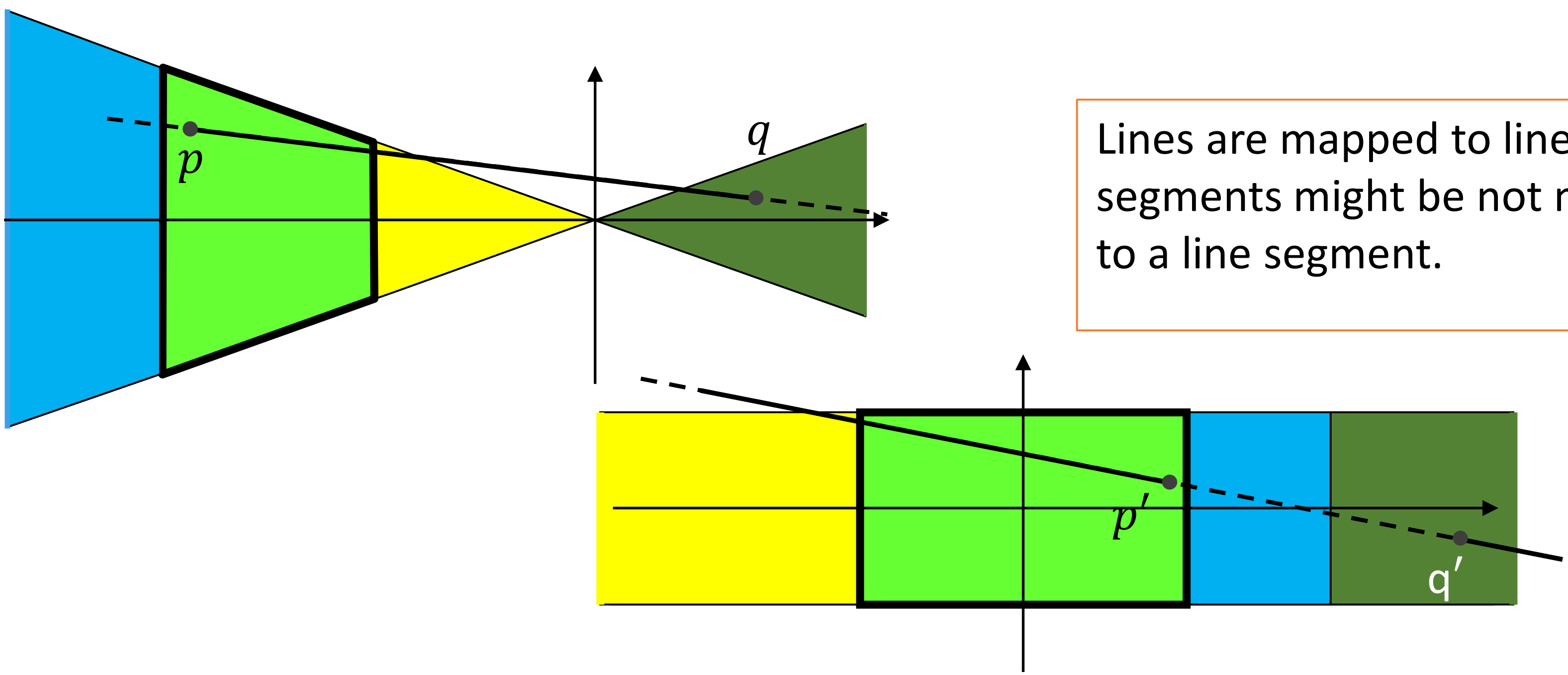
Normalizing Transformation

- Thus:
  Lines are mapped to lines
- But line segments are mapped to two rays…



Normalizing transformation

- Thus:
  If we have a line segment (p, q), and p is before, and q behind the camera, we cannot project p and q (to p' and q') and then simply render the line (p',q')…
  → homogeneous clipping



Lines are mapped to lines but line segments might be not mapped to a line segment.