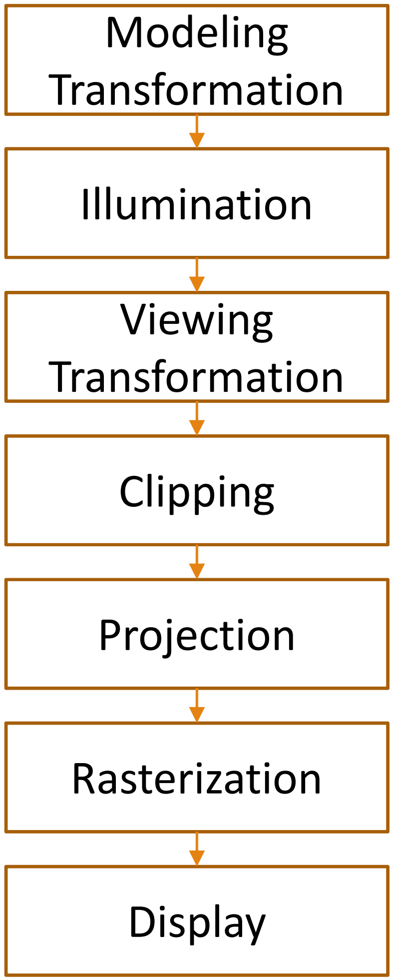
CT5510: Computer Graphics

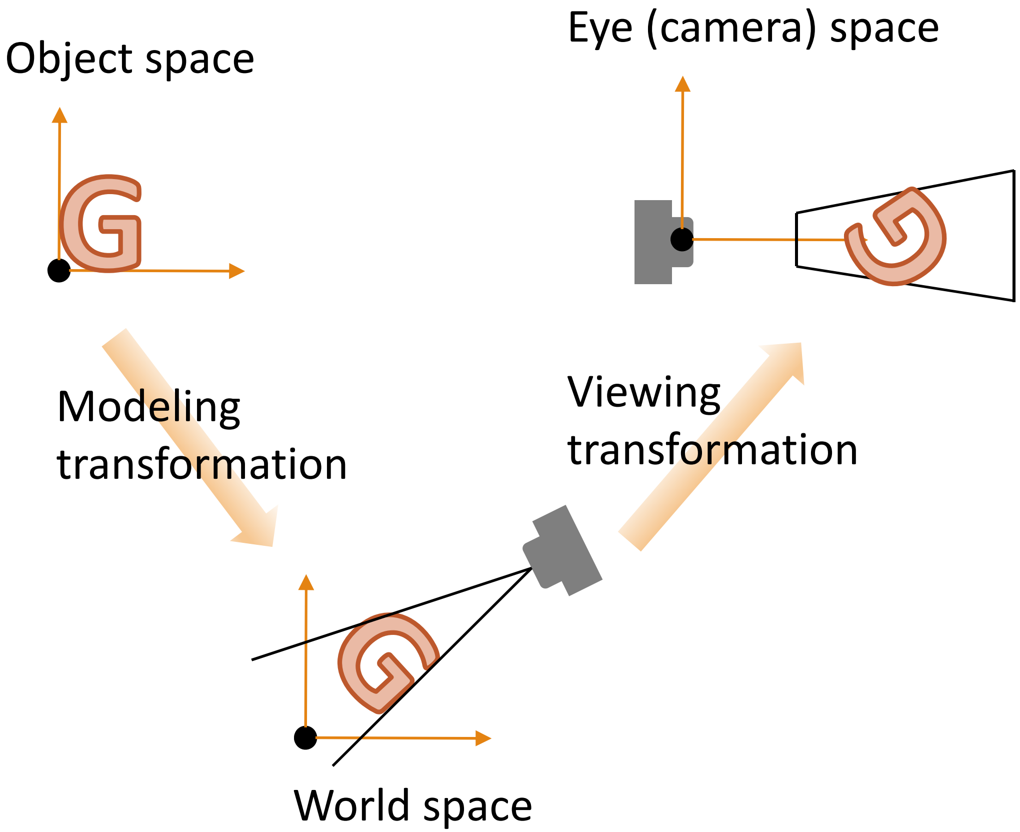
Projections

BOCHANG MOON

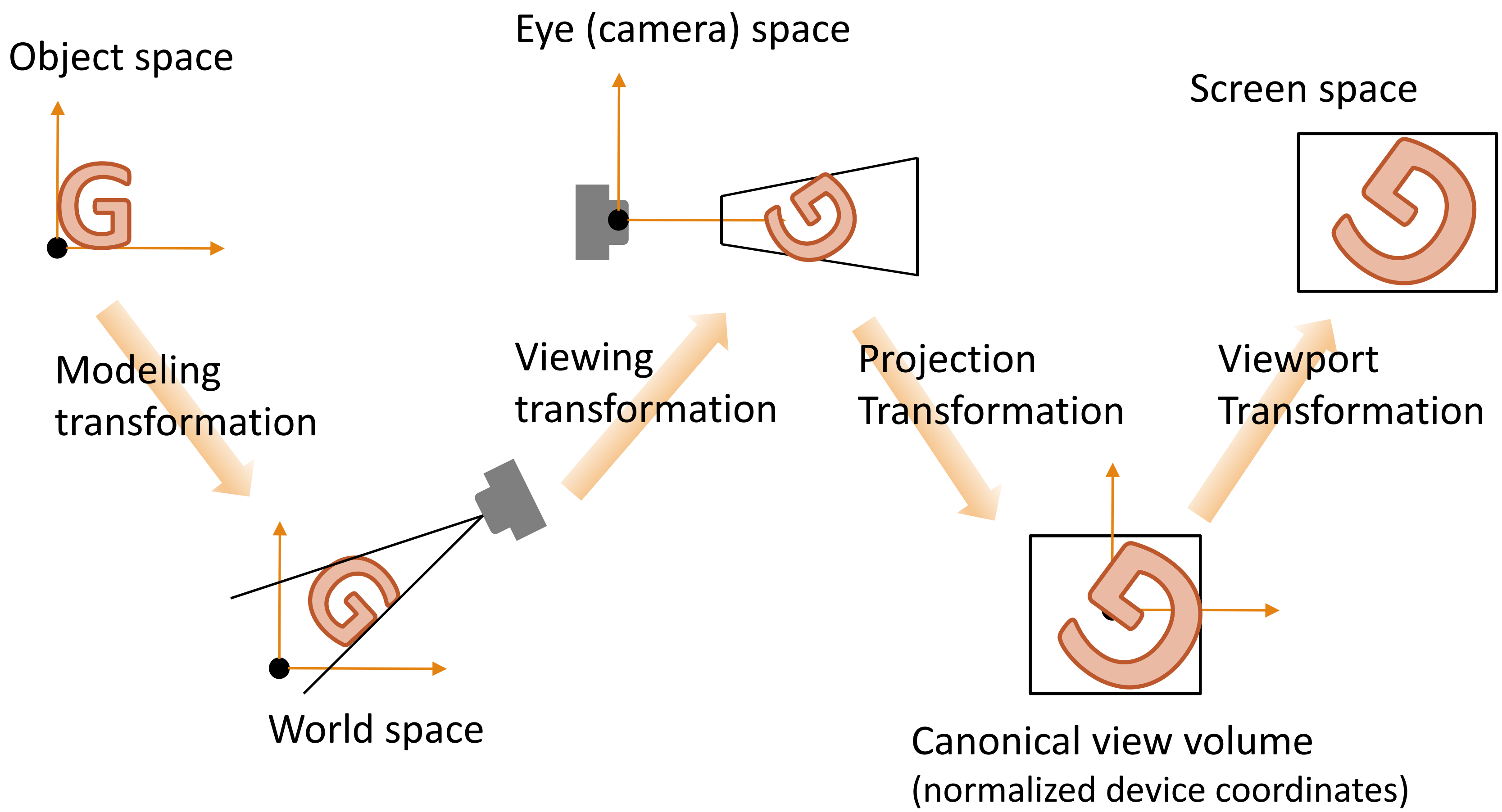
Graphics Pipeline



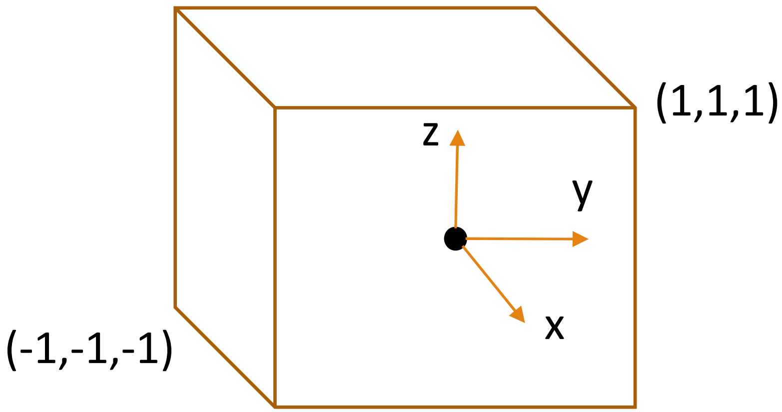
Sequence of Spaces and Transformations



Sequence of Spaces and Transformations

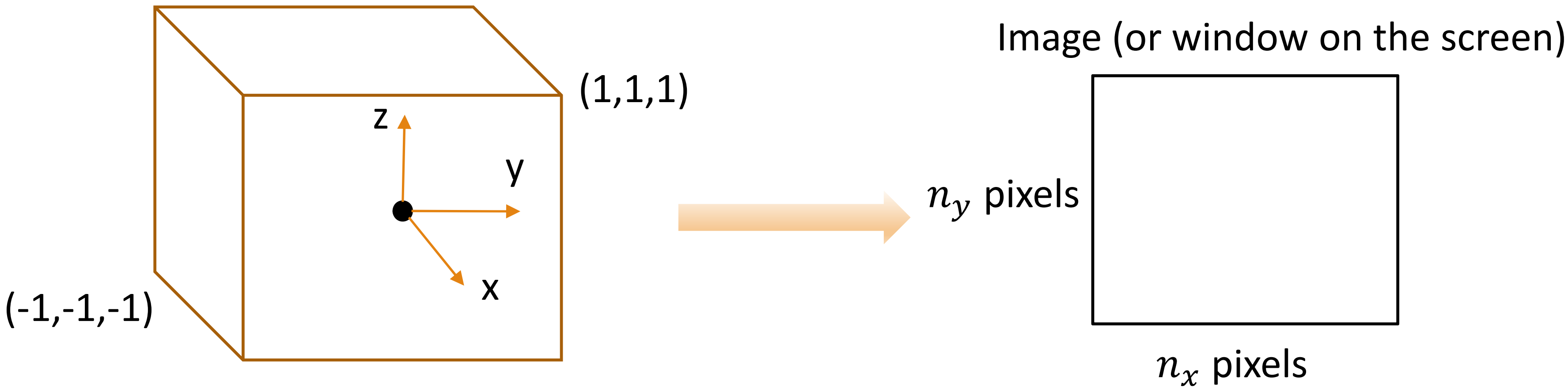


Canonical View Volume



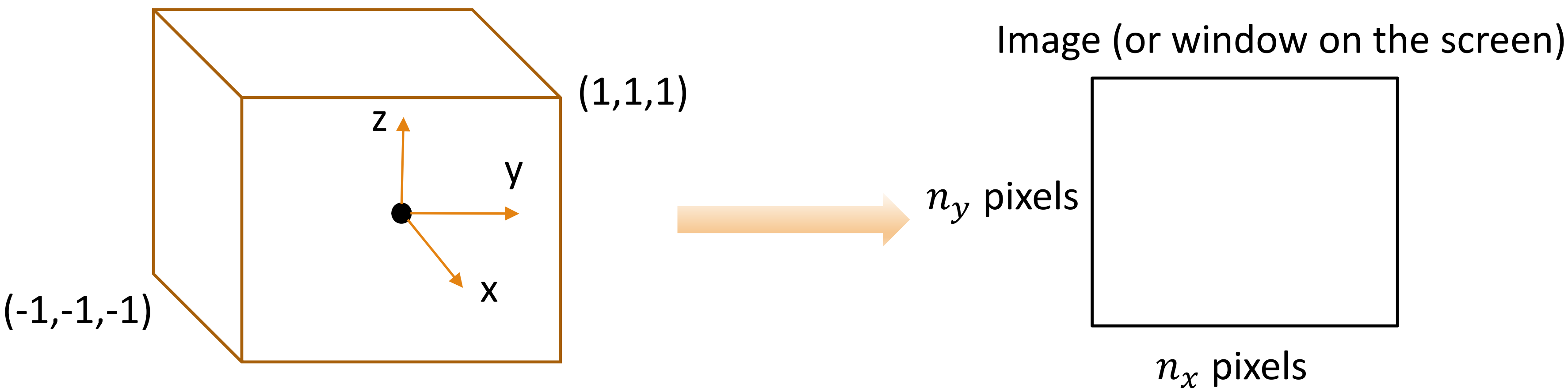
Viewport Transformation

- Primitives (or line segments) within the canonical view volume will be mapped to the image



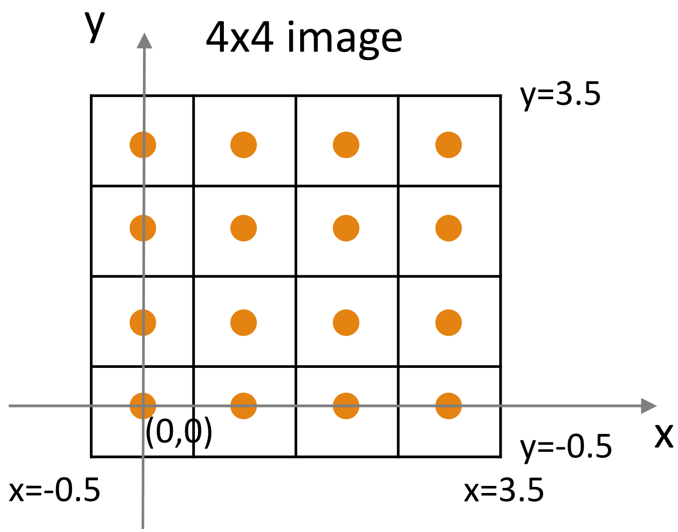
Viewport Transformation

- Ignore the z-coordinates of points for now
  - In practice, we need the z-coordinates and this will be covered later.
- Map the square $[-1,1]^2$ to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$



Raster Image (again)

- Ignore the z-coordinates of points for now
  - In practice, we need the z-coordinates and this will be covered later.
- Map the square $[-1,1]^2$ to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$
- Where do we need to locate pixels in 2D space?



Raster Image (again)

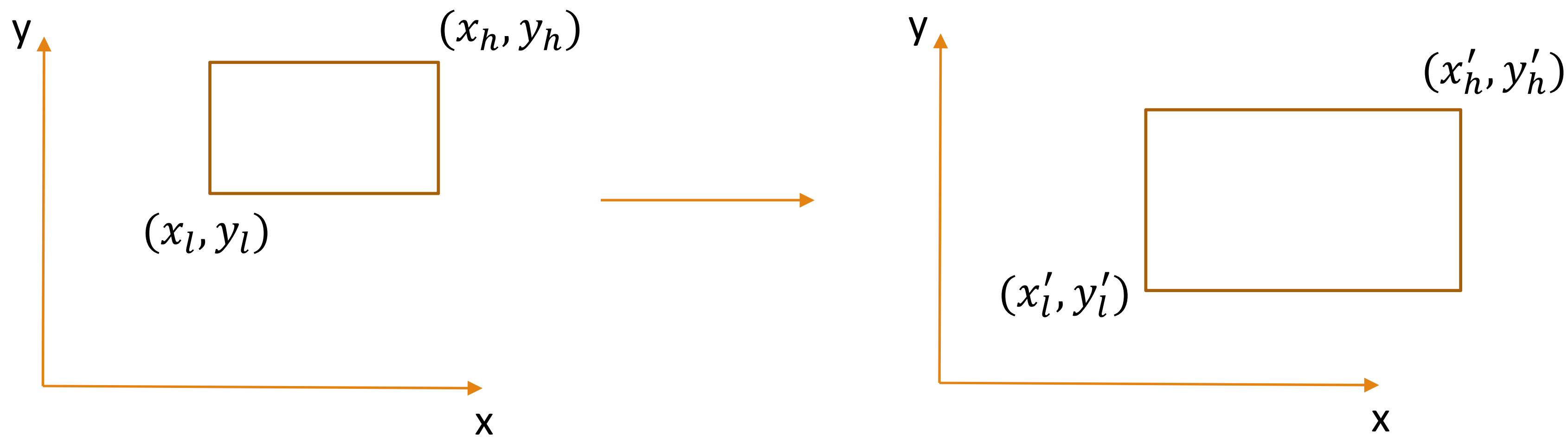
- Ignore the z-coordinates of points for now
  - In practice, we need the z-coordinates and this will be covered later.
- Map the square $[-1,1]^2$to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$
- Where do we need to locate pixels in 2D space?
- The rectangular domain of a $n_x \times n_y$ image
  - $R = [-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$

Viewport Transformation

- Ignore the z-coordinates of points for now
  - In practice, we need the z-coordinates and this will be covered later.
- Map the square $[-1,1]^2$ to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$
- Q. How do we transform a rectangle to another rectangle?

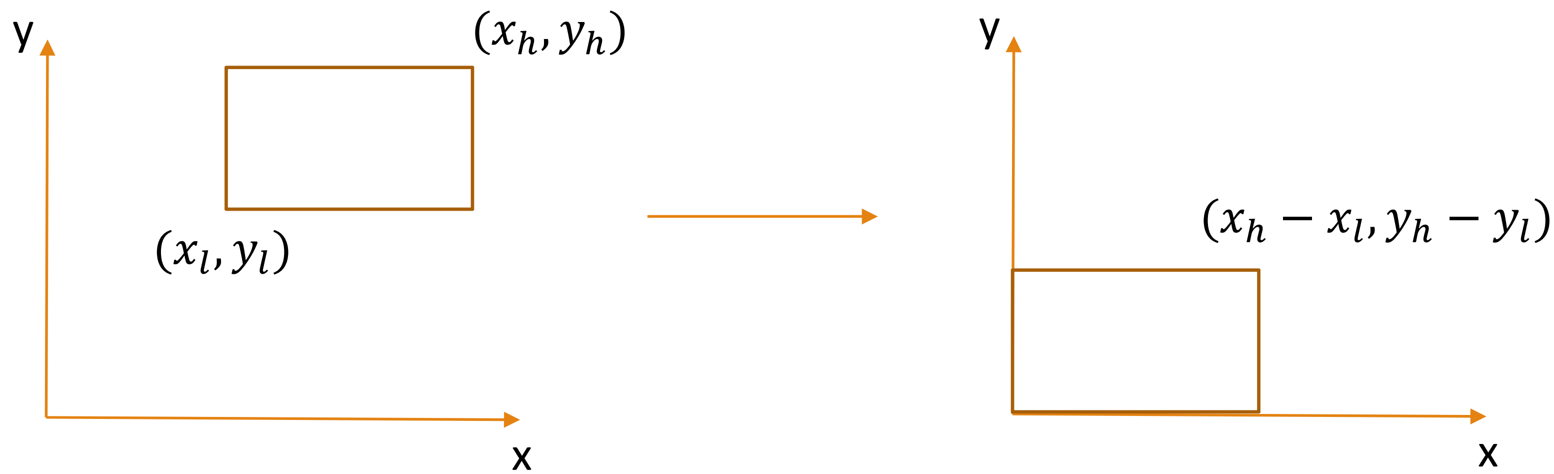
Example: Windowing Transform

- Problem specification: move a 2D rectangle into a new position



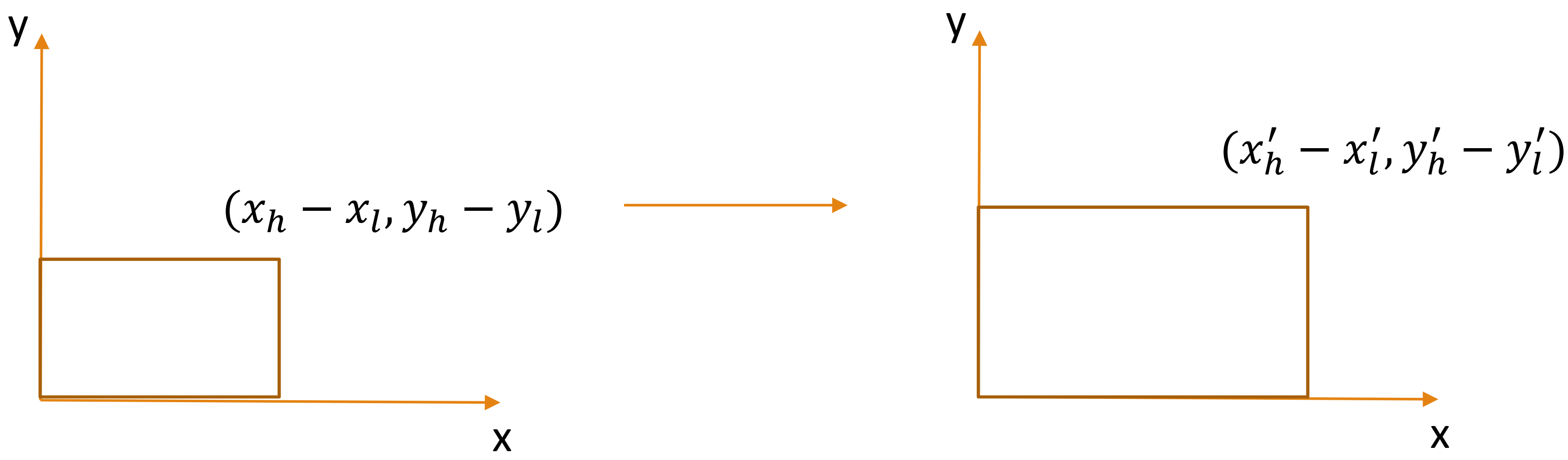
Example: Windowing Transform

- Problem specification: move a 2D rectangle into a new position
  - Step1. translate: move the point (x_l, y_l) to the origin



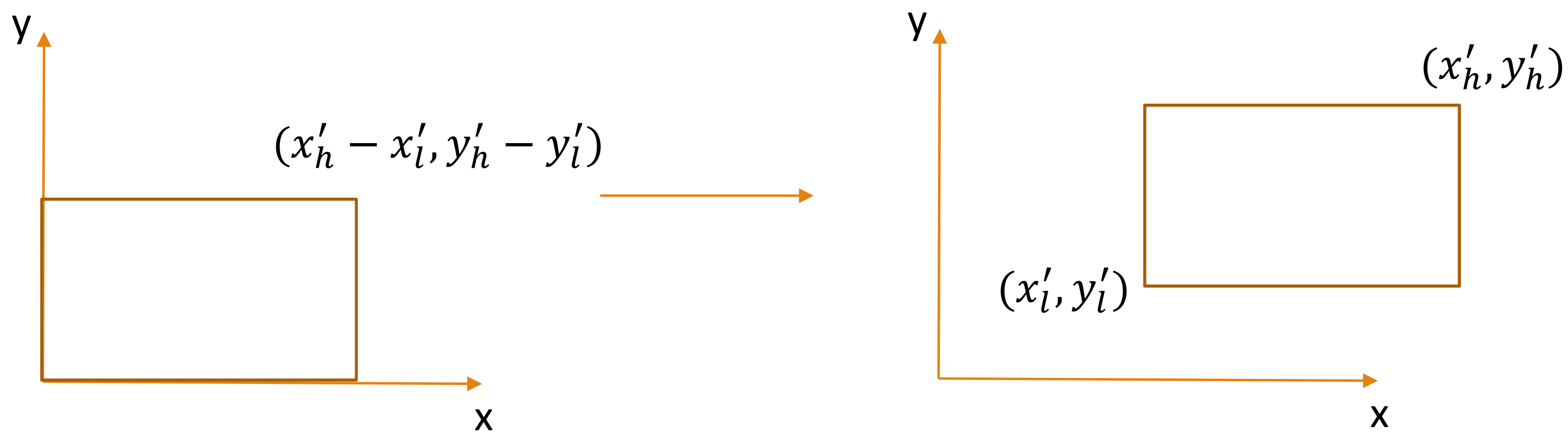
Example: Windowing Transform

- Problem specification: move a 2D rectangle into a new position
  - Step2. scale: resize the rectangle to be the same size of the target.



Example: Windowing Transform

- Problem specification: move a 2D rectangle into a new position
  - Step3. translate: move the origin to point (x_l', y_l')



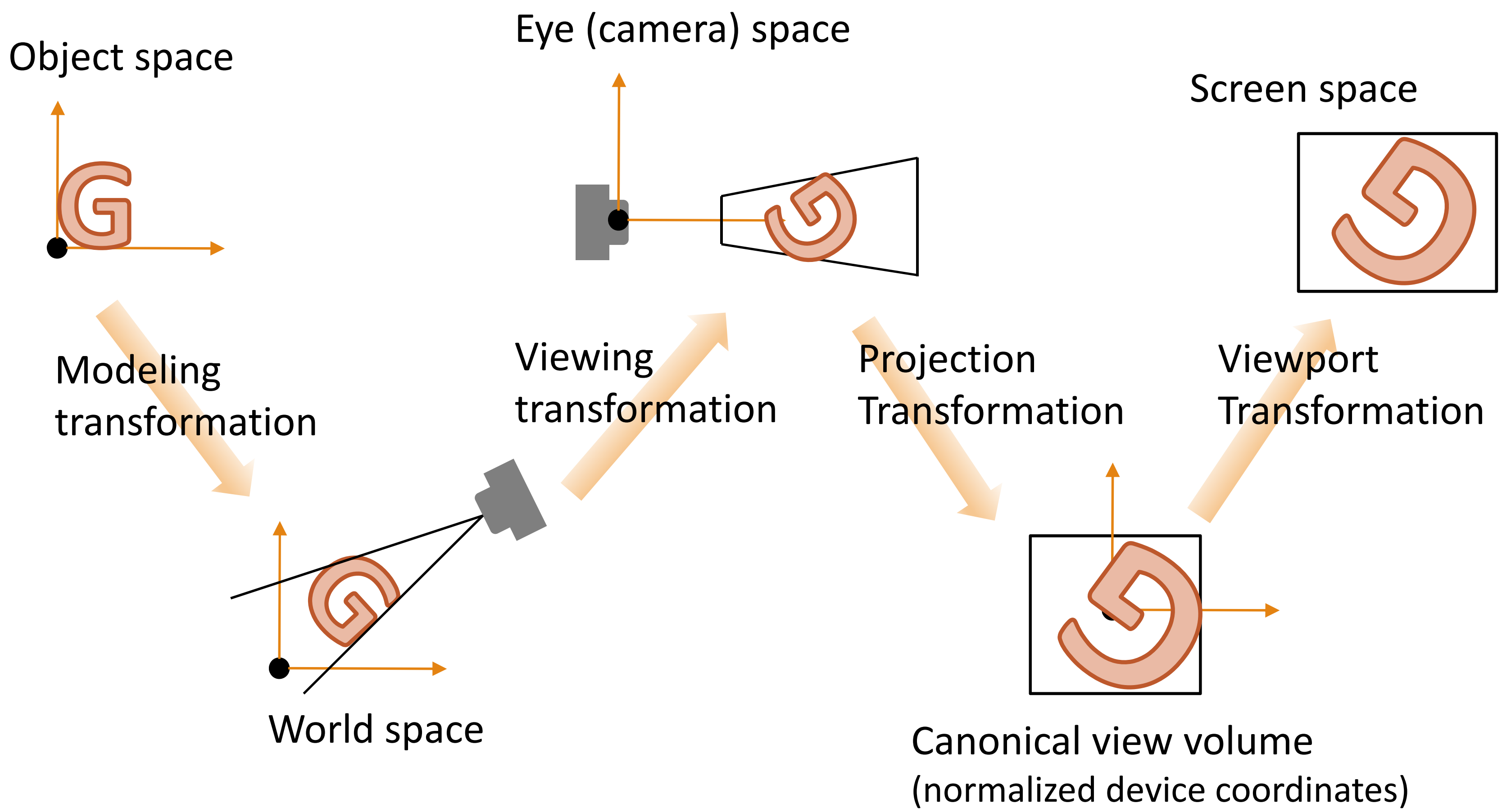
Example: Windowing Transform

- Problem specification: move a 2D rectangle into a new position
  - Target = translate(x_l', y_l') $scale\left(\frac{x_h'-x_l'}{x_h-x_l}, \frac{y_h'-y_l'}{y_h-y_l}\right)$ $translate(-x_l, -y_l)$
  - $= \begin{bmatrix} 1 & 0 & x_l' \\ 0 & 1 & y_l' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{x_h'-x_l'}{x_h-x_l} & 0 & 0 \\ 0 & \frac{y_h'-y_l'}{y_h-y_l} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_l \\ 0 & 1 & -y_l \\ 0 & 0 & 1 \end{bmatrix}$
  - $= \begin{bmatrix} \frac{x_h'-x_l'}{x_h-x_l} & 0 & \frac{x_l' x_h - x_h' x_l}{x_h-x_l} \\ 0 & \frac{y_h'-y_l'}{y_h-y_l} & \frac{y_l' y_h - y_h' y_l}{y_h-y_l} \\ 0 & 0 & 1 \end{bmatrix}$

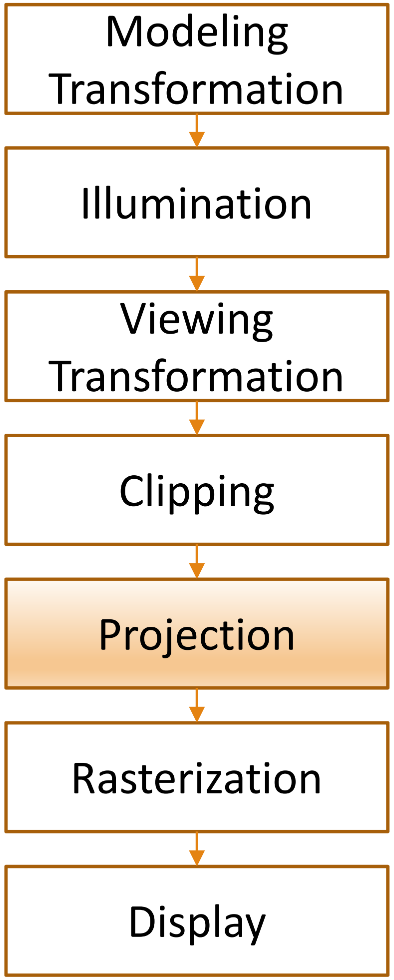
Viewport Transformation

- Ignore the z-coordinates of points for now
  - In practice, we need the z-coordinates and this will be covered later.
- Map the square $[-1,1]^2$to the rectangle $[-0.5, n_x - 0.5] \times [-0.5, n_y - 0.5]$
- $$\begin{bmatrix} x_{screen} \\ y_{screen} \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{n_x}{2} & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & \frac{n_{y-1}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{canonical} \\ y_{canonical} \\ 1 \end{bmatrix}$$
- For the case with z-coordinates,
  - $$M_{viewport} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x - 1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y - 1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

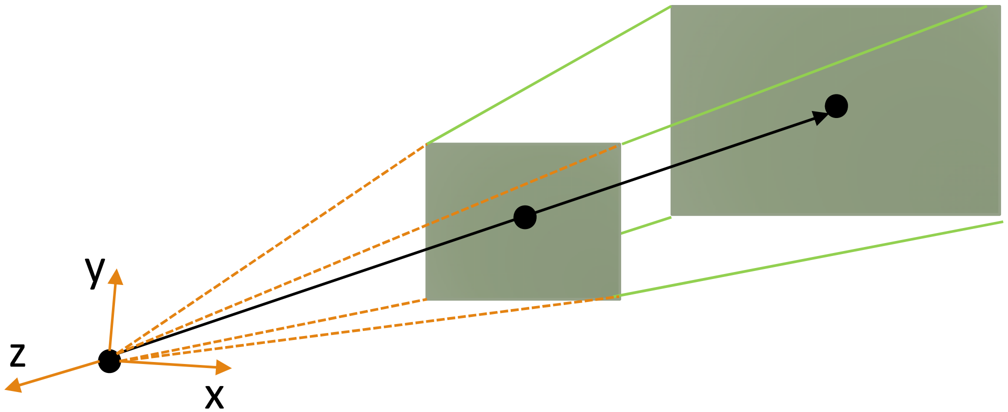
Sequence of Spaces and Transformations



Projections



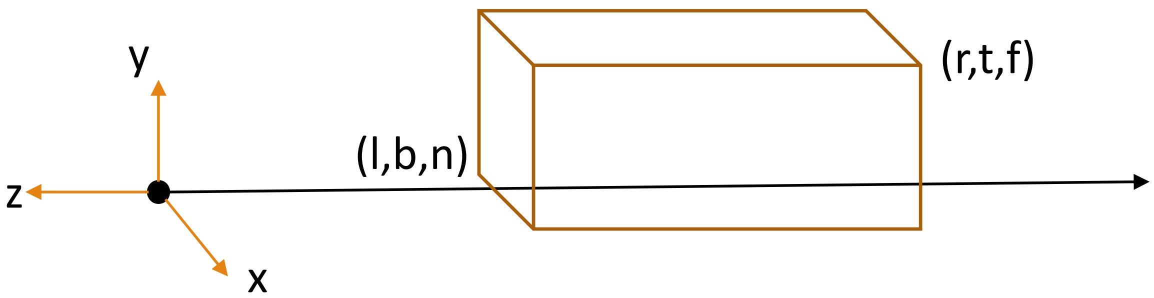
- Transform 3D points in *eye space* to 2D points in *image space*



- Two types of projections
  - Orthographic projection
  - Perspective projection

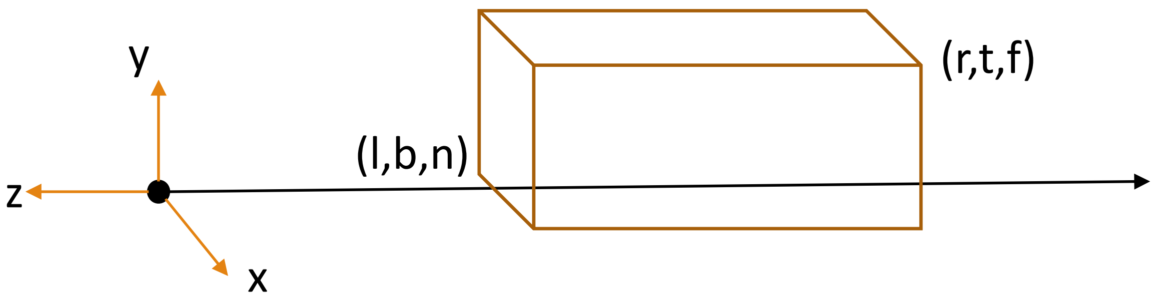
Orthographic Projection

- Assumption
  - A viewer is looking along the minus z-axis with his head pointing in the y-direction
    - Implies n > f



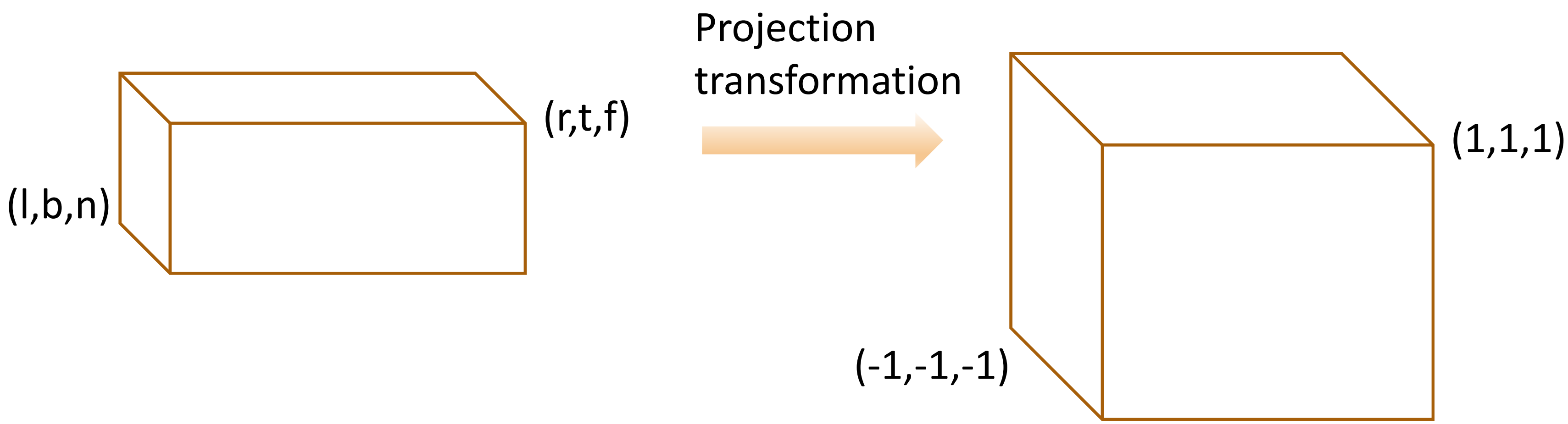
Orthographic Projection

- The view volume (*orthographic view volume*) is an axis-aligned box
  - [l, r] x [b, t] x [f, n]
- Notations
  - $x = l \equiv left\ plane, x = r \equiv right\ plane$
  - $y = b \equiv bottom\ plane, y = t \equiv top\ plane$
  - $z = n \equiv near\ plane, z = f \equiv far\ plane$



Orthographic Projection

- Transform points in orthographic view volume to the canonical view volume
  - Also windowing transform (3D)



Orthographic Projection

- Transform points in orthographic view volume to the canonical view volume
  - Also windowing transform (3D)
    - Map a box $[x_l, x_h] \times [y_l, y_h] \times [z_l, z_h]$ to another box $[x_l', x_h'] \times [y_l', y_h'] \times [z_l', z_h']$
    - $$\begin{bmatrix} \frac{x_h'-x_l'}{x_h-x_l} & 0 & 0 & \frac{x_l' x_h - x_h' x_l}{x_h - x_l} \\ 0 & \frac{y_h'-y_l'}{y_h-y_l} & 0 & \frac{y_l' y_h - y_h' y_l}{y_h - y_l} \\ 0 & 0 & \frac{z_h'-z_l{}'}{z_h-z_l} & \frac{z_l' z_h - z_h' z_l}{z_h - z_l} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Orthographic Projection

- Transform points in orthographic view volume to the canonical view volume
  - Also windowing transform (3D)
- $$M_{ortho} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{2}{n-f} & -\frac{n+f}{n-f} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Composite Transformation

- The matrix that transforms points in world space to screen coordinate:

- $M = M_{viewport} M_{ortho} M_{viewing}$

Orthographic Projection

- Transform points in orthographic view volume to the canonical view volume
  - Also windowing transform (3D)
- Tend to ignore relative distances between objects and eye
  - Unrealistic
- In practice,
  - We usually do not use this projection.
  - It can be useful in applications where relative lengths should be judged.

Orthographic Projection in OpenGL

- void glOrtho(GLdouble left, GLdouble right,
- GLdouble bottom, GLdouble top,
- GLdouble nearVal, GLdouble farVal);

Perspective Projection

- Objects in an image become smaller as their distance from the eye increases.
- History of perspective:
  - Artists from the Renaissance period employed the perspective property.

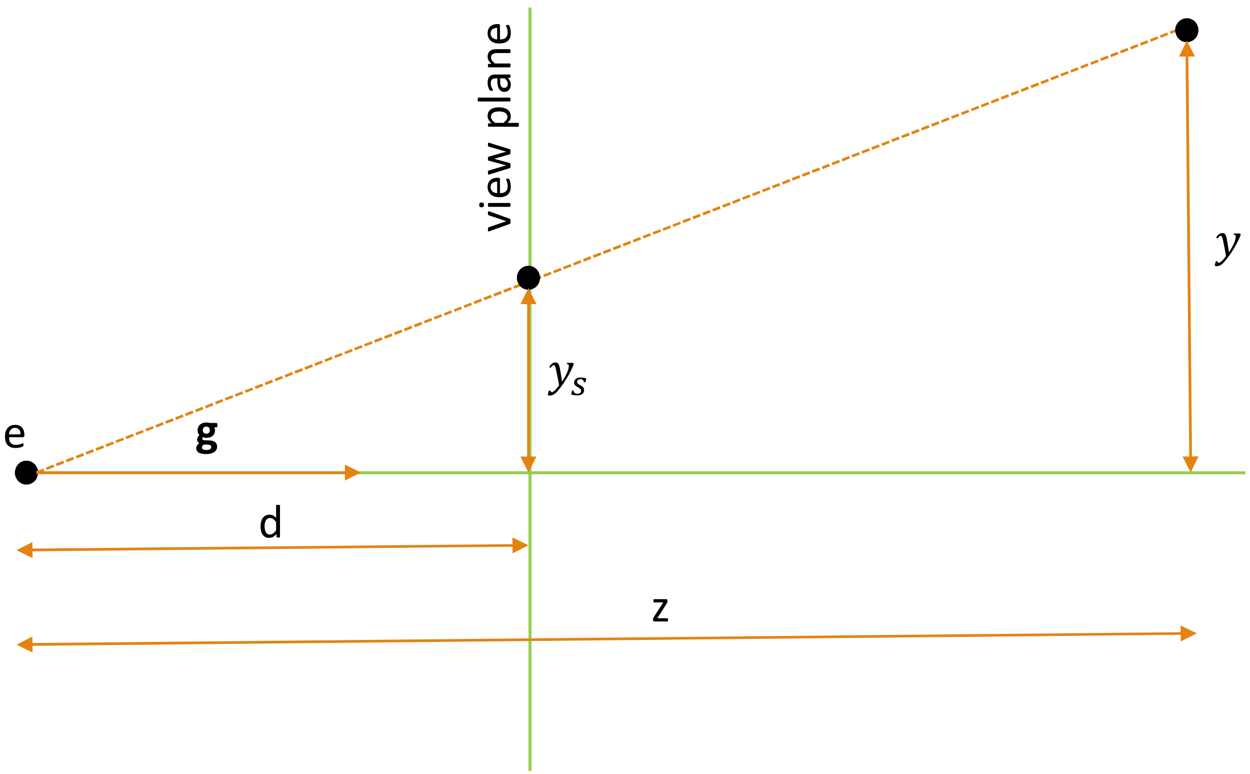
Images from wikipedia

Perspective Projection

- Objects in an image become smaller as their distance from the eye increases.
- History of perspective:
  - Artists from the Renaissance period employed the perspective property.
- In everyday life?

Perspective Projection

- $y_s = \frac{d}{z} y$
  - y_s: y-axis coordinate in view plane
  - y: distance of the point along the y-axis



Homogeneous Coordinate

- Represent a point (x, y, z) with an extra coordinate w
  - (x, y, z, w)
  - In the previous lecture, $w = 1$

- Let's define w to be the denominator of the x-, y-, z-coordinates
  - (x, y, z, w) represent the 3D point $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$
  - A special case, $w = 1$, is still valid.
  - w can be any values

Projective Transform

- Let's define w to be the denominator of the x-, y-, z-coordinates
  - (x, y, z, w) represent the 3D point $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$
  - A special case, $w = 1$, is still valid.
  - w can be any values

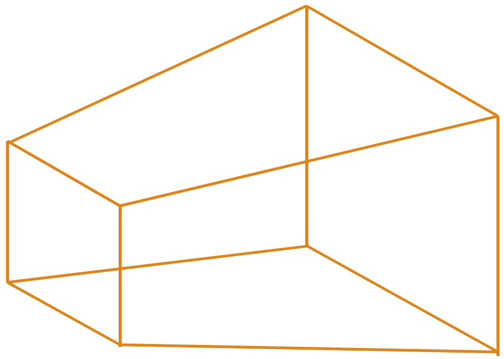
- Projective transformation
  - $$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ e & f & g & h \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$
  - $(x', y', z') = (\frac{\tilde{x}}{\tilde{w}}, \frac{\tilde{y}}{\tilde{w}}, \frac{\tilde{z}}{\tilde{w}})$

Perspective Projection

- Example with 2D homogeneous vector $[y\ z\ 1]^T$
  - $\begin{bmatrix} y_s \\ 1 \end{bmatrix} = \begin{bmatrix} d & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}$
  - This is corresponding to the perspective equation, $y_s = \frac{d}{z} y$.

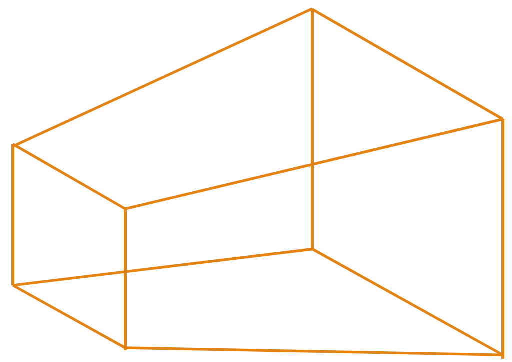
Perspective Projection

- Some info. for perspective matrix
  - Define our project plane as the near plane
  - Distance to the near plane: $-n$
  - Distance to the far plane: $-f$
- Perspective equation: $y_s = \frac{n}{z} y$
- *Perspective matrix*
  - $P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$



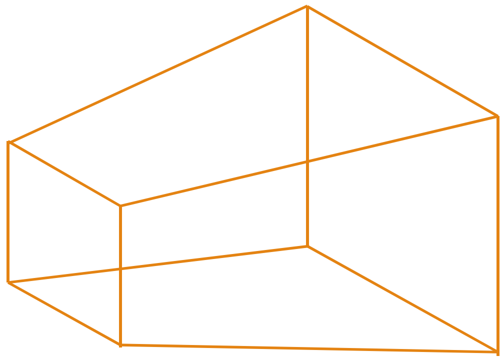
Perspective Projection

- *Perspective matrix*
  - $P = \begin{bmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ 0 & 0 & n+f & -fn \\ 0 & 0 & 1 & 0 \end{bmatrix}$
- A mapping with the perspective matrix:
  - $P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n+f-\frac{fn}{z} \\ 1 \end{bmatrix}$



Perspective Projection

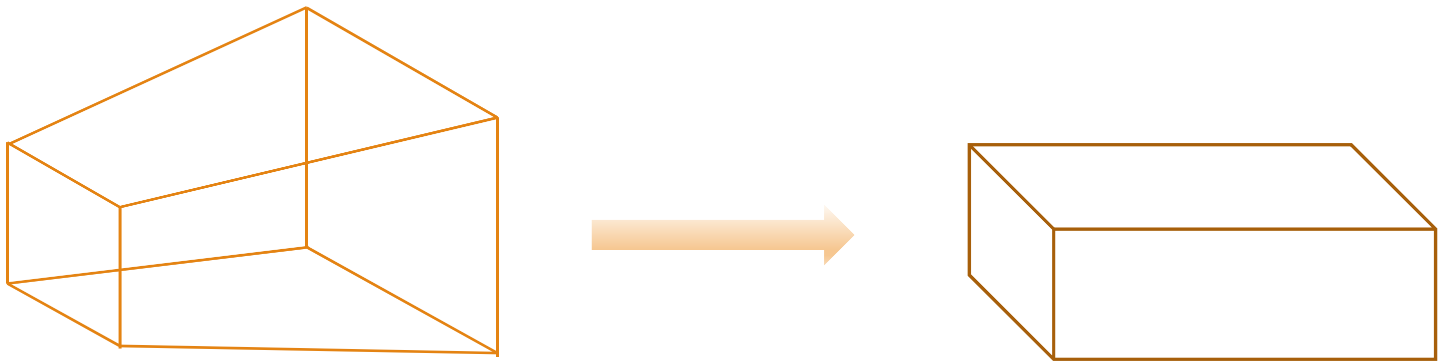
- A mapping with the perspective matrix:
  - $P\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} nx \\ ny \\ (n+f)z - fn \\ z \end{bmatrix} = \begin{bmatrix} \frac{nx}{z} \\ \frac{ny}{z} \\ n + f - \frac{fn}{z} \\ 1 \end{bmatrix}$



- Properties
  - The first, second, and fourth rows are for the perspective equation.
  - The third row is for keeping z coordinate at least approximately.
    - E.g., when z = n, transformed z coordinate is still n.
    - E.g., when $z > n$, we cannot preserve the z coordinate exactly, but relative orders between points will be preserved.

Perspective Projection

- Perspective matrix
  - Map the perspective view volume to the orthographic view volume.

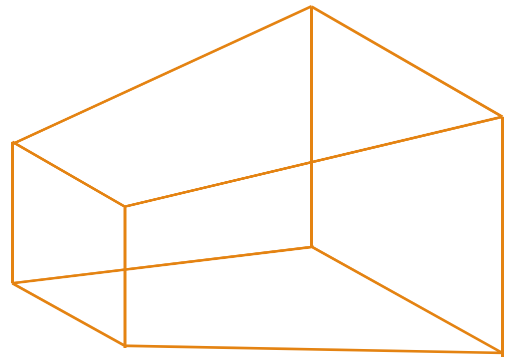


Composite Transformation

- The matrix that transforms points in world space to screen coordinate:
- $M = M_{viewport} M_{ortho} P M_{viewing} = M_{viewport} M_{per} M_{viewing}$
- $M_{per} = M_{ortho} P$ *(perspective projection matrix)*
  - $M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$

Perspective Projection in OpenGL

- void glFrustum(GLdouble left, GLdouble right,
- GLdouble bottom, GLdouble top,
- GLdouble nearVal, GLdouble farVal);

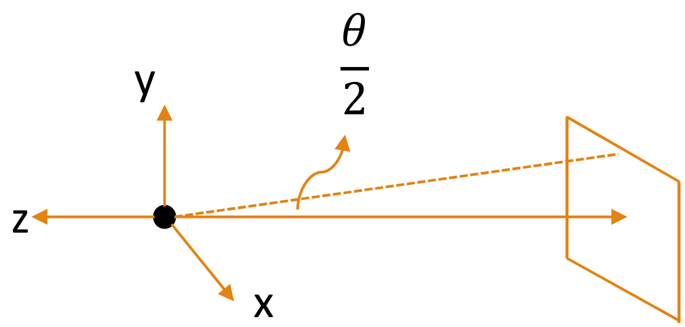


Perspective Projection in OpenGL

- void gluPerspective(GLdouble fovy, GLdouble aspect,
- GLdouble zNear, GLdouble zFar);

- Parameters
  - fovy: field of view (in degrees) in the y direction
  - aspect: aspect ratio is the ratio of x (width) to y (height)

- Symmetric constraints are implicitly applied.
  - l = -r, b = -t



- A constraint to prevent image distortion
  - $\frac{n_x}{n_y} = \frac{r}{t}$

Further Reading

- In our textbook, Fundamentals of Computer Graphics (4th edition)
  - Chapter 7