# Computer Graphics

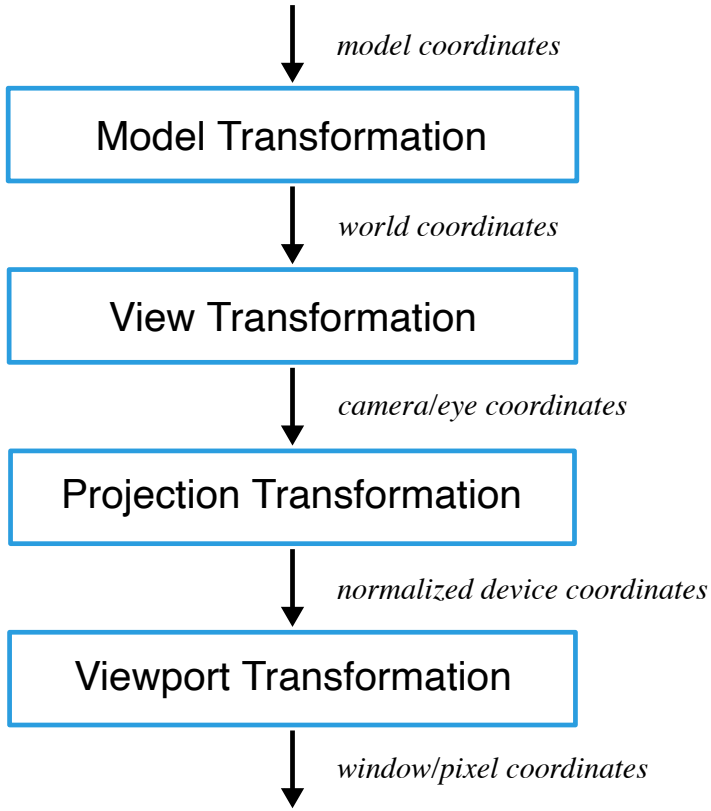
## *Viewing*

Mark Pauly

Geometric Computing Laboratory

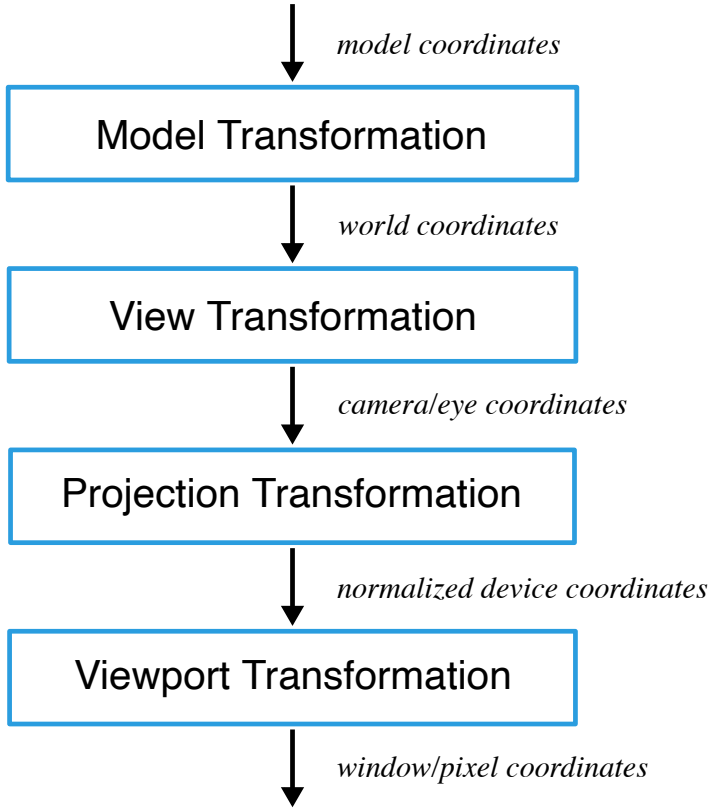
# Transformation Pipeline

- Split complex transformation from 3D to 2D into a sequence of simpler transformations.
- We discussed model transformations before
- **Now we discuss view, projection, and viewport transformations**



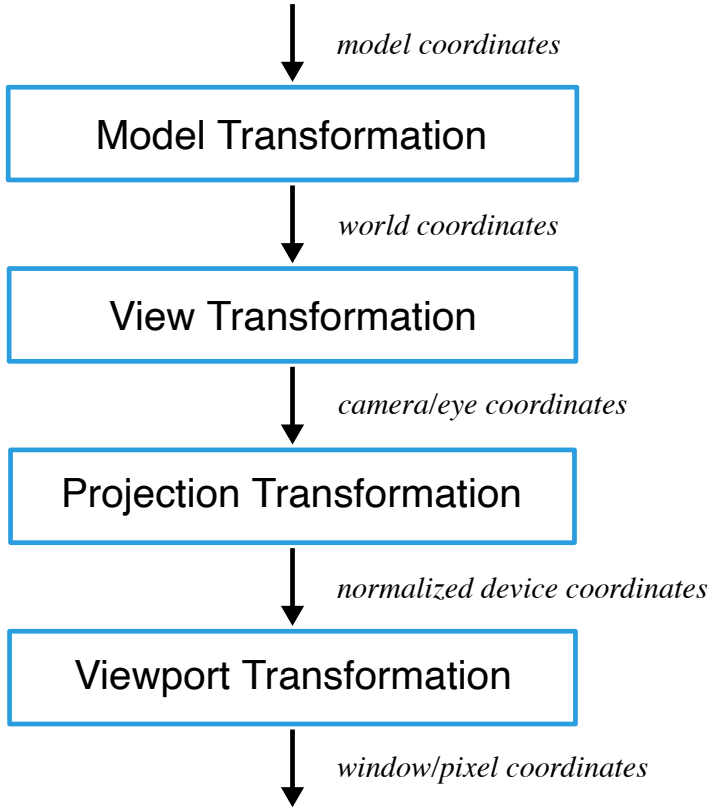
# Transformation Pipeline

- **View Transformation**
  - setup extrinsic camera parameters: position and orientation
- **Projection Transformation**
  - setup intrinsic camera parameters: opening angle and depth range
- **Viewport Transformation**
  - setup image parameters/resolution: width and height



# Transformation Pipeline

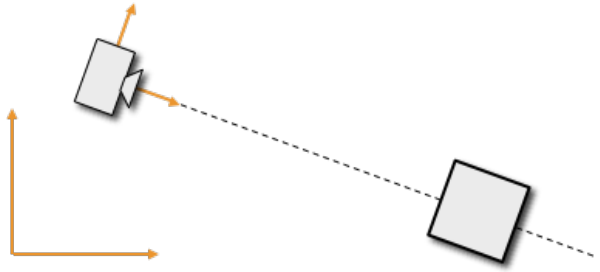
- **Model coordinates**
  - local coordinate system for each model
- **World coordinates**
  - one global coordinate system
- **Camera / eye coordinates**
  - world transformed to standard camera
- **Normalized device coordinates**
  - viewing volume mapped to $[-1,1]^3$
- **Window coordinates**
  - from $[-1,1]^3$ to $[-1,1]^2$ to pixel coordinates



# View Transformation

# View Transformation

- Specify extrinsic camera parameters
  - Camera/eye location $\mathbf{e}$
  - Viewing direction $\mathbf{v}$
  - Up direction $\mathbf{u}$
  - Right direction $\mathbf{r}$
  - Assume $\mathbf{v}$, $\mathbf{u}$, $\mathbf{r}$ are **orthonormal**



- Transform scene to *standard camera*
  - Camera/eye location $(0, 0, 0)$
  - Viewing direction $(0, 0, -1)$
  - Up direction $(0, 1, 0)$
  - Right direction $(1, 0, 0)$

# View Transformation

- Matrix from standard camera to $\mathbf{e}, \mathbf{r}, \mathbf{u}, \mathbf{v}$:

$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- The inverse matrix does the job

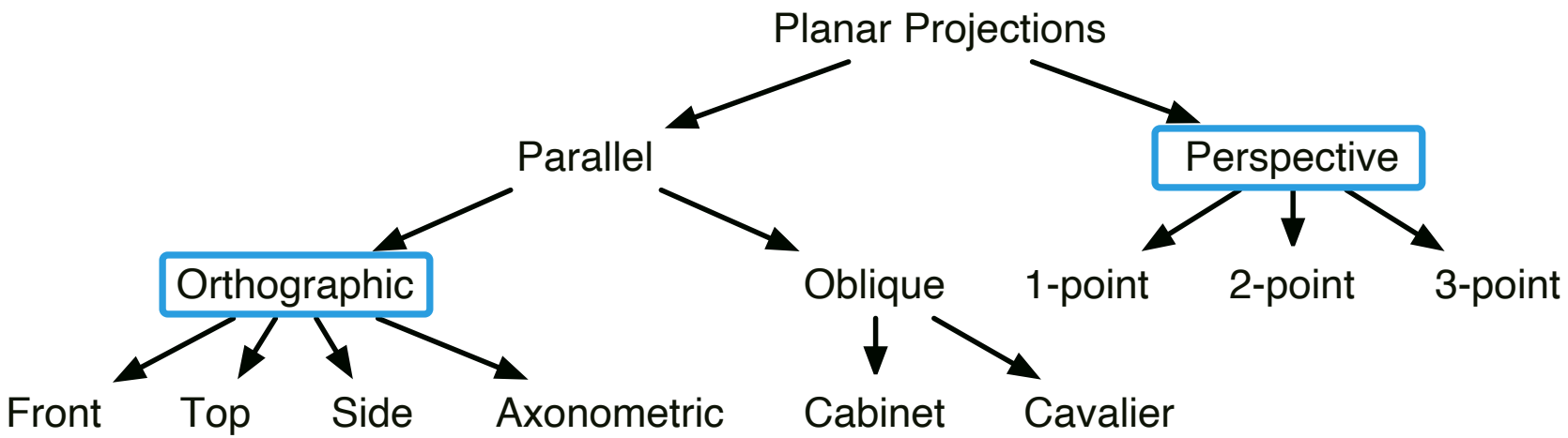
$$\begin{pmatrix} r_x & u_x & -v_x & e_x \\ r_y & u_y & -v_y & e_y \\ r_z & u_z & -v_z & e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ -v_x & -v_y & -v_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Projections

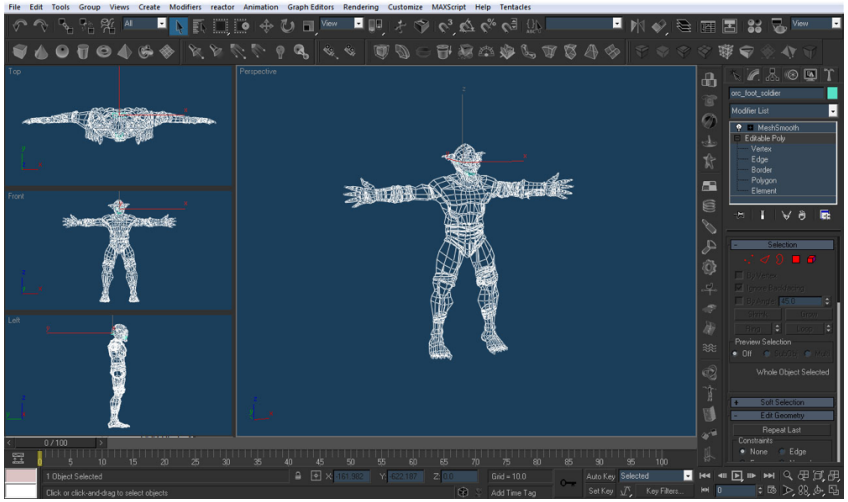
# What is a projection?

- In mathematics
  - An *idempotent* mapping: $\mathbf{P}(\mathbf{P}(\mathbf{x})) = \mathbf{P}(\mathbf{x})$
- In computer graphics
  - A mapping from 3D space to a planar 2D image
  - Parallel or perspective projection
  - Transforms viewing volume to normalized device coordinates $[-1, 1]^3$

# Classification of Projections



# Parallel vs. Perspective Projection



*Autodesk 3D Studio Max*

- **Parallel projections**
  - One direction of projection for all points
  - Orthographic: direction perpendicular to image plane
  - Preserve parallelism & lengths
- **Perspective projections**
  - Each point projects towards center of projection
  - Perspective foreshortening, realistic appearance

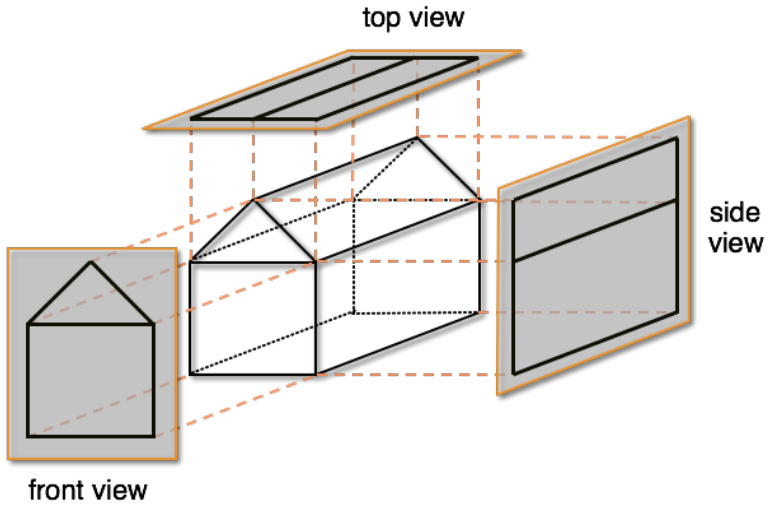
Image from Wikipedia

# Generic Orthographic Projection

- Standard camera setup
  - located at origin
  - looking down negative z-axis
- Orthogonal projection onto xy-plane

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  - xy-coordinates do not change
  - remove z-coordinate
  - keep w-coordinate



# OpenGL Orthographic Projection

- Specify viewing volume as axis-aligned box

$$[l, r] \times [b, t] \times [-n, -f]$$

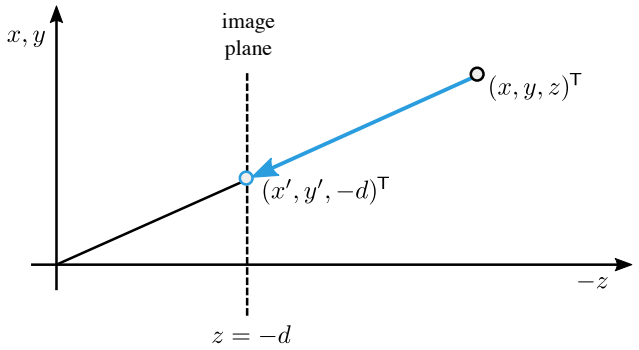
  with left/right, bottom/top, near/far boundaries

- Map viewing box to unit cube $[-1, 1]^3$
  - translate box center to origin
  - scale box dimensions to $2 \times 2 \times 2$
  - z values should be mirrored

$$\begin{pmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -\frac{l+r}{2} \\ 0 & 1 & 0 & -\frac{b+t}{2} \\ 0 & 0 & 1 & -\frac{-n-f}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{l+r}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{b+t}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{n+f}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Generic Perspective Projection

- Standard projection
  - Center of projection: $(0, 0, 0)$
  - Image plane at $z = -d$




$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \end{pmatrix} \overset{?}{\longleftrightarrow} \begin{pmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

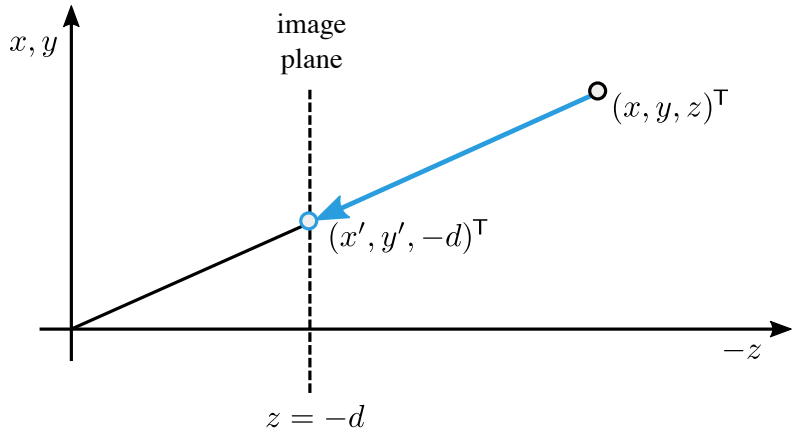
Don’t put $z$ into the matrix!

# Homogeneous Coordinates

- Use homogeneous coordinates $(x, y, z, w)$!
  - Vectors are represented by $(x, y, z, 0)^\mathsf{T}$
  - Points are represented by $(wx, wy, wz, w)^\mathsf{T}$ for any $w \neq 0$
  - Divide a point $(wx, wy, wz, w)^\mathsf{T}$ by $w$ to get its canonical representation $(x, y, z, 1)^\mathsf{T}$ (this process is called *"homogenization"*)

# Generic Perspective Projection

- Standard projection
  - Center of projection: $(0, 0, 0)$
  - Image plane at $z = -d$



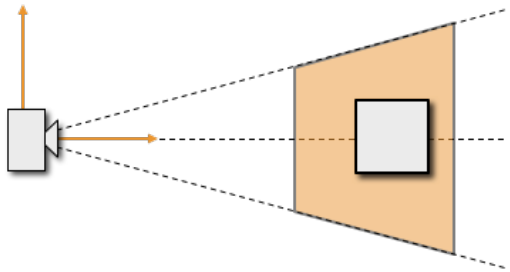

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \end{pmatrix} \quad \longleftrightarrow \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ -z/d \end{pmatrix} \cong \begin{pmatrix} x \frac{-d}{z} \\ y \frac{-d}{z} \\ -d \\ 1 \end{pmatrix}$$

# Homogeneous Coordinates

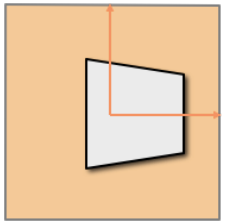
- Use homogeneous coordinates $(x, y, z, w)$!
  - Vectors are represented by $(x, y, z, 0)^\mathsf{T}$
  - Points are represented by $(wx, wy, wz, w)^\mathsf{T}$ for any $w \neq 0$
- Divide a point $(wx, wy, wz, w)^\mathsf{T}$ by $w$ to get its canonical representation $(x, y, z, 1)^\mathsf{T}$
  - Defer this homogenization until the very last step
- We can now use $4 \times 4$ matrices for
  - linear transformations (scaling, rotation, …)
  - affine transformations (linear map + translation)
  - projective transformations (affine map + projection)

# OpenGL Perspective Projection

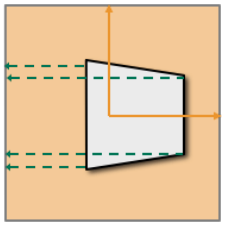
- Specify viewing volume by
  - near and far
  - left and right (or opening angle in x)
  - bottom and top (or opening angle in y)



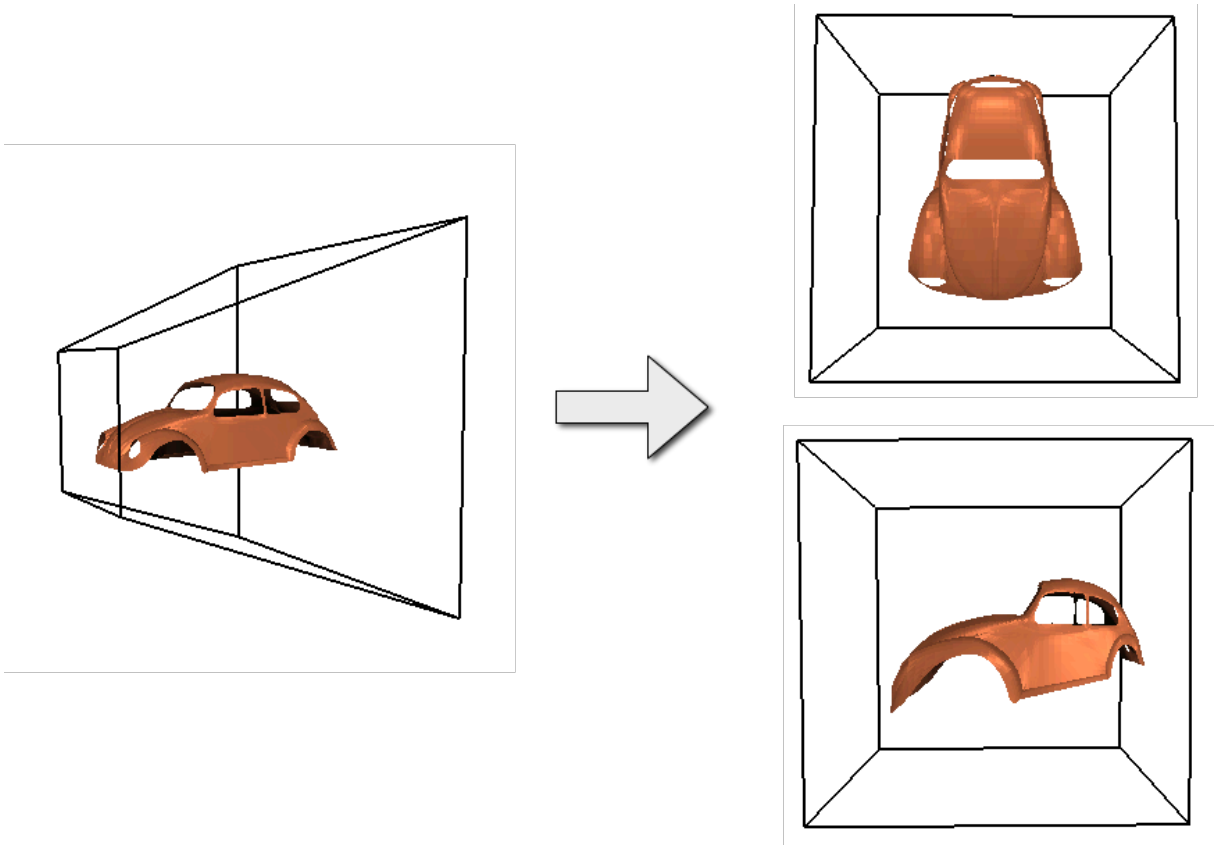
- Transform this *frustum* to unit cube $[-1,1]^3$
  - so-called *frustum mapping*



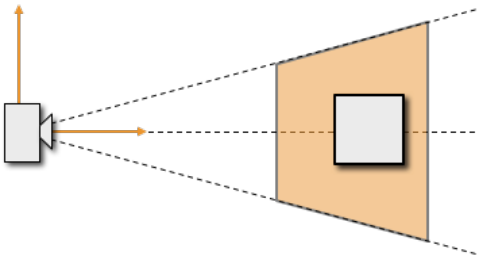
- Then perform parallel projection onto xy-plane and viewport transformation

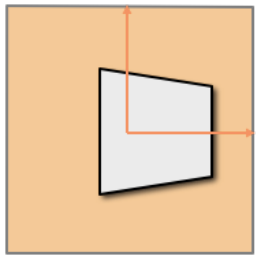


# Frustum Mapping



# Frustum Mapping





- How to transform $(x, y, z)$?

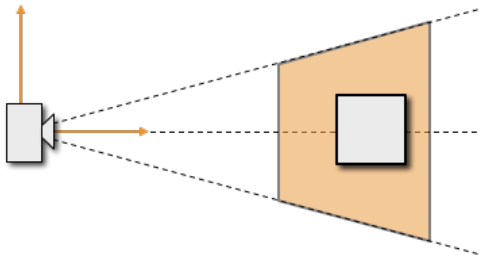
$$x \mapsto \left( x \frac{n}{-z} - \frac{l+r}{2} \right) \cdot \frac{2}{r-l}$$

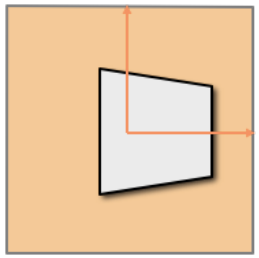
$$y \mapsto \left( y \frac{n}{-z} - \frac{b+t}{2} \right) \cdot \frac{2}{t-b}$$

- Matrix representation:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ ? & ? & ? & 0 \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

# Frustum Mapping





- Transform

$$z \mapsto \frac{a \cdot z + b}{z}$$

such that $-n \mapsto -1$ and $-f \mapsto +1$.

- This leads to

$$a = -\frac{n+f}{f-n}, \quad b = -\frac{2nf}{f-n}$$

- Matrix representation:

$$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{l+r}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{b+t}{t-b} & 0 \\ 0 & 0 & -\frac{n+f}{f-n} & -\frac{2nf}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

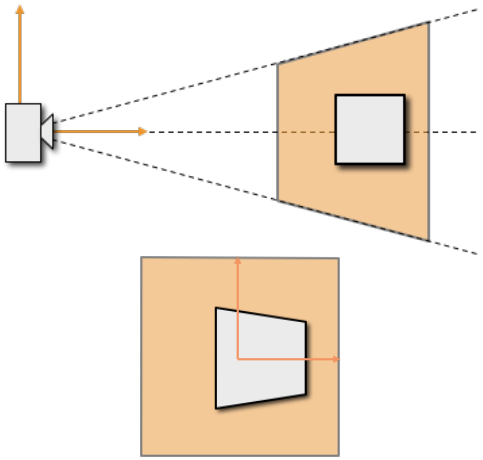
# Frustum Mapping

```
# define frustum matrix
var('l,r,b,t,n,f')
M = matrix([[2*n/(r-l), 0, (l+r)/(r-l), 0], [0, 2*n/(t-b), (b+t)/(t-
show(M)

# function that divides a 4D vector by its w-component
def homogenize(v):
    return vector([ v[0]/v[3],
                    v[1]/v[3],
                    v[2]/v[3],
                    v[3]/v[3] ])

# function that symbolically simplifies of each component of 4D vect
def simplify(v):
    return vector([ v[0].full_simplify(),
                    v[1].full_simplify(),
                    v[2].full_simplify(),
                    v[3].full_simplify() ])

# Now let's test some corners of the frustum
show( simplify( homogenize( M * vector([ l, b, -n, 1]) ) ) )
```



# Viewport Transform

# Viewport Mapping

- Simple scaling of normalized device coordinates

$$[-1,1] \times [-1,1] \times [-1,1]$$

  to window pixel coordinates

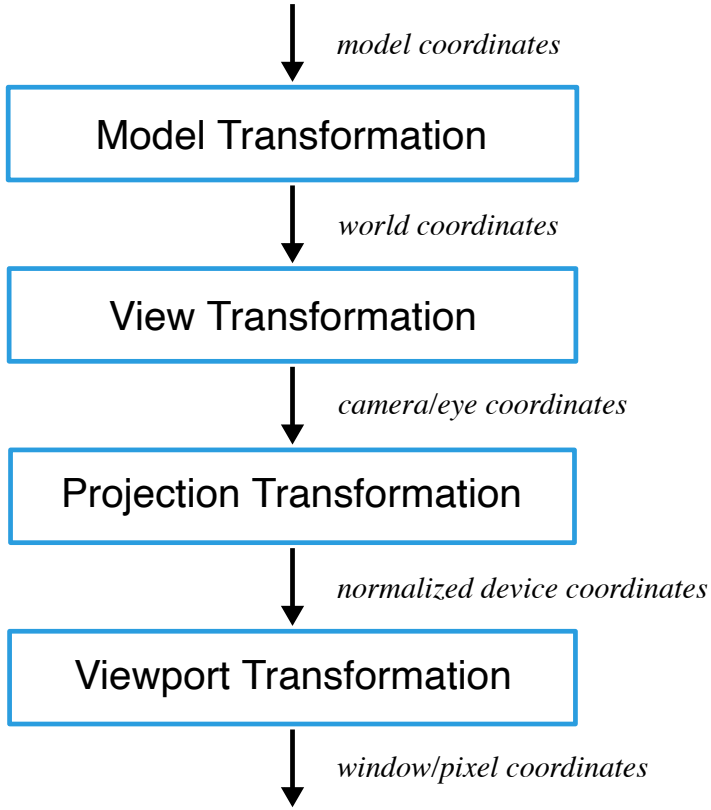
$$[l, l+w] \times [b, b+h] \times [0,1]$$

- Matrix representation

$$\begin{pmatrix} \frac{w}{2} & 0 & 0 & \frac{w}{2}+l \\ 0 & \frac{h}{2} & 0 & \frac{h}{2}+b \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

# Transformation Pipeline

- **View Transformation**
  - setup extrinsic camera parameters: position and orientation
- **Projection Transformation**
  - setup intrinsic camera parameters: opening angle and depth range
- **Viewport Transformation**
  - setup image parameters/resolution: width and height



# Try it yourself!

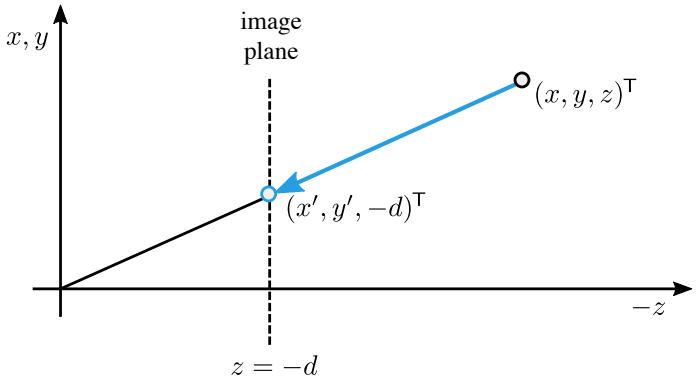


*Model transformation* $\mathbf{M} = \mathbf{R}_y(\cdot)\,\mathbf{T}_z(\cdot)\,\mathbf{S}_x(\cdot)$

# Quiz: Projections

Which matrix computes this projection?

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \begin{pmatrix} x \cdot \frac{-d}{z} \\ y \cdot \frac{-d}{z} \\ -d \end{pmatrix}$$




A: $\begin{pmatrix} \frac{-d}{z} & 0 & 0 & 0 \\ 0 & \frac{-d}{z} & 0 & 0 \\ 0 & 0 & \frac{-d}{z} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

B: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{-1}{d} & 0 \end{pmatrix}$

C: $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{-z}{d} \end{pmatrix}$

# Ponzo Illusion

Source

# Ponzo Illusion

Source

# Literature

- Marschner & Shirley: *Fundamentals of Computer Graphics*, 5th Edition, AK Peters, 2021.
  - Chapter 8