UNIVERSITY OF DERBY

# 5CM507 Graphics

## Lecture 04 Viewing and Projection

Dr Youbing Zhao

October 12, 2025

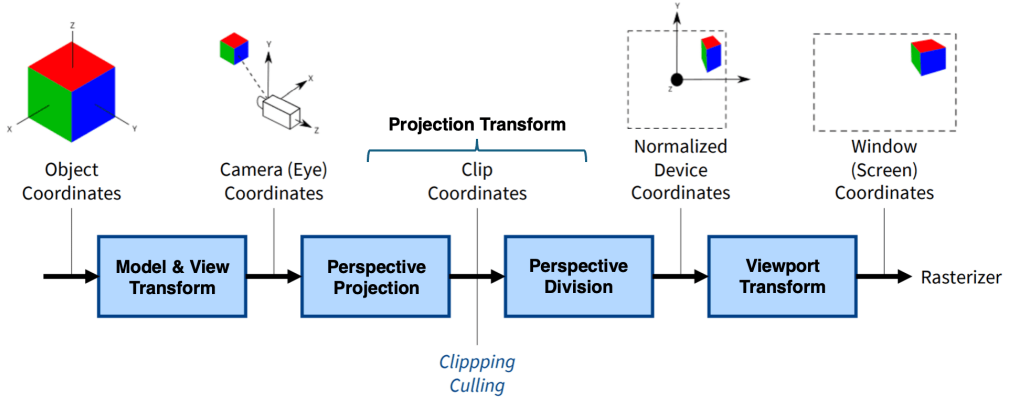
# Last Week

- Position a Model: Model Transform
- Build a scene: Hierarchical Transformation
- Transformation of frames
- Position and orient the Camera: View Transform
- The Model-View Matrix

# Contents

- Camera control
- Orthographic Projection
- Perspective Projection

# Modelling, Viewing and Projection




[Youngdo Lee]

# The View Matrix - the LookAt function

The LookAt function specifies the major axis vecor of the camera space in the world space.

$${}^{W}\vec{X} = \begin{pmatrix} {}^{A}X_B^x & {}^{A}X_B^y & {}^{A}X_B^z \end{pmatrix}^T, {}^{W}\vec{Y} = \begin{pmatrix} {}^{A}Y_B^x & {}^{A}Y_B^y & {}^{A}Y_B^z \end{pmatrix}^T, {}^{W}\vec{Z} = \begin{pmatrix} {}^{A}Z_B^x & {}^{A}Z_B^y & {}^{A}Z_B^z \end{pmatrix}^T$$

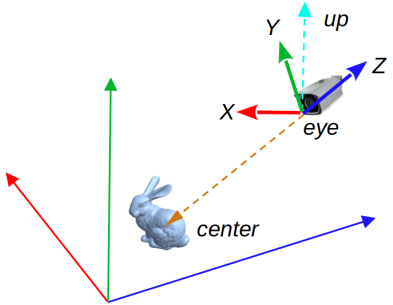
```
glm::dmat4 glm::lookAt(glm::dvec3 eye, glm::dvec3 center, glm::dvec3 up);
```

which defines the view matrix V:

$$\mathbf{V} = {}^{c}_{w}\mathbf{M} = {}^{w}_{c}\mathbf{M}^{-1} = \left({}^{w}_{c}\mathbf{T}\,{}^{w}_{c}\mathbf{R}\right)^{-1} = {}^{w}_{c}\mathbf{R}^{-1}\mathbf{T}(-\mathbf{e_x}, -\mathbf{e_y}, -\mathbf{e_z})$$

$$= \begin{pmatrix} {}^{A}X_B^x & {}^{A}X_B^y & {}^{A}X_B^z & 0 \\ {}^{A}Y_B^x & {}^{A}Y_B^y & {}^{A}Y_B^z & 0 \\ {}^{A}Z_B^x & {}^{A}Z_B^y & {}^{A}Z_B^z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

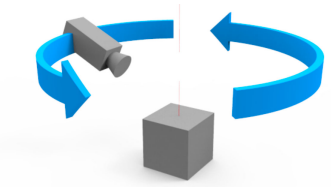
$$= \begin{pmatrix} {}^{w}X_c^x & {}^{w}X_c^y & {}^{w}X_c^z & -\vec{e} \cdot {}^{w}\vec{X}_c \\ {}^{w}Y_c^x & {}^{w}Y_c^y & {}^{w}Y_c^z & -\vec{e} \cdot {}^{w}\vec{Y}_c \\ {}^{w}Z_c^x & {}^{w}Z_c^y & {}^{w}Z_c^z & -\vec{e} \cdot {}^{w}\vec{Z}_c \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



# Basic Camera Control

## Position and orientation

- Position and orient : `glm::lookAt()`
  - You can achieve the bullet time effect !
- Translation - change the position only
  - Easy way: use `glm::lookAt()` again
    - update the position
    - update the center based on the original orientation
  - But how about using translation matrices ?



Camera orbit

Camera dolly

# Basic Camera Control

## Translation

$\mathbf{H} = \mathbf{PVM}$

- Hint: a camera is move of $T(t_x, t_y, t_z)$ is equiv. to the scene/object's with the opposite move of $T(-t_x, -t_y, -t_z)$
- Move with regard to the world frame ${}^{w}T(t_x, t_y, t_z)\ \mathbf{C}' = \mathbf{PV}\,{}^{w}T(-t_x, -t_y, -t_z)\mathbf{M}$

$$\mathbf{V}' = \mathbf{V}\,{}^{w}T(-t_x, -t_y, -t_z) \begin{bmatrix} {}^{w}X_c^x & {}^{w}X_c^y & {}^{w}X_c^z & -(e+\vec{t})\cdot {}^{w}\vec{X}_c \\ {}^{w}Y_c^x & {}^{w}Y_c^y & {}^{w}Y_c^z & -(e+\vec{t})\cdot {}^{w}\vec{Y}_c \\ {}^{w}Z_c^x & {}^{w}Z_c^y & {}^{w}Z_c^z & -(e+\vec{t})\cdot {}^{w}\vec{Z}_c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Move with regard to the camera frame ${}^{c}T(t_x, t_y, t_z)\ \mathbf{C}' = \mathbf{P}\,{}^{c}T(-t_x, -t_y, -t_z)\mathbf{VM}$

$$\mathbf{V}' = {}^{c}T(-t_x, -t_y, -t_z)\mathbf{V} \begin{bmatrix} {}^{w}X_c^x & {}^{w}X_c^y & {}^{w}X_c^z & -e\cdot {}^{w}\vec{X}_c - t_x \\ {}^{w}Y_c^x & {}^{w}Y_c^y & {}^{w}Y_c^z & -e\cdot {}^{w}\vec{Y}_c - t_y \\ {}^{w}Z_c^x & {}^{w}Z_c^y & {}^{w}Z_c^z & -e\cdot {}^{w}\vec{Z}_c - t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Basic Camera Control

## Pan, Tilt, Zoom

- Pan : camera frame rotates $\theta_y$ about its $Y$ axis
  - equiv. to the scene rotates $-\theta_y$ about the camera $Y$ axis
- Tilt : camera frame rotates $\theta_x$ about its $X$ axis
  - equiv. to the scene rotates $-\theta_x$ about the camera $X$ axis
- Zoom : Dolly zoom: Camera moving along its Z axis; Optical zoom: change FoV of the projection


Example:

- A camera pan: $\mathbf{V}' = \mathbf{R}_{pan}(-\theta_y)\mathbf{V}$
- A camera pan followed by a tilt: $\mathbf{V}' = \mathbf{R}_{tilt}(-\theta_x)\mathbf{R}_{pan}(-\theta_y)\mathbf{V}$
- A camera zoom in by $d_{in} < 0$, with a pan followed by a tilt, then zoom out by $d_{out} > 0$: $\mathbf{V}' = \mathbf{T}_{out}(-T_{out})\mathbf{R}_{tilt}(-\theta_x)\mathbf{R}_{pan}(-\theta_y)\mathbf{T}_{in}(-d_{in})\mathbf{V}$

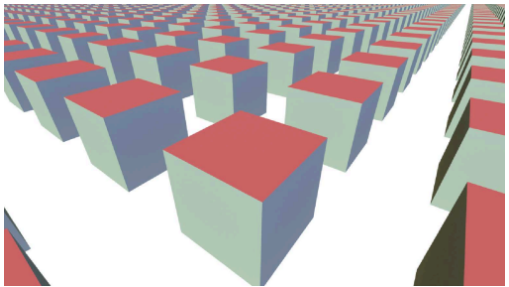
Matrix order : right to left, negative rotation angles or translation distance.

# Projections

# Perspective vs Orthographic Projections

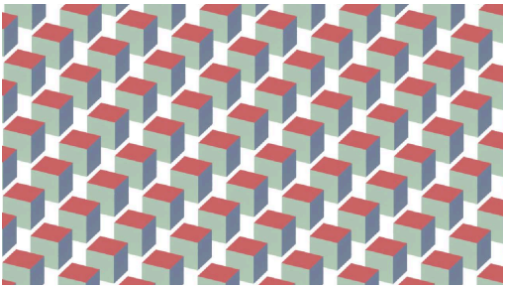
## Perspective

- Does not preserve parallelism, proportions
- Provides a sense of depth
- 1 to 3 vanishing points

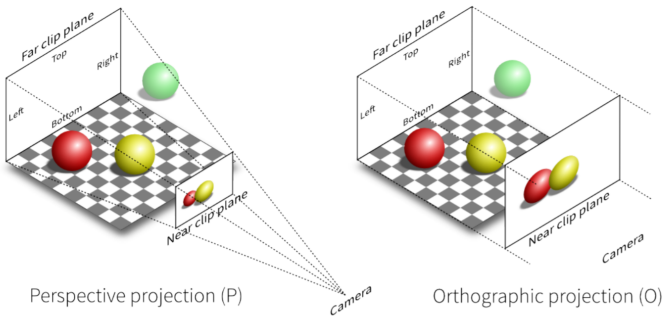


## Orthographic

- A view preserving parallelism, proportions
- Lack a sense of depth

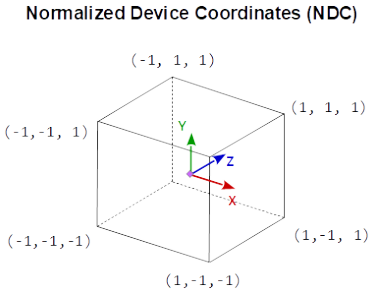


# View volume and Normalised Device Coordinates (NDC)




Perspective projection (P)

Orthographic projection (O)



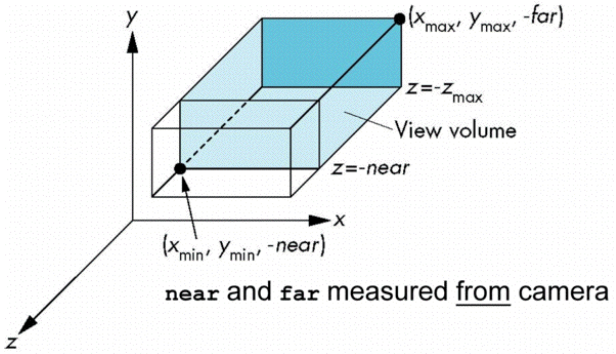

OpenGL NDC

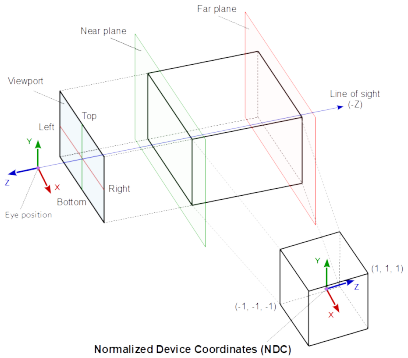
# Orthographic Projection

# Orthographic Projection

OpenGL orthographic projection function:

```
glm::mat4 glm::ortho(left, right, bottom, top, zNear, zFar);
(left < right, bottom < top, 0 < near < far)
```



# Orthographic Projection Matrix

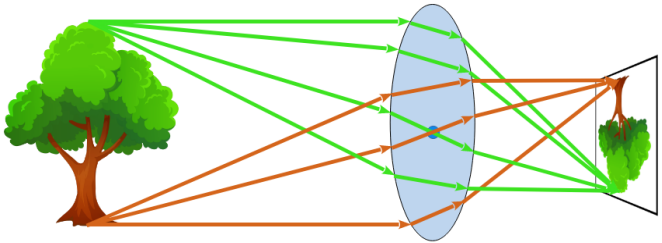
The orthographic projection matrix is a translation to the centre of the view volume, following by scaling.

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

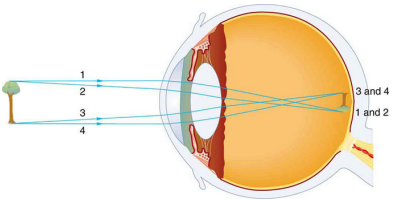
$$= \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & -\frac{2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

# Perspective Projection

# A camera simulates an eye



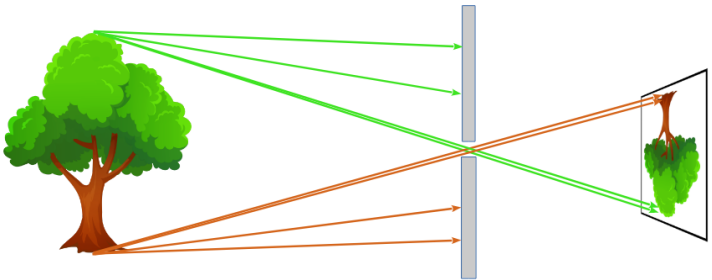
Imaging with lens



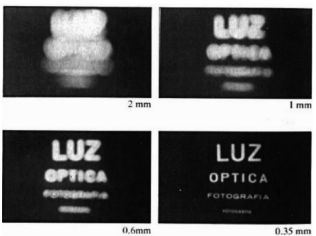

The human vision

# The Pinhole Camera

- The pinhole effect : described by a great Chinese thinker Mo Di (Mozi) dating back to around 400BC.
- The pinhole camera model
  - computes the size of image projections
  - unable to simulate depth of field effects

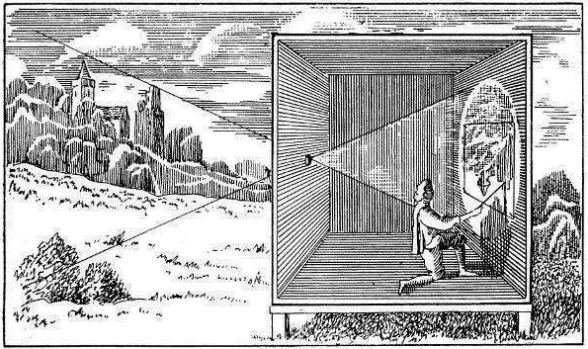


Imaging with a pinhole




Imaging with different pinhole sizes

# Camera Obscura (dark chamber)



Engraving of a "portable" camera obscura in Athanasius Kircher's Ars Magna Lucis Et Umbrae (1645)

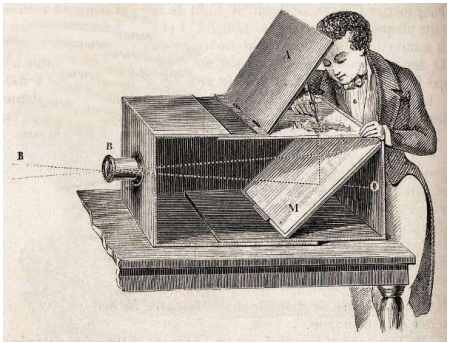
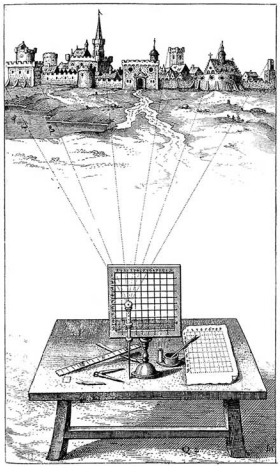
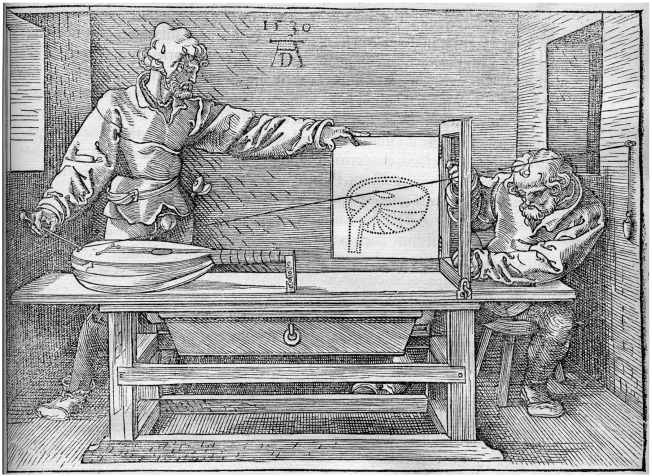



Illustration from Adolphe Ganot, An Elementary Treatise on Physics, 1882

# Perspective Drawing Machines



Robert Fludd's sighting grid (1617)



Albrecht Duerer: "The draughtsman of the lute" (1525)

# Perspective Drawing in Renaissance

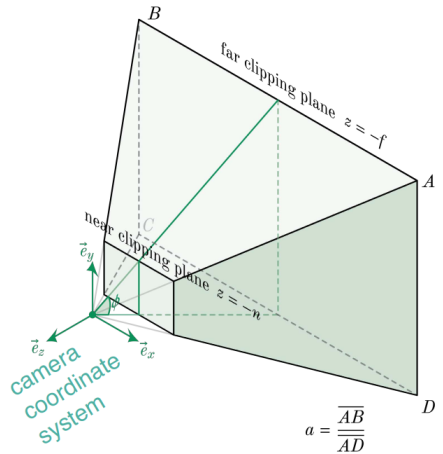
Annunciation, Leonardo da Vinci, c.1472-1476

# Perspective View Frustum

## Parameters

In graphics, a view frustum is the 3D region that is visible on the screen.

- Parameters for a viewing frustum:
  - field of view (FoV): angle $\phi$
  - aspect ratio $a = \frac{width}{height}$
  - near clipping plan distance $n > 0$
  - far clipping plan distance $f > n$
- Camera zooming: adjusting the field of view angle $\phi$
  - Wide angle zoom: wide FoV with a large $\phi$
  - Telephoto zoom: narrow FoV with a small $\phi$




[UCSD CSE167, Chern]

# Perspective can be done simply as well ?

Let's have a try as the Renaissance draft man !

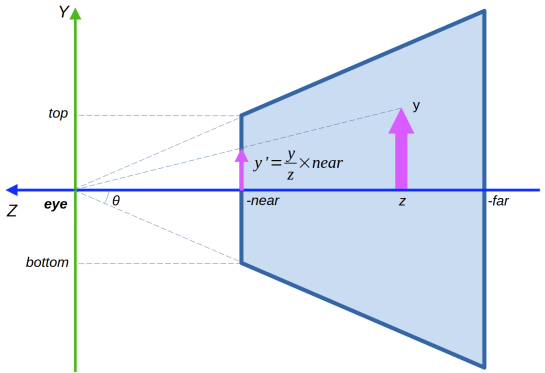
- Bring me my thread of simulated light.
- Bring me my target of sight.

$$\begin{cases} x' = \frac{x}{z}n \\ y' = \frac{y}{z}n \end{cases}$$ are almost right

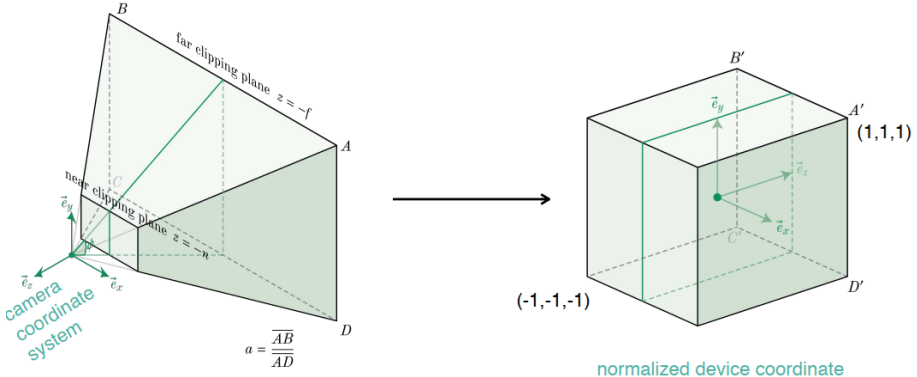
Design of $z$ Transform: no need in drawing but needed in CG for hidden surface removal, clipping and interpolation (we are projecting vertices).

Let' try $z' = \frac{z-n}{f-n}$

- OK for vertex depth but not preserve lines
- Linearity is needed for clipping and interpolation
- We like to see a factor of $\frac{1}{z}$ in all of $x', y', z'$



# Parameters for Perspective Projection



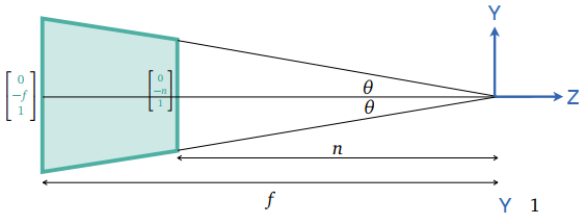

A projection matrix transforms a view frustum to a canonical cube

```
// Symmetric Viewing Volume
glm::mat4 perspective(double fovy, double aspect, double zNear, double zFar);
// General Viewing Volume
glm::mat4 glm::frustum(double left, double right, double bottom, double top, double zNear, double zFar);
```

# Symmetric Perspective Projection Matrix

A translation along $Z$, followed by a $z$-based scaling.

$$\mathbf{P}_{sym}V = \begin{bmatrix} \frac{1}{a\tan\theta} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l}x \\ \frac{2n}{t-b}y \\ -\frac{f+n}{f-n}z - \frac{2fn}{f-n} \\ -z \end{bmatrix}$$



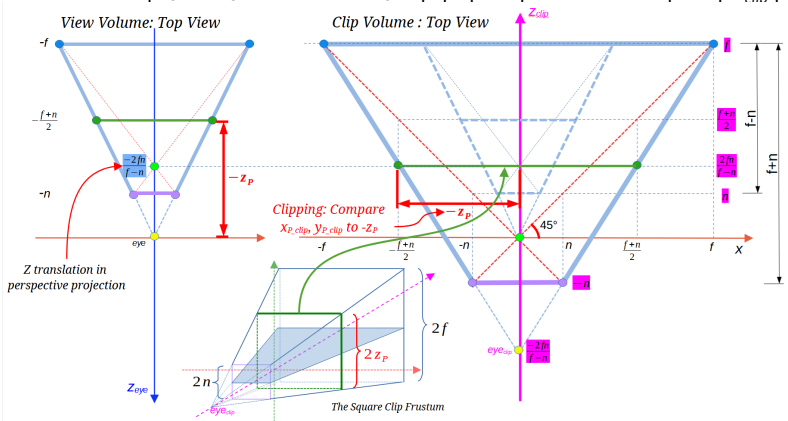

$$\theta = \frac{\phi}{2}$$

$$\tan\theta = \frac{top-bottom}{2near} \qquad a\tan\theta = \frac{right-left}{top-bottom}\frac{top-bottom}{2near} = \frac{right-left}{2near}$$

# Projection Step 1: The Clip Volume

The perspective matrix transforms a point $v(x, y, z, 1)$ to the clip space, before normalised by $w_{clip} = -z$.

$$v_{clip} = \mathbf{P}v = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l}x \\ \frac{2n}{t-b}y \\ -\frac{f+n}{f-n}z - \frac{2fn}{f-n} \\ -z \end{bmatrix} = \begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix}$$

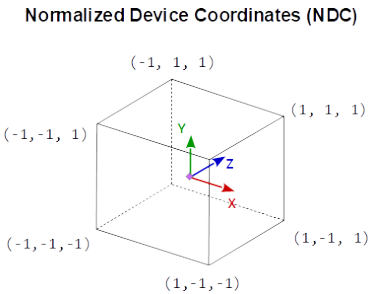


# Projection Step 2: Perspective Divide (Normalisation)

- Clip volume: every cross section parallel to the image plane shares a constant $w_{clip} = -z$ and spans $[-w_{clip}, w_{clip}] = [-z, z]$ in both $x$ and $y$, speeding up clipping.
- Divide $x, y, z$ by the weight component $-z$ results in the normalised coordinate system (NCS).

$$v_{norm} = \begin{bmatrix} x_{norm} \\ y_{norm} \\ z_{norm} \\ 1 \end{bmatrix} = \begin{bmatrix} x_{clip}/(-z) \\ y_{clip}/(-z) \\ z_{clip}/(-z) \\ (-z)/(-z) \end{bmatrix} = \begin{bmatrix} -\frac{2n}{r-l}\frac{x}{z} \\ -\frac{2n}{t-b}\frac{y}{z} \\ \frac{f+n}{f-n} + \frac{2fn}{(f-n)}\frac{1}{z} \\ 1 \end{bmatrix}$$

Vanishing point: $(-\frac{r+l}{r-l}, -\frac{t+b}{t-b})$, $(0, 0)$ for symmetric projection




OpenGL NDC

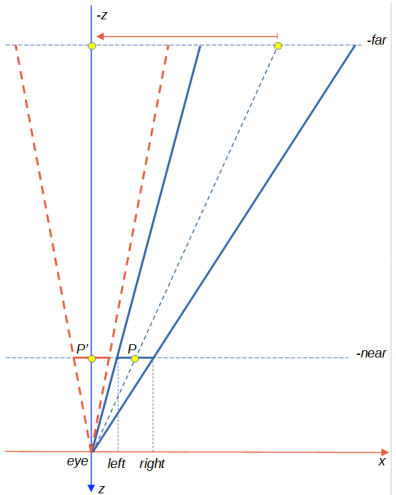
# The General Perspective Projection Matrix

- Oblique viewing volumes
  - Not common for single display games
  - Use a shear transform of $X$ and $Y$ to convert into a symmetric volume.

$$\mathbf{P} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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

$$V_{norm} = \begin{bmatrix} -\frac{2n}{r-l}\frac{x}{z} - \frac{r+l}{r-l} \\ -\frac{2n}{t-b}\frac{y}{z} - \frac{t+b}{t-b} \\ \frac{f+n}{f-n} + \frac{2fn}{(f-n)}\frac{1}{z} \\ 1 \end{bmatrix}$$



# Distortions resulted from $\frac{1}{z}$

- Marginal distortion ( large $x, y$ offsets with a small $-z$ leading to large $\frac{x}{-z}$)
- Close-up distortion: (with a smaller $n$, $\frac{-z}{n} \to \infty$, $\frac{n}{-z} \to 0$, projections converge to the vanishing point)
- Vertical distortion: buildings appears leaning if tilt the camera up (vanishing points caused by $\frac{1}{-z}$)

Vertical distortion

(a) A wide-angle photo with distortions on subjects' faces. (b) Distortion-free photo by our method.

Marginal distortion [Google]

85mm @ 200cm 35mm @ 85cm 16mm @ 40cm 12mm @ 30cm 8mm @ 20cm

Close-up(Wide angle) distortion [Daniel]

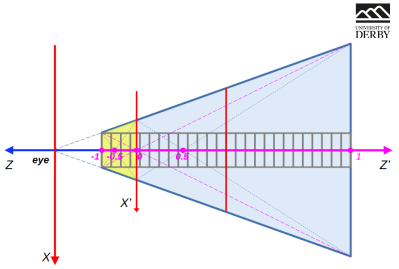
# Transformed $z$ is not uniform

$$z' = \frac{f+n}{f-n} + \frac{2fn}{f-n}\frac{1}{z}$$ is not linear

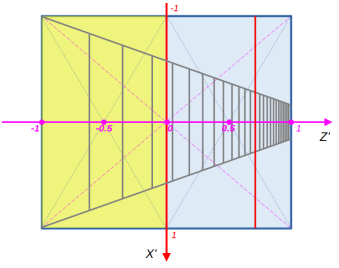
- Closer to the camera => greater numerical precision
- Most points are squeezed close to $z' = 1$
- May cause z-fighting when precision is lacking
- Points in the middle ranges are pushed much backwards away from the camera.

Choose near and far planes that tightly fit the objects to make full use of the $[-1, 1]$ range.

- Common mistake: set near = 0, far = infinity
- Near plane: in decent distance from the camera




The $z'$ quartiles before projection




$Z'$ distribution after projection

# FoV Setting

- Desktop / VR preview 45°–60° Feels natural; similar to a 50 mm camera lens
- First-person games 60°–90° Wider for situational awareness
- VR headset (per eye) 90°–110° Matches immersive field

# Summary

- Camera control
- Normalised Device Coordinates (NDC)
- Orthographic Projection Matrix
- Perspective Projection Matrices

Questions?

You are welcome to take the poll !