Viewing

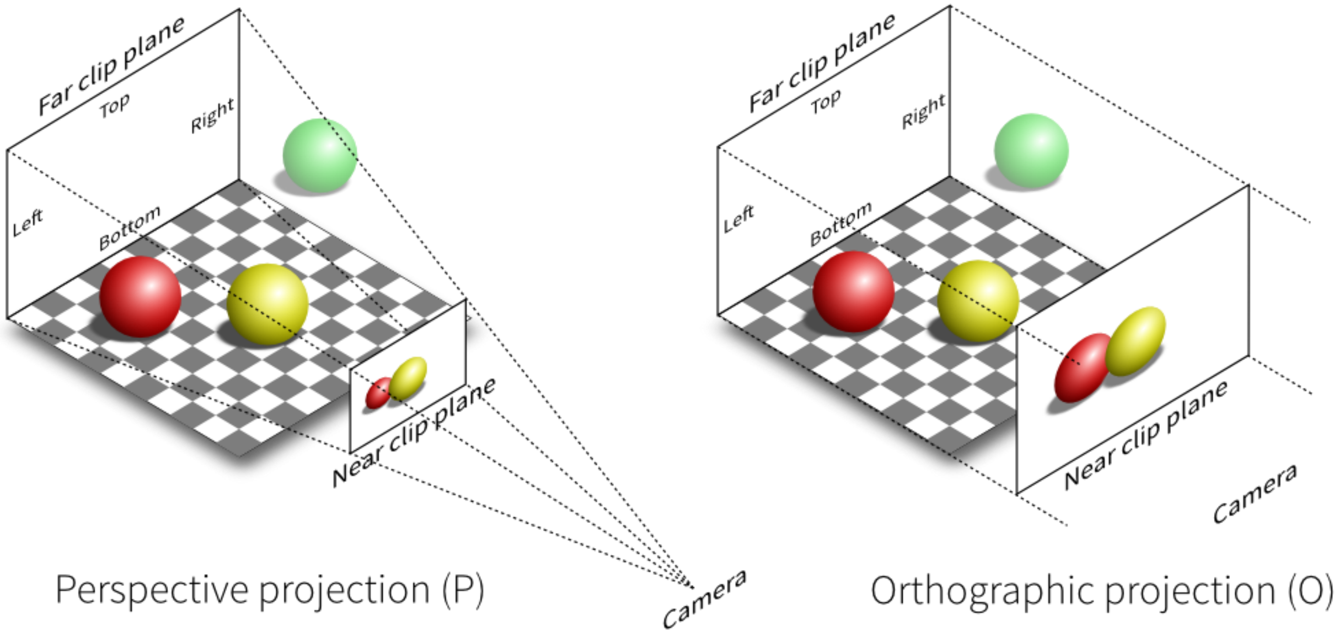
CSU44052 Computer Graphics

Binh-Son Hua

Credits: Some slides taken from Carol O'Sullivan and Robb T. Koether

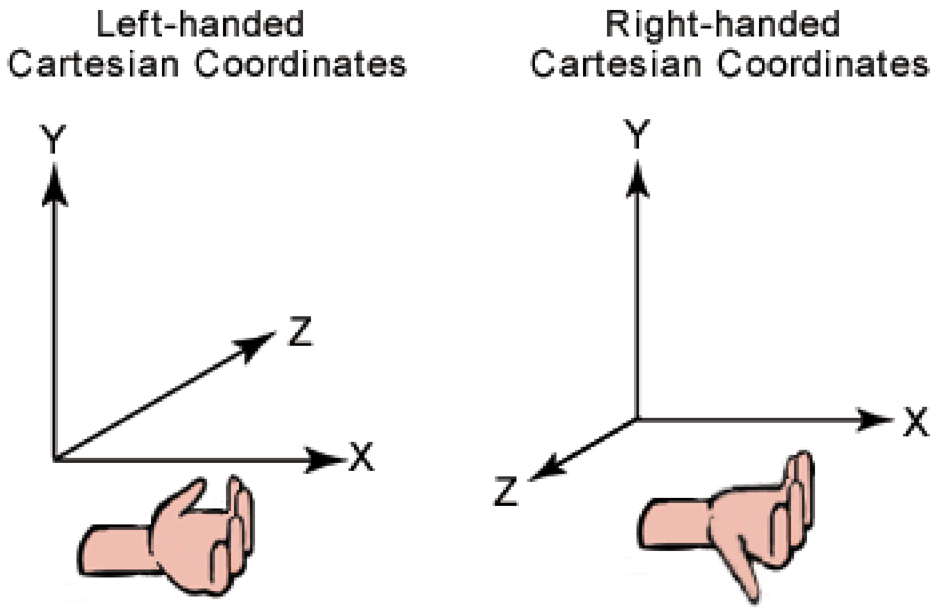
Overview

- Viewing
  - Model transform
  - View transform
  - Perspective projection
  - Viewport



Coordinate System

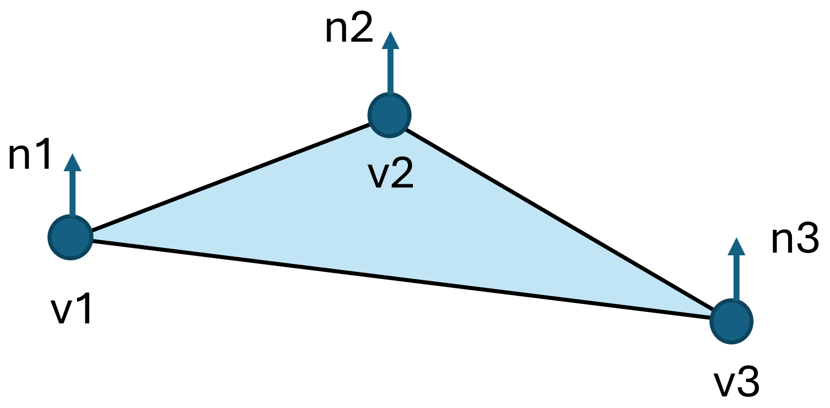
- Left-handed (DirectX, Unity, Unreal)
- Right-handed (OpenGL)
- Assume right-handed coordinate system



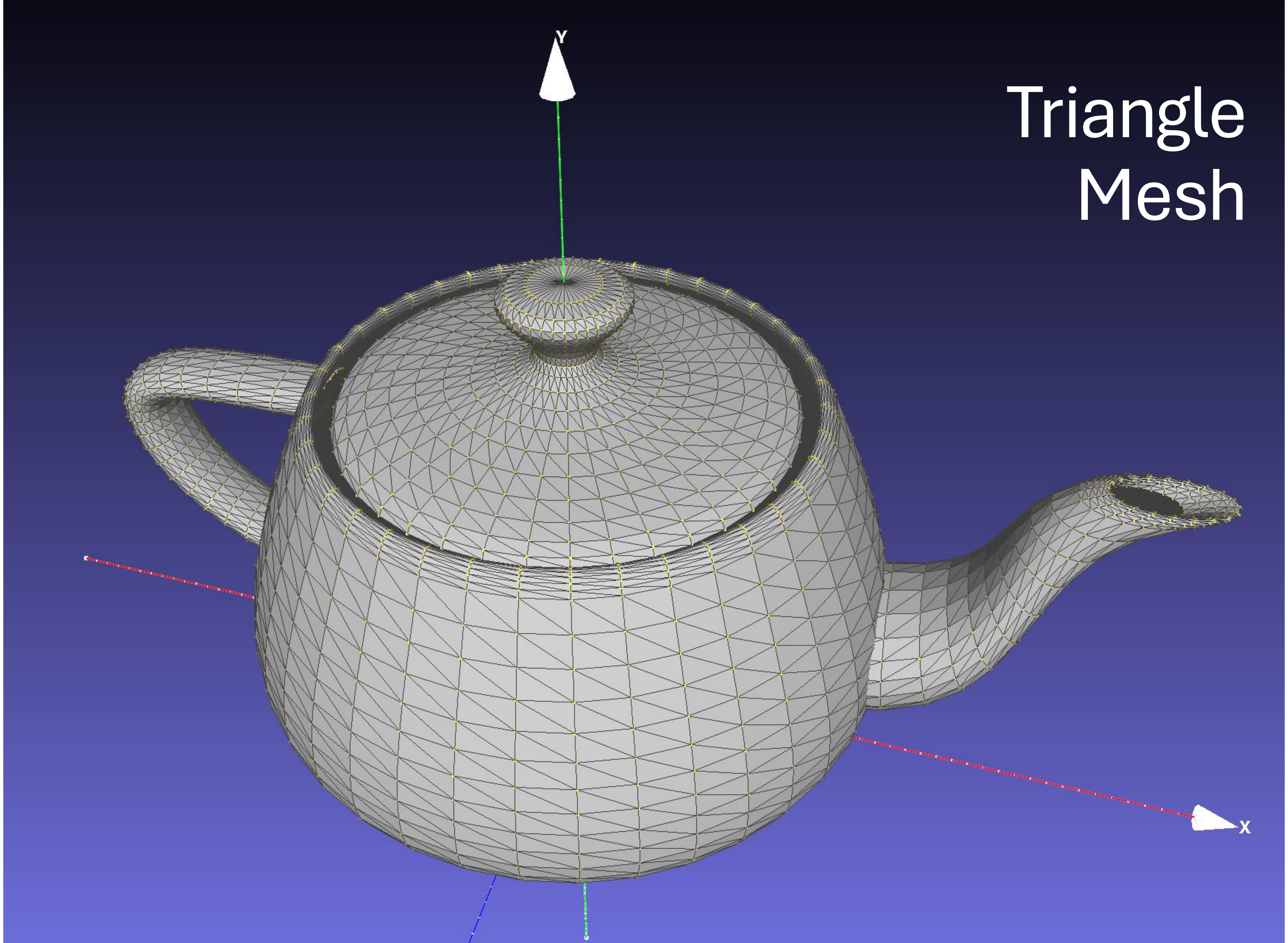
https://learn.microsoft.com/en-us/windows/win32/direct3d9/coordinate-

Primitives

- Vertex is represented by a vector v = (x, y, z).
- Normal vector is represented by a unit vector n.
- Triangle is represented by three vertices v1, v2, v3.
- Attributes of a vertex: normal vector, RGB color, texture coordinates.

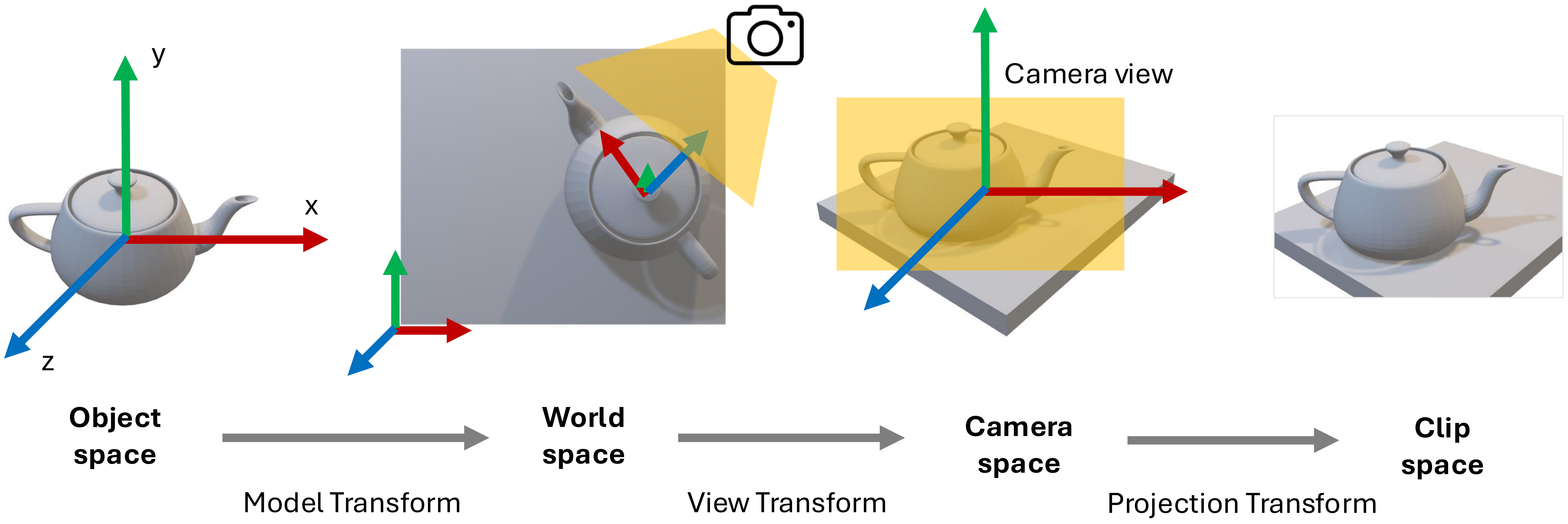


Triangle Mesh



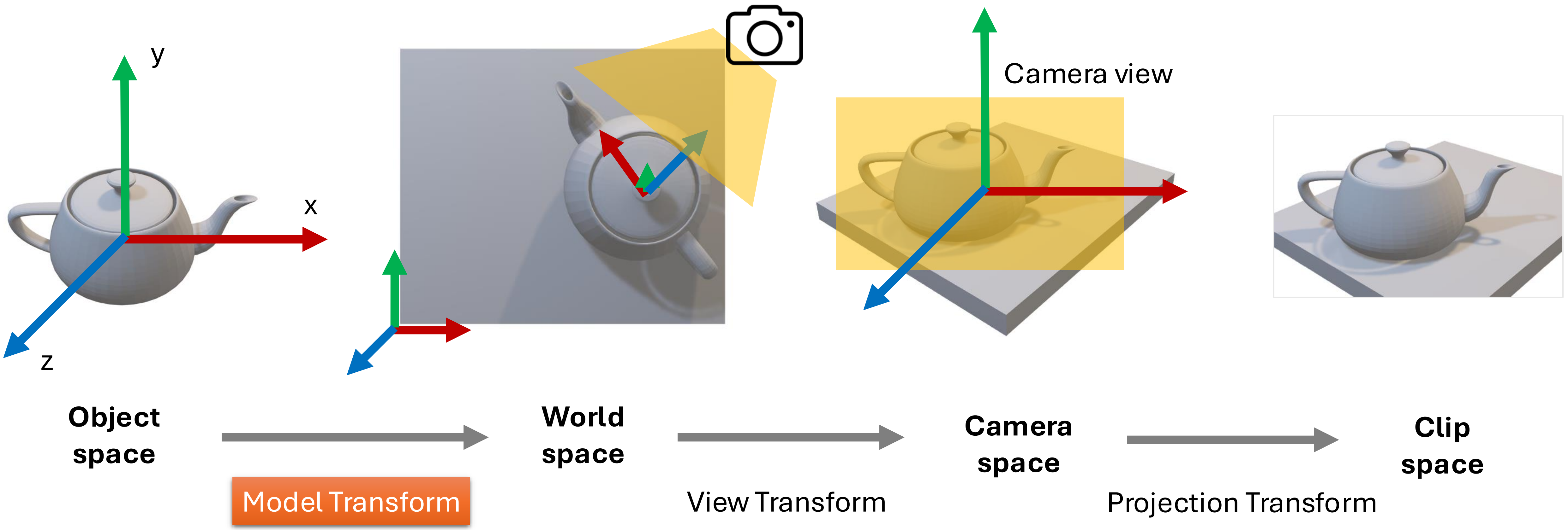
Viewing Transformation 1

- Per-vertex coordinate transformation



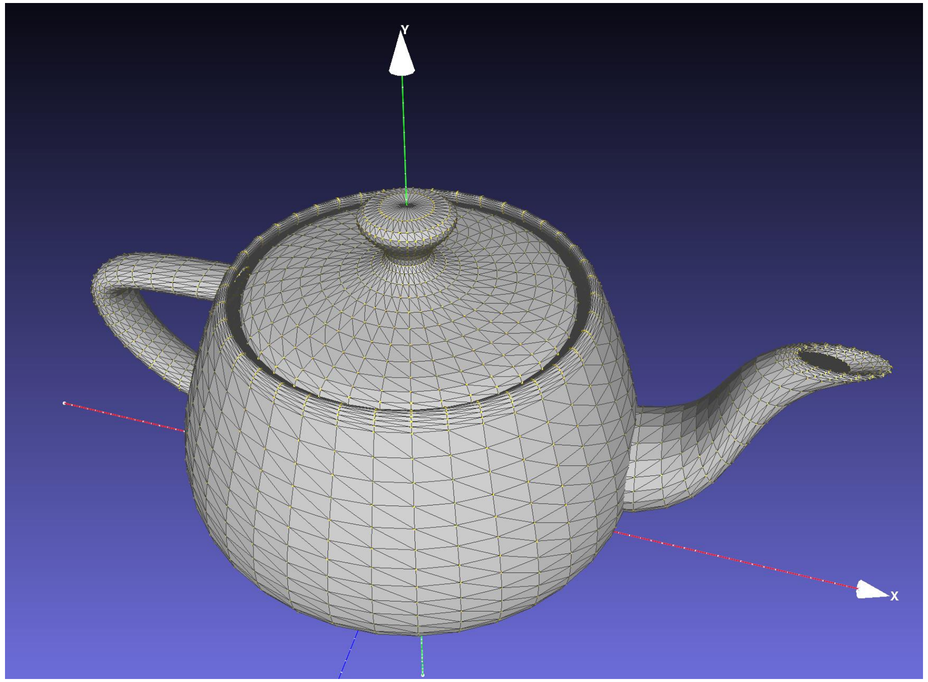
Viewing Transformation 1

- Per-vertex coordinate transformation



Model Transform

- When you create a triangle or load a mesh from a file, e.g., teapot
- The teapot has a (0,0,0) origin, local to its particular mesh
- To position the teapot in a virtual world, we can translate, rotate, scale it
- Multiply its points with a model matrix ("model-to-world matrix")



Model Transform

Scale

$$S(s) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Translation

$$T(d) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

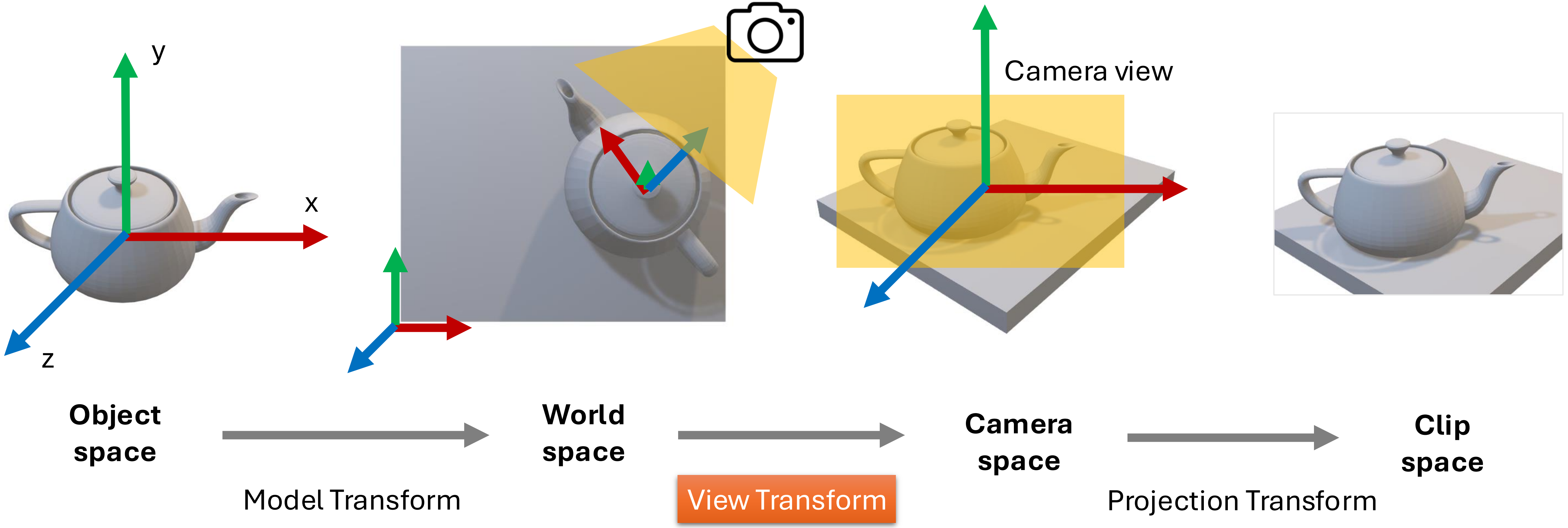
$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transformation 1

- Per-vertex coordinate transformation



View Transform

OpenGL view transform

- Eye position

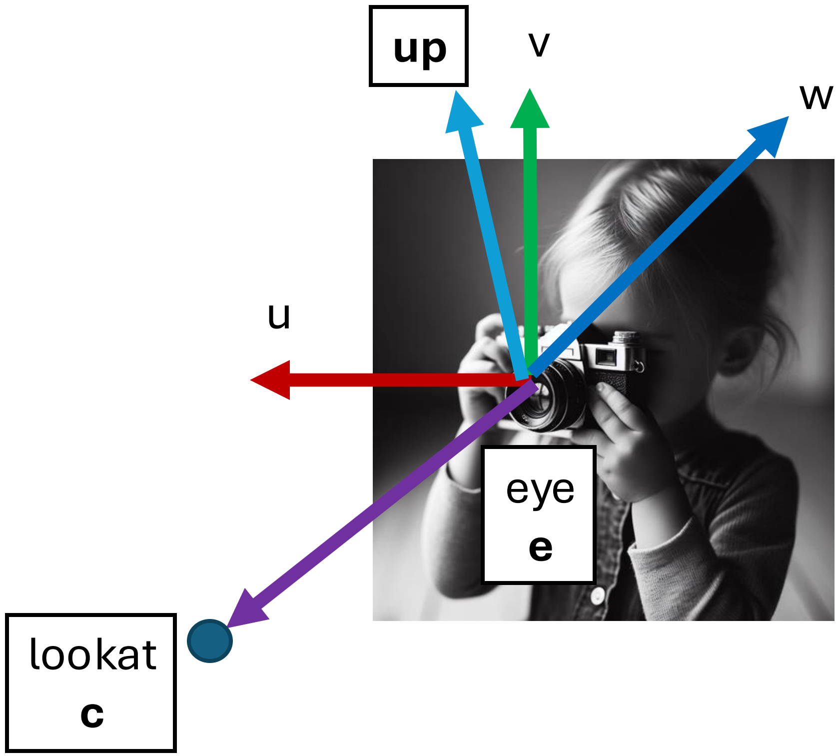
$$\mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$$

- Lookat position

$$\mathbf{c} = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix}$$

- Up vector

$$\mathbf{up} = \begin{bmatrix} up_x \\ up_y \\ up_z \end{bmatrix}$$

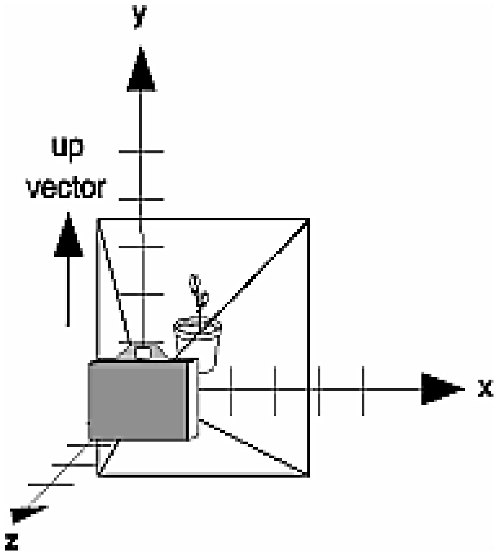


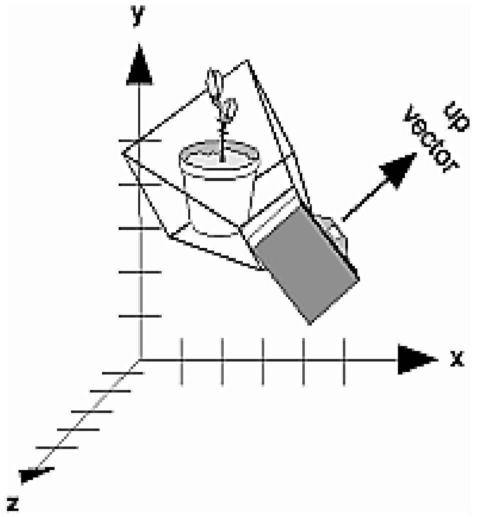
Note: lookat direction is the negative z-axis.

Image created by DALL-E 3

Up Vector

- Perpendicular to the line of sight
- Must not be parallel
- Tells which direction is up (i.e. the direction from the bottom to the top of the viewing volume)





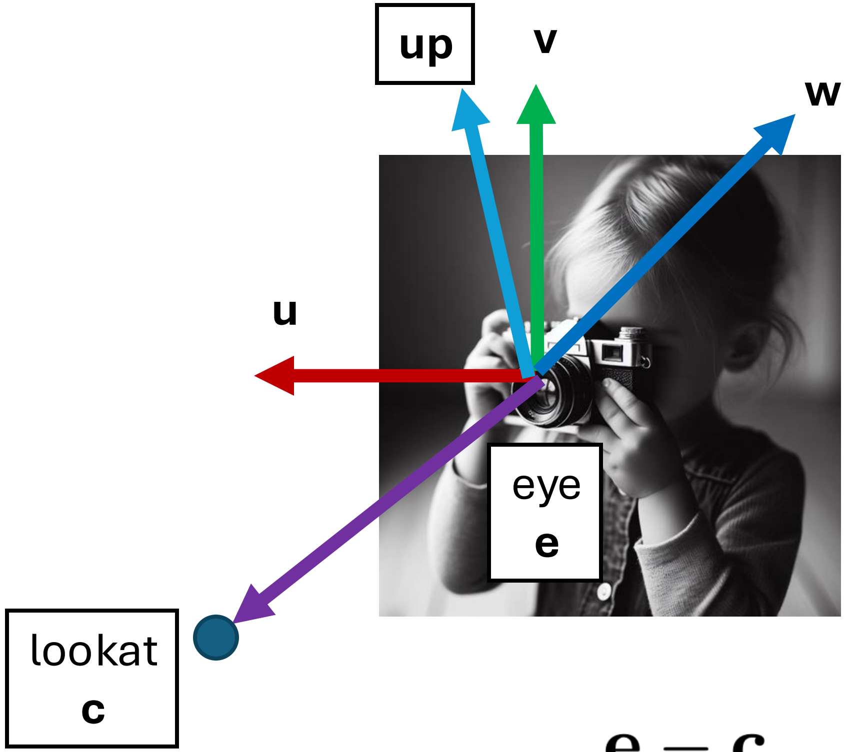
World-to-Camera Transform

$$\begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & 0 \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & 0 \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -\mathbf{e}_x \\ 0 & 1 & 0 & -\mathbf{e}_y \\ 0 & 0 & 1 & -\mathbf{e}_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation　　　　Translation

View matrix

$$\begin{bmatrix} \mathbf{u}_x & \mathbf{u}_y & \mathbf{u}_z & -\mathbf{u}^\top\mathbf{e} \\ \mathbf{v}_x & \mathbf{v}_y & \mathbf{v}_z & -\mathbf{v}^\top\mathbf{e} \\ \mathbf{w}_x & \mathbf{w}_y & \mathbf{w}_z & -\mathbf{w}^\top\mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



$$\mathbf{w} = \frac{\mathbf{e} - \mathbf{c}}{\|\mathbf{e} - \mathbf{c}\|_2}$$

$$\mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{\|\mathbf{up} \times \mathbf{w}\|_2}$$

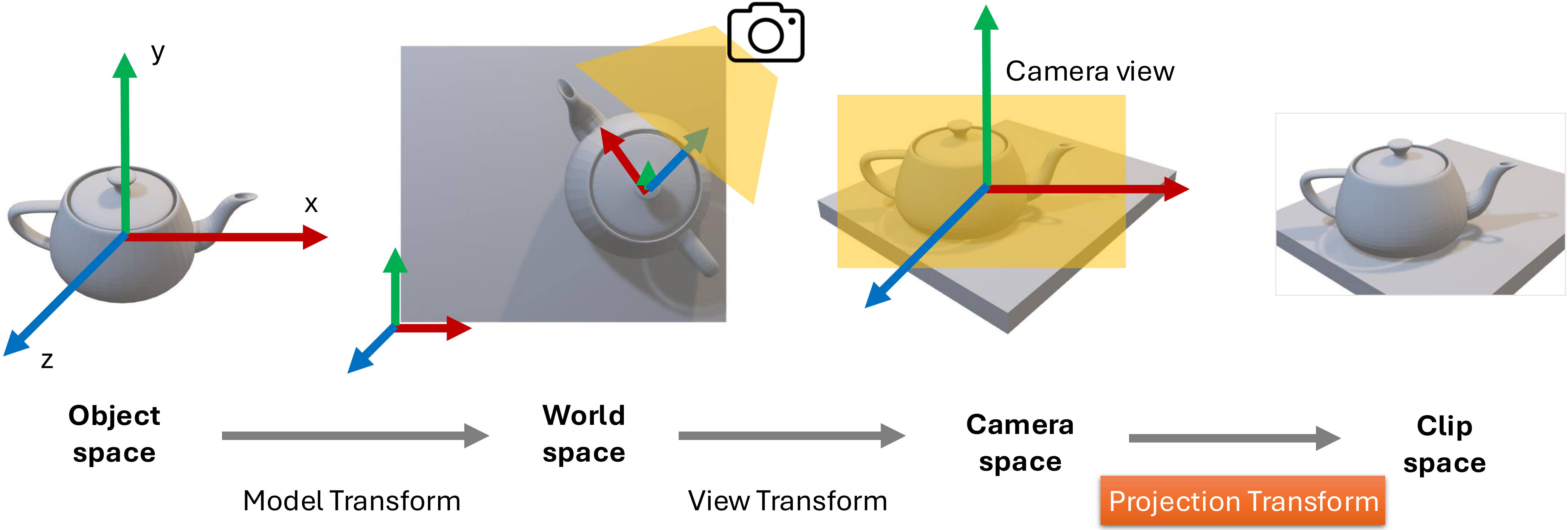
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}$$

Note: lookat direction is the negative z-axis.

Image created by DALL-E 3

Viewing Transformation 1

- Per-vertex coordinate transformation

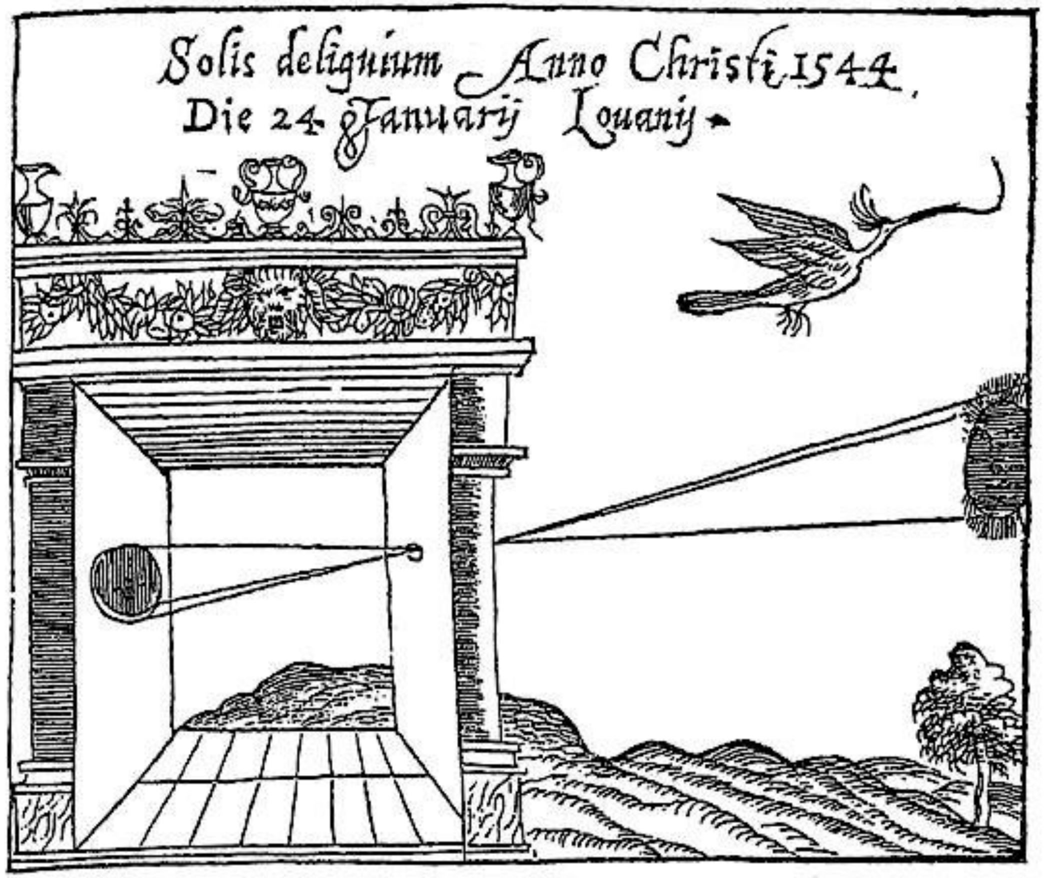


Projection Transform

- Basic principle known to
  Mozi (470-390 BCE),
  Aristotle (384-322 BCE)

- Drawing aid for artists: described by
  Leonardo da Vinci (1452-1519)

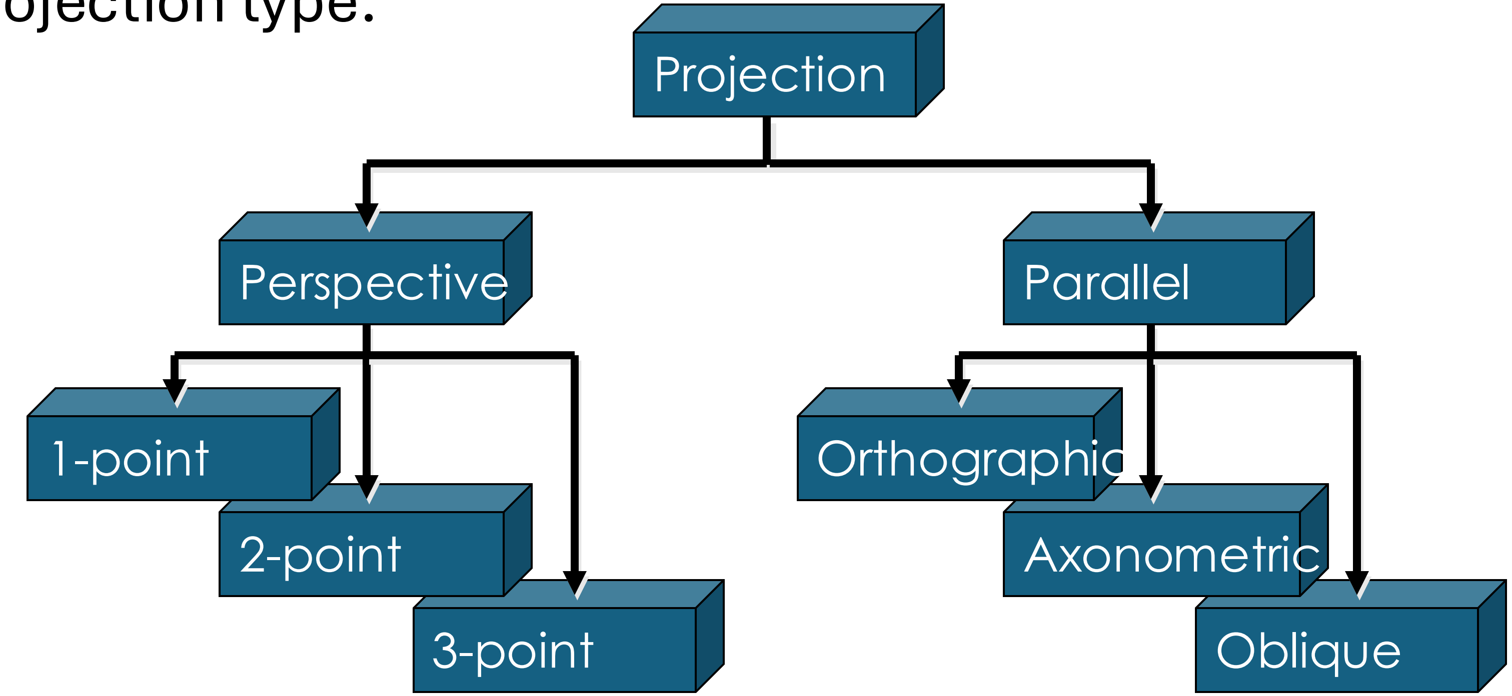
- In a dark room,
  observe light leaking through
  a very small hole on a window onto
  a piece of A4 white paper.



One of the earliest known representation of pinhole model,
by Gemma Frisius, 1558

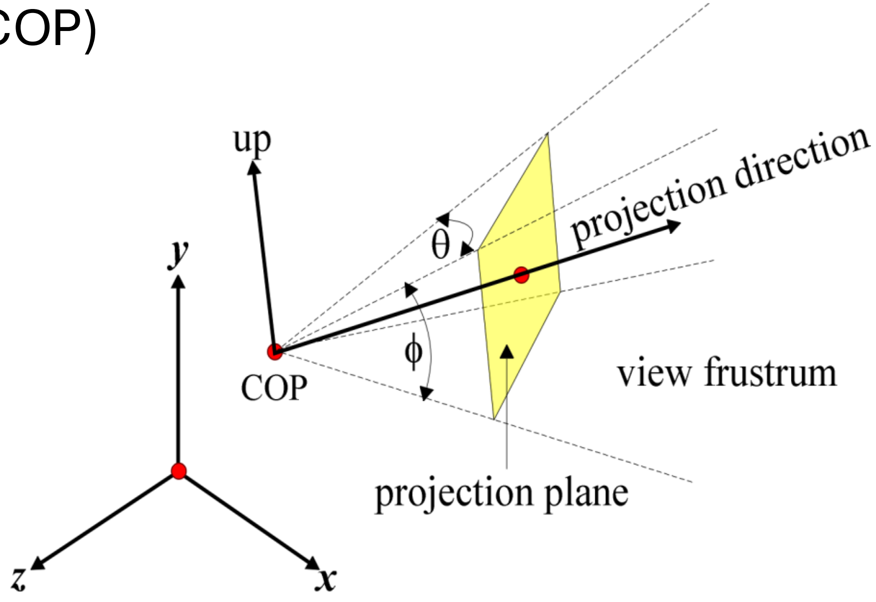
3D to 2D Projection

- Type of projection depends on a number of factors:
  - location and orientation of the viewing plane (viewport)
  - direction of projection (described by a vector)
  - projection type:



Perspective Projections

- Perspective projections exhibit *fore-shortening* (parallel appear to converge at points).
- Objects further away appear smaller
- Parameters:
  - centre of projection (COP)
  - field of view (θ, ϕ)
  - projection direction
  - up direction



Homogenous Coordinates

- Up to now, for a point, w = 1

$$\begin{bmatrix} x \\ y \\ z \\ \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

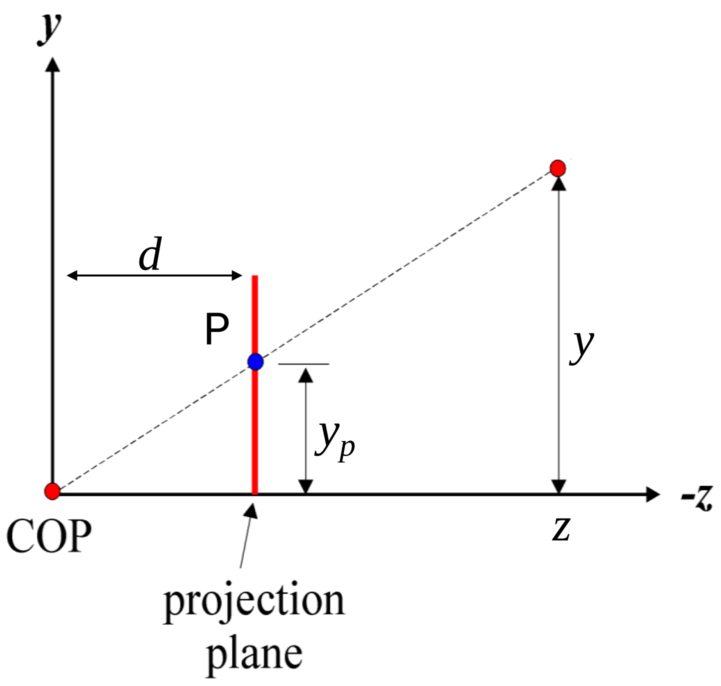
- We now make use of the 4th element w to handle perspective projection. w is no longer 1.

- Property: scaling a homogeneous coordinate results in the same point.

$$\begin{bmatrix} x/W \\ y/W \\ z/W \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix}$$

Perspective Projections

Consider a perspective projection with the viewpoint at the origin and a viewing direction oriented along the -***z*** axis and the view-plane located at *z* = -*d* with d > 0.



$$\frac{y}{z} = \frac{y_P}{-d} \quad \Rightarrow \quad y_P = d\,\frac{y}{-z}$$

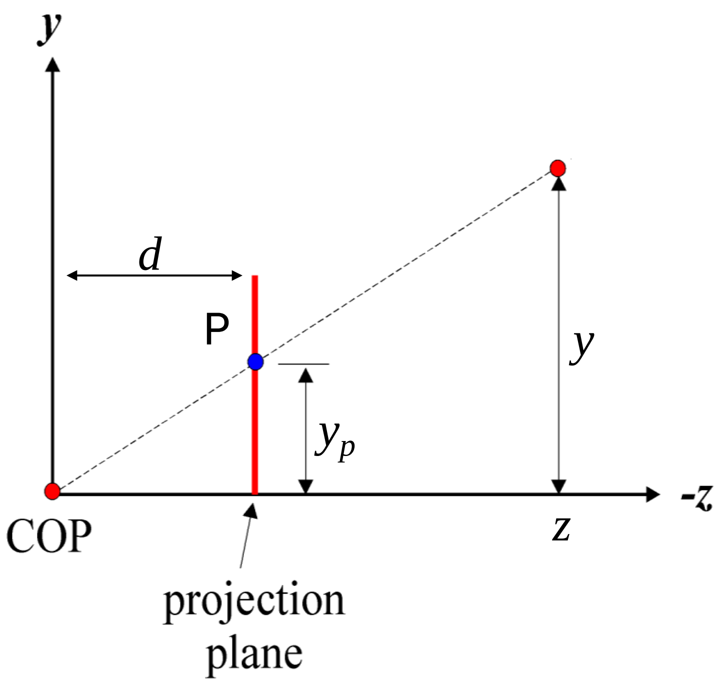
The negative sign is a convention as we assume d > 0 and we view toward –z axis.

Non-uniform foreshortening: the value y_p depends on z.

a similar construction applies for x_p

Perspective Projections

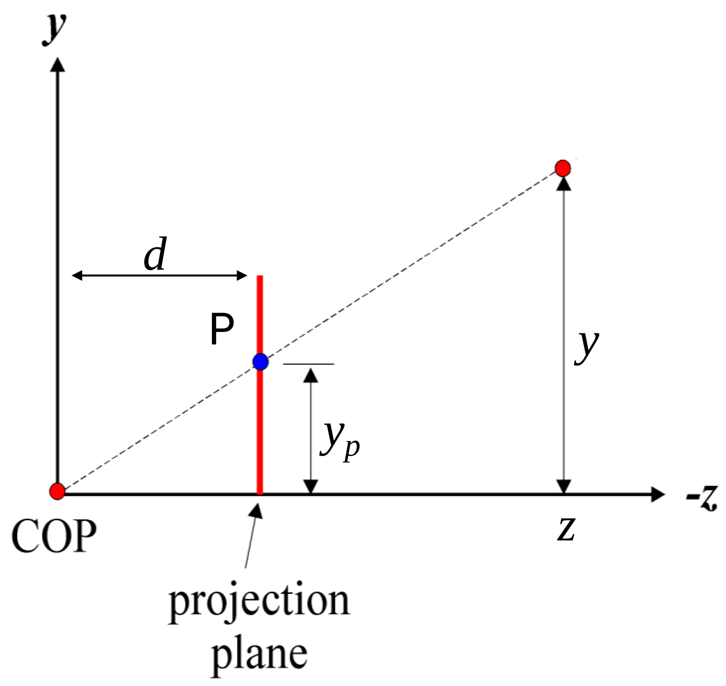
Consider a perspective projection with the viewpoint at the origin and a viewing direction oriented along the -***z*** axis and the view-plane located at *z* = -*d* with d > 0.



$$\begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} = \begin{bmatrix} \dfrac{x}{-z/d} \\ \dfrac{y}{-z/d} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z \\ -z/d \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projections

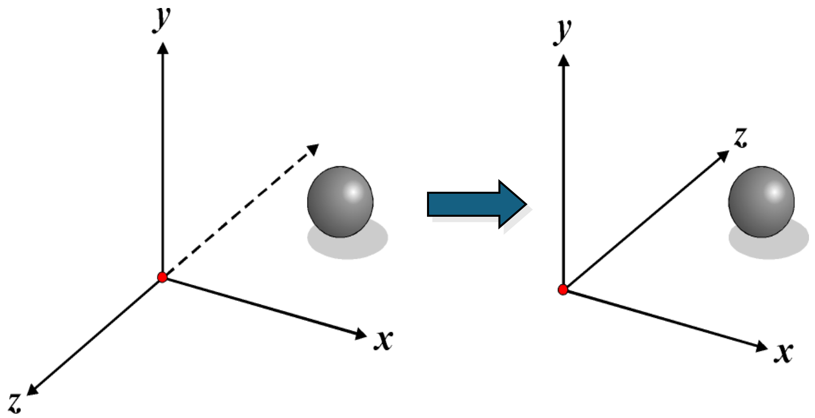
Consider a perspective projection with the viewpoint at the origin and a viewing direction oriented along the -***z*** axis and the view-plane located at $z = -d$ with d > 0.



$$\begin{bmatrix} x_P \\ y_P \\ z_P \\ 1 \end{bmatrix} = \begin{bmatrix} d\,\dfrac{x}{-z} \\ d\,\dfrac{y}{-z} \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} d\,x \\ d\,y \\ -d\,z \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -d & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

An alternative way of writing the transformation matrix by multiplying the homogeneous coordinates with d. Note that all points remain the same.

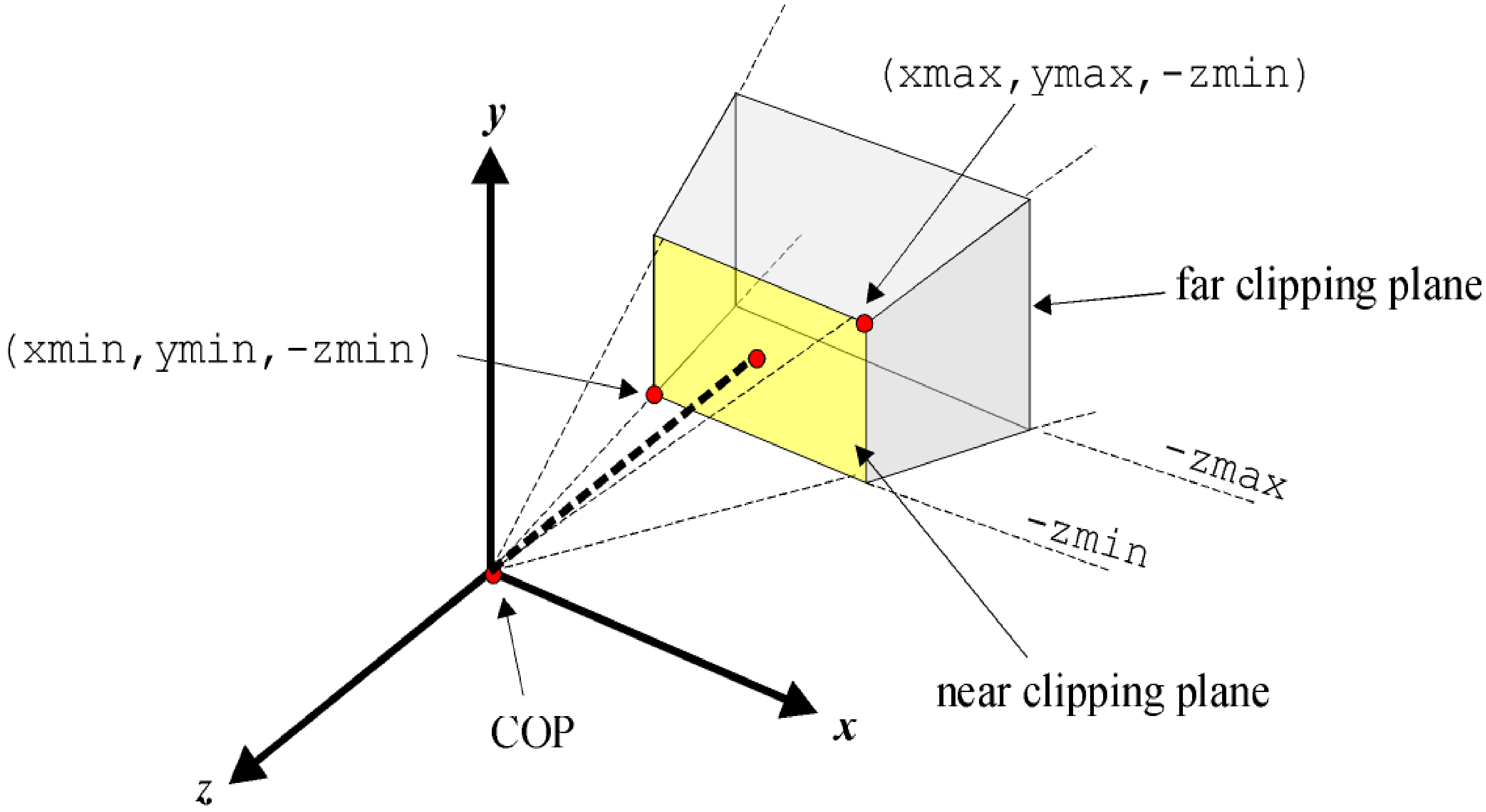
Perspective Projections Details



$$\begin{bmatrix} x \\ y \\ -z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

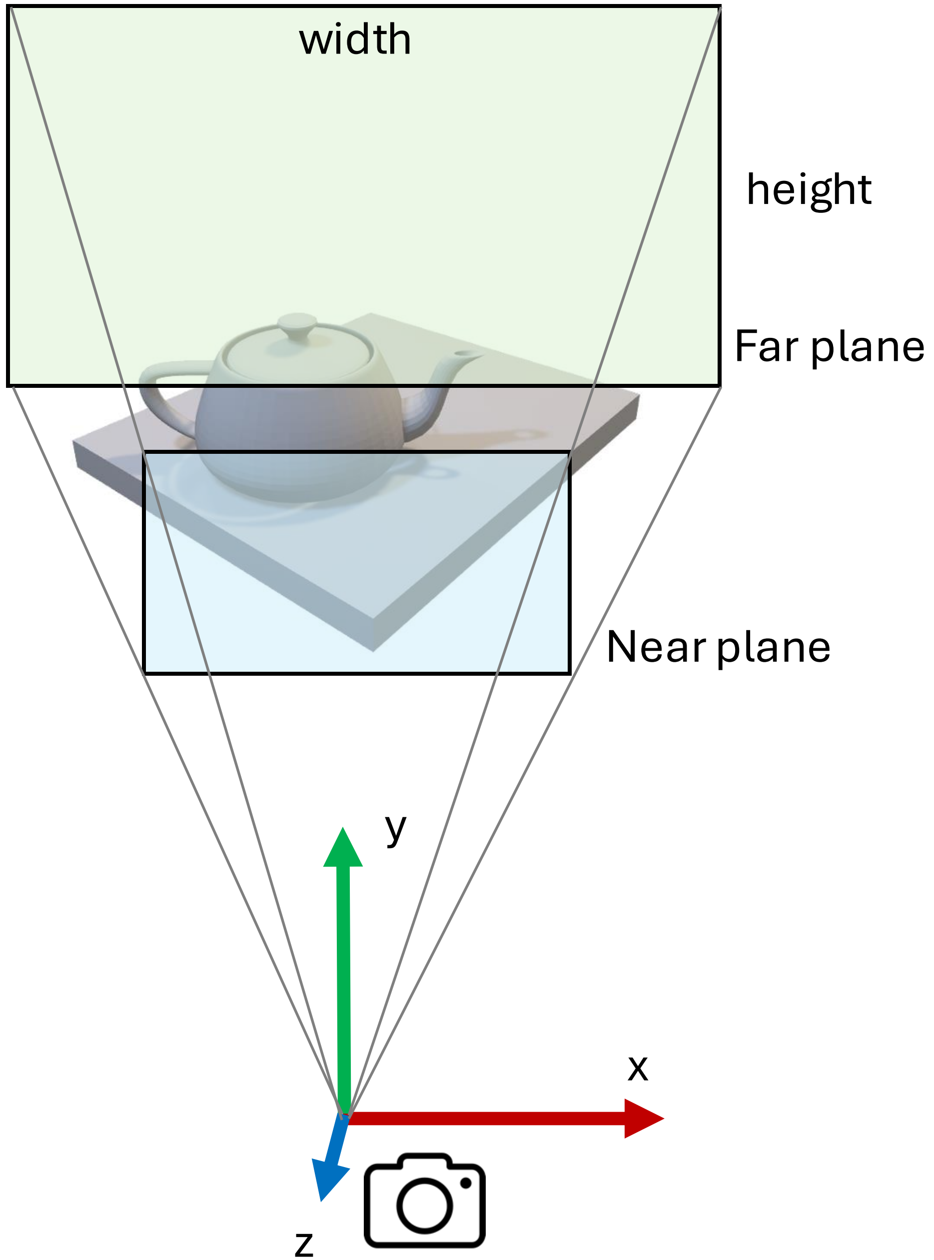
- Flip ***z*** to transform to a left handed co-ordinate system
- Increasing *z* values mean increasing distance from the viewer.

Perspective Projections

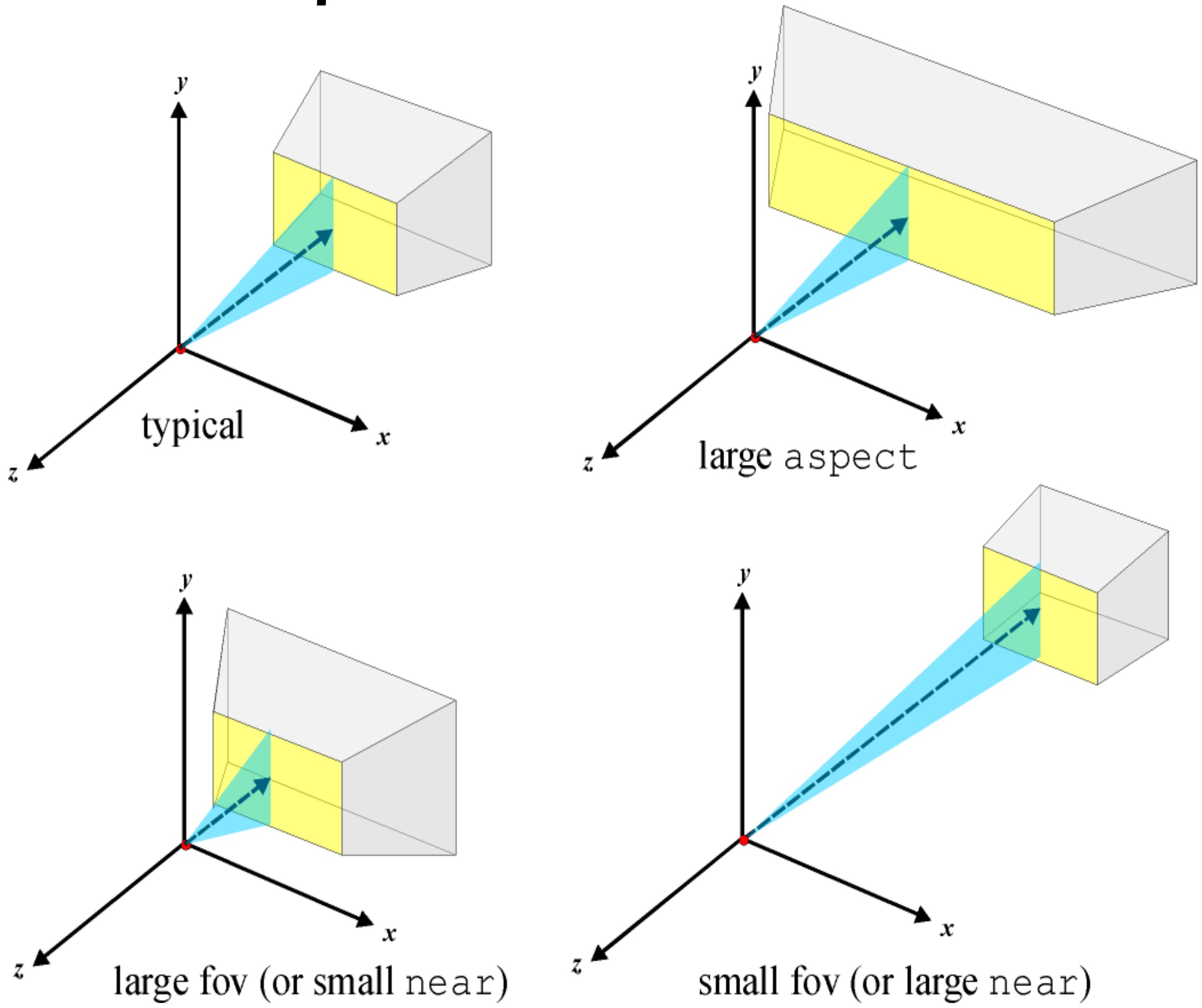


Perspective Projection in OpenGL

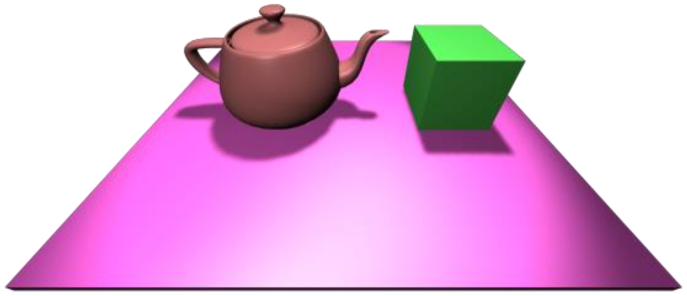
- Symmetric view frustum case
- Field of view (along y-axis)
- Aspect ratio: ratio of width and height of the viewport
- Near plane: near clipping distance relative from camera
- Far plane: far clipping distance relative from camera



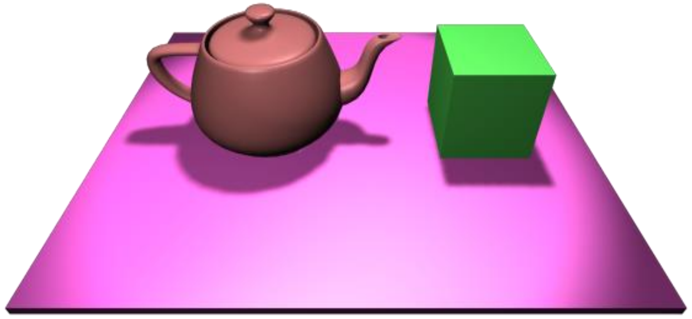
Perspective Projections



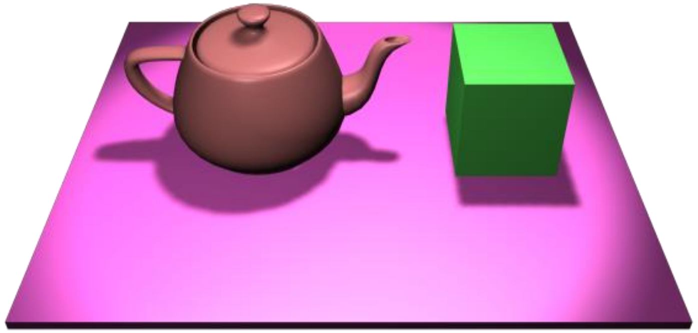
Lens Configurations



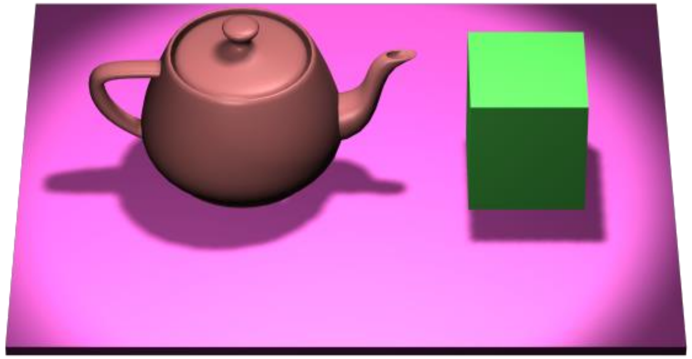
10mm Lens (fov = 122°)



20mm Lens (fov = 84°)



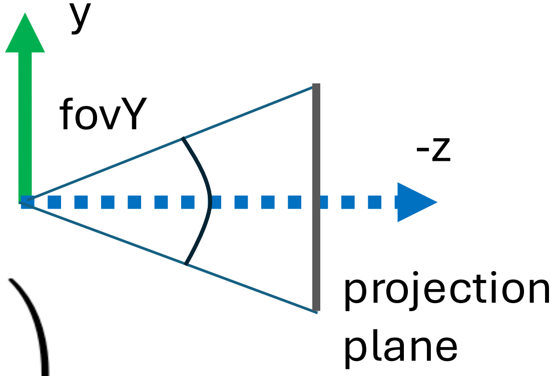
35mm Lens (fov = 54°)



200mm Lens (fov = 10°)

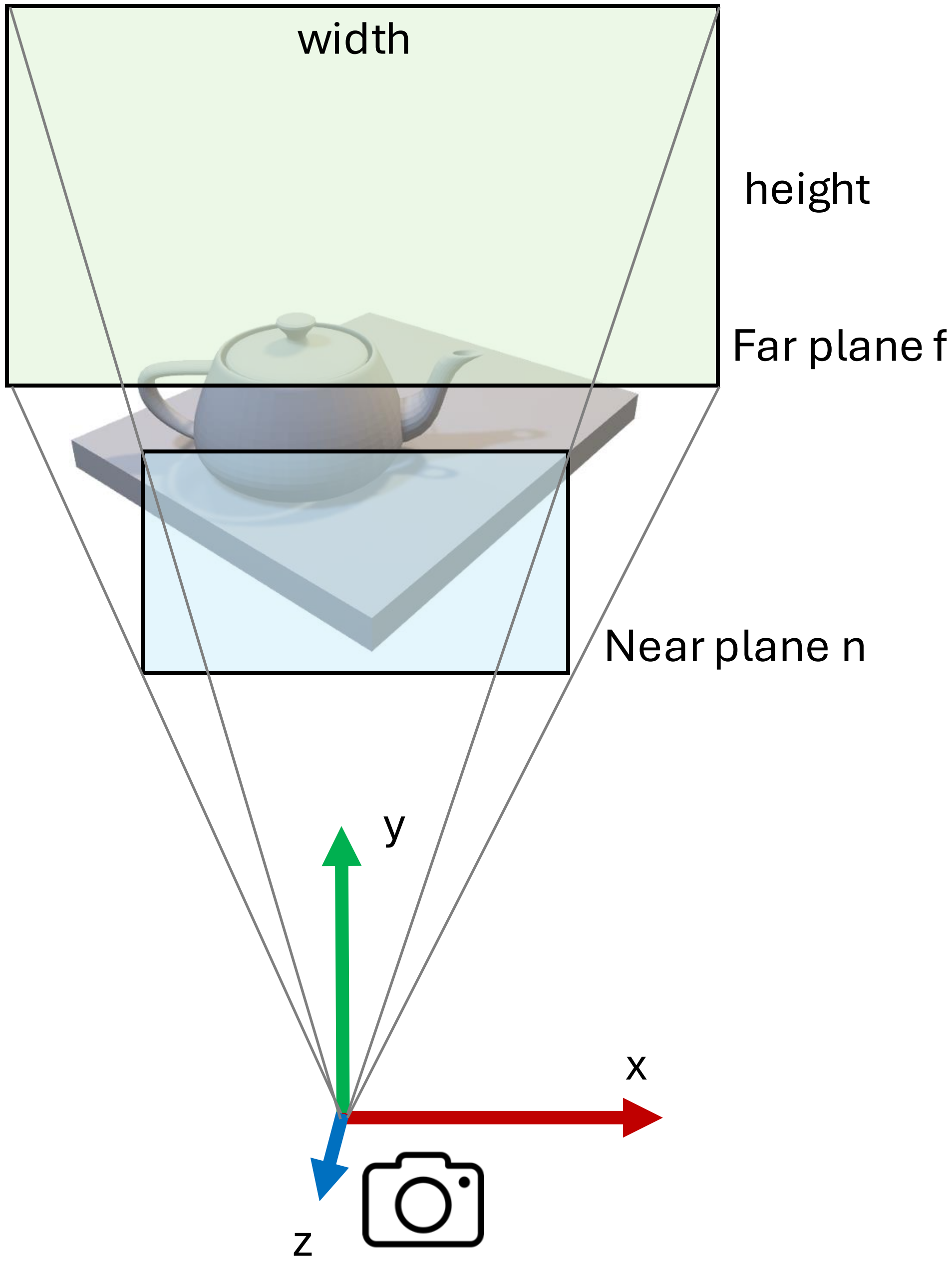
Perspective Projection in OpenGL

- Projection matrix



$$F = \cot\left(\frac{\text{fovY}}{2}\right)$$

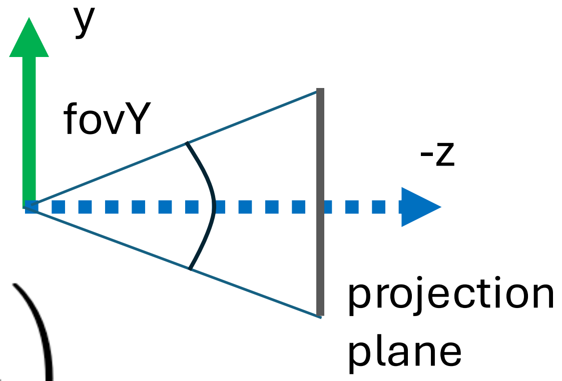
$$\begin{bmatrix} \frac{F}{\text{aspect}} & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



Derivation of OpenGL projection matrix: http://www.songho.ca/opengl/gl_projectionmatrix.html

Perspective Projection in OpenGL

- Projection matrix



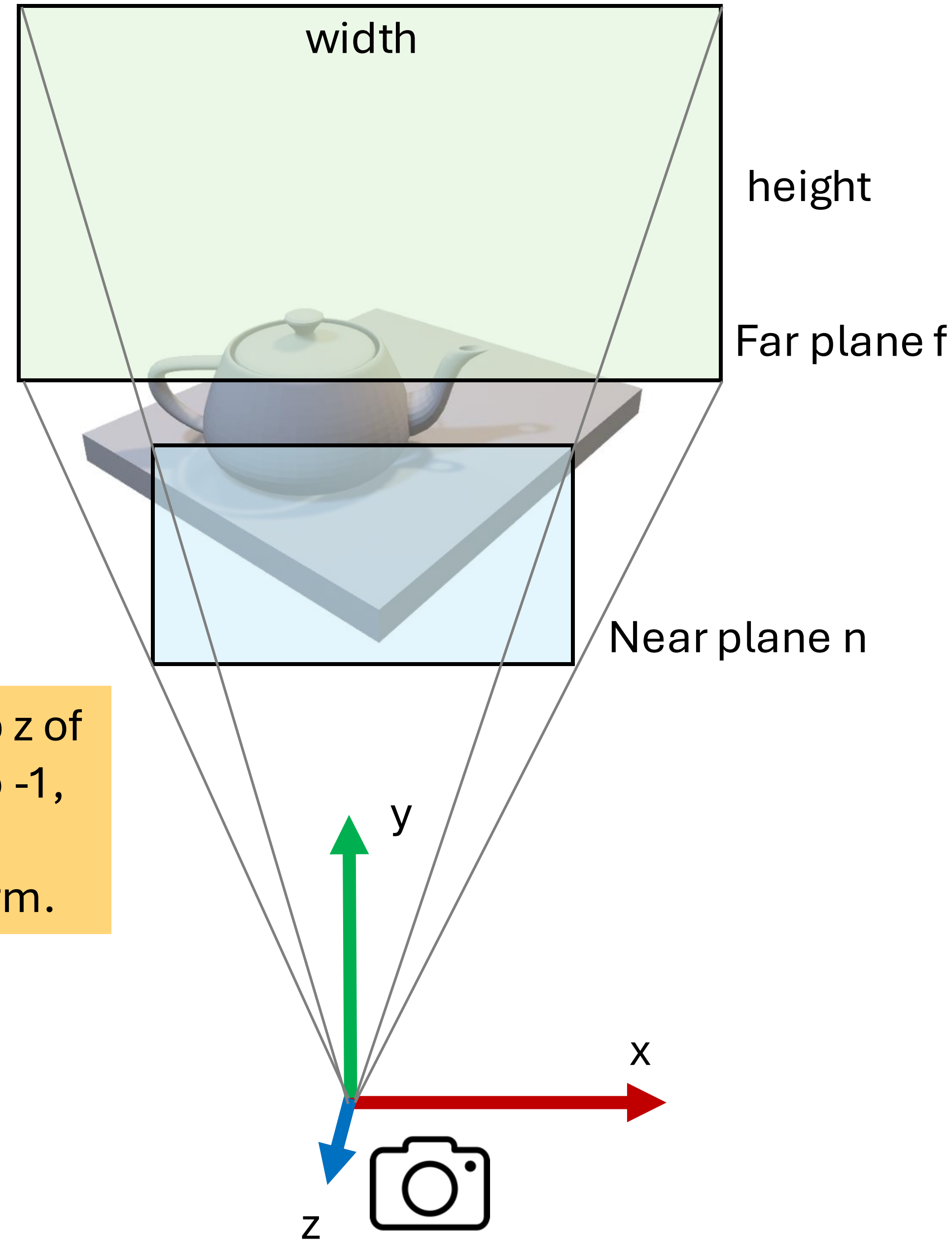
Projection plane ratio

$$F = \cot\left(\frac{\text{fovY}}{2}\right)$$

Aspect ratio

Scale to map z of near plane to -1, far plane to 1 after transform.

$$\begin{bmatrix} \frac{F}{\text{aspect}} & 0 & 0 & 0 \\ 0 & F & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



Derivation of OpenGL projection matrix: http://www.songho.ca/opengl/gl_projectionmatrix.html

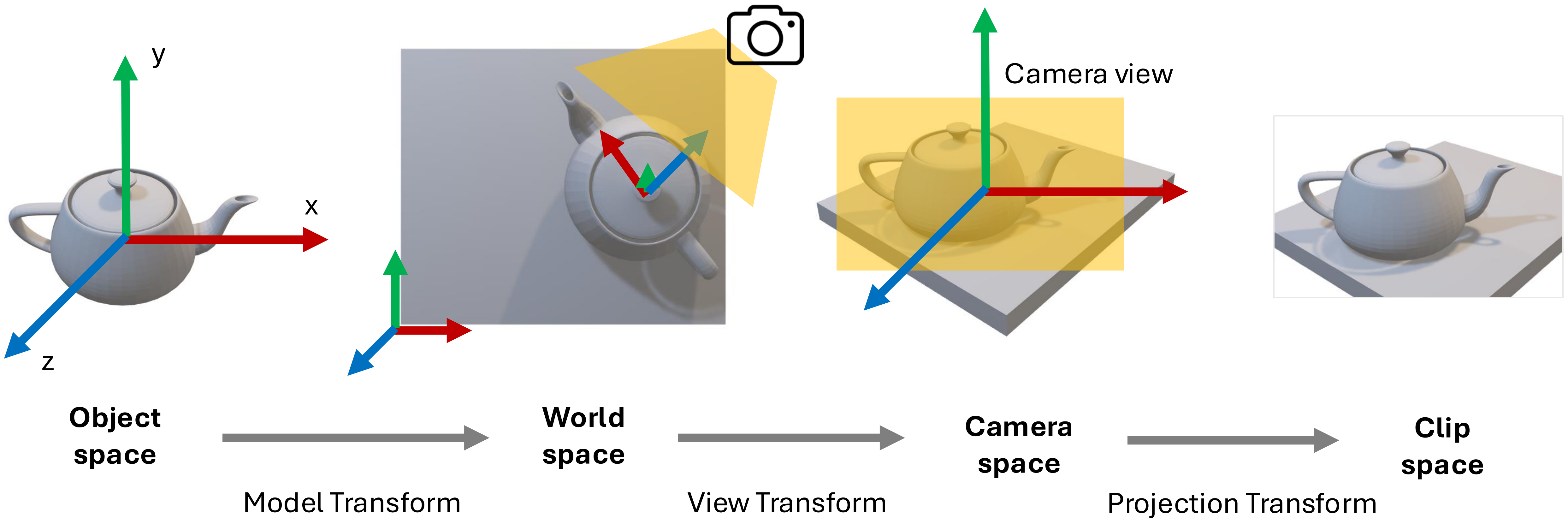
Perspective Projections

- So far, our model and view transformations are affine, i.e., it preserves **lines** and **parallelism**.

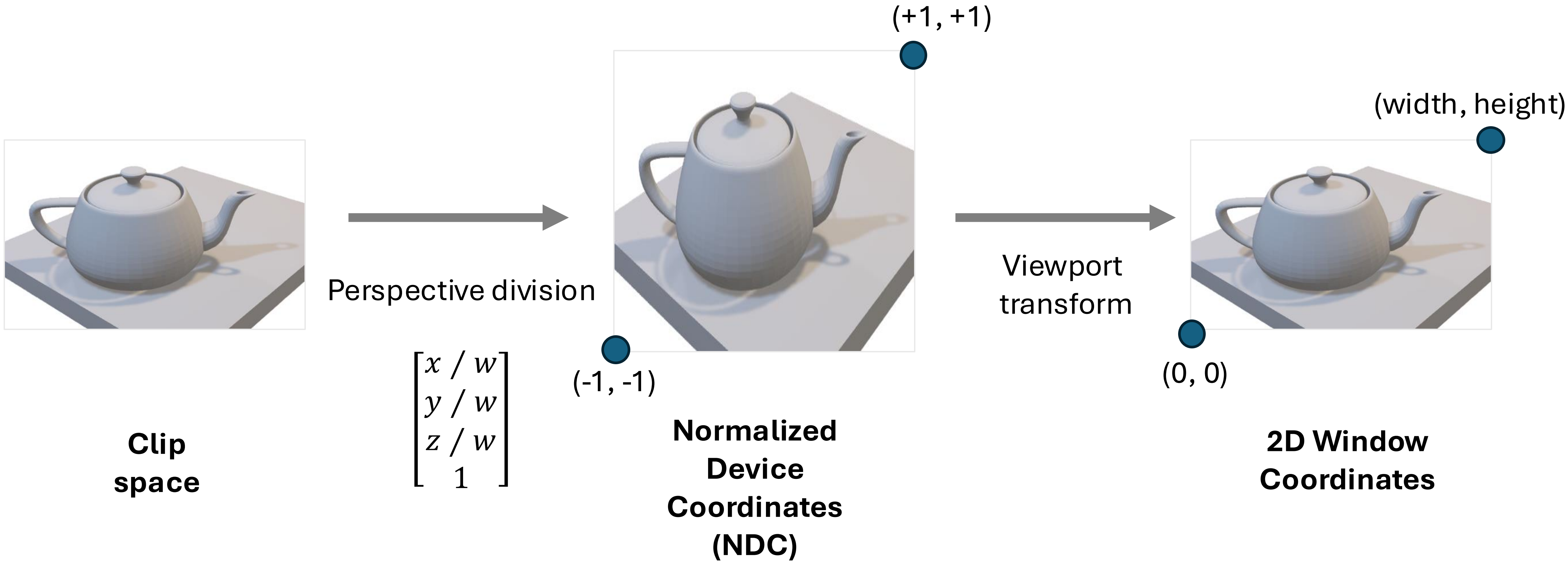
- Perspective transformations <u>preserve lines</u>. After perspective projection, parallel lines can intersect at vanishing points!

- Perspective transformation is irreversible.
  All points along a projector project onto the same point, we cannot recover a point from its projection.

View Transformation 1



View Transformation 2



Clip Space

- Perform all vertex transformations using 4x4 matrix multiplications

$$\mathbf{v}_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot \mathbf{v}$$

Clip-space coordinates

Camera-space coordinates

Object-space or world-space coordinates

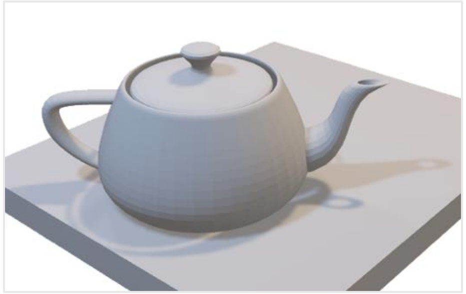
$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = M_{proj} \cdot M_{modelview} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Projection matrix

Modelview matrix

Clip Space

- OpenGL expects clip space coordinates at the end of the vertex shader, after performing projection.
- Clipping is then performed to remove out-of-range coordinates.
- A point is visible when
  $-w < x < w$
  $-w < y < w$
  $-w < z < w$



Clip space

Normalized Device Coordinates (NDC)

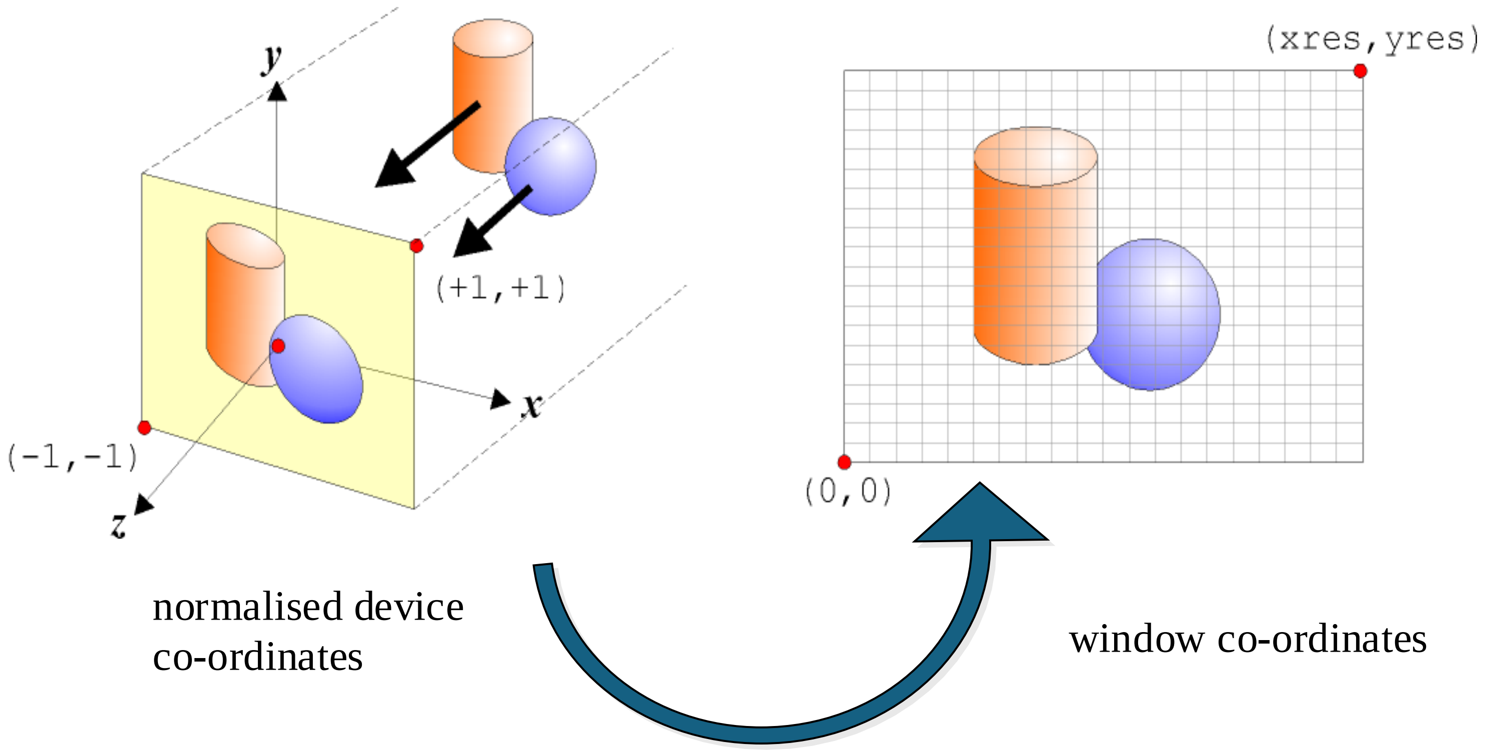
- Perspective division to get NDC coordinates.

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} \rightarrow \begin{bmatrix} x_{clip}/w_{clip} \\ y_{clip}/w_{clip} \\ z_{clip}/w_{clip} \\ 1 \end{bmatrix} \in [-1, 1]$$

- Note that the output of a vertex shader should be in the clip space. OpenGL will then perform the perspective division for us.
- Conventionally, OpenGL's NDC is left-handed.

The Viewport

- We need to associate the 2D *viewport co-ordinate system* with the *window co-ordinate system* in order to determine the correct pixel associated with each vertex.



Viewport to Window Transformation

- **`glViewport`** used to relate the co-ordinate systems:

```
glViewport(int x, int y, int width, int height);
```

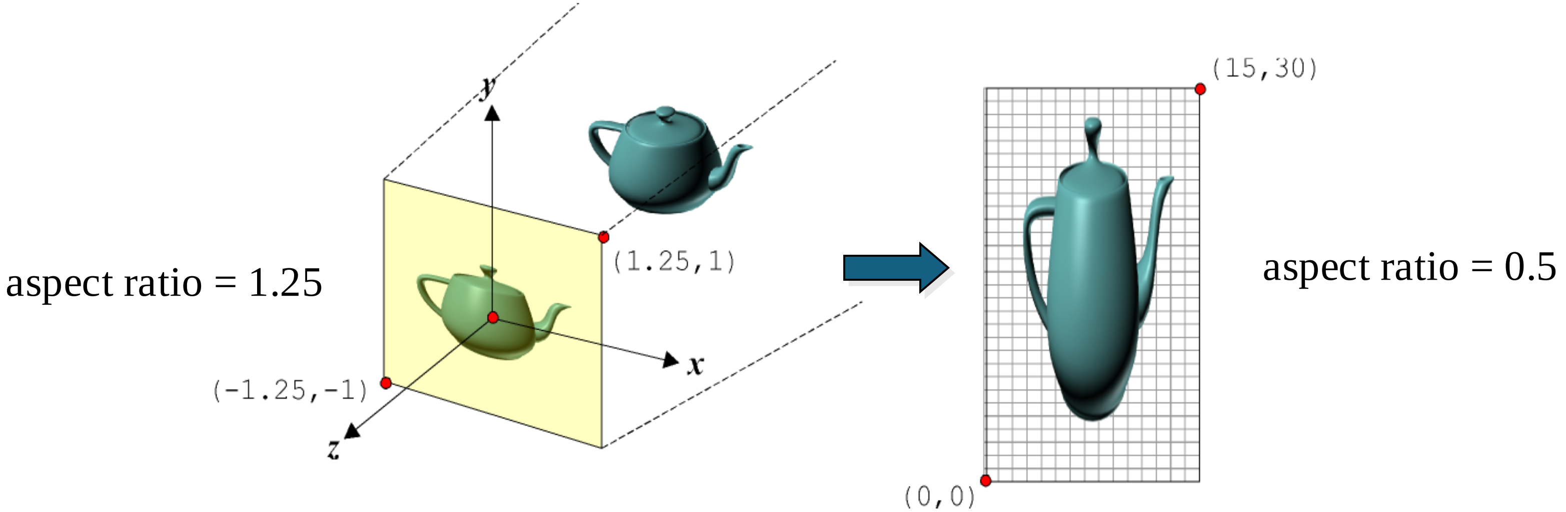
- The NDC coordinates are mapped to the window using

$$x_w = (x_n + 1)\left(\frac{\text{width}}{2}\right) + \mathrm{x} \quad y_w = (y_n + 1)\left(\frac{\text{height}}{2}\right) + \mathrm{y}$$

- `x,y` is location of bottom left of viewport within the window
- `width,height` is the dimension in pixels of the viewport

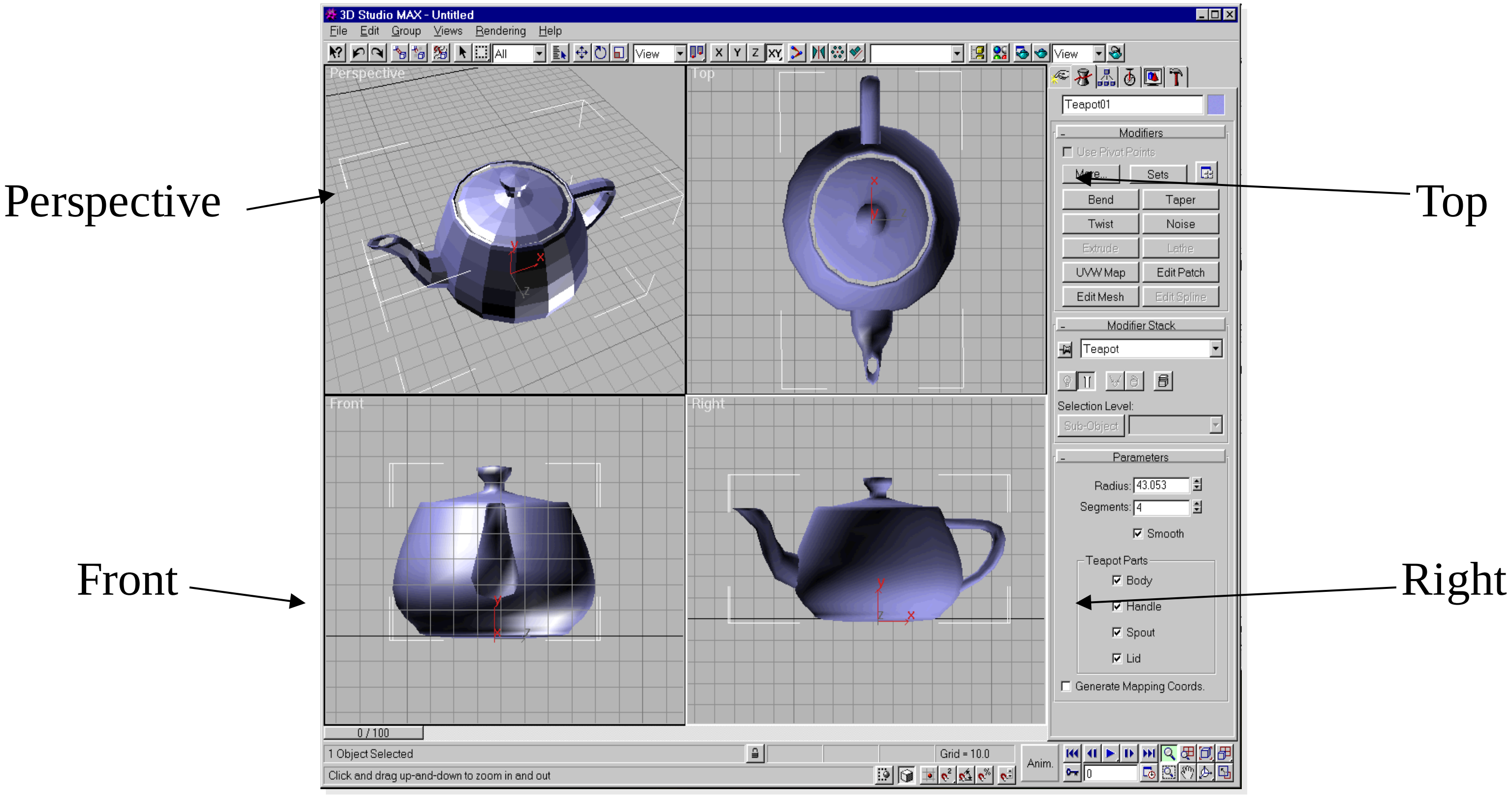
Aspect Ratio

- The *aspect ratio* defines the relationship between the width and height of an image.
- Using **`Perspective`** matrix, a viewport aspect ratio may be explicitly provided, otherwise the aspect ratio is a function of the supplied viewport width and height.
- The aspect ratio of the window (defined by the user) must match the viewport aspect ratio to prevent unwanted *affine* distortion:



Multiple Projections

- To help 3D understanding, it can be useful to have *multiple projections* available at any given time
  - usually: plan (top) view, front & left or right elevation (side) view



Reading List & Practical Tasks

- Interactive Computer Graphics, A Top-down Approach with OpenGL, 6th edition, Chapter 4 on Viewing
  - Edward Angel
- Fundamentals of Computer Graphics, 3rd Edition, Shirley and Marschner, Chapter 7
  - Equation 6.7 shows derivation of scale and translate for Orthographic matrix
  - Section 7.1 Discusses Viewing Transformations
- Akenine Moeller et. al "Real-Time Rendering" Ch. 2 and 4.6 "Projections"
- Nice video tutorial on creating a camera in OpenGL, by Jamie King
  - https://www.youtube.com/watch?v=zHlxQoJYUhw
- Know how to work out the pipeline by hand on paper for 1 vertex & M, V, and P
- Derivation of OpenGL Projection Matrix: http://www.songho.ca/opengl/gl_projectionmatrix.html
- Hint: add a "print_matrix(m)" function to check contents