Camera

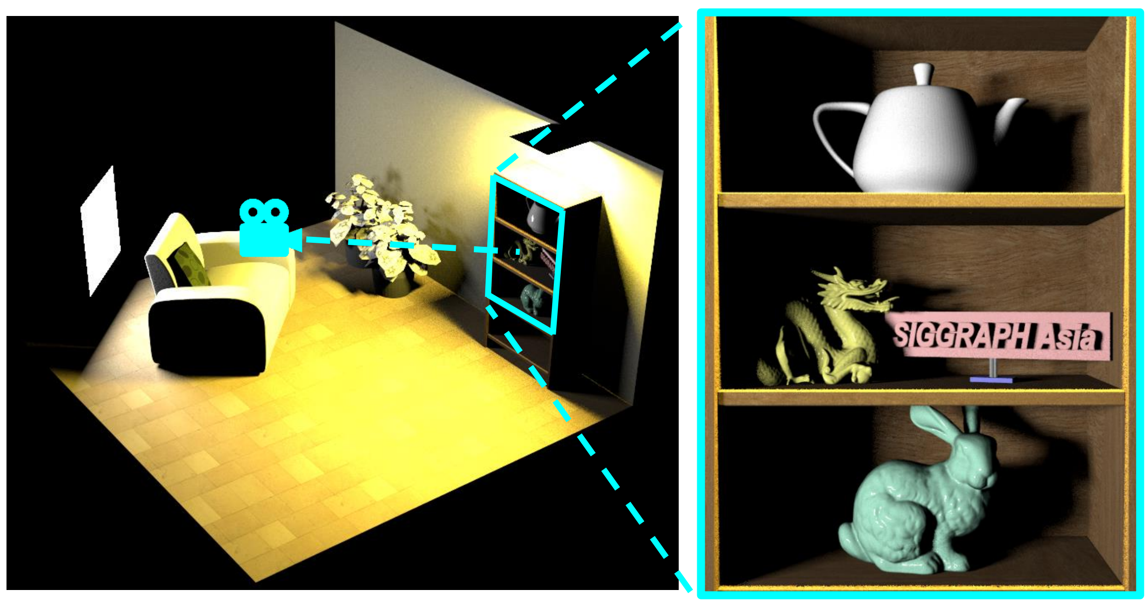
Introduction to Computer Graphics

Yu-Ting Wu

(Some of this slides are borrowed from Prof. Yung-Yu Chuang)

Recap.

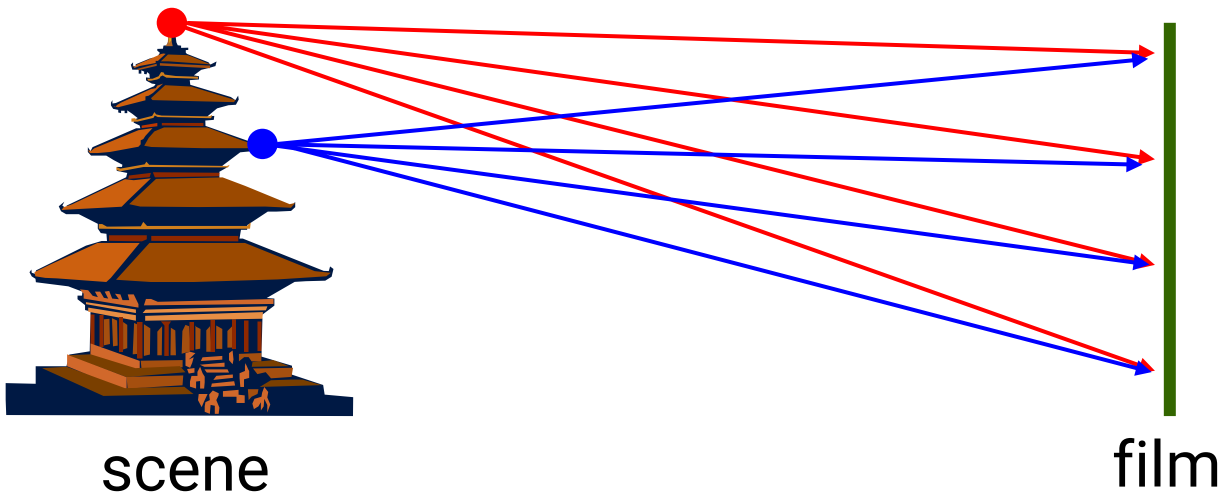
- In computer graphics, we generate an **image** from a **virtual 3D world**
  - We are going to introduce the **virtual camera** and **its projection** used to render the scene



3D virtual world rendered image

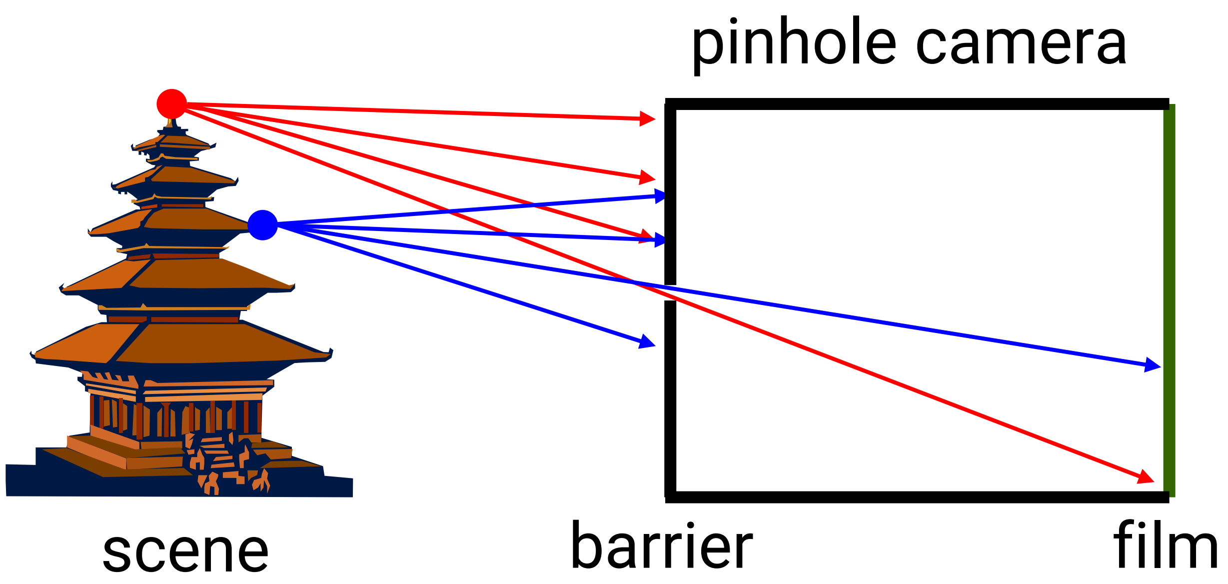
How a Real-world Camera Works

Camera Trail



Put a piece of film in front of an object

Pinhole Camera

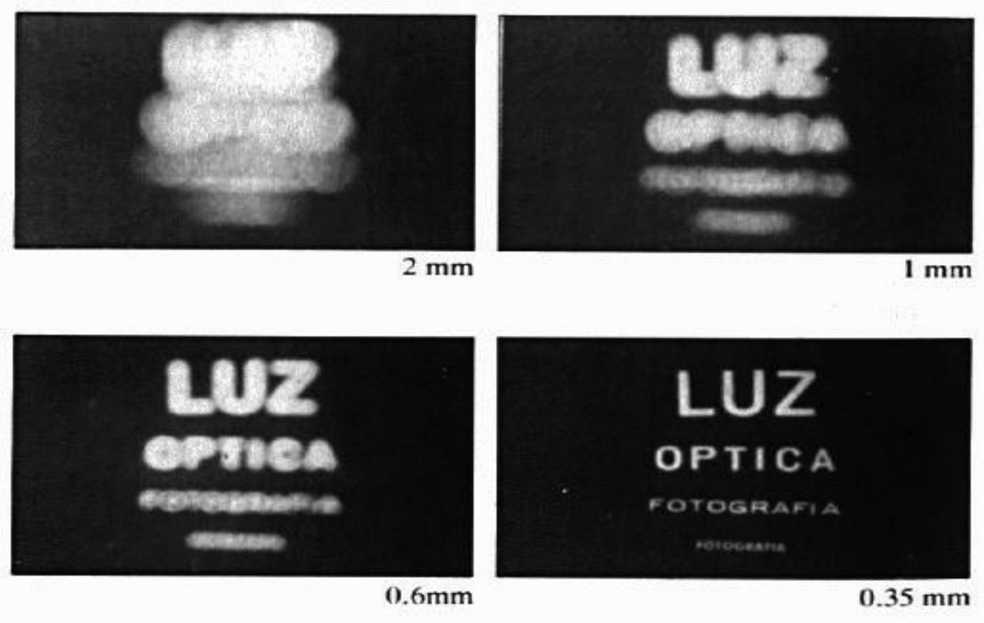


Add a barrier to block off most of the rays

- It reduces blurring
- The pinhole is known as the aperture
- The image is inverted

Pinhole Camera (cont.)

- Shrink the aperture

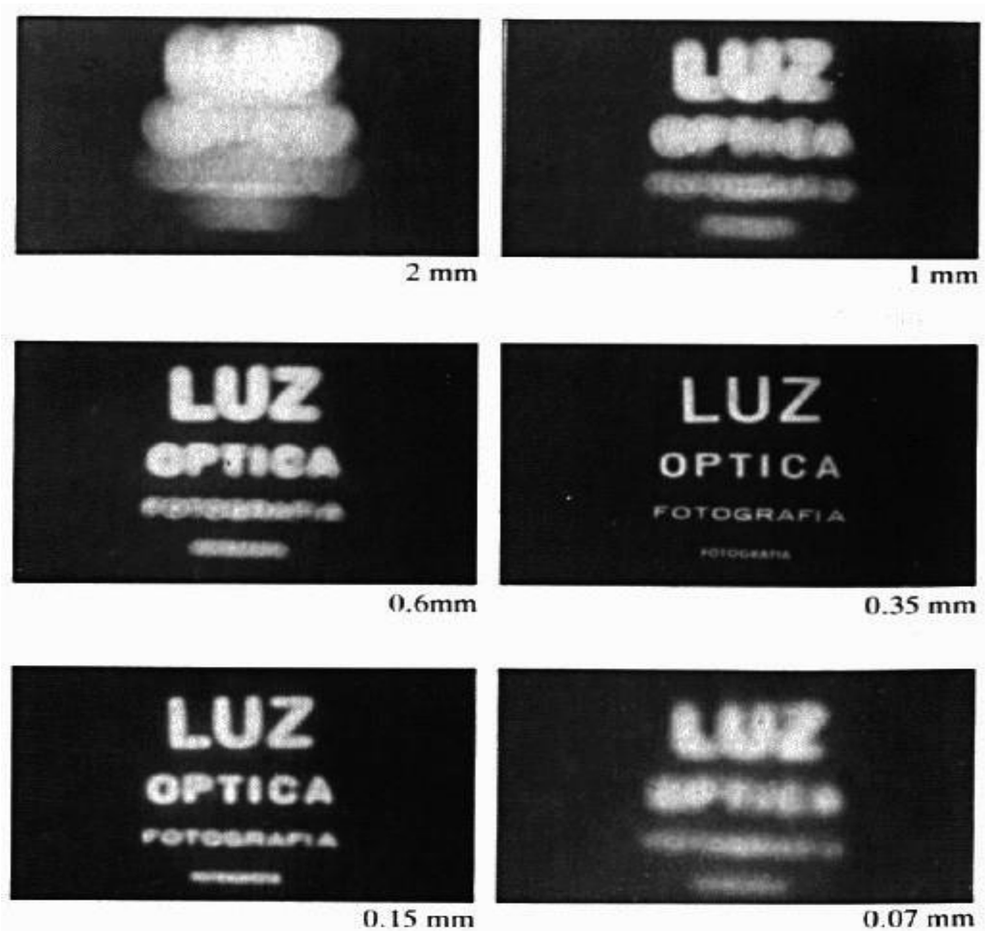


Why not make the aperture as small as possible?

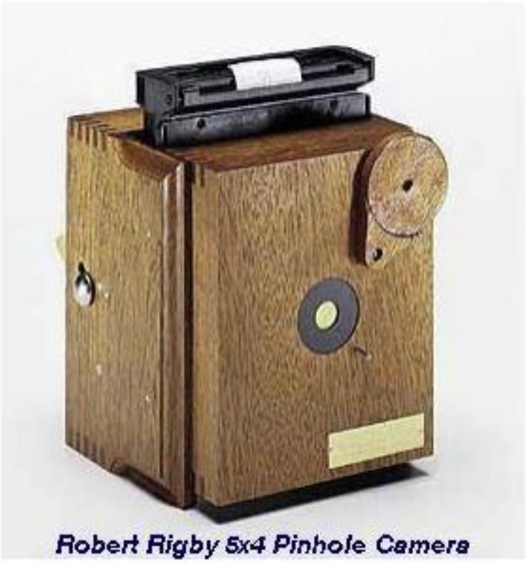
- Less light gets through
- Diffraction effect

Pinhole Camera (cont.)

- Shrink the aperture

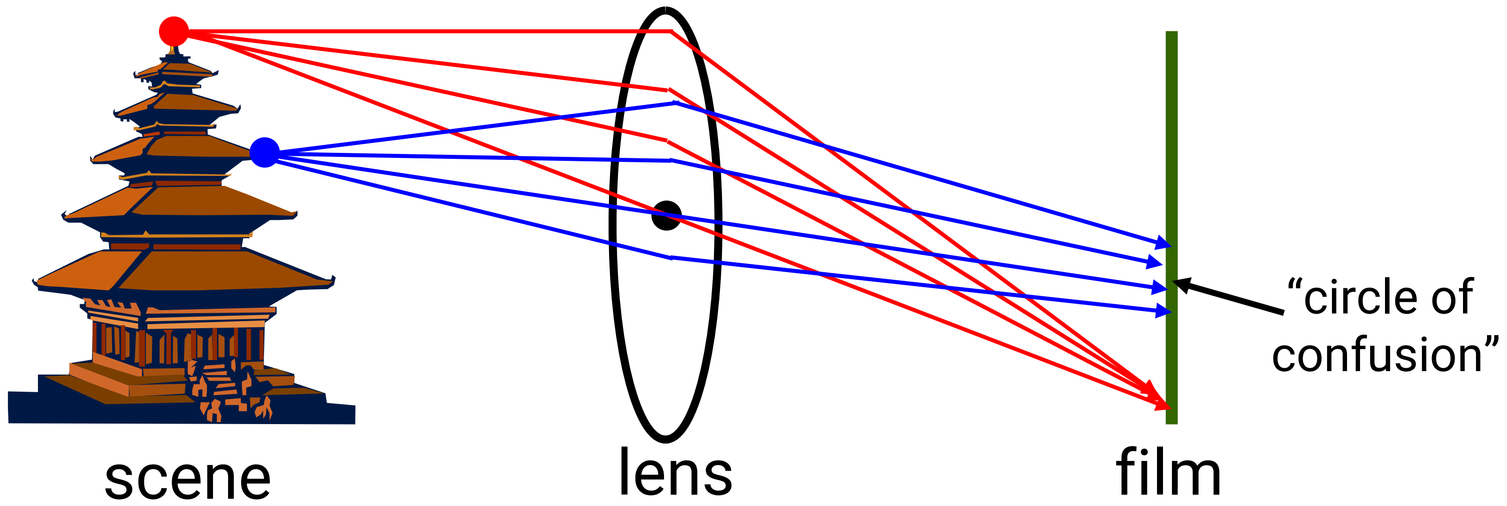


Pinhole Camera (cont.)



$200~$700

Camera with Lens



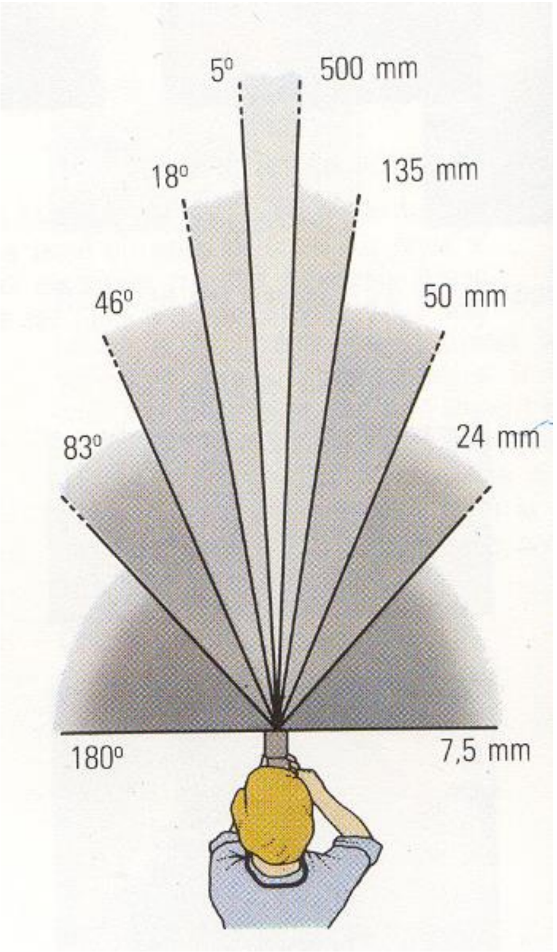
A lens **focuses** light onto the film

- There is a specific distance at which objects are “in focus”
- Other points project to a “circle of confusion” in the image

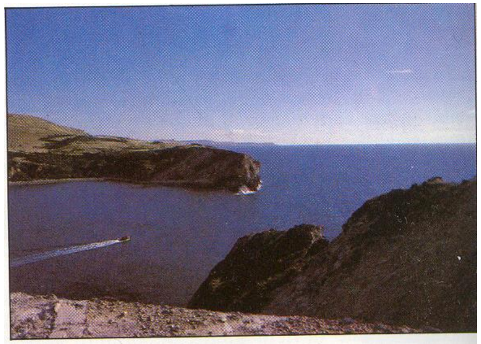
Current digital cameras replace the film with a **sensor array** (CCD or CMOS)

Camera with Lens (cont.)

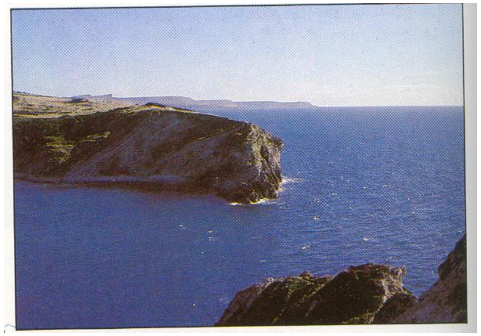
field of view



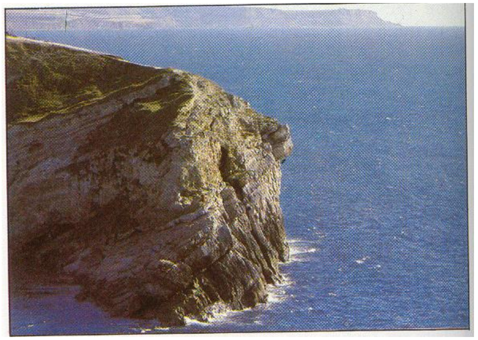
24mm



50mm

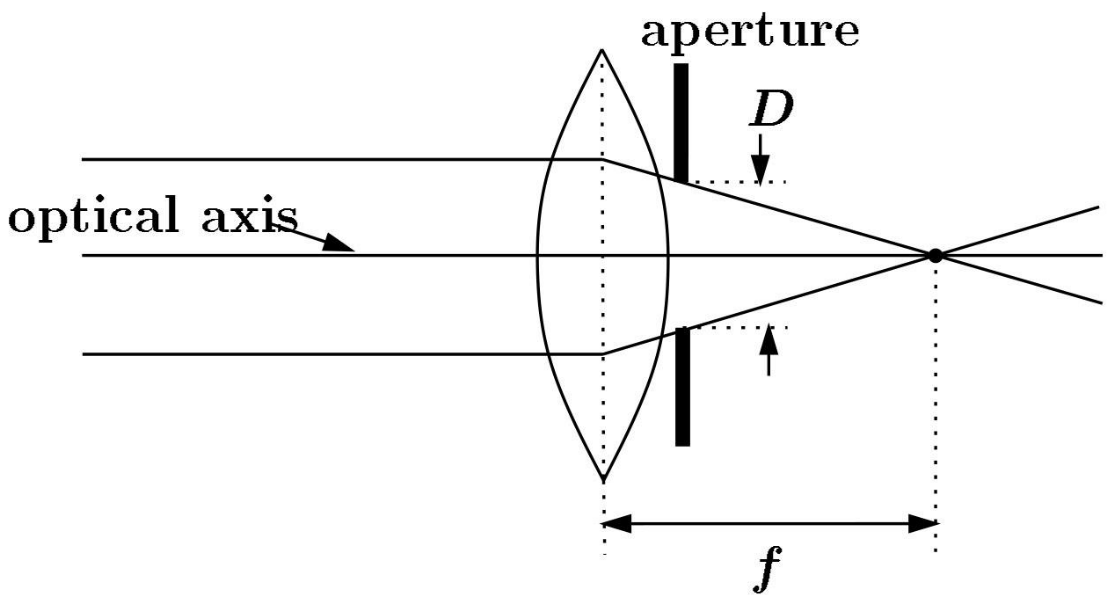


135mm



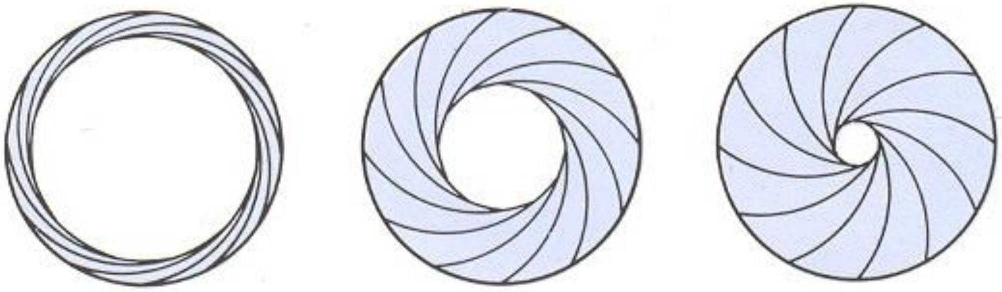
Exposure

- **Exposure = aperture + shutter speed**
  - Aperture of diameter ***D*** restricts the range of rays (aperture may be on either side of the lens)
  - Shutter speed is the amount of time that light is allowed to pass through the aperture

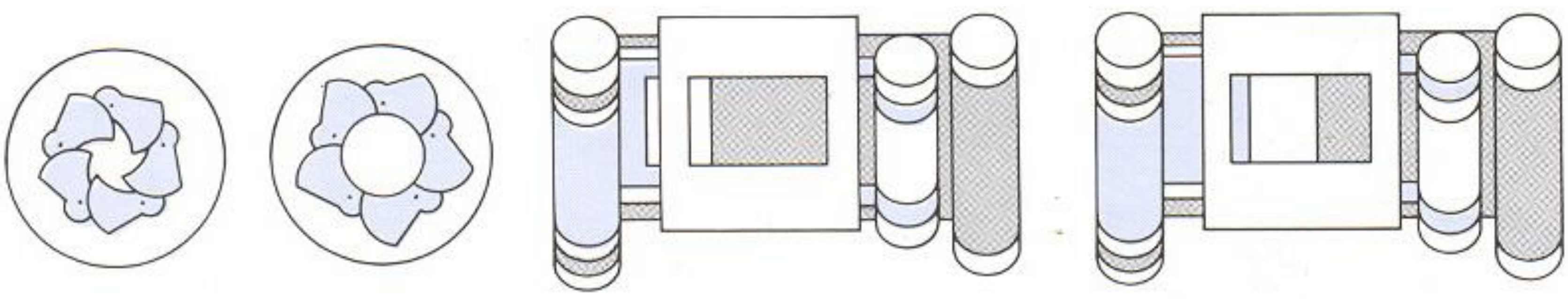


Exposure

- Aperture (in f stop)

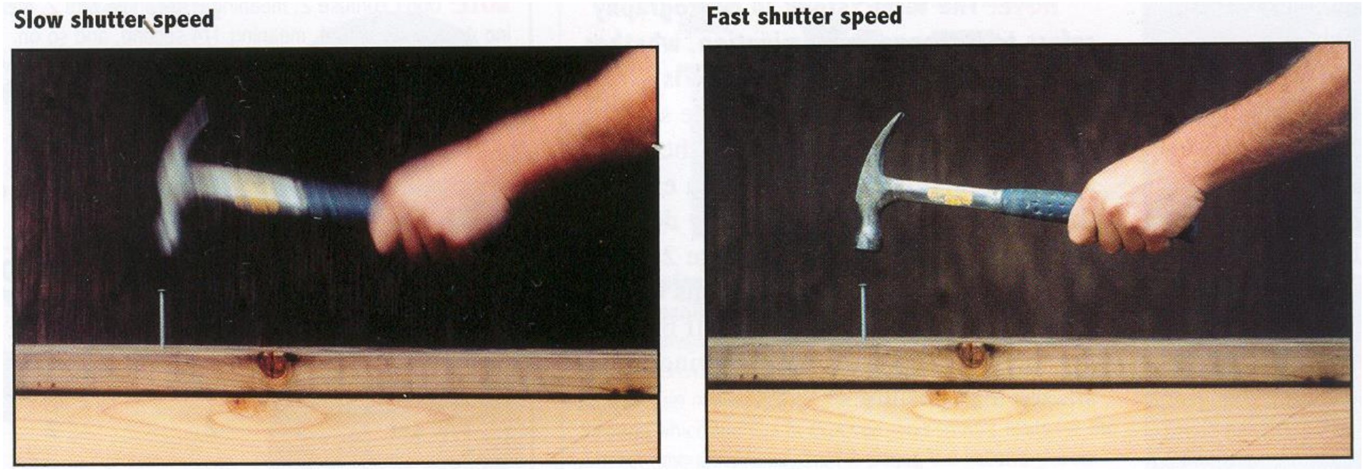


- Shutter speed (in fraction of a second)

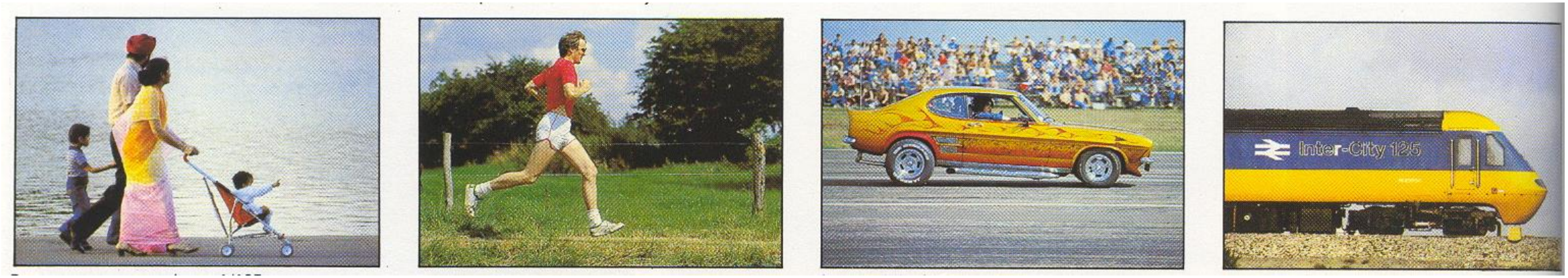


Effect of Shutter Speeds

- Slow shutter speed ➔ more light, but more motion blur



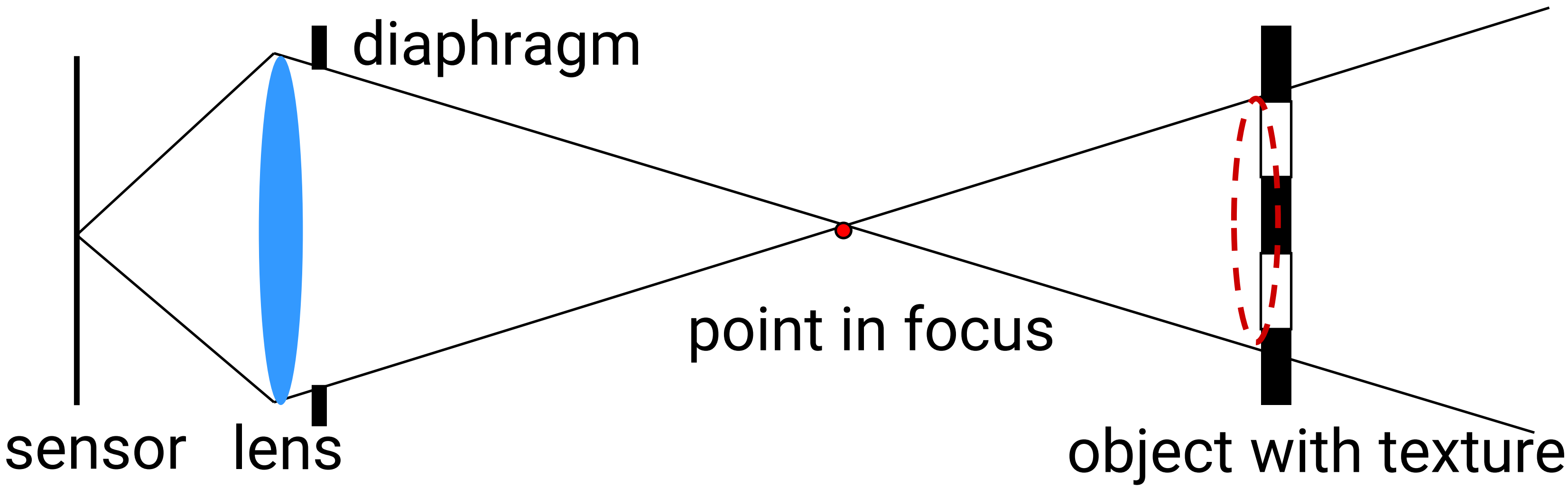
- Faster shutter speed freezes motion



1/125 1/250 1/500 1/1000

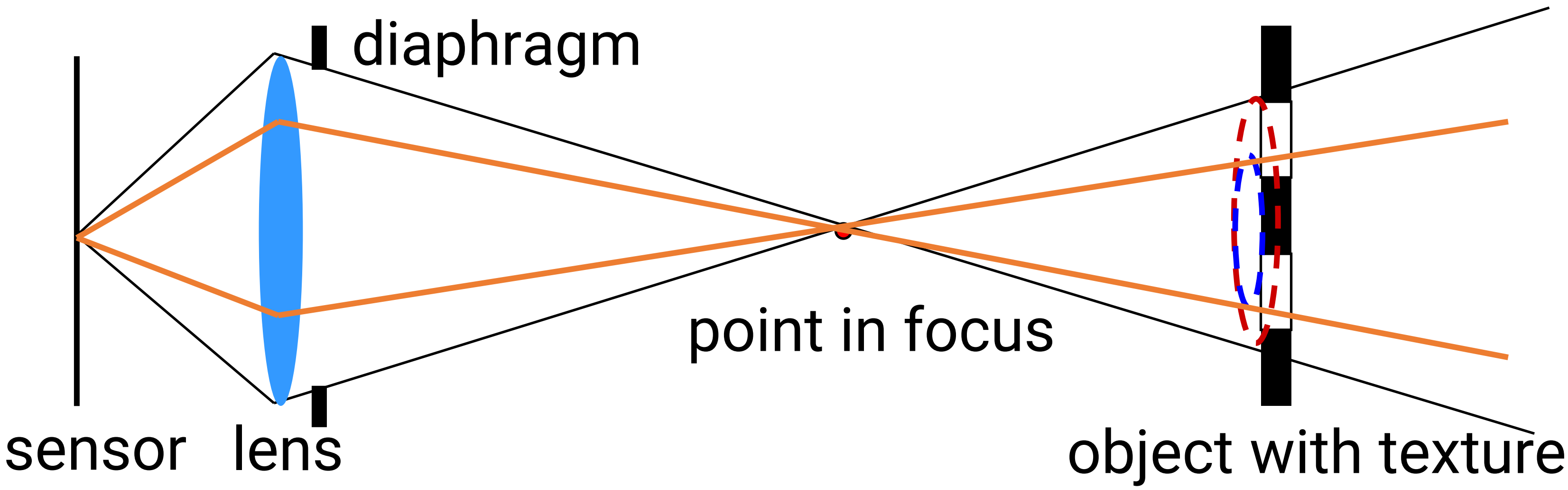
Depth of Field

- Changing the aperture size affects depth of field
  - A smaller aperture increases the range in which the object is approximately in focus

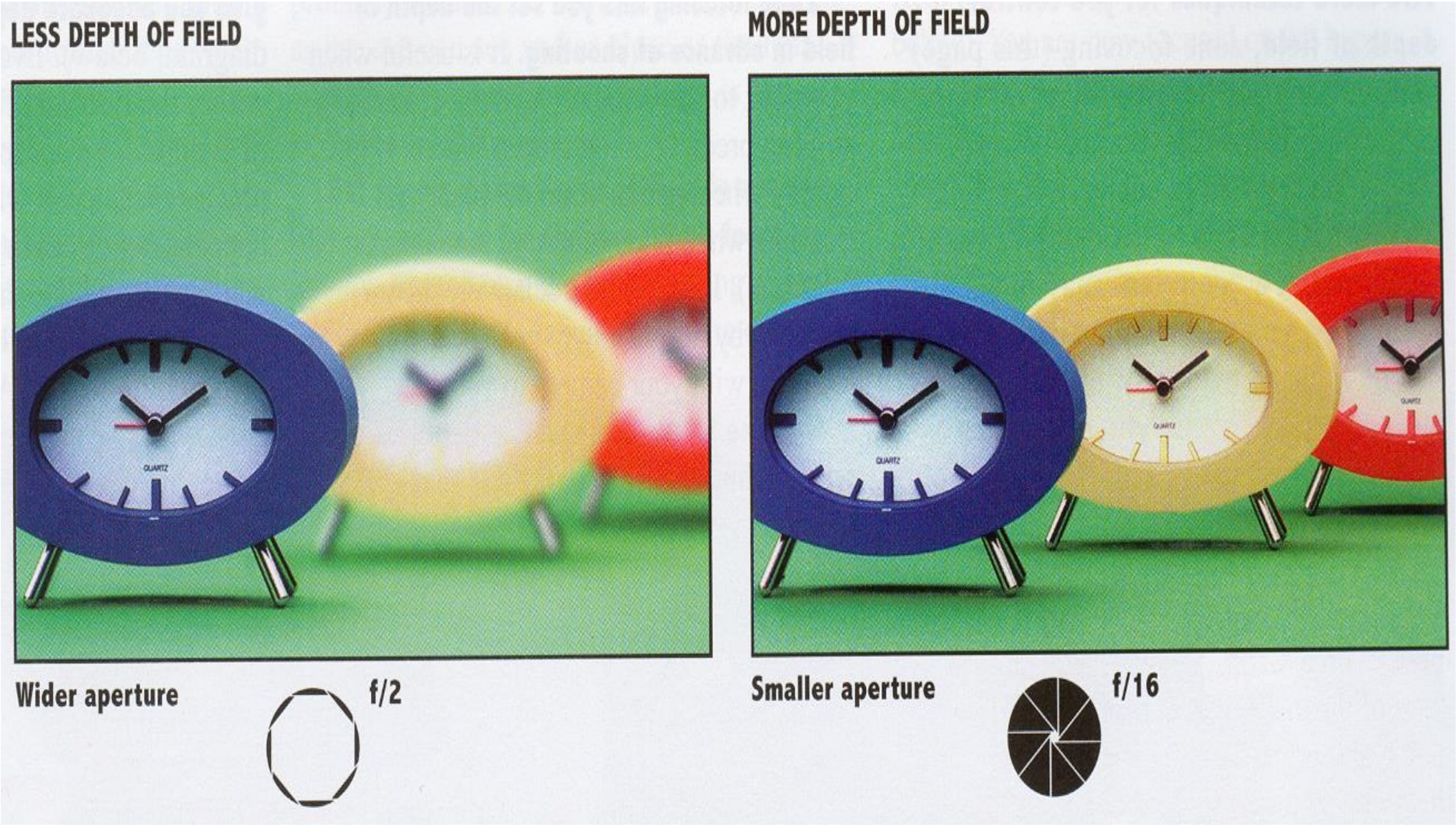


Depth of Field (cont.)

- Changing the aperture size affects depth of field.
  - A smaller aperture increases the range in which the object is approximately in focus



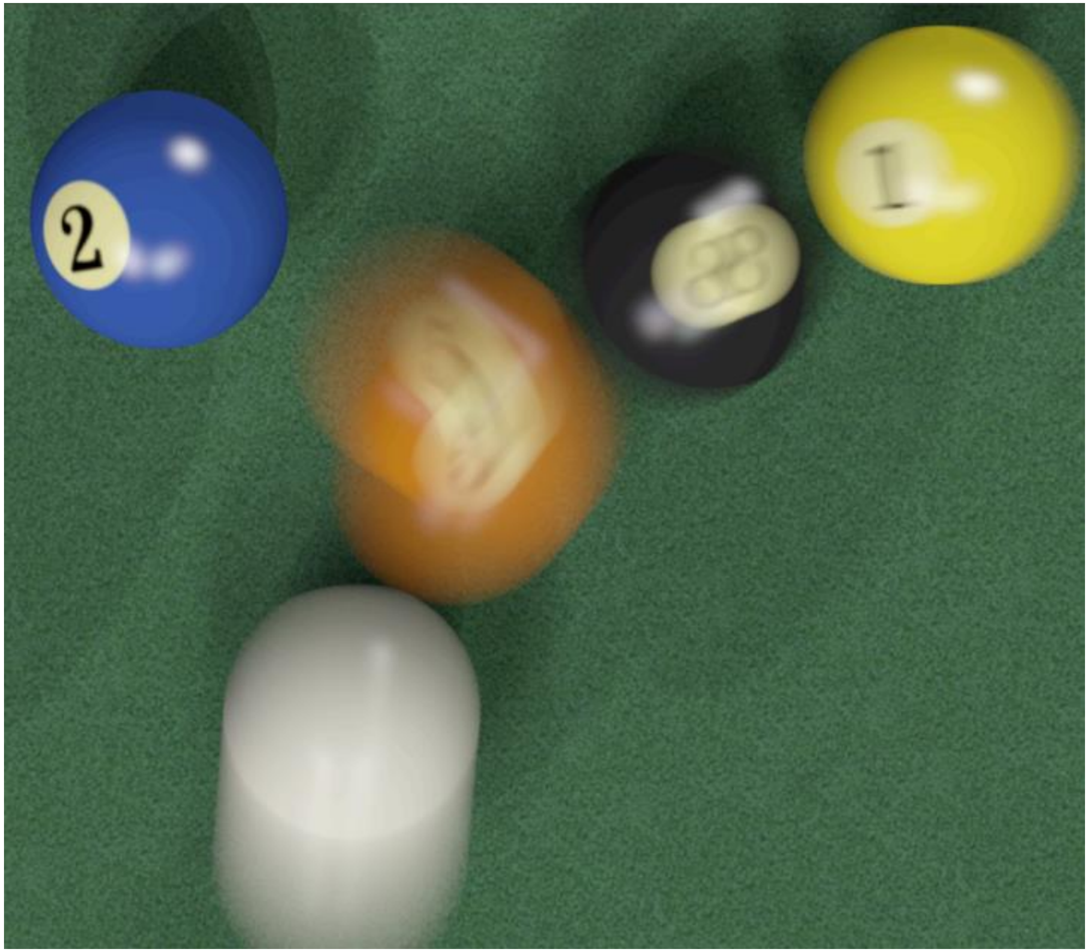
Effect of Depth of Field



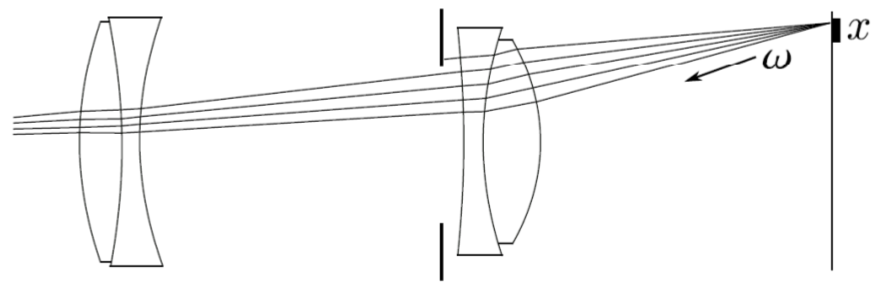
Computer Graphics Camera

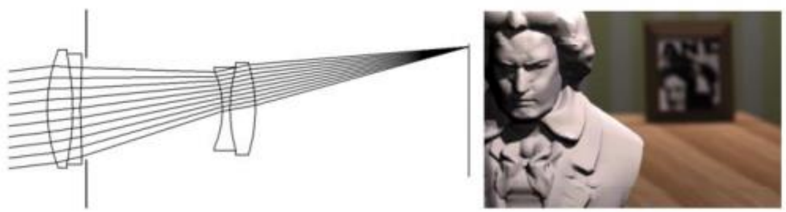
- To mimic the real-world functionality of a real-world camera
- In offline (high-quality) graphics, we can simulate all the imaging processes of a camera using ray tracing

Advanced Simulation of Camera Lens

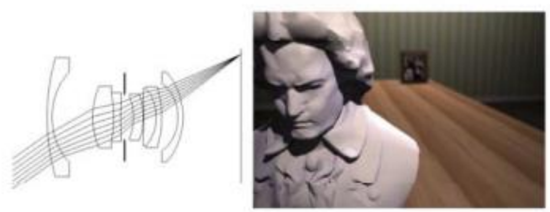


Advanced Simulation of Camera Lens





200 mm telephoto



35 mm wide-angle

50 mm double-gauss

16 mm fisheye

Computer Graphics Camera

- To mimic the real-world functionality of a real-world camera
- In offline (high-quality) graphics, we can simulate all the imaging processes of a camera using ray tracing
- In interactive or real-time graphics, we usually use a **pinhole camera** for its simplicity
  - Every object will always be in-focus
  - Depth of field and motion blur are simulated by other rendering techniques

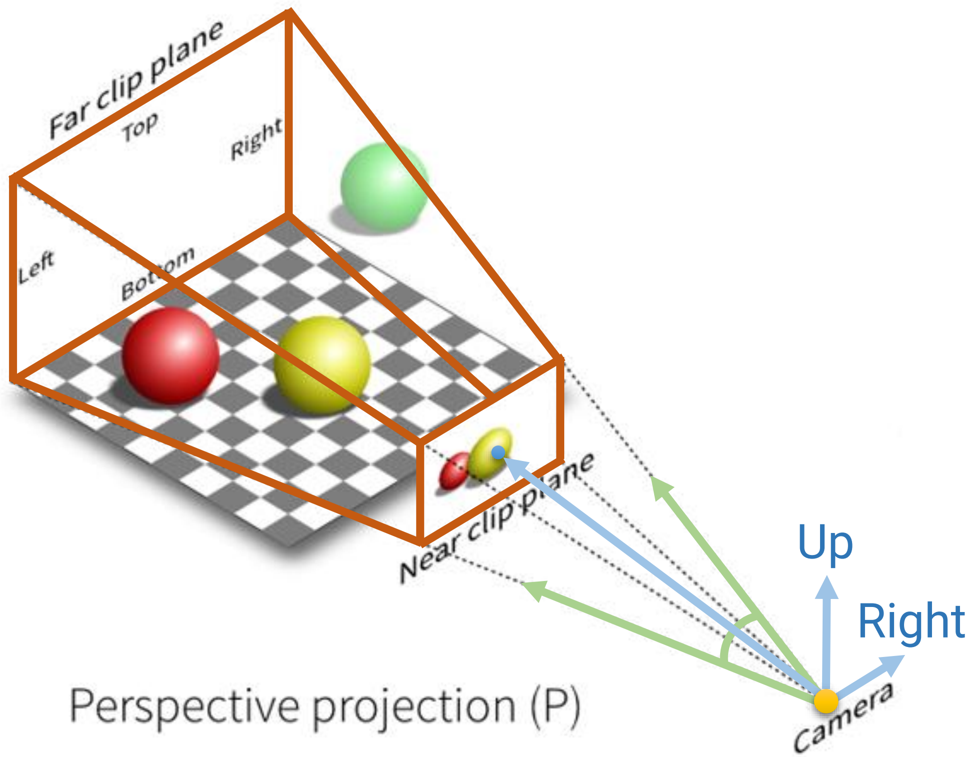
Computer Graphics Camera (cont.)

Camera Properties

- The film is **in front of** the camera (to avoid up-side-down)
- **Basic properties**
  - Camera position
  - Viewing direction
  - Camera local frame
  - Field of view
  - Aspect ratio

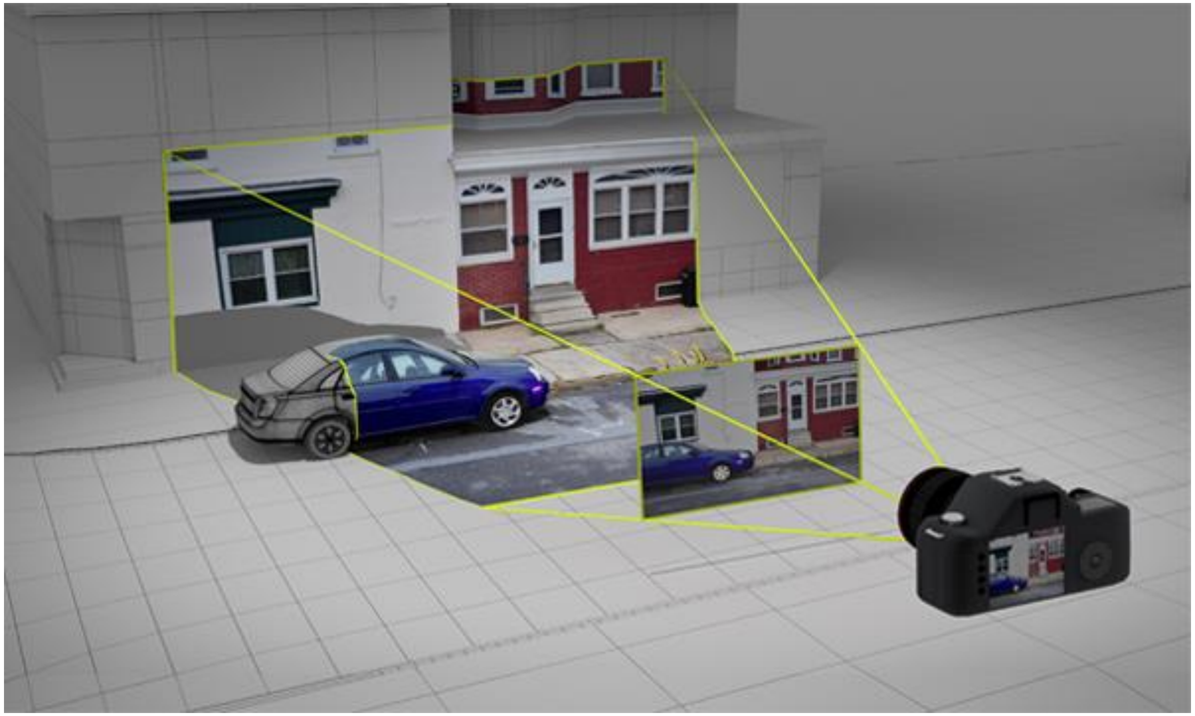
viewing volume
(view frustum)

- **Advanced properties**
  - Shutter speed
  - Lens system



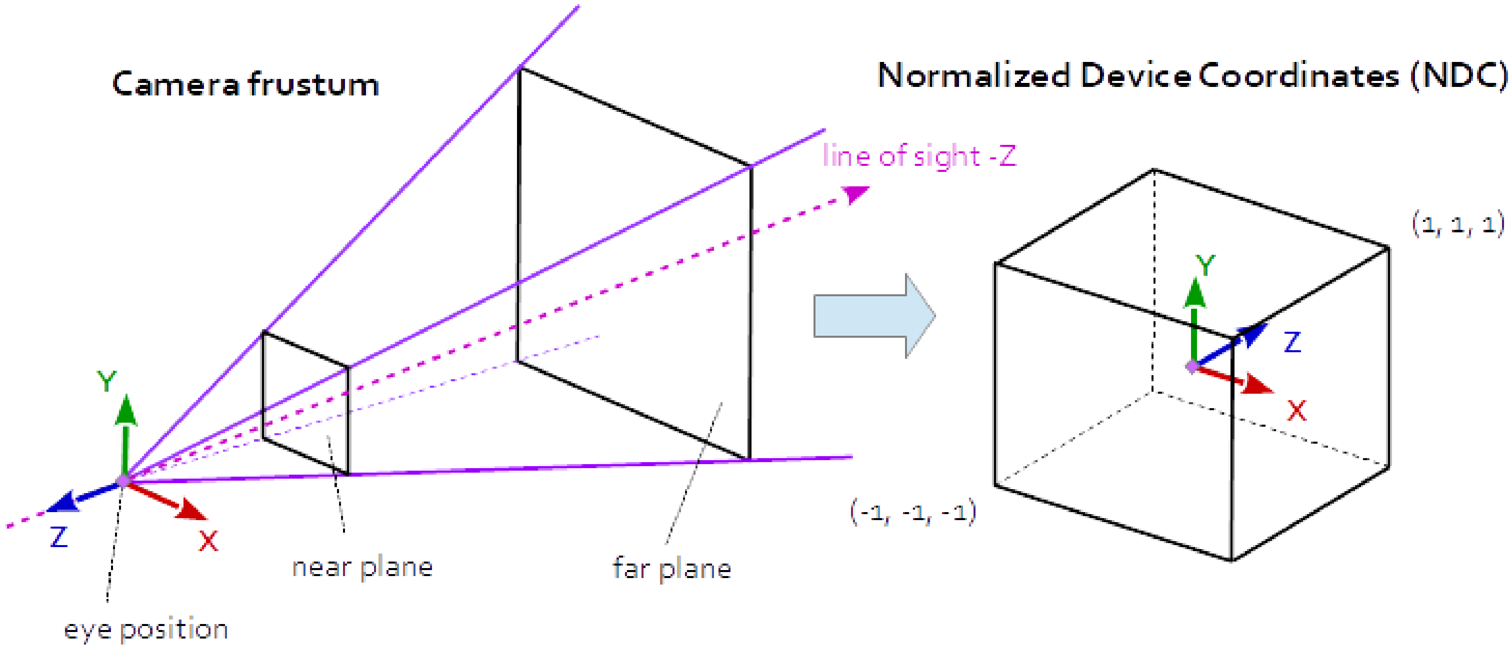
Camera (View) Transform

- The camera can be at an arbitrary position and have an arbitrary viewing direction in the **world space**
- This makes the projection difficult in terms of mathematics



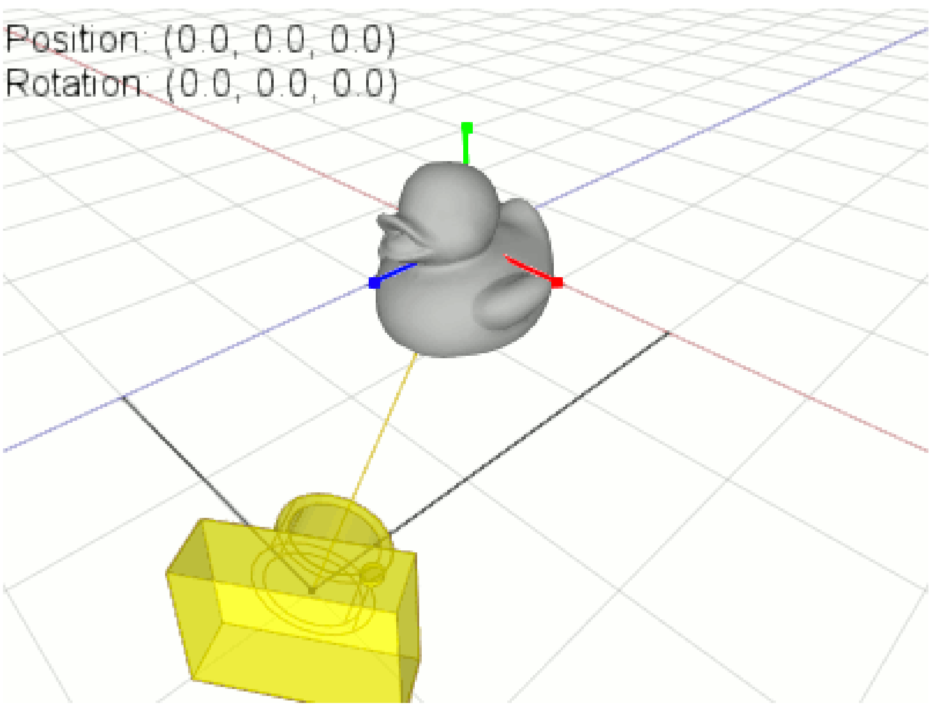
Camera (View) Transform (cont.)

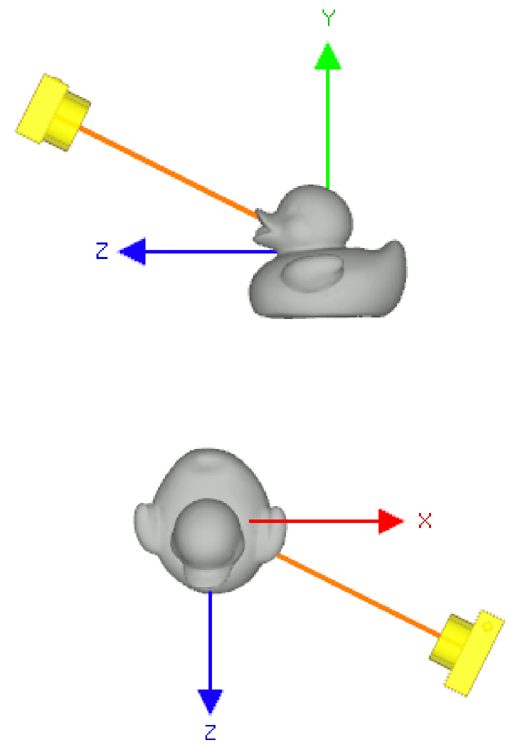
- To keep the math of projection simpler, we additionally define a **camera (view, eye) space**
  - In the camera space, the camera is **at the origin *(0, 0, 0)*** and **looking at the negative Z-axis**



Camera (View) Transform (cont.)

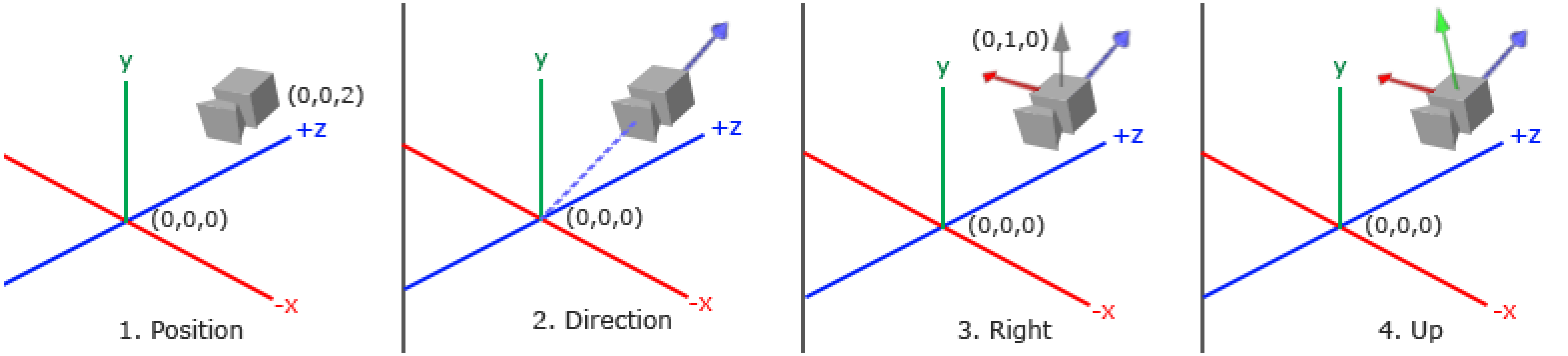
- OpenGL itself is not familiar with the concept of a camera
- Instead, we simulate one by moving all objects in the scene in the reverse direction





Camera (View) Transform (cont.)

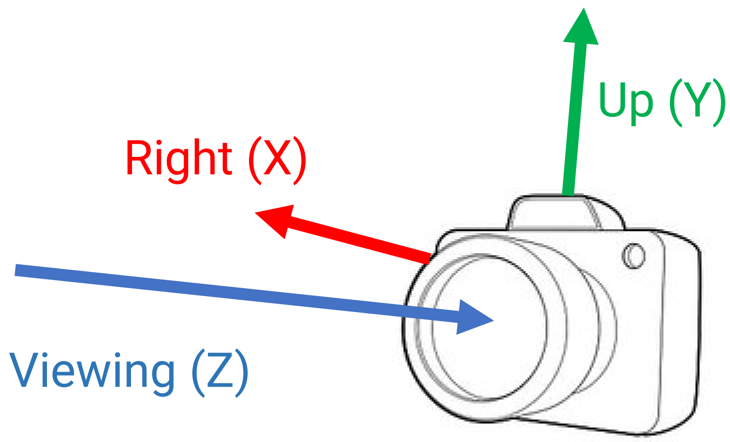
- To do this, we need to define the camera's local frame



- For each object, we transform its world coordinate to the camera coordinate by
  - Moving it with the inverse translation of the camera's position
  - Rotate the object to match the camera's local frame

Camera (View) Transform (cont.)

- **Camera's local frame**
  - Formed by the **view direction (D)**, **right (R)**, and **up (U)** vectors of the camera
  - The three axes of the local frame should be **orthogonal**



Camera (View) Transform (cont.)

- Set camera's local frame
  - However, it is usually difficult for a user to specify an orthogonal basis
  - OpenGL will do it for you (with the **Gram-Schmidt process**)

Camera (View) Transform (cont.)

- Steps for setting camera's local frame
  - Determine the **viewing dir.** with the position of the camera and a target point

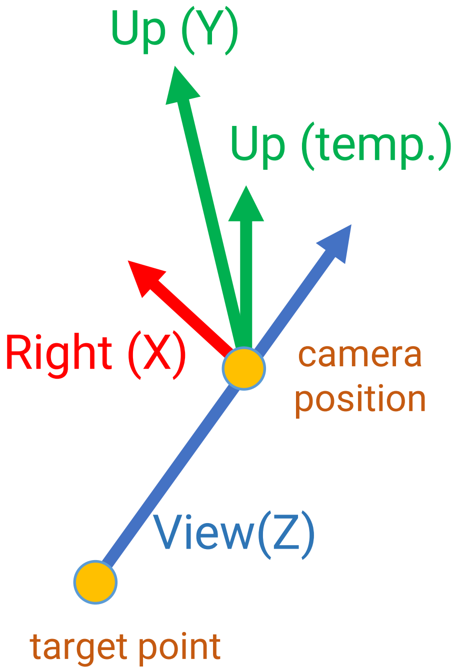
viewing direction = normalize(cameraPos – targetPos)

  - Assume a temporal "up vector"
    - In most cases, we use the up direction (0, 1, 0) in the world frame
  - Obtain the right vector by computing the **cross product** of the **up vector** and the **viewing dir.**

camera right = normalize(cross(up, viewing direction))

  - Obtain the **new up vector** by computing the **cross product** of the **viewing dir.** and the **right vector**

camera up = normalize(cross(viewing direction, camera right))



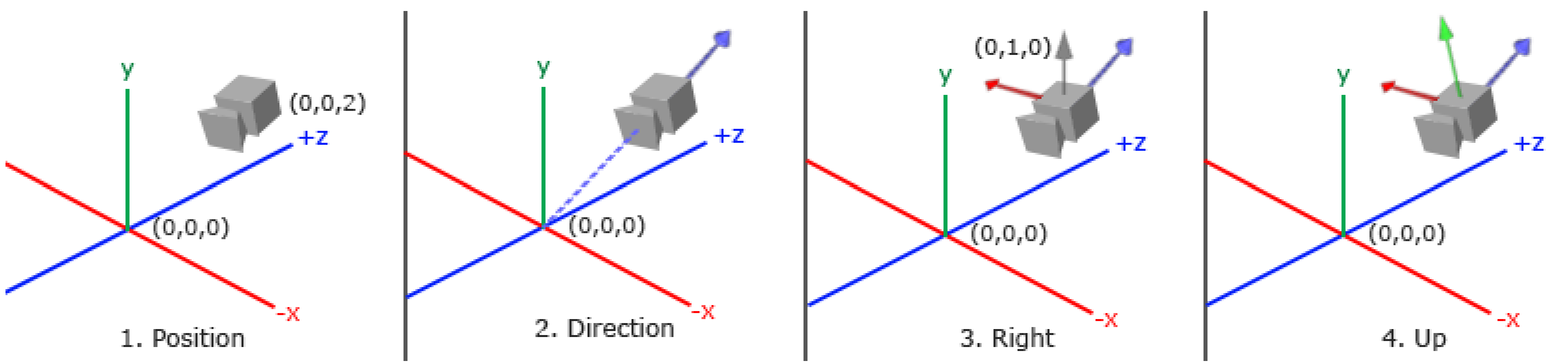
Camera (View) Transform (cont.)

- Camera (view) transformation

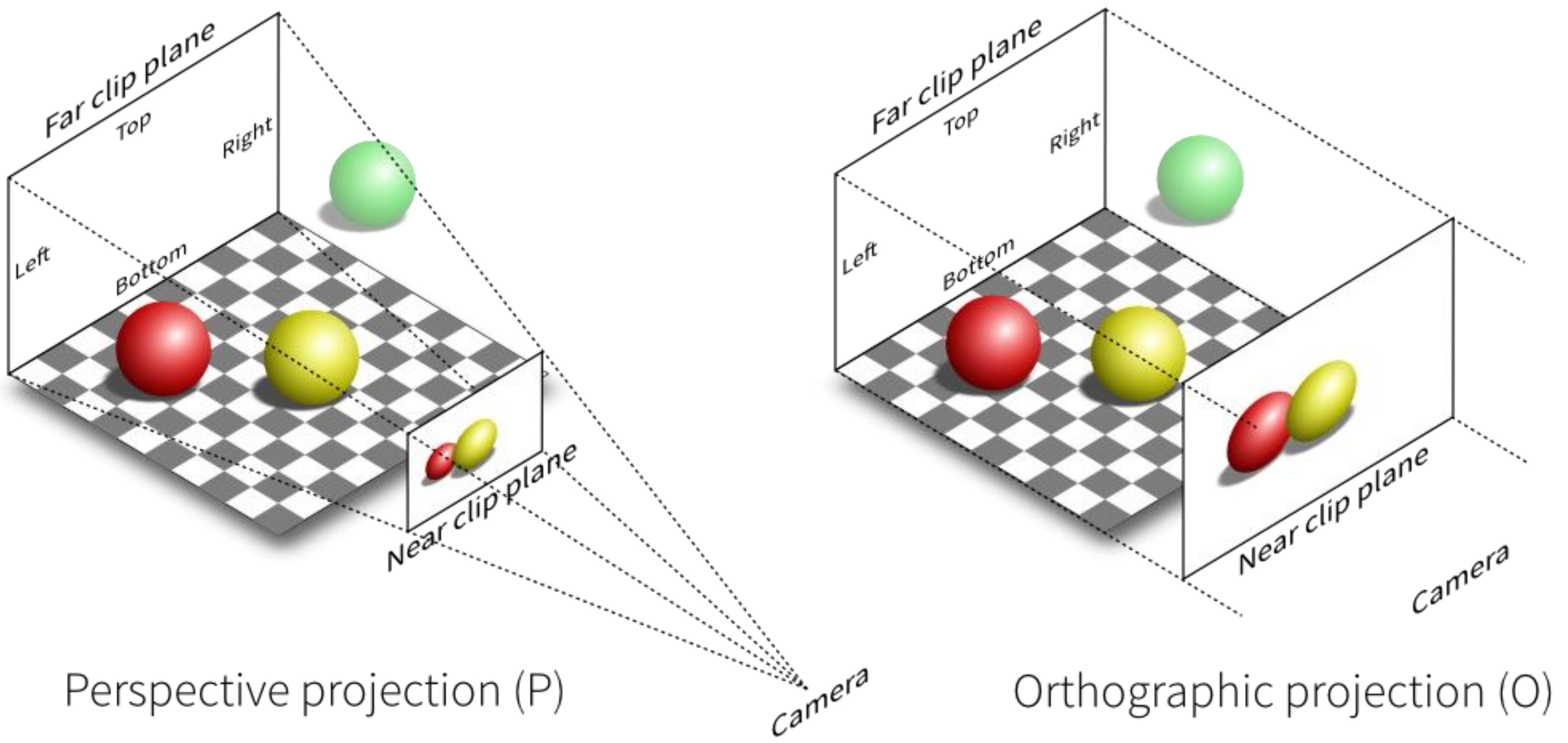
(P_x, P_y, P_z) is the camera's position

$$\begin{array}{r} \text{right vector} \\ \text{up vector} \\ \text{viewing vector} \\ \end{array} \begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

rotation matrix translation matrix

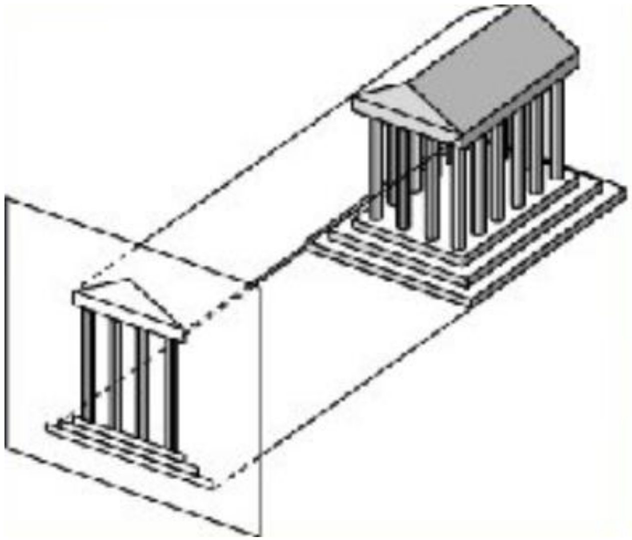


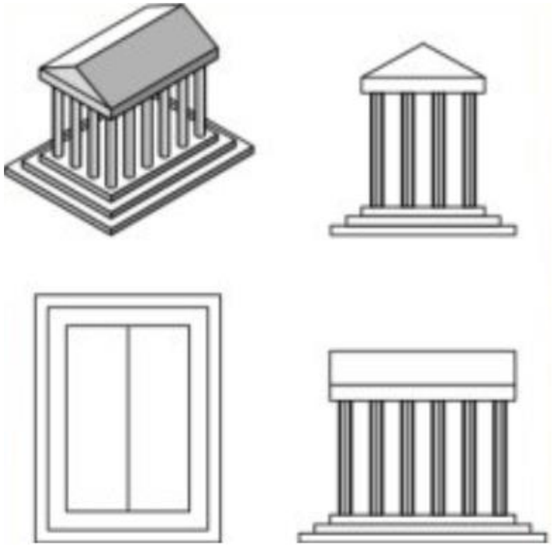
Projective Camera Models



Orthographic Projection

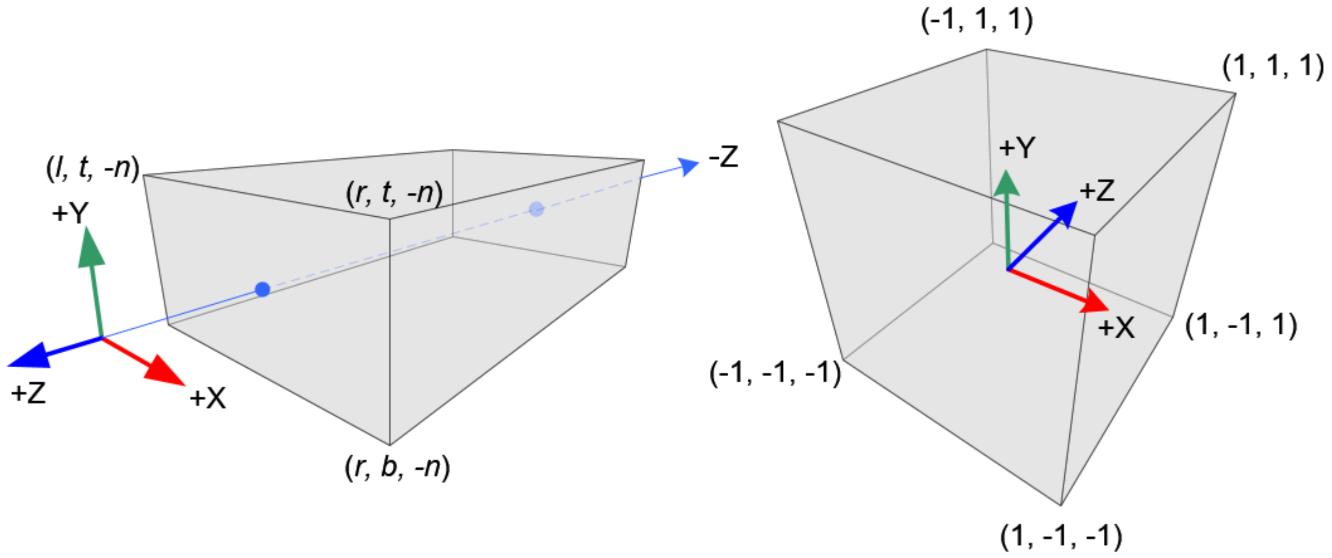
- Parallel projection with projectors perpendicular to the projection plane
- Preserve distance and angle
- Often used as front, side, and top views for 3D design





Orthographic Projection (cont.)

- Need to define the viewing volume with its six planes: left, right, top, bottom, near, and far
  - The viewing volume (frustum) is cube-like
- Map the xyz-coordinate to the range [-1, 1]



Orthographic Projection (cont.)

- Let the ***l***, ***r***, ***t***, ***b***, ***n***, ***f*** be the boundaries of the left, right, top, bottom, near, and far planes

$$l \leq x \leq r \quad \Rightarrow \quad 0 \leq x - l \leq r - l$$

$$\Rightarrow \quad 0 \leq \frac{x-l}{r-l} \leq 1 \quad \Rightarrow \quad 0 \leq 2\left(\frac{x-l}{r-l}\right) \leq 2$$

$$\Rightarrow \quad -1 \leq 2\left(\frac{x-l}{r-l}\right) - 1 \leq 1 \quad \Rightarrow \quad -1 \leq \frac{2\boxed{x}}{r-l} - \frac{r+l}{r-l} \leq 1$$

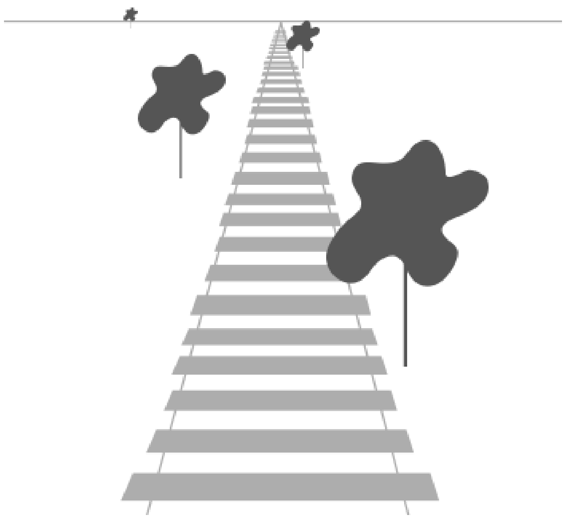
Orthographic Projection (cont.)

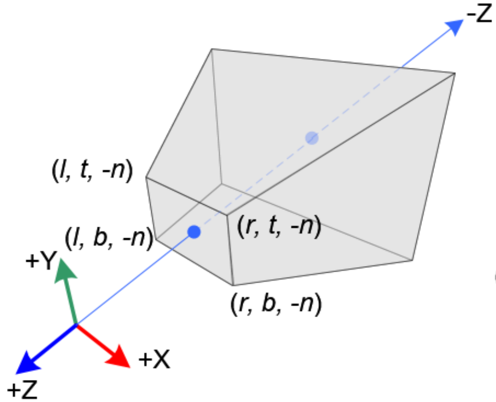
- Let the ***l***, ***r***, ***t***, ***b***, ***n***, ***f*** be the boundaries of the left, right, top, bottom, near, and far planes
- An orthographic projection matrix can be written as

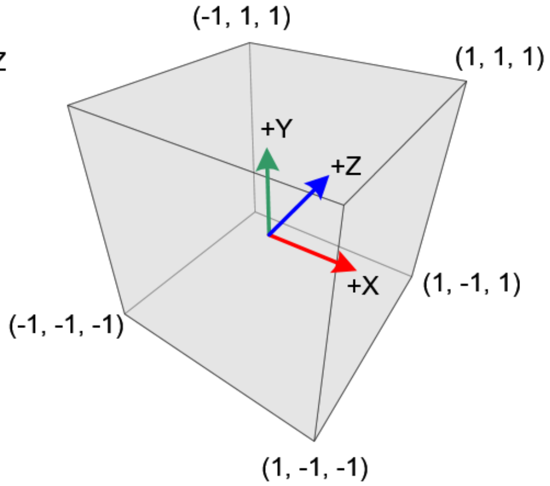
$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Perspective Projection

- In our real lives, the objects that are farther away appear much smaller
- This effect is called **perspective**
- A perspective projection tries to mimic the vision of human eyes





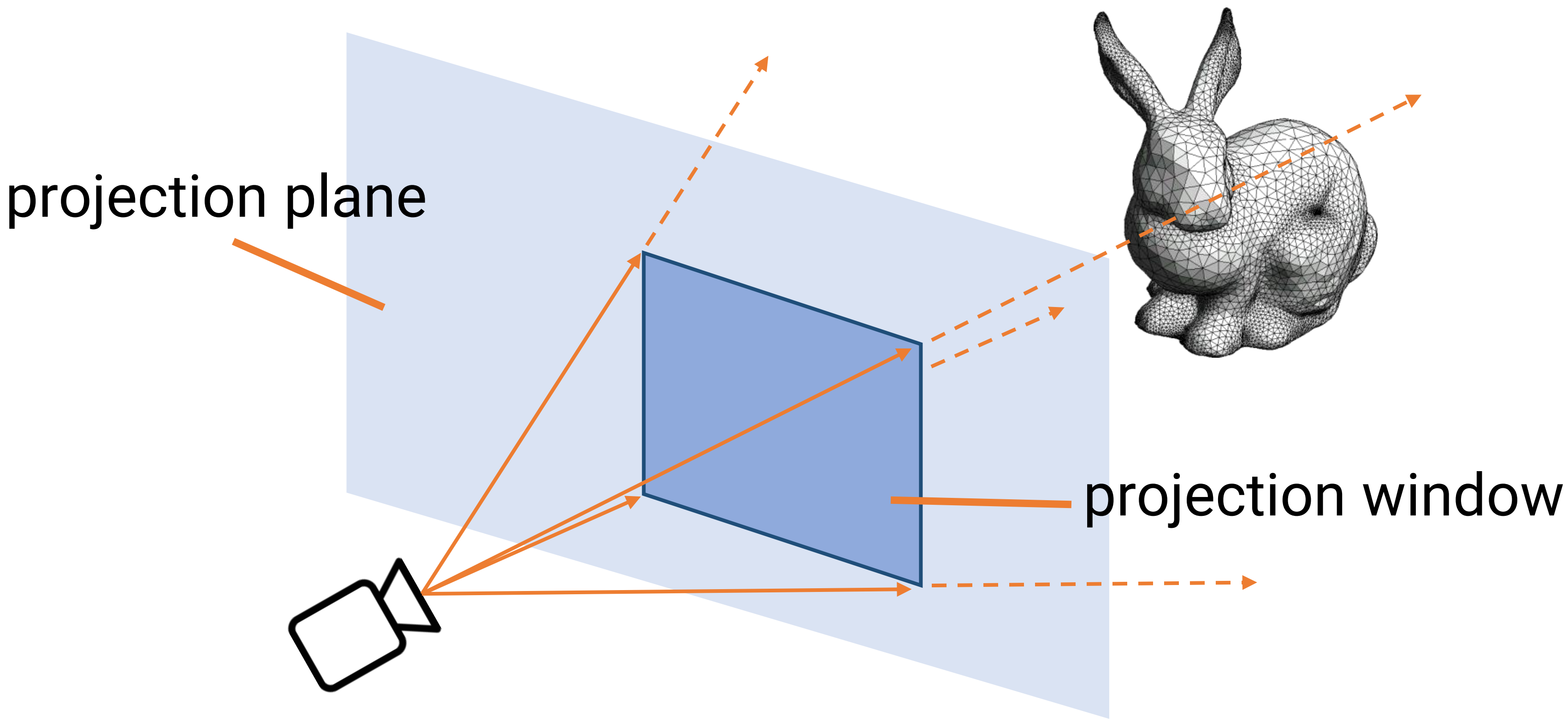


Perspective Projection (cont.)

- Four components for the perspective projection matrix
  - **The aspect ratio of the screen**
    - The ratio between the width and the height
  - **The vertical field of view**
    - The vertical angle of the camera through which we are looking at the world
  - **The location of the near Z plane**
    - Used to clip objects that are too close to the camera
  - **The location of the far Z plane**
    - Used to clip objects that are too distant from the camera

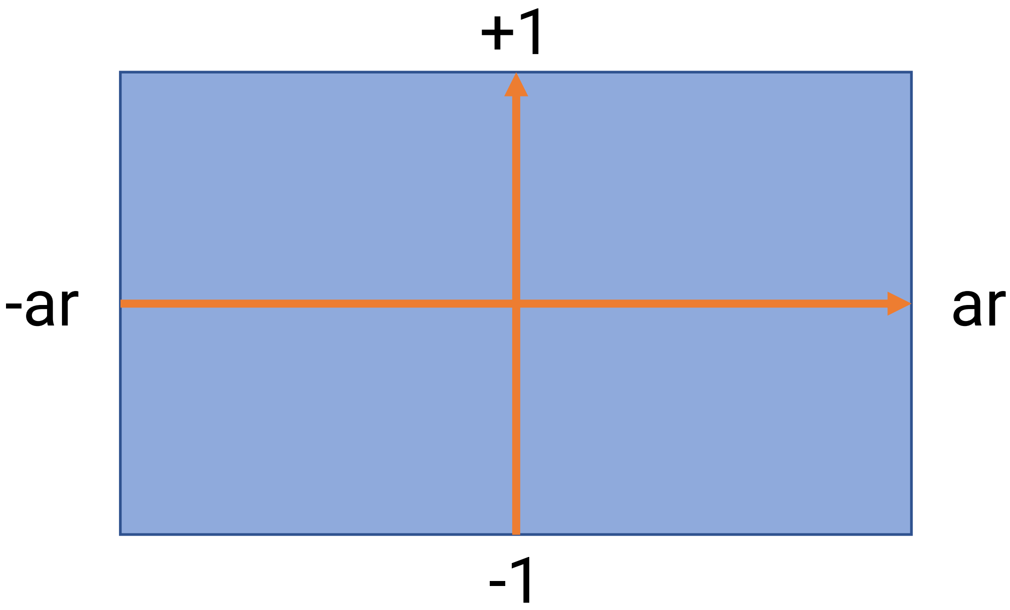
Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - The projection plane and the projection window



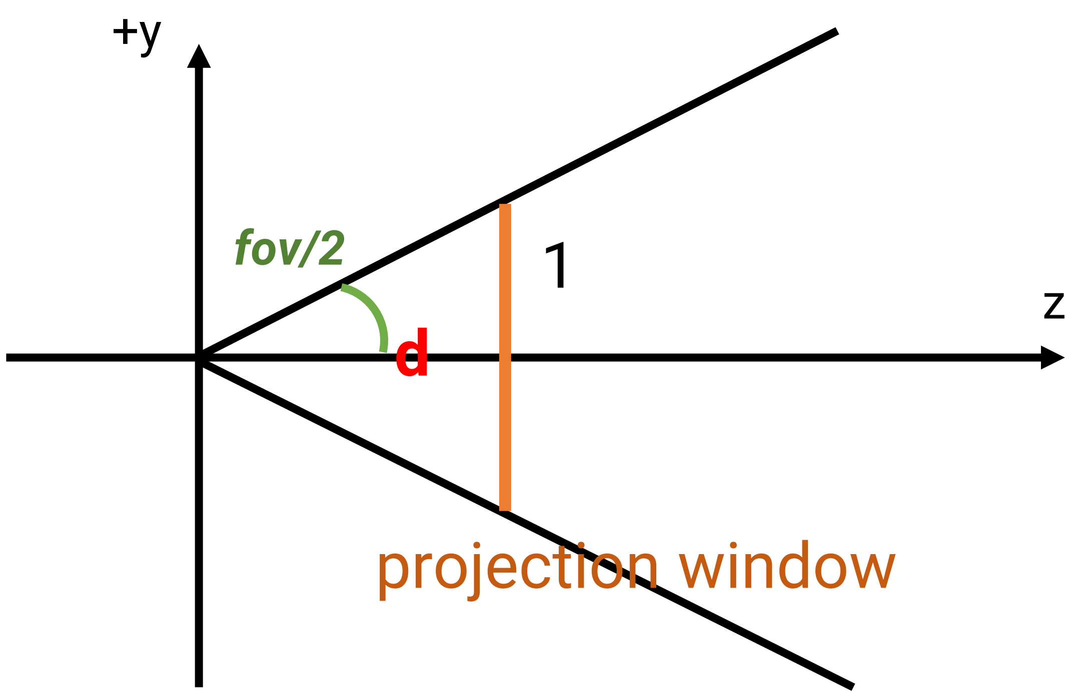
Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Determine the height of the projection window as 2
  - The width of the projection window becomes 2 times the aspect ratio (ar)



Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - We can determine the distance from the camera to the projection window based on the field of view (fov)

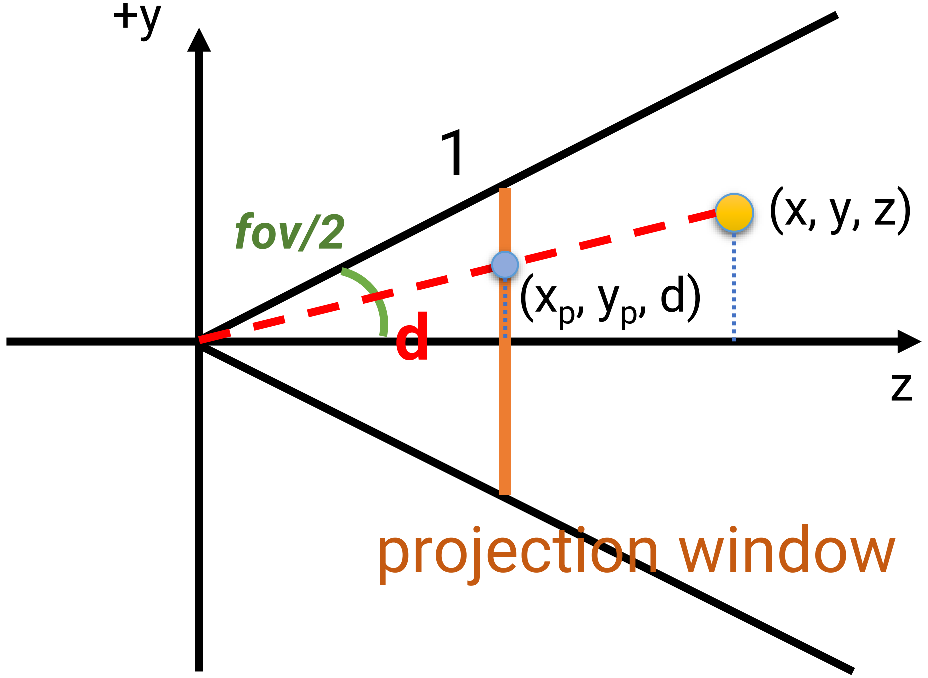


$$\tan\left(\frac{\alpha}{2}\right) = \frac{1}{d}$$

$$\Rightarrow d = \frac{1}{\tan\left(\frac{\alpha}{2}\right)}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Assume we want to find the projected coordinate (x_p, y_p) of a 3D point (x, y, z)
  - The y component can be derived as …



$$\frac{y_p}{d} = \frac{y}{-z}$$

$$\Rightarrow \quad y_p = \frac{y \cdot d}{-z}$$

$$\Rightarrow \quad y_p = \frac{y}{-z \cdot \tan(\frac{\alpha}{2})}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Do the same derivation for the x component
    - Note in the x-direction we have to multiply the aspect ratio ***ar***
  - After that, we can obtain the following equations

$$x_p = \frac{x}{ar \cdot (-z) \cdot \tan(\frac{\alpha}{2})}$$

$$y_p = \frac{y}{-z \cdot \tan(\frac{\alpha}{2})}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Fill-in the matrix, based on the following conditions

$$x_p = \frac{x}{ar \cdot (\boxed{-z}) \cdot \boxed{\tan(\frac{\alpha}{2})}} \qquad y_p = \frac{y}{\boxed{-z} \cdot \boxed{\tan(\frac{\alpha}{2})}}$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w \end{bmatrix} = \begin{bmatrix} \longleftarrow \mathbf{f(x)} \longrightarrow \\ \longleftarrow \mathbf{f(y)} \longrightarrow \\ \longleftarrow \mathbf{f(z)} \longrightarrow \\ \longleftarrow \mathbf{f(w)} \longrightarrow \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
    - Fill-in the matrix, based on the following conditions

$$x_p = \frac{x}{ar \cdot (\boxed{-z}) \cdot \boxed{\tan(\frac{\alpha}{2})}} \qquad y_p = \frac{y}{\boxed{-z} \cdot \boxed{\tan(\frac{\alpha}{2})}}$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w \end{bmatrix} = \left[\begin{array}{cccc} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ \multicolumn{4}{c}{\longleftarrow\quad \mathbf{f(z)} \quad\longrightarrow} \\ 0 & 0 & -1 & 0 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Fill-in the matrix, based on the following conditions
    - Assume the *Z* function has a shape $f(z) = A(-z) + B$
    - After perspective division, it becomes

$$f(z) = A - \frac{B}{z}$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ w \end{bmatrix} = \begin{bmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Fill-in the matrix, based on the following conditions

$$f(-nearZ) = -1 \Rightarrow A - \frac{B}{-nearZ} = -1 \Rightarrow A = -1 - \frac{B}{nearZ}$$

$$f(-farZ) = 1 \Rightarrow A - \frac{B}{-farZ} = 1 \Rightarrow A = 1 - \frac{B}{farZ}$$

$$2 = \frac{B}{farZ} - \frac{B}{nearZ}$$

$$\Rightarrow \frac{B \cdot nearZ - B \cdot farZ}{farZ \cdot farZ} = 2$$

$$\Rightarrow B(nearZ - farZ) = 2 \cdot farZ \cdot farZ$$

$$\Rightarrow \boxed{\begin{aligned} B &= \frac{2 \cdot farZ \cdot farZ}{nearZ - farZ} \\ A &= \frac{-nearZ - farZ}{nearZ - farZ} \end{aligned}}$$

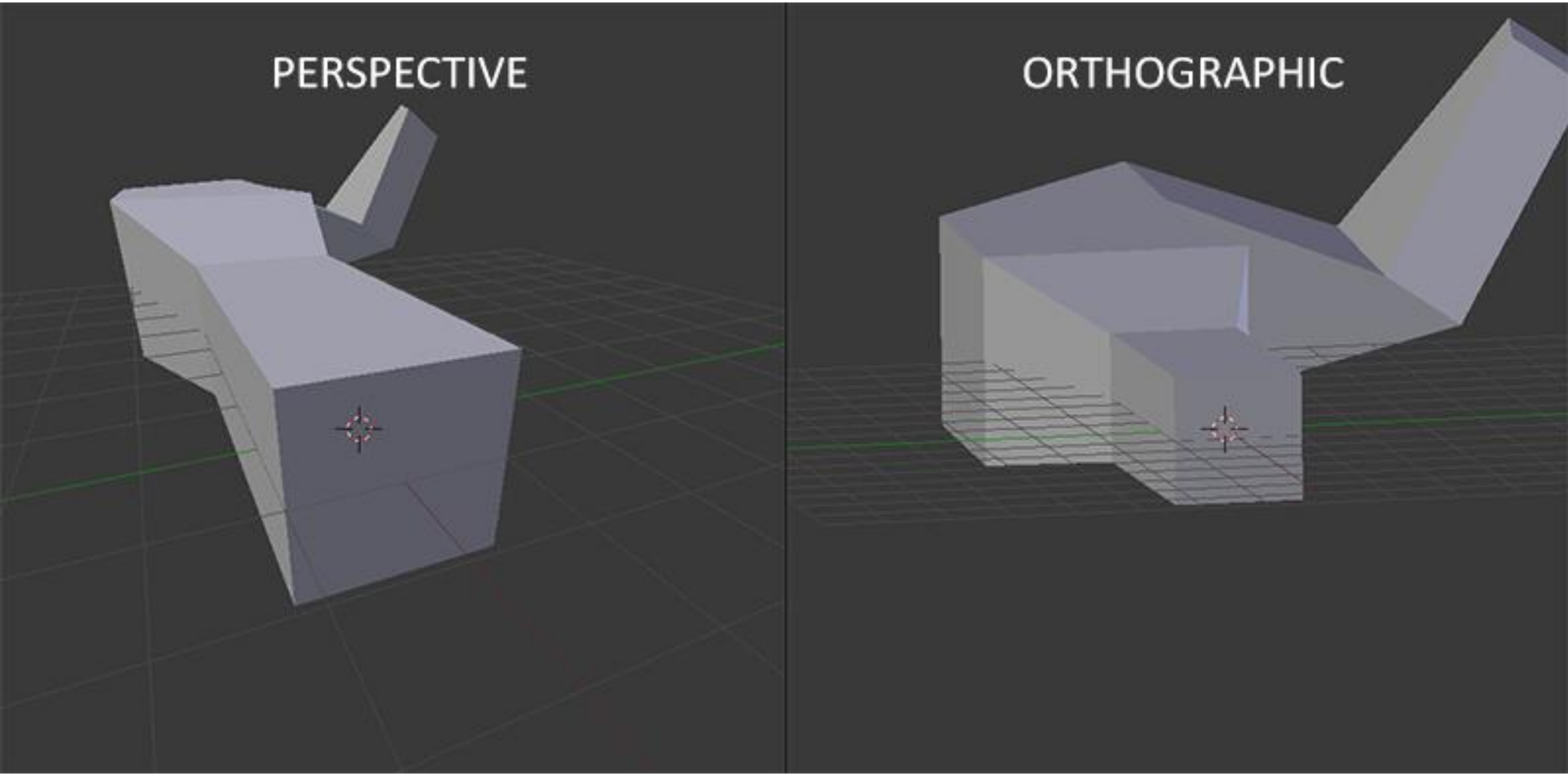
Perspective Projection (cont.)

- Derivation of the perspective projection matrix
  - Fill-in the matrix, based on the following conditions

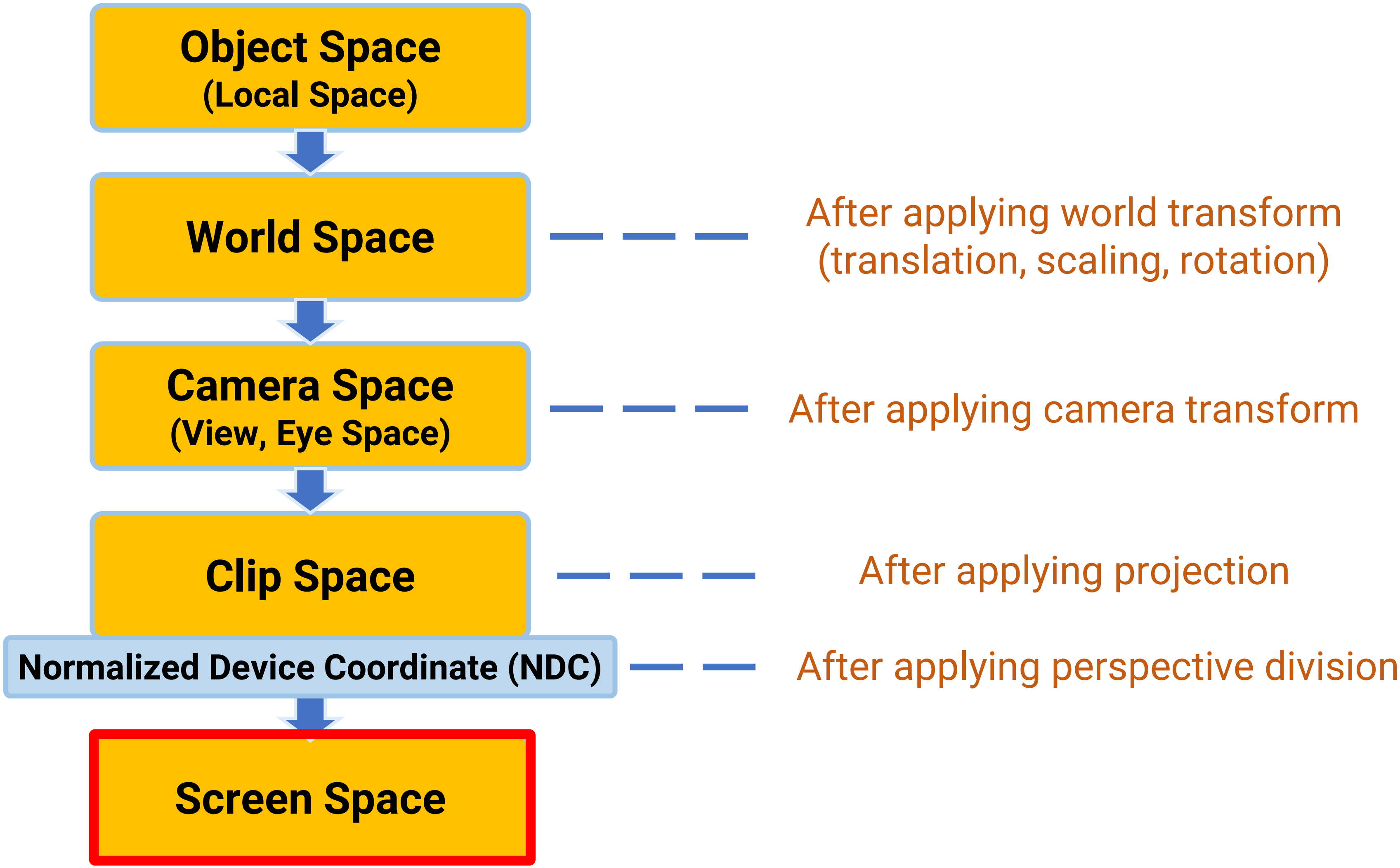
$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-nearZ - farZ}{nearZ - farZ} & \frac{2 \cdot farZ \cdot nearZ}{nearZ - farZ} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Camera Models Comparison

Camera Models Comparison (cont.)



The Full Vertex Transform Pipeline



Any Questions?