Camera

Computer Graphics

Yu-Ting Wu

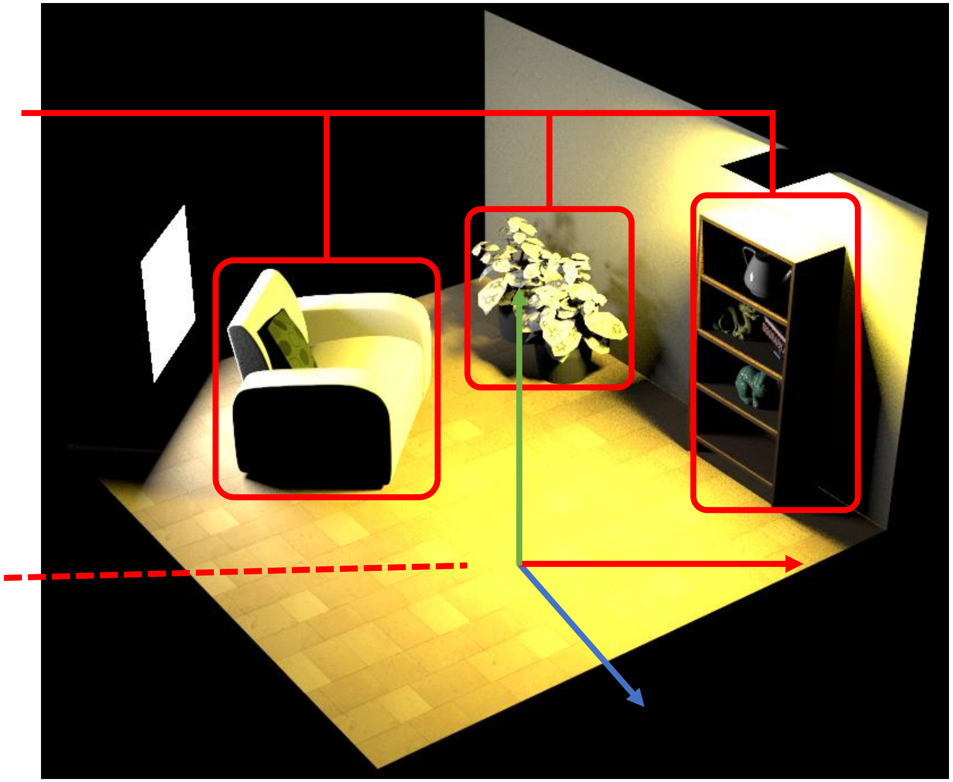
(Some of this slides are borrowed from Prof. Yung-Yu Chuang)

Outline

- Introduction to real-world cameras
- Introduction to computer graphics cameras
- Camera space and camera transformation
- Projective cameras
- OpenGL Implementation

Recap.

- So far, we have introduced how to represent a virtual 3D world



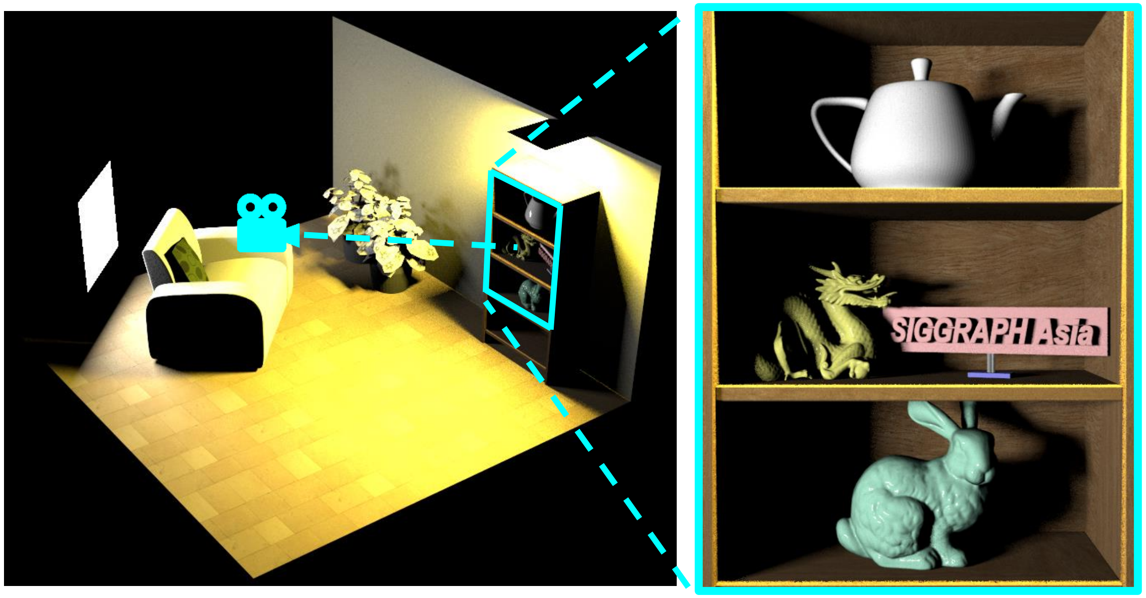
Sofa, plant, bookshelf, and the room
vertex data ➔ (vertex buffer)
vertex adjacency ➔ (index buffer)
defined in **Object Space**

Objects are put into a shared
World Space
by **transformation**
(translation, scaling, rotation)

3D virtual world

Recap. (cont.)

- 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

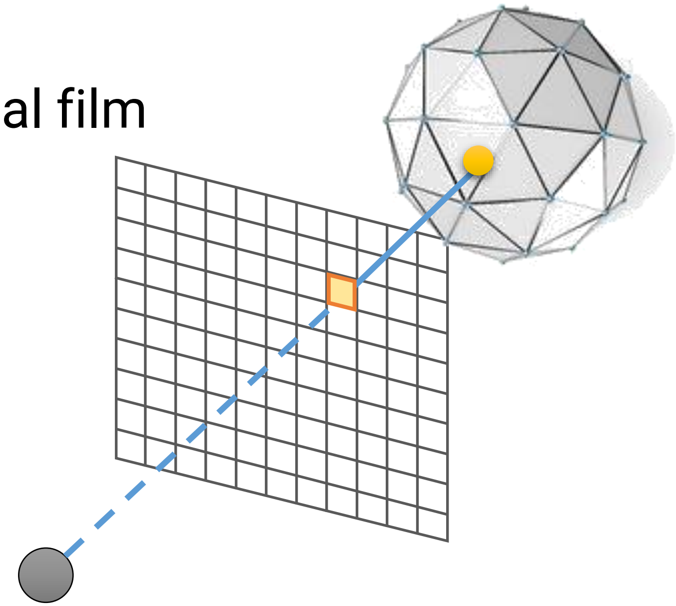


3D virtual world rendered image

Recap. (cont.)

- Two ways for generating synthetic images

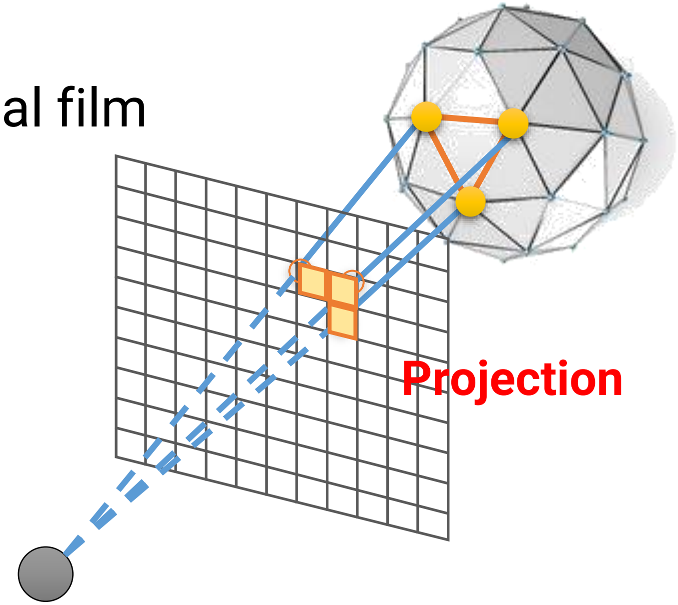
Ray tracing



virtual film

virtual camera

Rasterization



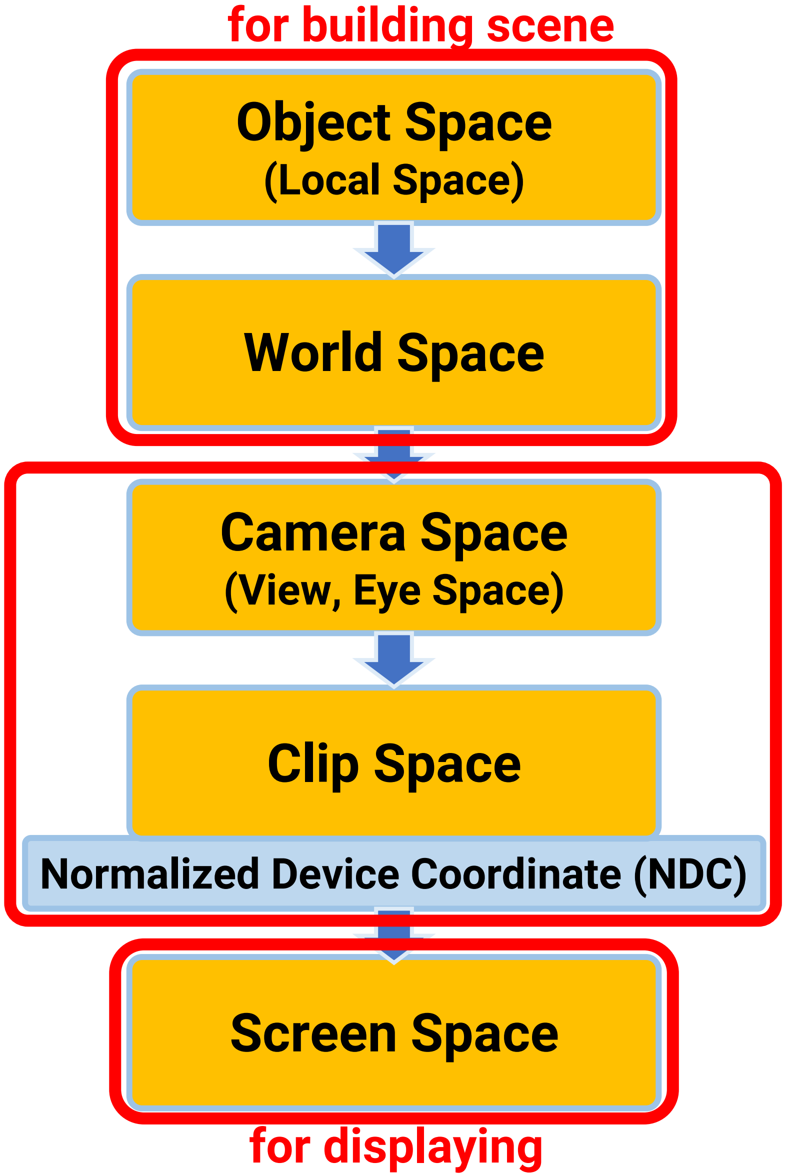
virtual film

Projection

virtual camera

Spoiler

- There are other spaces
- We will introduce **camera space**, **clip space**, and **NDC** today



for building scene

Object Space
(Local Space)

World Space

for assisting rendering

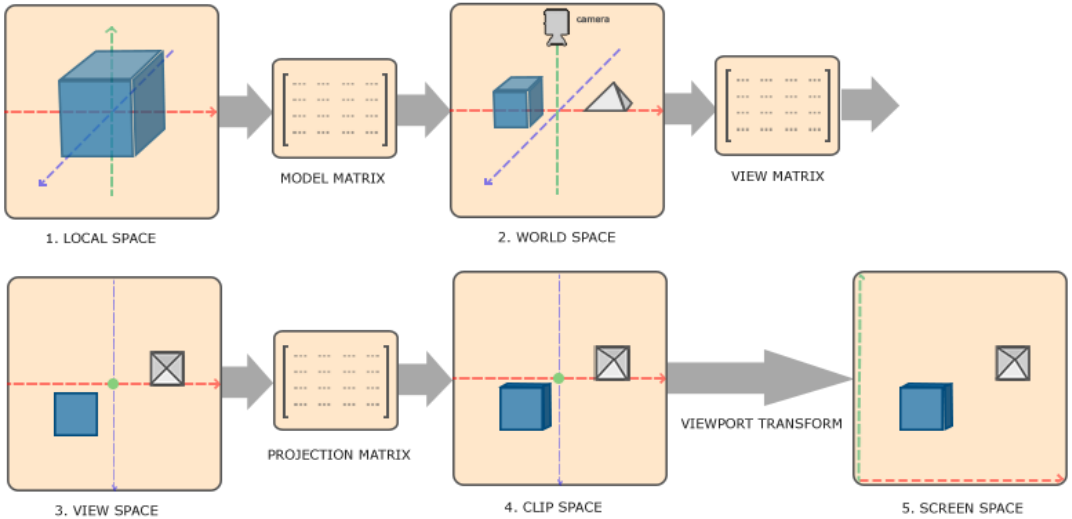
Camera Space
(View, Eye Space)

Clip Space

Normalized Device Coordinate (NDC)

Screen Space

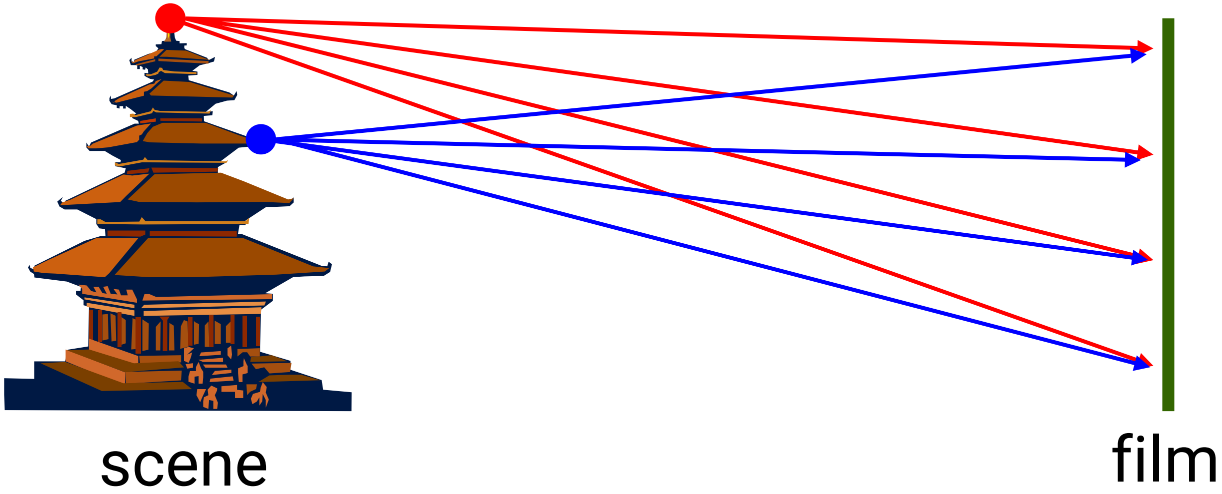
for displaying



Outline

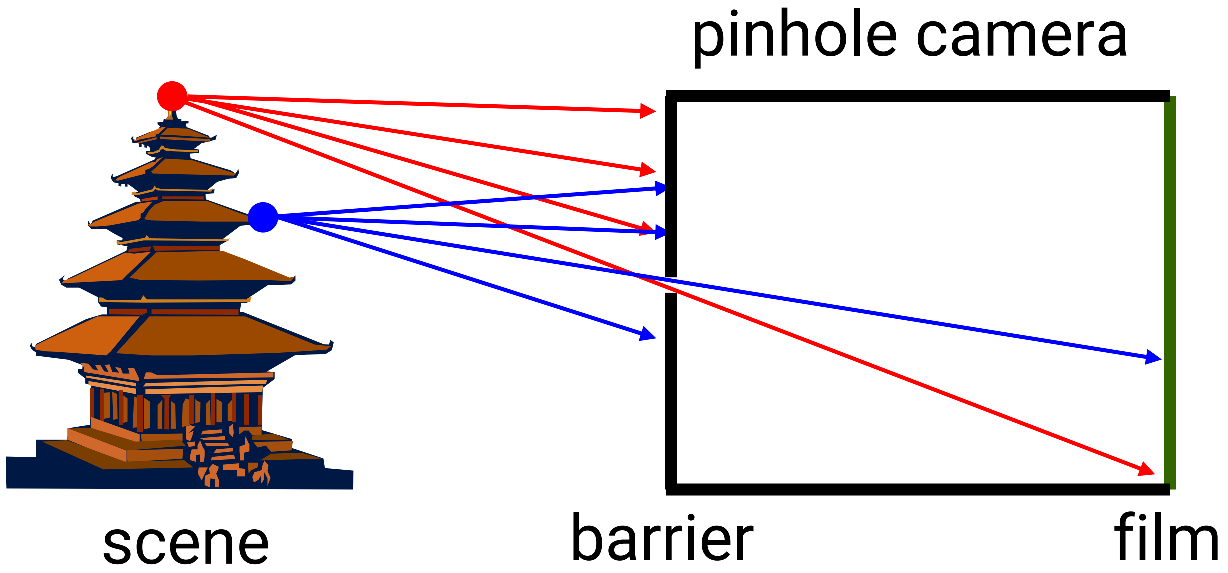
- **Introduction to real-world cameras**
- Introduction to computer graphics cameras
- Camera space and camera transformation
- Projective cameras
- OpenGL Implementation

Camera Trail



Put a piece of film in front of an object

Pinhole Camera

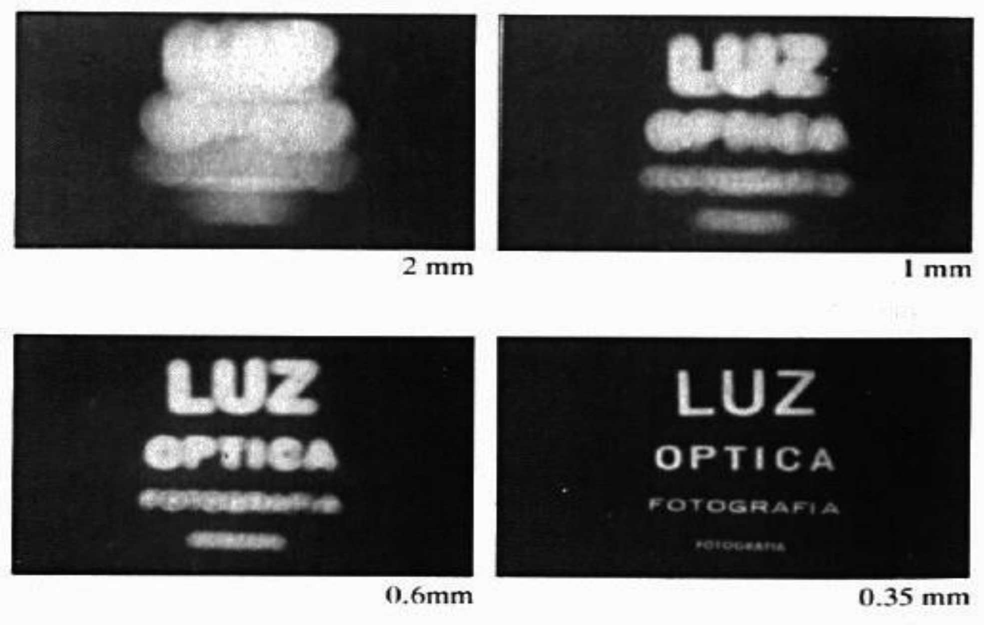


Add a barrier to block off most of the rays

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

Pinhole Camera (cont.)

- Shrink the aperture

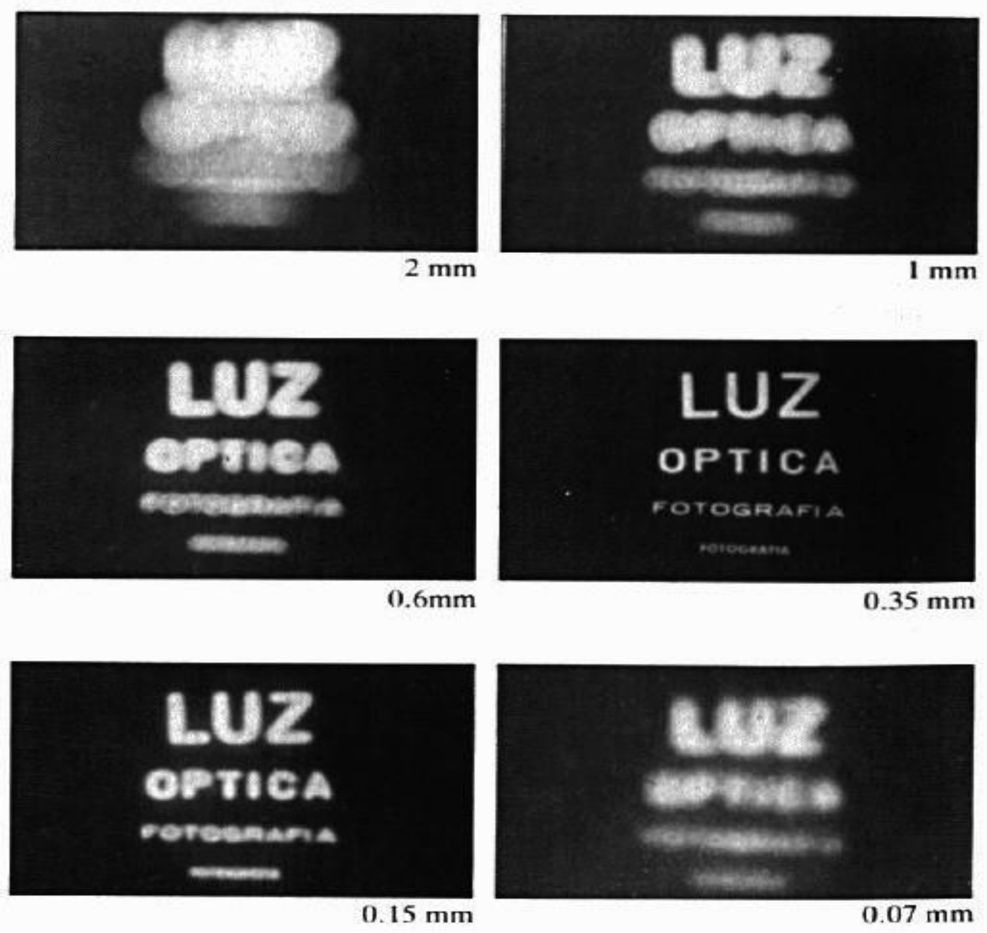


Why not make the aperture as small as possible?

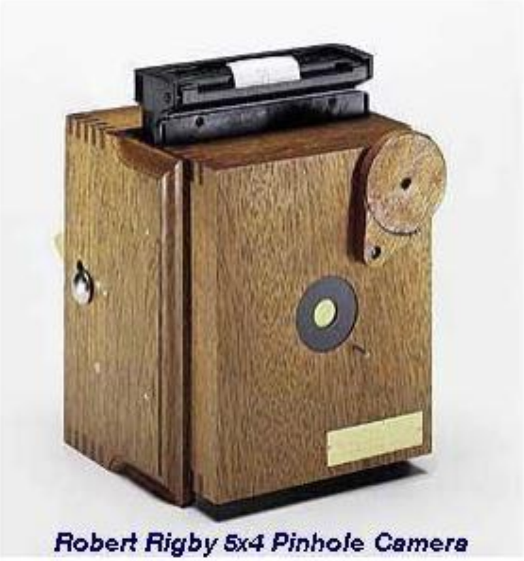
- Less light gets through
- Diffraction effect

Pinhole Camera (cont.)

- Shrink the aperture



Pinhole Camera (cont.)

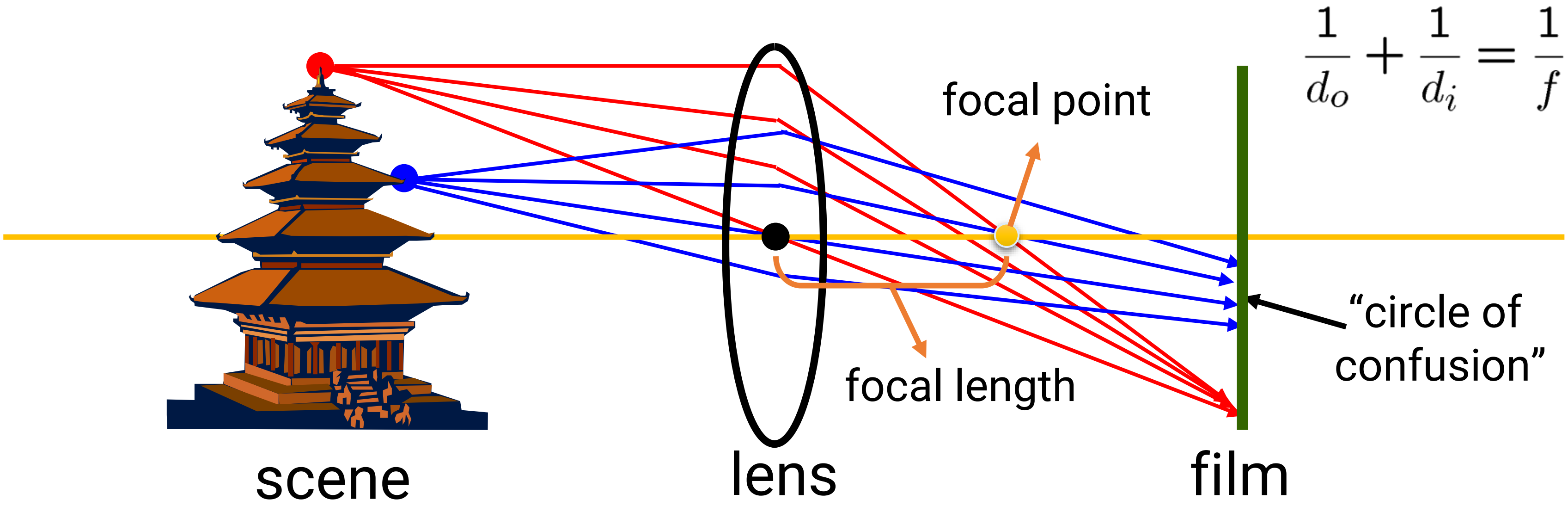


Robert Rigby 5x4 Pinhole Camera

$200~$700

© 2002 HEIDI CRABBE

Camera with Lens



$$\frac{1}{d_o} + \frac{1}{d_i} = \frac{1}{f}$$

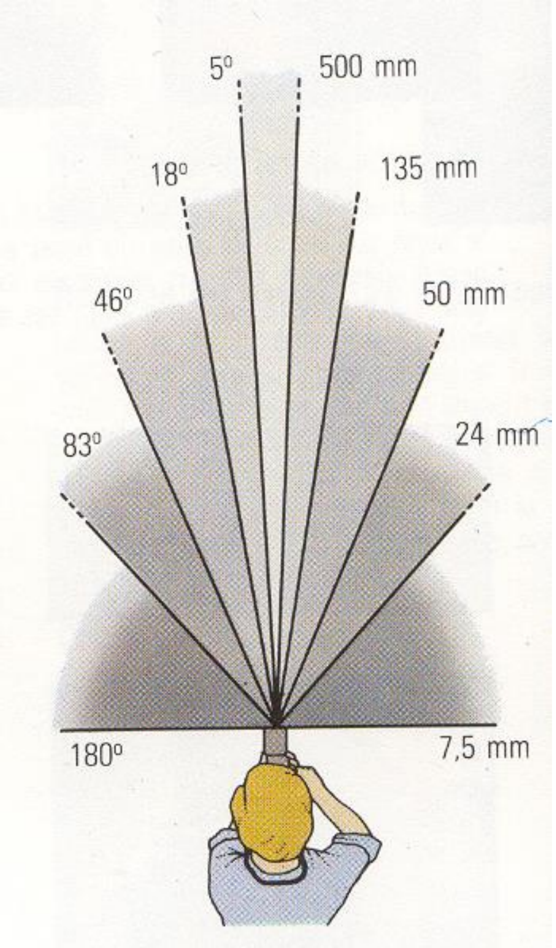
A lens **focuses** light onto the film

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

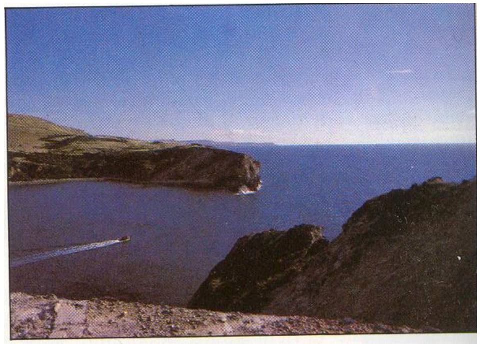
Current digital cameras replace the film with a **sensor array** (CCD or CMOS)

Camera with Lens (cont.)

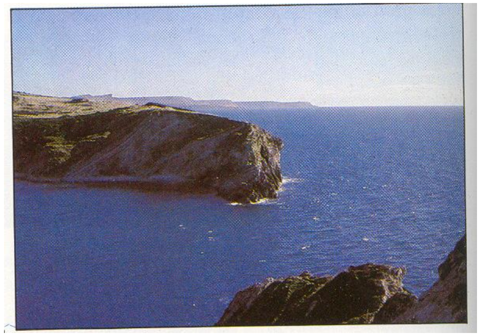
field of view (FOV)



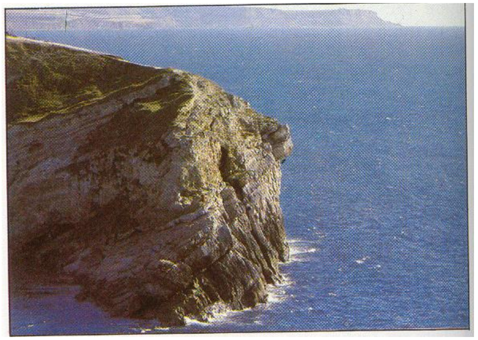
24mm



50mm

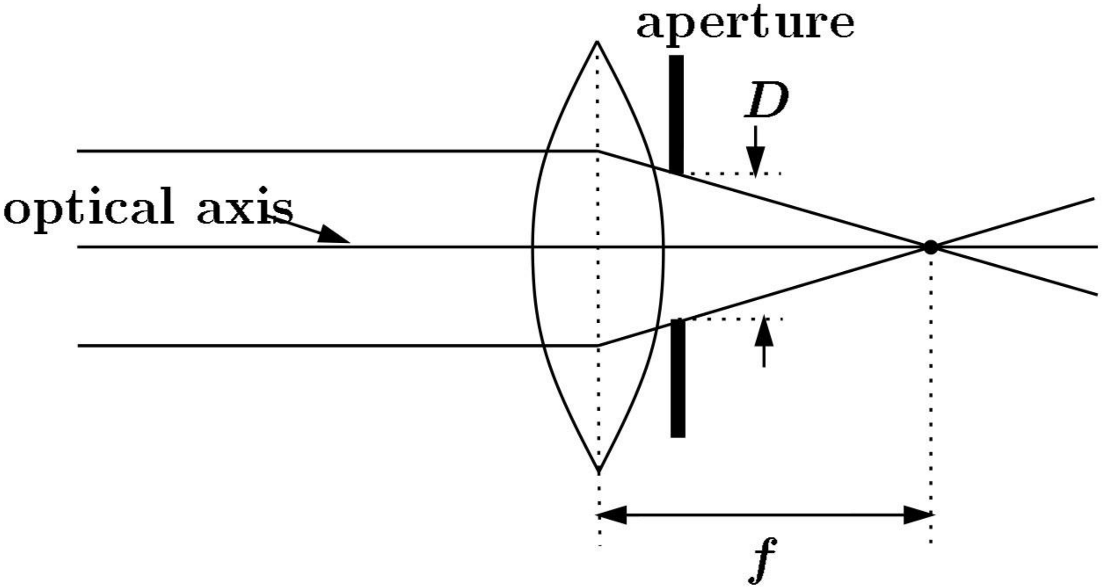


135mm



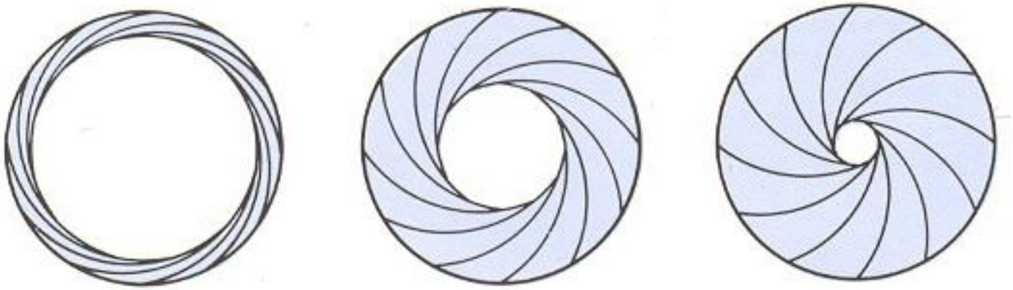
Exposure

- **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



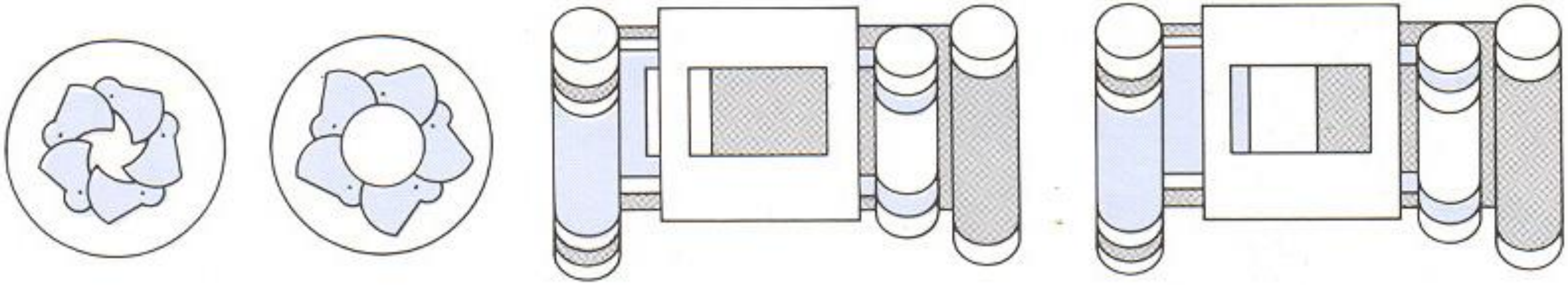
Exposure

- Aperture (in f stop)



Full aperture
Medium aperture
Stopped down

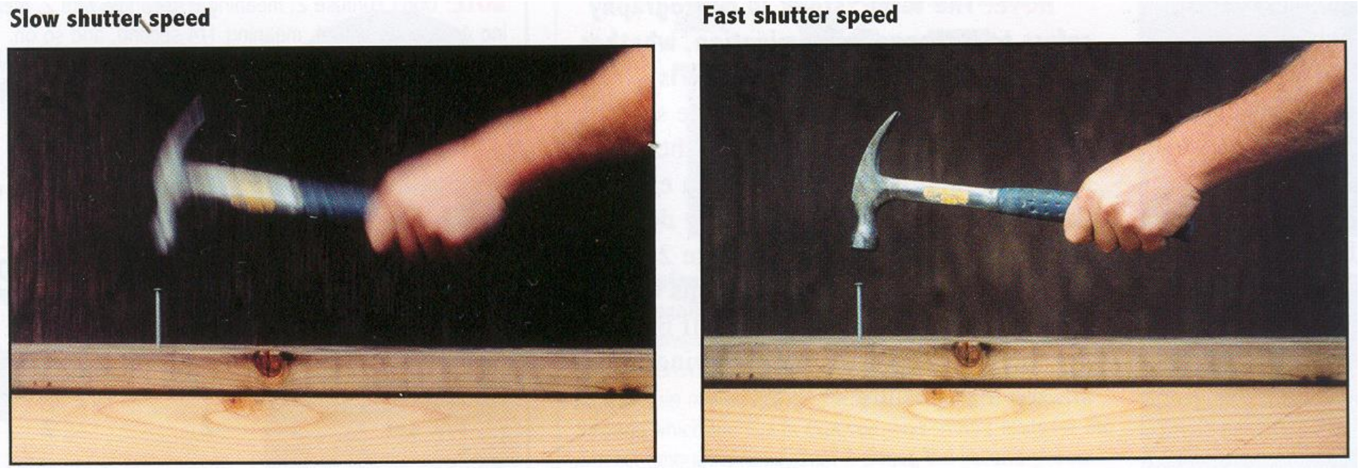
- Shutter speed (in fraction of a second)



Blade (closing)
Blade (open)
Focal plane (closed)
Focal plane (open)

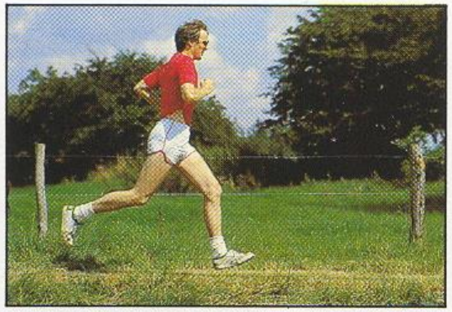
Effect of Shutter Speeds

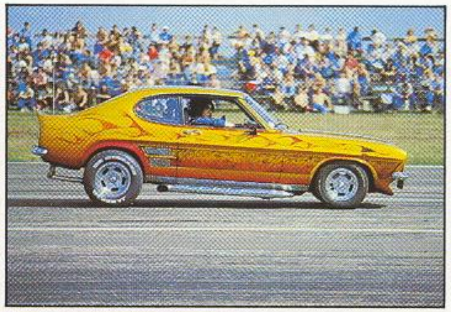
- Slow shutter speed ➔ more light, but more motion blur



- Faster shutter speed freezes motion


1/125

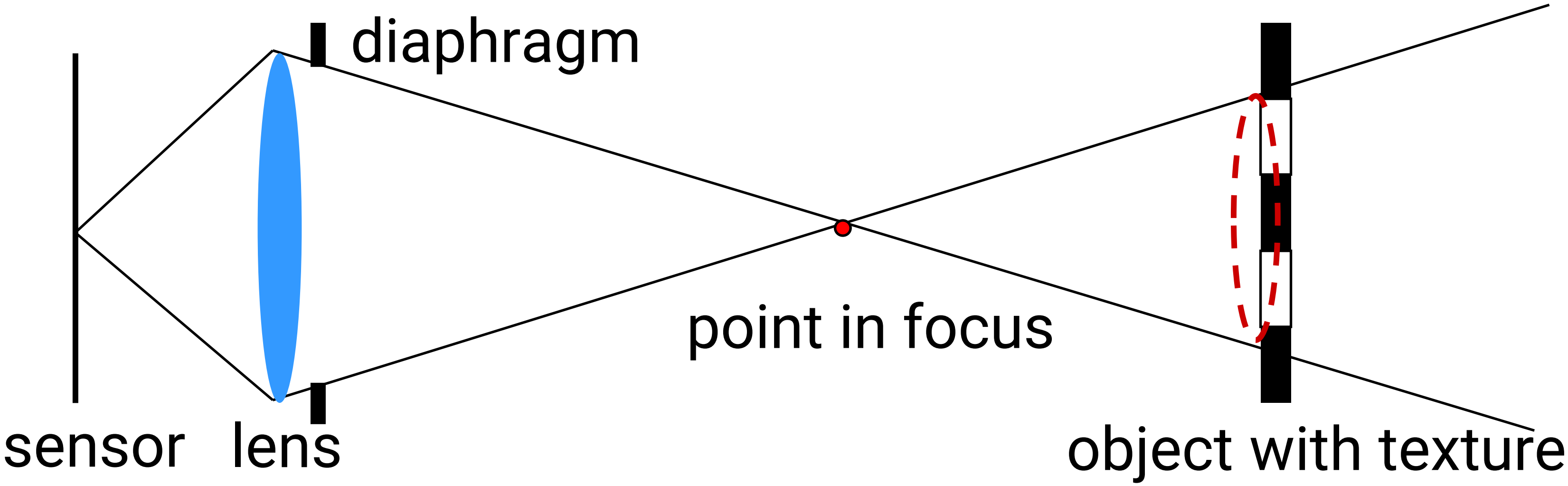

1/250


1/500

1/1000

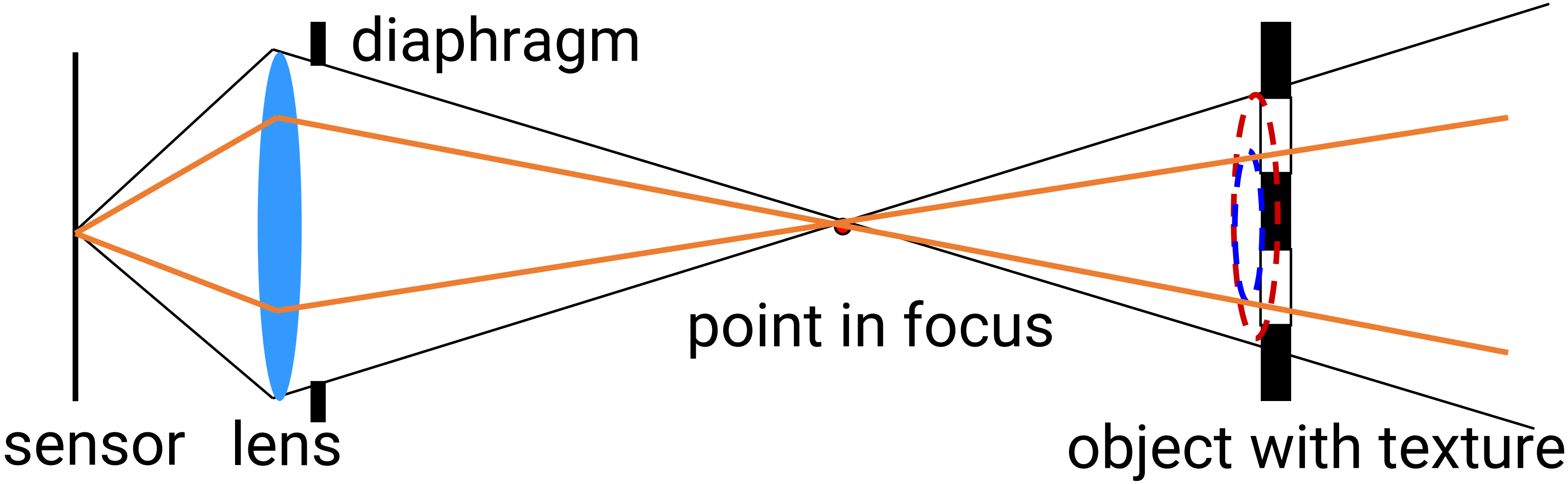
Depth of Field

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

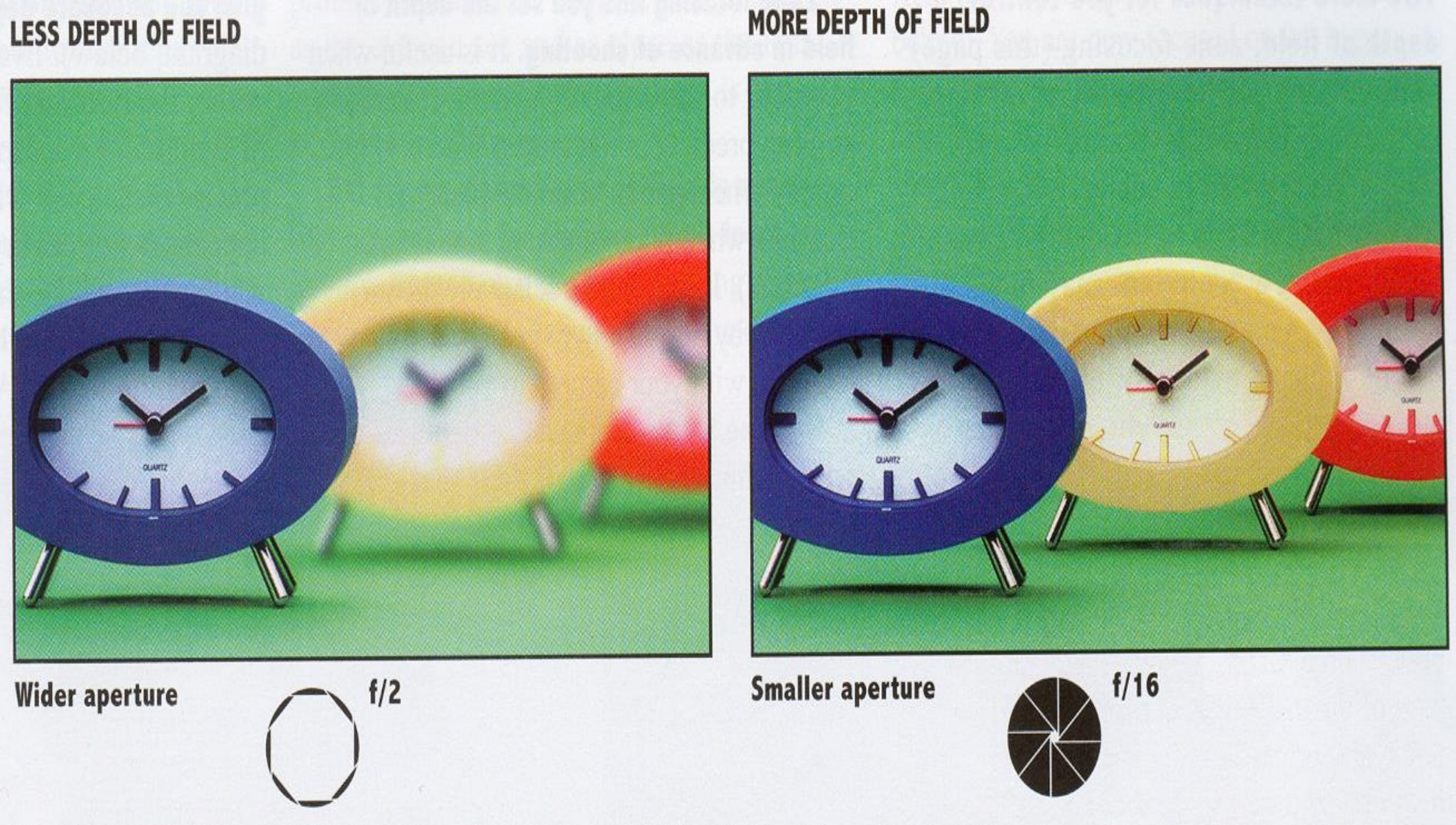


Depth of Field (cont.)

- 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



LESS DEPTH OF FIELD

Wider aperture f/2

MORE DEPTH OF FIELD

Smaller aperture f/16

Outline

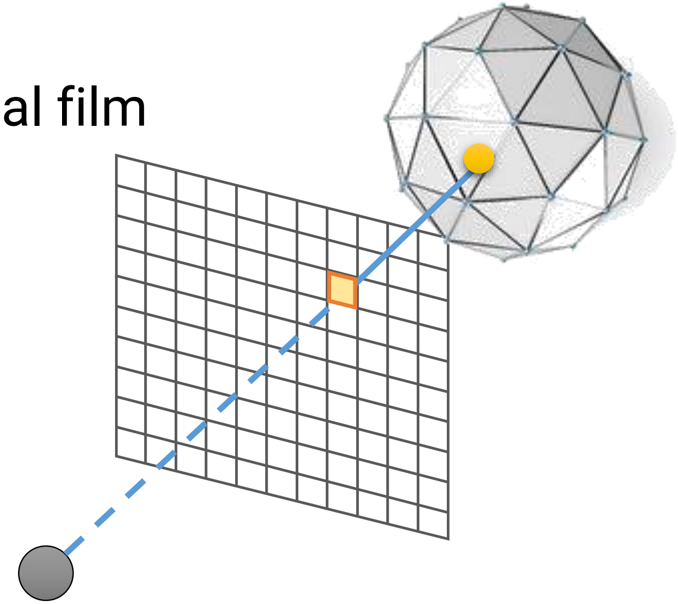
- Introduction to real-world cameras
- **Introduction to computer graphics cameras**
- Camera space and camera transformation
- Projective cameras
- OpenGL Implementation

Recap. Again !

- Two ways for generating synthetic images

Ray tracing

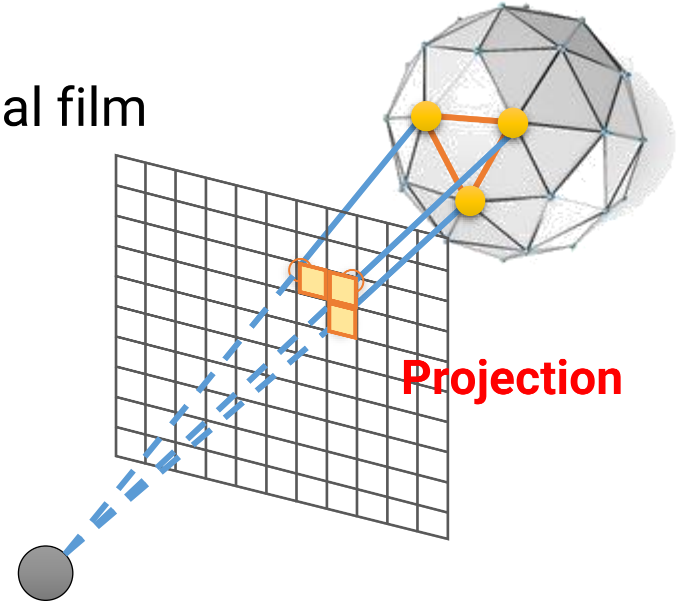
virtual film



virtual camera

Rasterization

virtual film

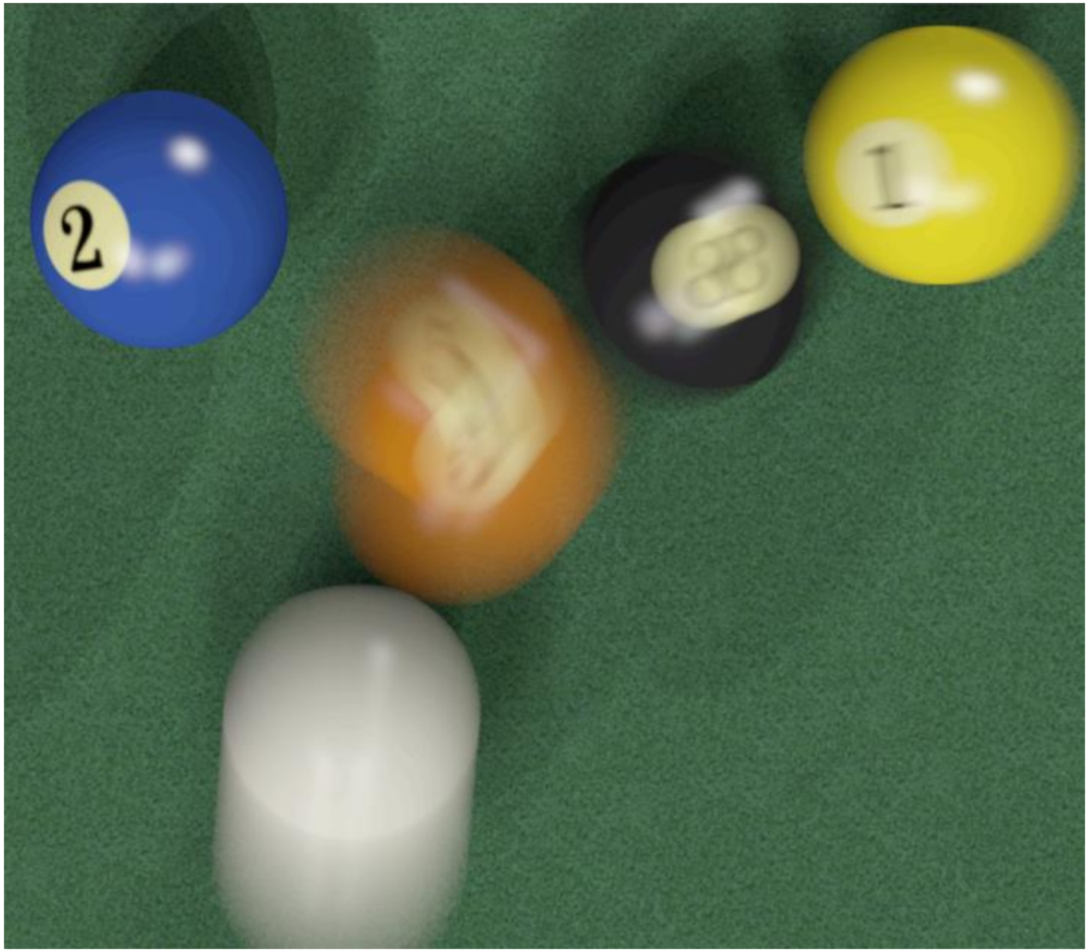


virtual camera

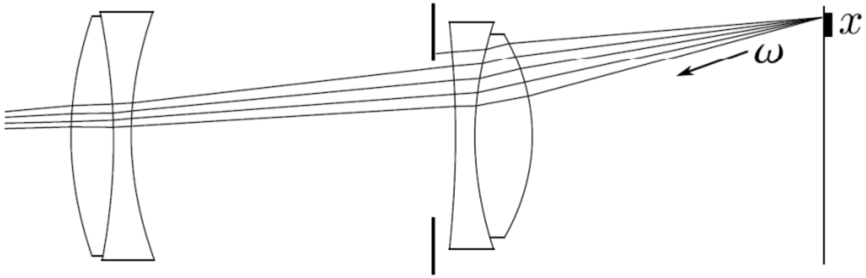
Computer Graphics Cameras

- To mimic the real-world functionalities 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 Cameras

- 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 for **projection**
  - Every object will always be in-focus
  - Depth of field and motion blur are simulated by other rendering techniques

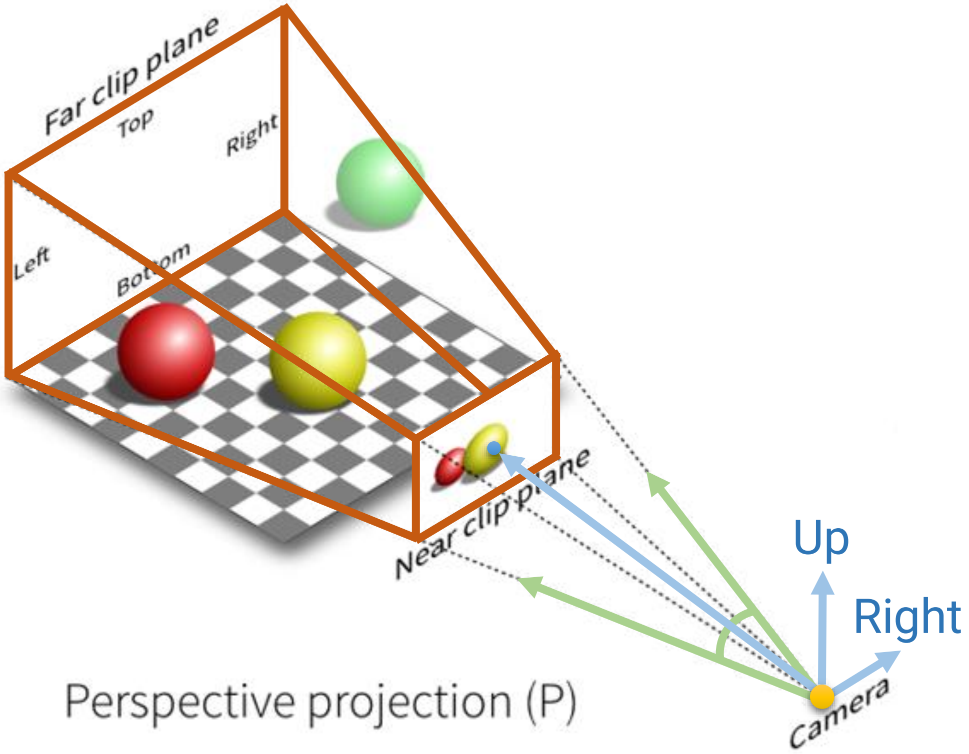
Computer Graphics Camera (cont.)

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



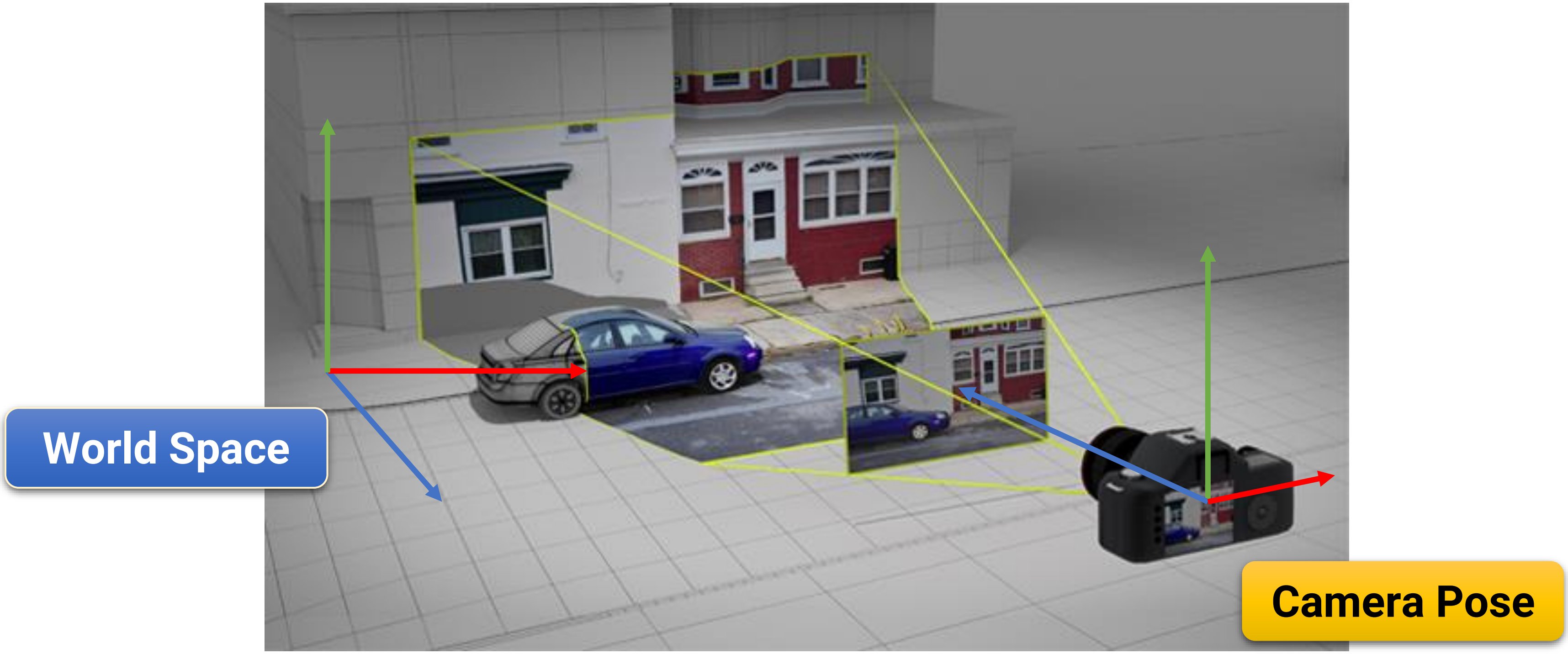
Perspective projection (P)

Outline

- Introduction to real-world cameras
- Introduction to computer graphics cameras
- **Camera space and camera transformation**
- Projective cameras
- OpenGL Implementation

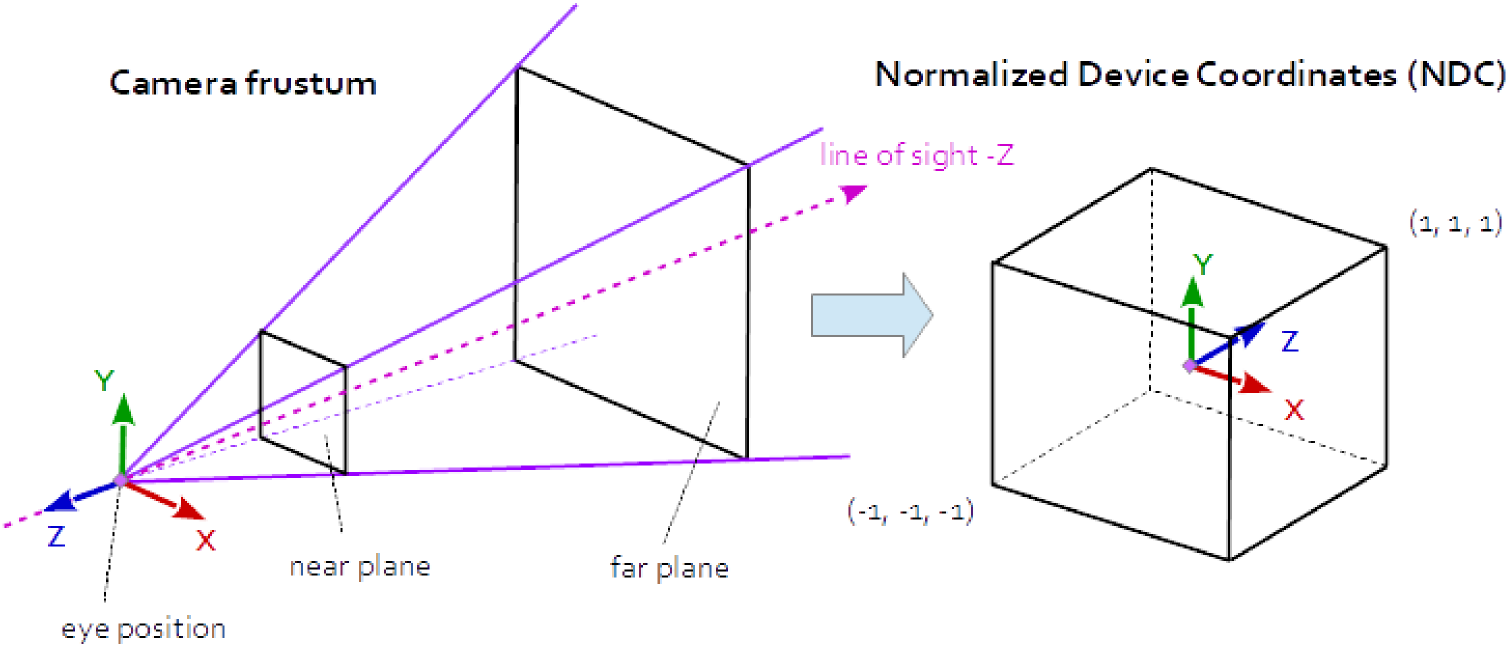
Camera (View) Transform

- 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



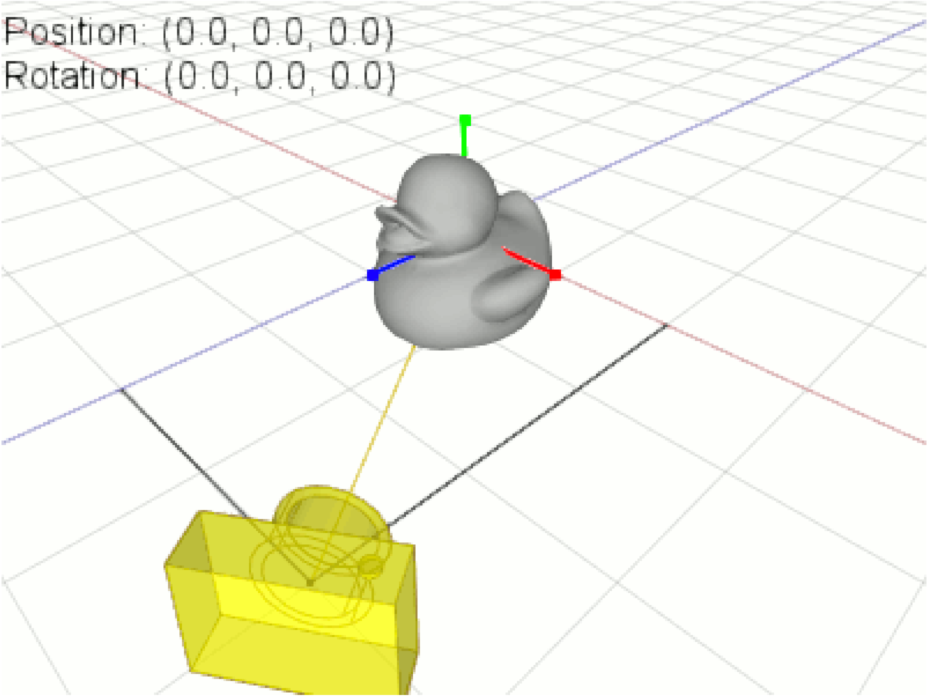
Camera (View) Transform (cont.)

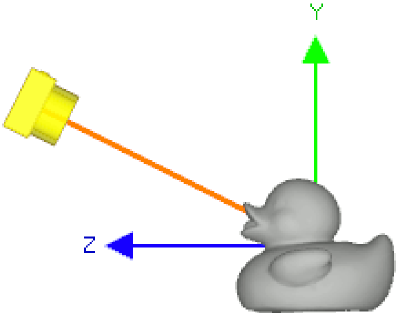
- To keep the math of projection simpler, we additionally define a **camera (view, eye) space**
  - In the camera space, the camera is **at the origin (0, 0, 0)** and **looking at the negative Z-axis**

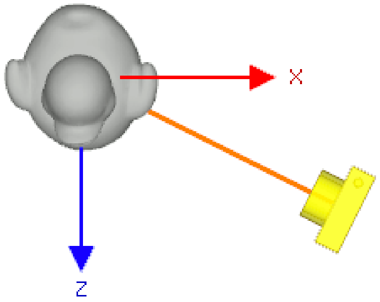


Camera (View) Transform (cont.)

- 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

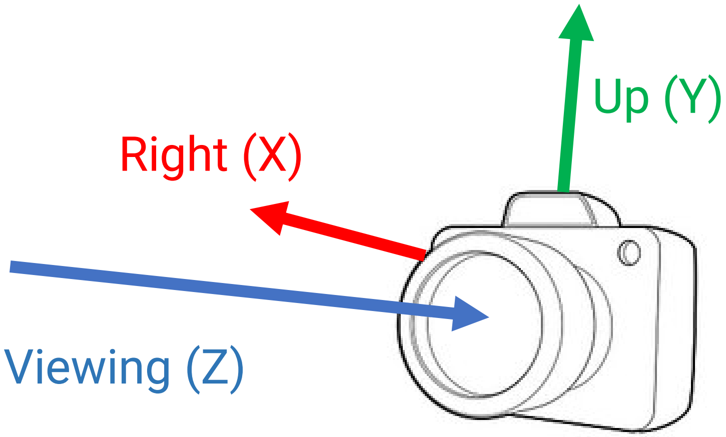






Camera (View) Transform (cont.)

- 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**



- 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

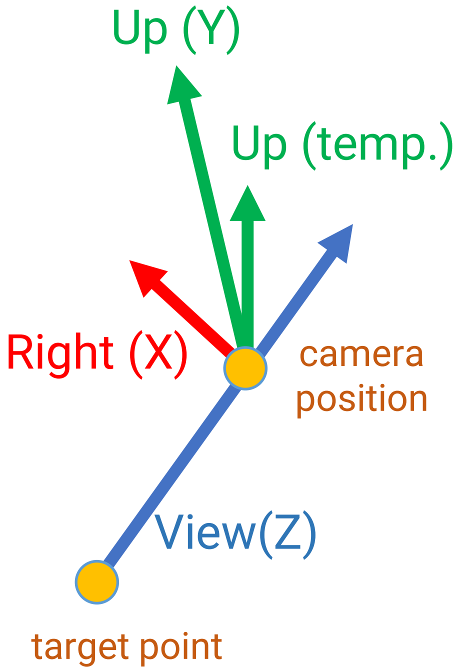
viewing direction = normalize(cameraPos – targetPos)

  - 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))

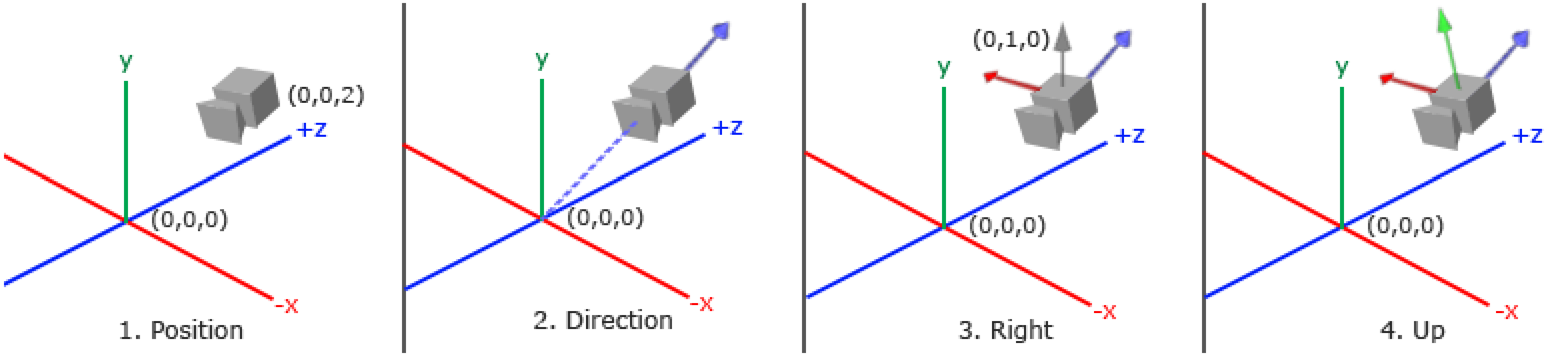


Camera (View) Transform (cont.)

- Camera (view) transformation

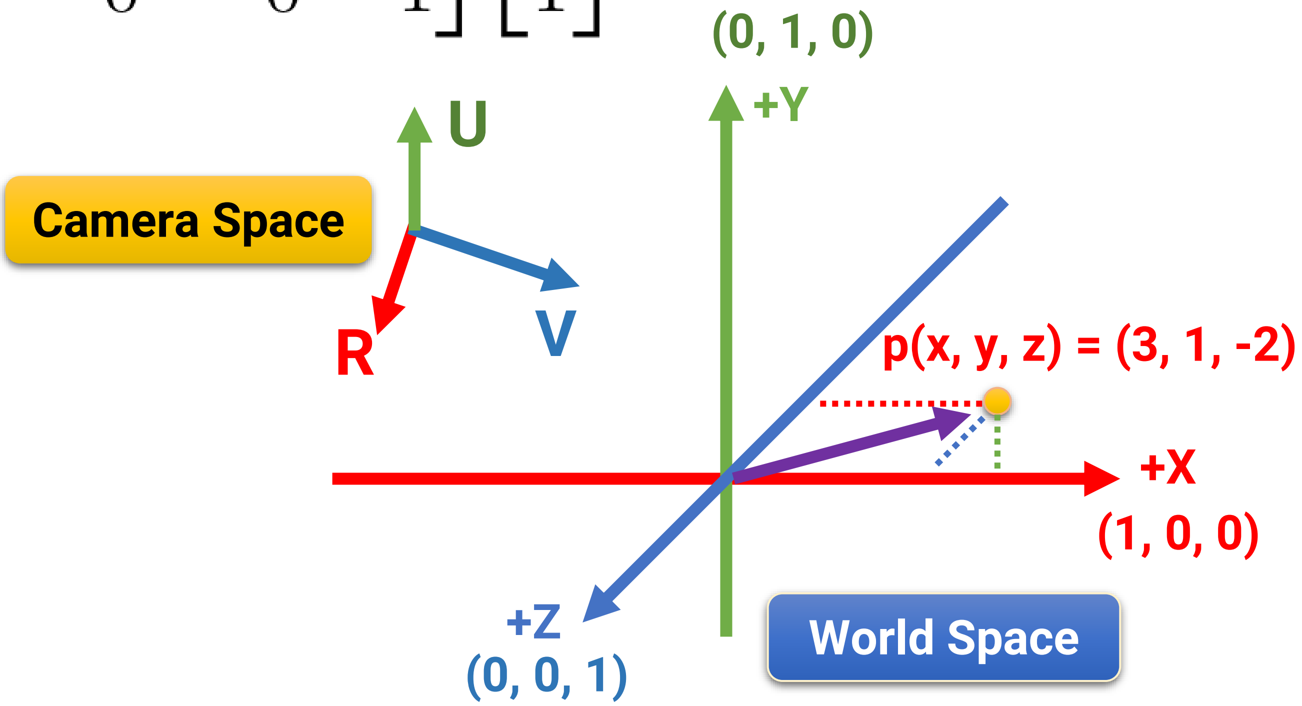
(P_x, P_y, P_z) is the camera's position

$$\begin{array}{r} \text{right vector} \\ \text{up vector} \\ \text{viewing vector} \\ \end{array} \underbrace{\begin{bmatrix} \boxed{R_x \quad R_y \quad R_z} & 0 \\ \boxed{U_x \quad U_y \quad U_z} & 0 \\ \boxed{D_x \quad D_y \quad D_z} & 0 \\ 0 \quad\; 0 \quad\; 0 & 1 \end{bmatrix}}_{\text{rotation matrix}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{translation matrix}}$$



Camera (View) Transform (cont.)

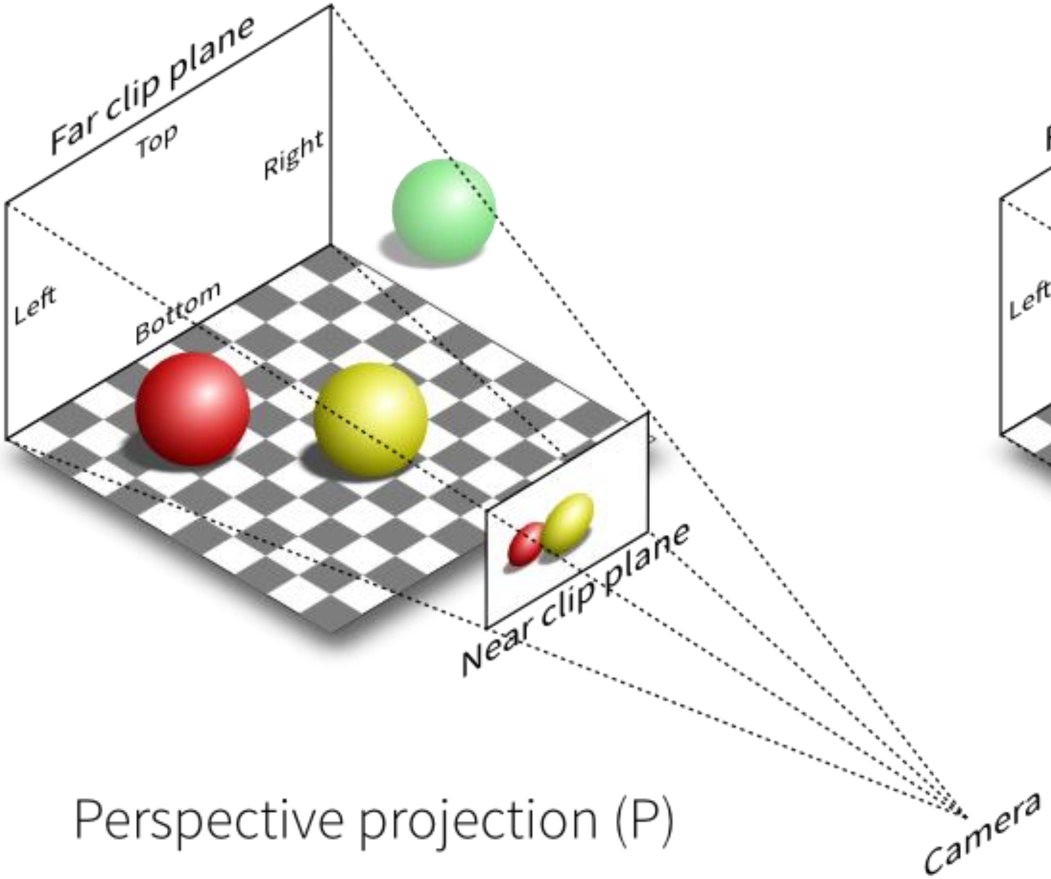
$$\begin{array}{r} \text{right vector} \\ \text{up vector} \\ \text{viewing vector} \\ \\ \end{array} \begin{bmatrix} \boxed{R_x \quad R_y \quad R_z} & 0 \\ \boxed{U_x \quad U_y \quad U_z} & 0 \\ \boxed{D_x \quad D_y \quad D_z} & 0 \\ 0 \quad\; 0 \quad\; 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



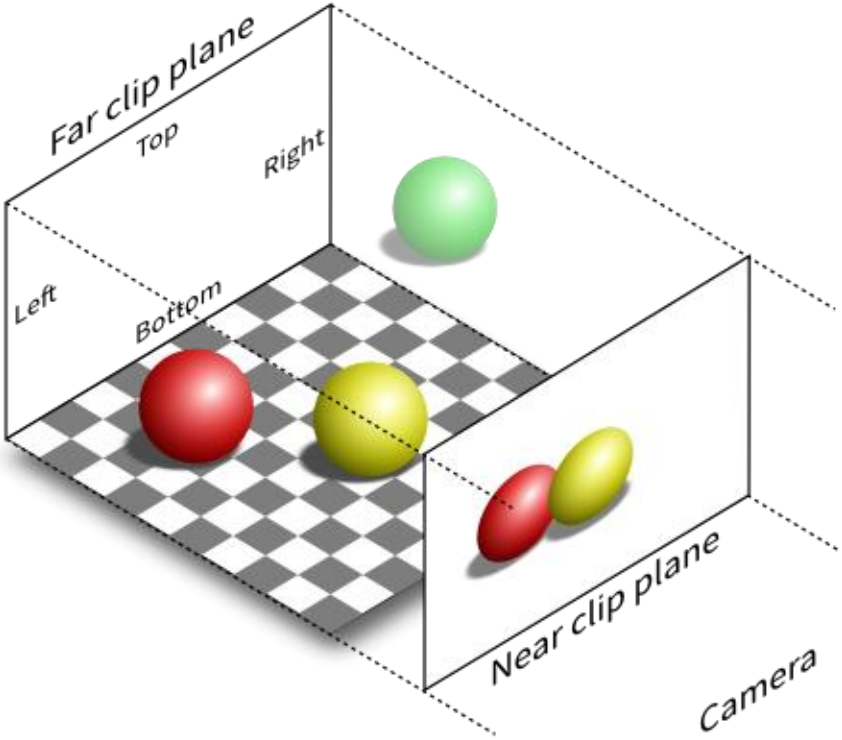
Outline

- Introduction to real-world cameras
- Introduction to computer graphics cameras
- Camera space and camera transformation
- **Projective cameras**
- OpenGL Implementation

Projective Camera Models



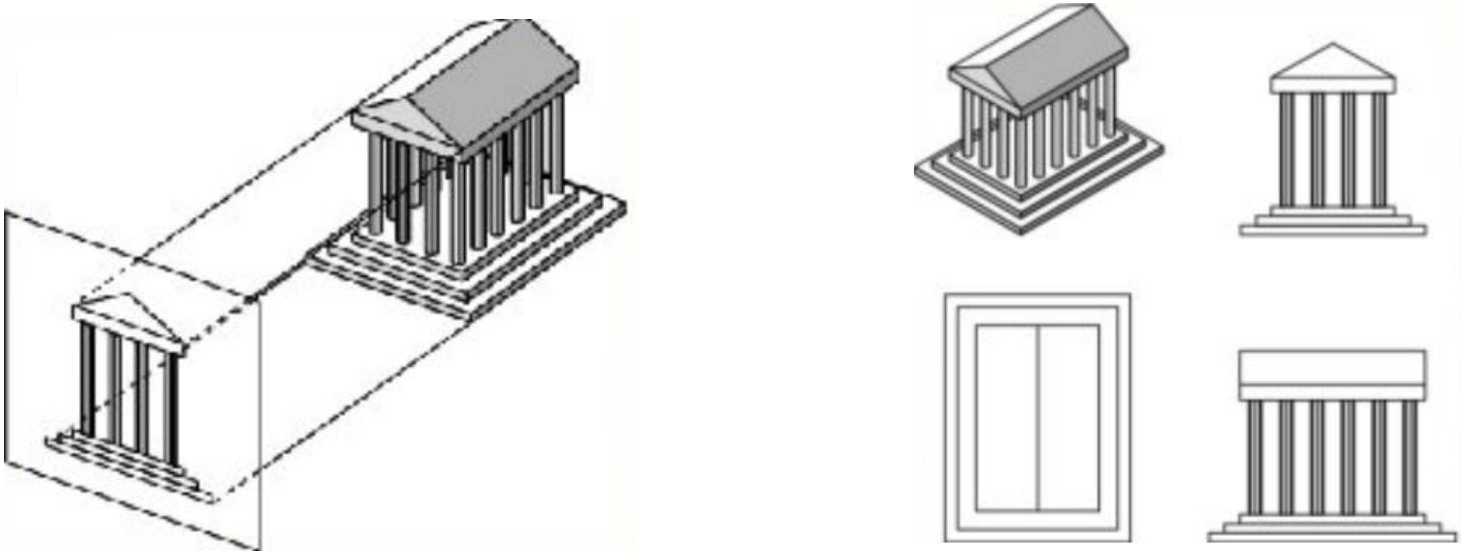
Perspective projection (P)



Orthographic projection (O)

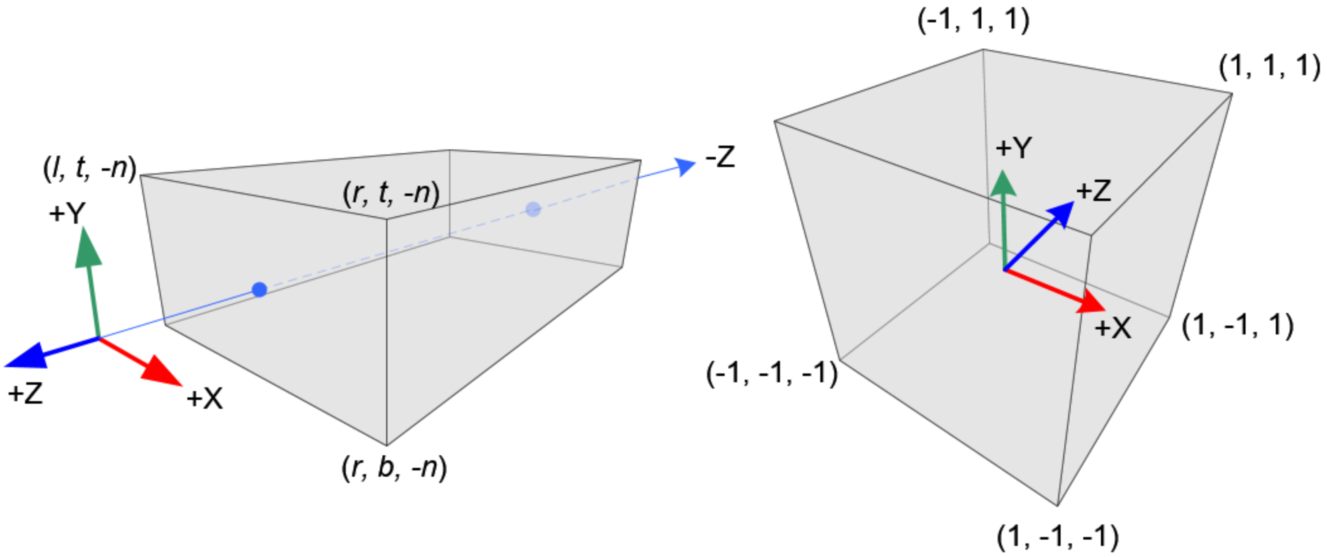
Orthographic Projection

- 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

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

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

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

Orthographic Projection (cont.)

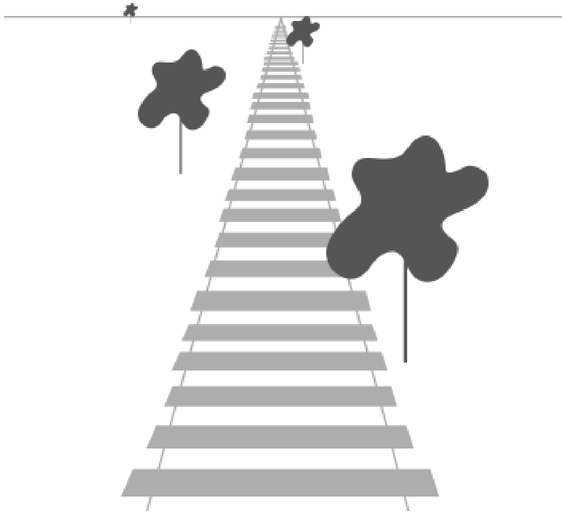
- Let the **l**, **r**, **t**, **b**, **n**, **f** be the boundaries of the left, right, top, bottom, near, and far planes
- An orthographic projection matrix can be written as

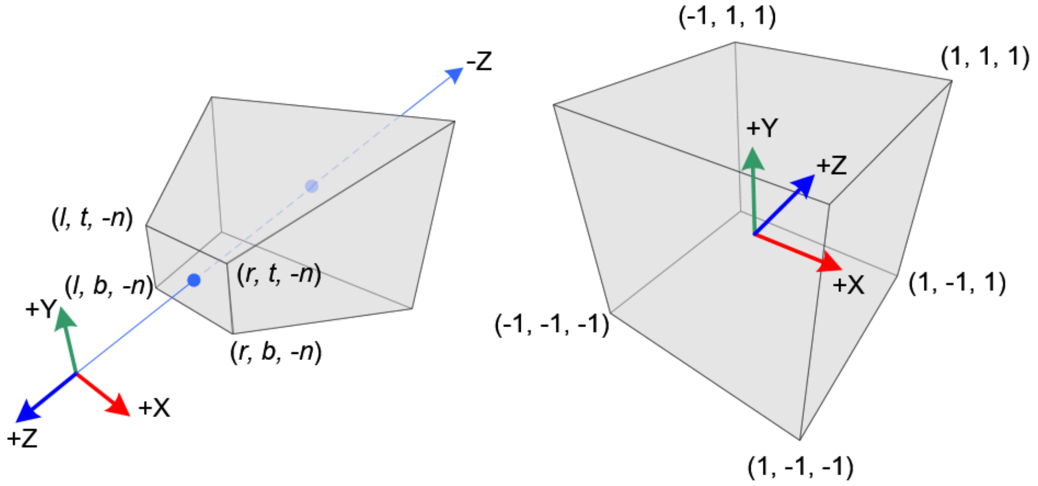
$$\begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$-1 \leq \frac{2x}{r-l} - \frac{r+l}{r-l} \leq 1$$

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



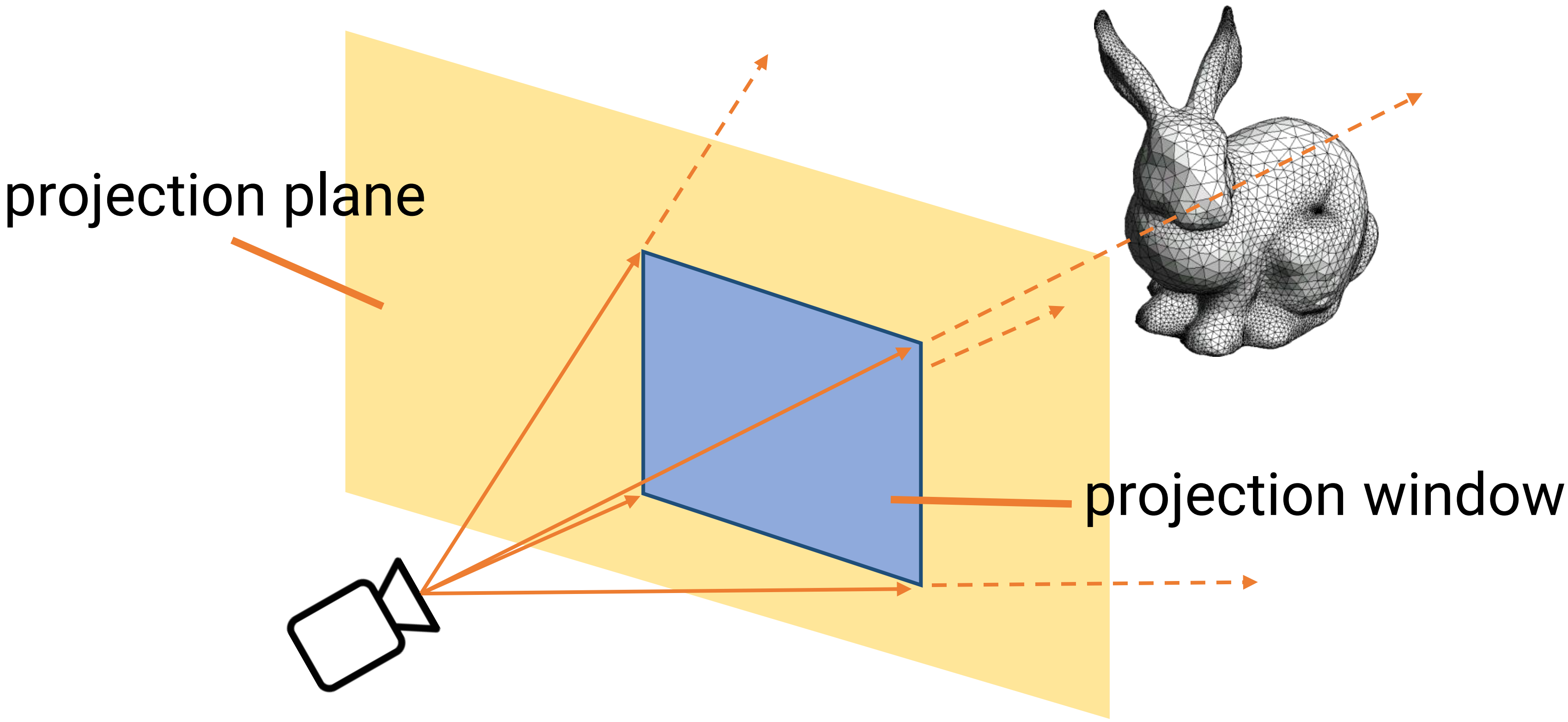


Perspective Projection (cont.)

- Four components for the perspective projection matrix
  - **The aspect ratio of the screen**
    - The ratio between the width and the height (W/H)
  - **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

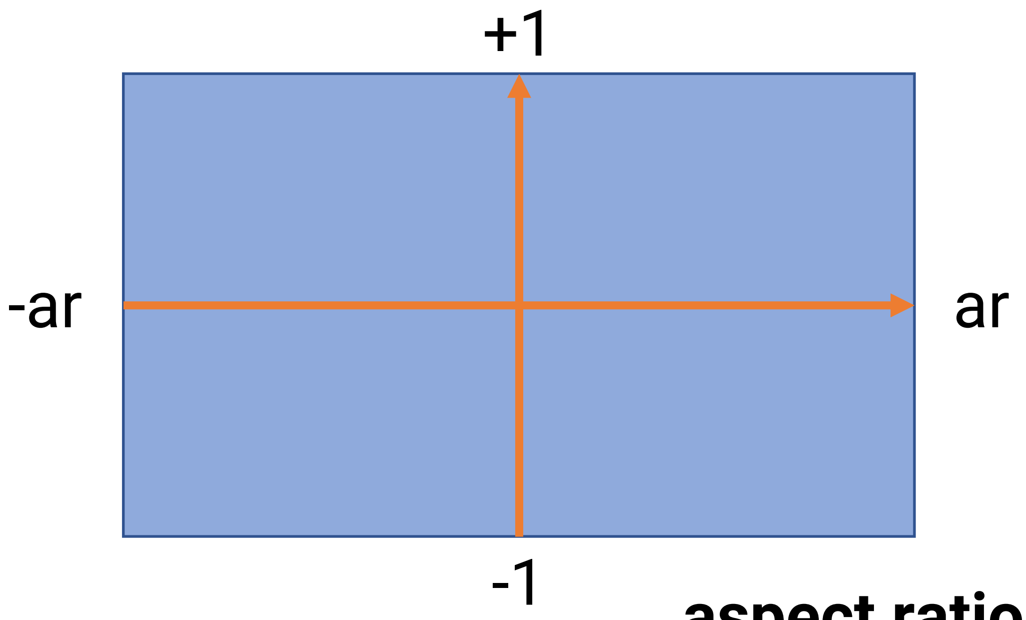
Perspective Projection (cont.)

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



Perspective Projection (cont.)

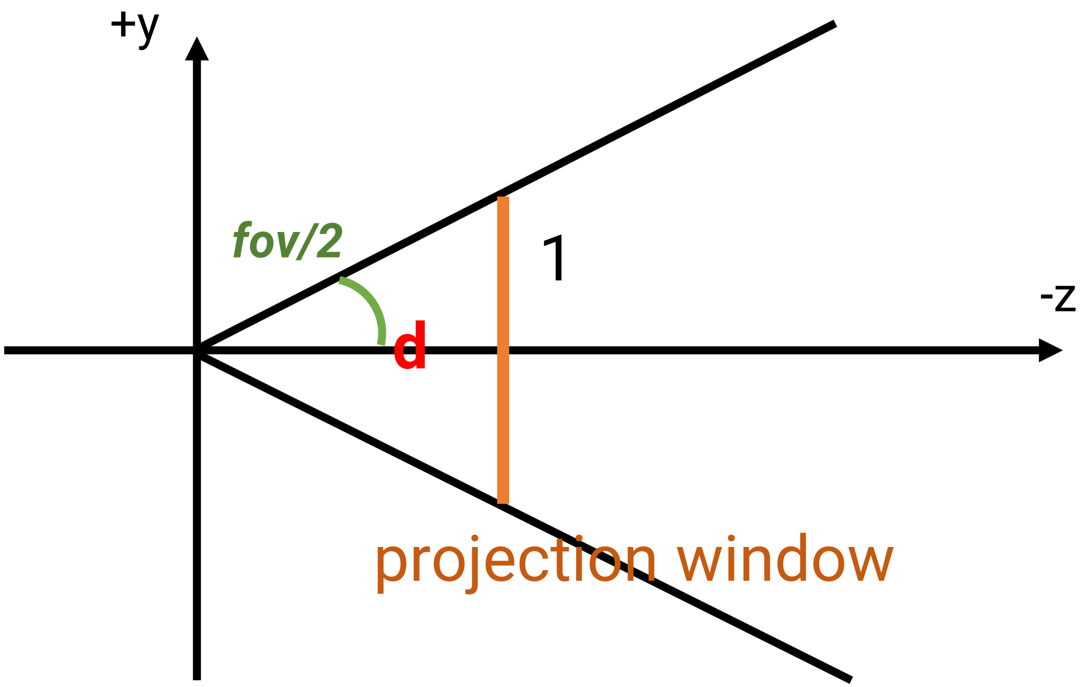
- Derivation of the perspective projection matrix
  - Determine the height of the projection window as 2
  - The width of the projection window becomes 2 times the aspect ratio (ar)



aspect ratio (ar) = W/H

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)

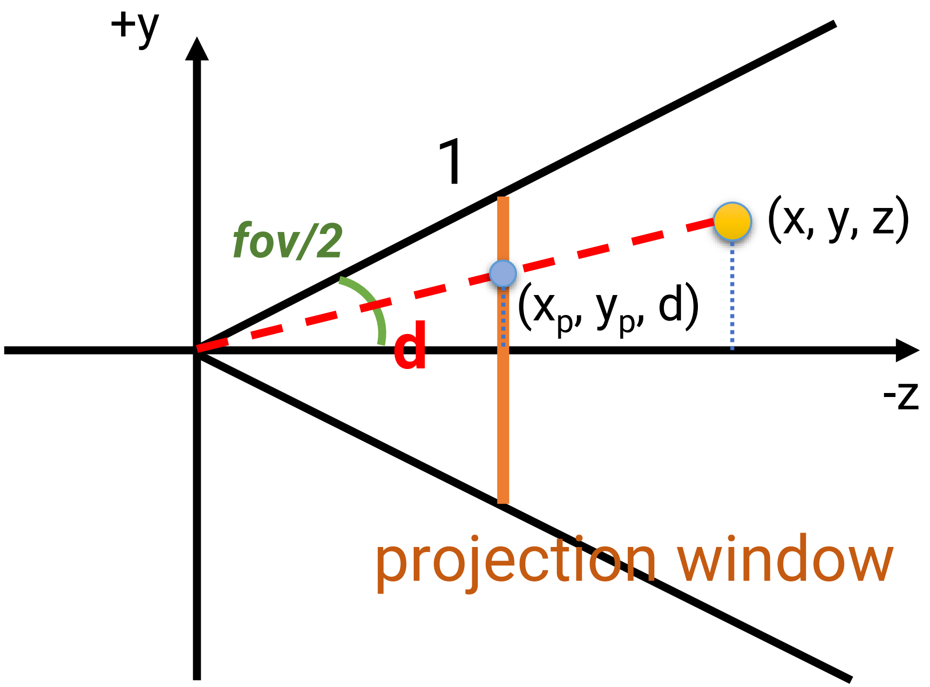


$$\tan(\frac{\alpha}{2}) = \frac{1}{d}$$

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

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 …



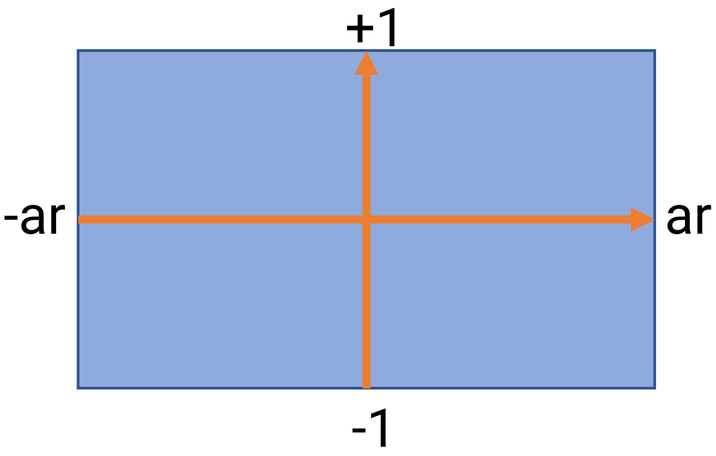
$$\frac{y_p}{d} = \frac{y}{-z}$$

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

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

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

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

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

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

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

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

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

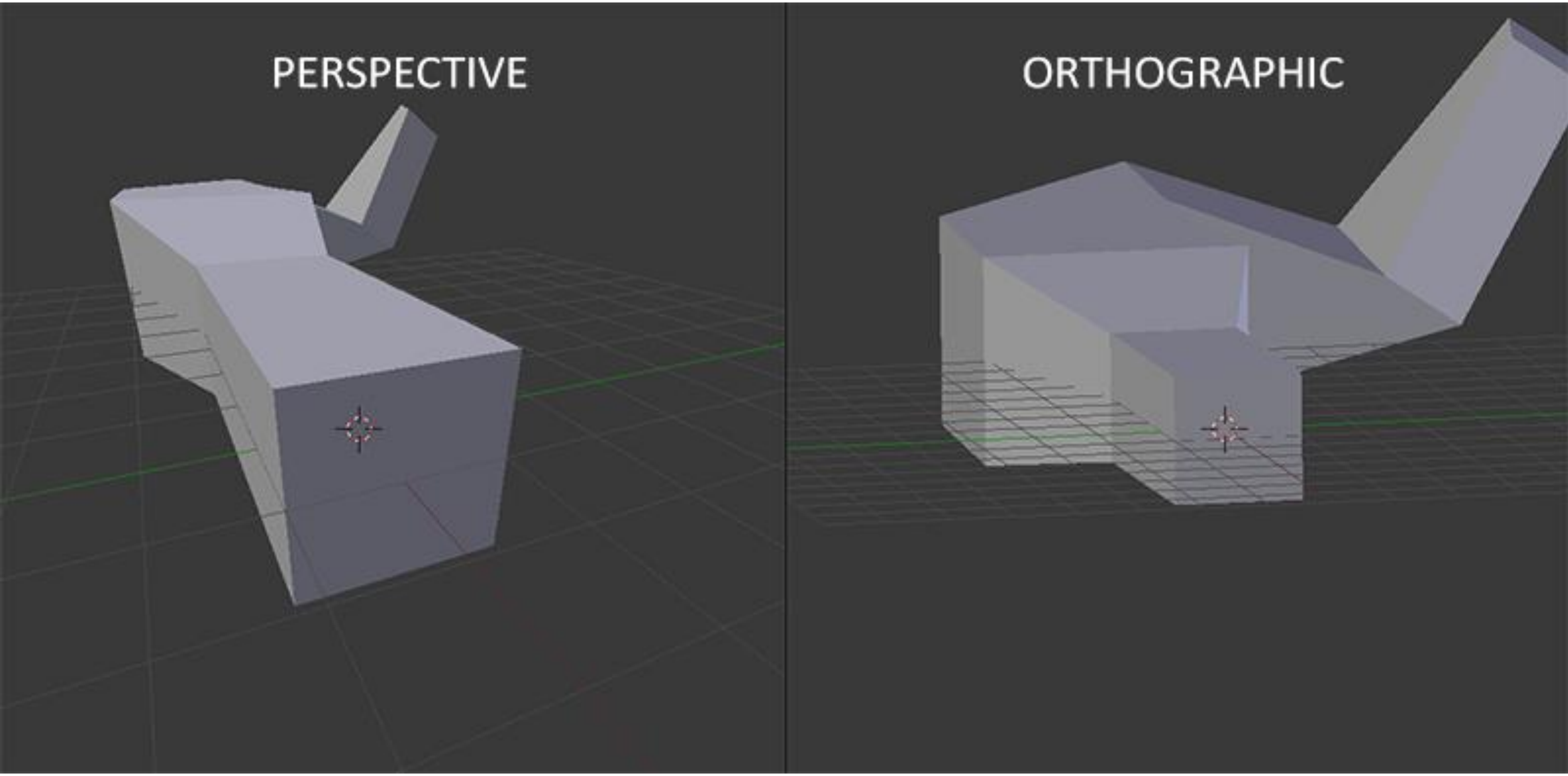
Perspective Projection (cont.)

- 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

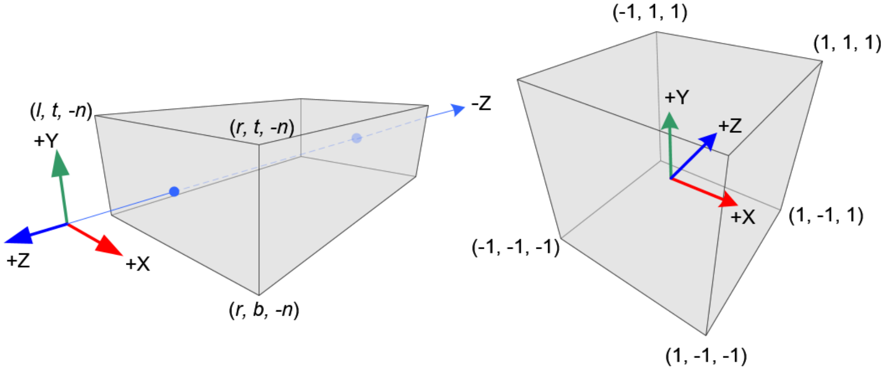
Camera Models Comparison (cont.)



Outline

- Introduction to real-world cameras
- Introduction to computer graphics cameras
- Camera space and camera transformation
- Projective cameras
- **OpenGL Implementation**

Ortho Projection Matrix



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

- glm::mat4x4 ortho(const float left, const float right,
  const float bottom, const float bottom,
  const float near, const float far)

```
glm::mat4x4 goP = glm::ortho(-5.0f, 5.0f, -5.0f, 5.0f, 0.01f, 100.0f);
```

Perspective Projection Matrix

$$\begin{bmatrix} \frac{1}{ar \cdot \tan(\frac{\alpha}{2})} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\frac{\alpha}{2})} & 0 & 0 \\ 0 & 0 & \frac{-nearZ - farZ}{nearZ - farZ} & \frac{2 \cdot farZ \cdot nearZ}{nearZ - farZ} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

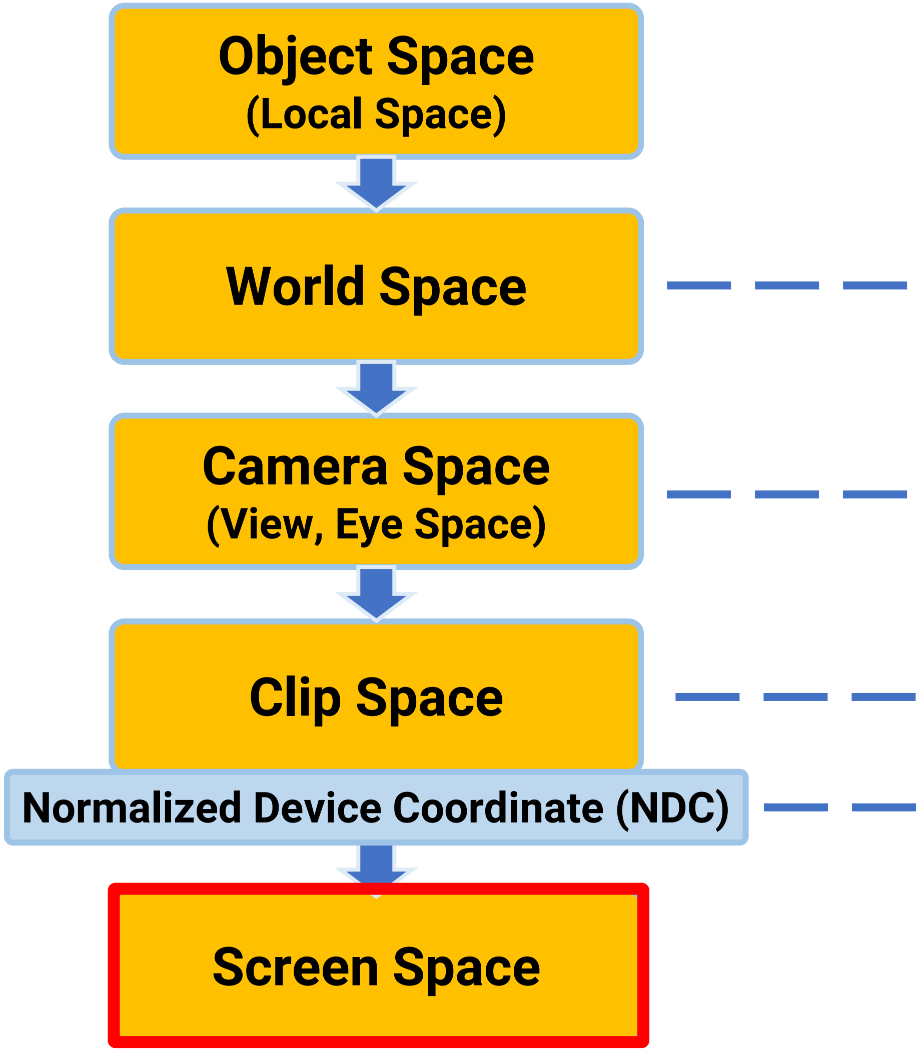
- glm::mat4x4 perspective(const float fovy ,
  const float aspectRatio ,
  const float near ,
  const float far)

use radian, not degree

width / height

```
float fovy = glm::radians(30.0f);
float aspectRatio = 640.0f / 360.0f;
float nearZ = 0.1f;
float farZ = 100.0f;
glm::mat4x4 gP = glm::perspective(fovy, aspectRatio, nearZ, farZ);
```

The Full Vertex Transform Pipeline



After applying world transform (translation, scaling, rotation)

After applying camera transform

After applying projection

After applying perspective division

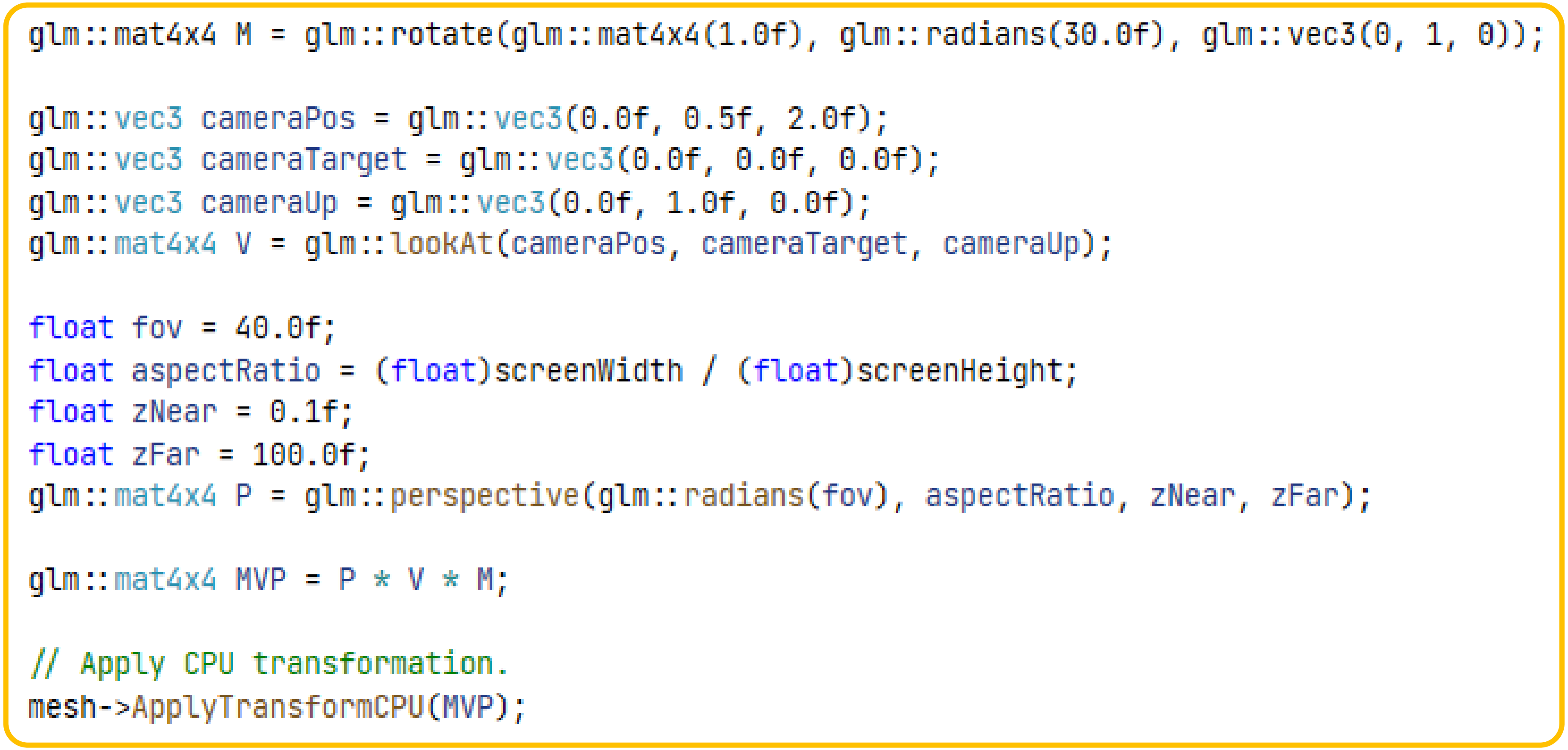
XYZ in the range [-1, 1]

The Z component is kept for depth comparison (closest surface to the camera)

Apply the Transformation on CPU

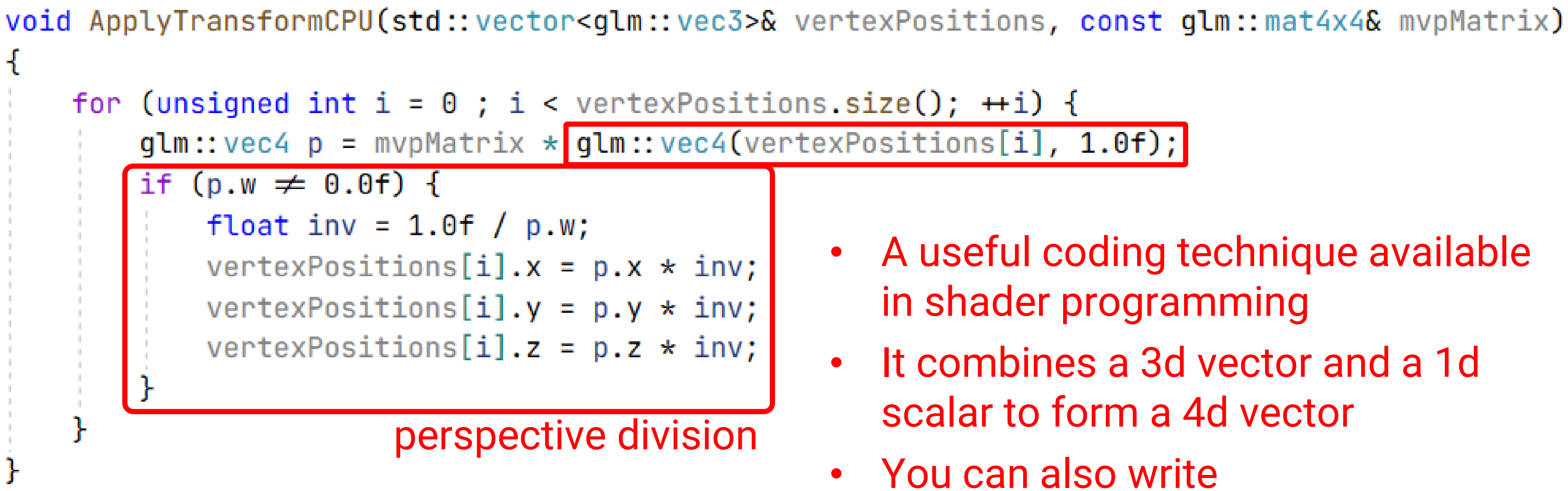
- To transform a vertex from object space to clip space, we multiply its position with the **model-view-projection (MVP)** matrix
- We can pre-multiply part of the matrix if some of them are fixed
  - For example, we can pre-multiply the camera (view) and the projection matrix to form a VP matrix, and change the model matrix to perform object animation
- Remember to do the **perspective division**

Apply the Transformation on CPU (cont.)



```
glm::mat4x4 M = glm::rotate(glm::mat4x4(1.0f), glm::radians(30.0f), glm::vec3(0, 1, 0));

glm::vec3 cameraPos = glm::vec3(0.0f, 0.5f, 2.0f);
glm::vec3 cameraTarget = glm::vec3(0.0f, 0.0f, 0.0f);
glm::vec3 cameraUp = glm::vec3(0.0f, 1.0f, 0.0f);
glm::mat4x4 V = glm::lookAt(cameraPos, cameraTarget, cameraUp);

float fov = 40.0f;
float aspectRatio = (float)screenWidth / (float)screenHeight;
float zNear = 0.1f;
float zFar = 100.0f;
glm::mat4x4 P = glm::perspective(glm::radians(fov), aspectRatio, zNear, zFar);

glm::mat4x4 MVP = P * V * M;

// Apply CPU transformation.
mesh->ApplyTransformCPU(MVP);
```

Apply the Transformation on CPU (cont.)



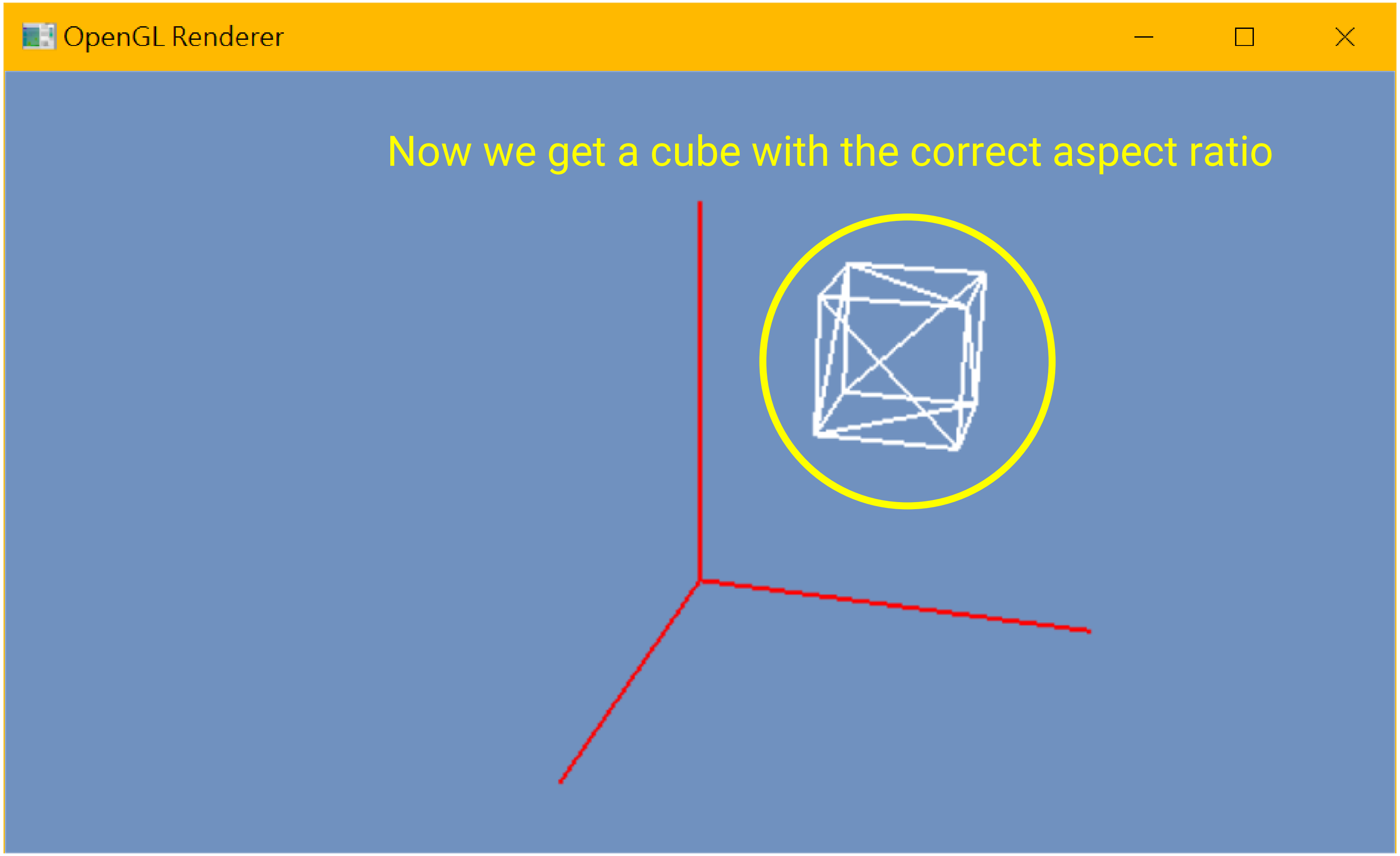
```
void ApplyTransformCPU(std::vector<glm::vec3>& vertexPositions, const glm::mat4x4& mvpMatrix)
{
    for (unsigned int i = 0 ; i < vertexPositions.size(); ++i) {
        glm::vec4 p = mvpMatrix * glm::vec4(vertexPositions[i], 1.0f);
        if (p.w ≠ 0.0f) {
            float inv = 1.0f / p.w;
            vertexPositions[i].x = p.x * inv;
            vertexPositions[i].y = p.y * inv;
            vertexPositions[i].z = p.z * inv;
        }
    }
}
```

perspective division

- A useful coding technique available in shader programming
- It combines a 3d vector and a 1d scalar to form a 4d vector
- You can also write

```
glm::vec4(vertexPositions[i].x,
          vertexPositions[i].y,
          vertexPositions[i].z,
          1.0f);
```

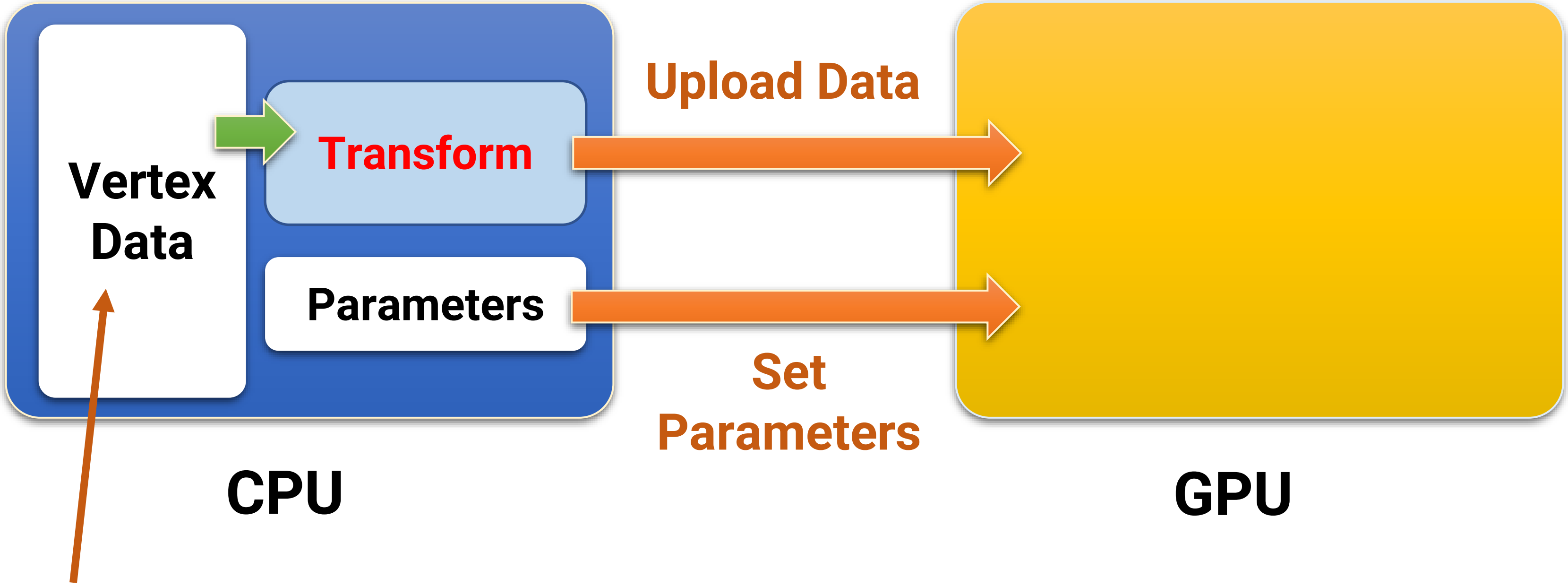
Apply the Transformation on CPU (cont.)



Now we get a cube with the correct aspect ratio

Apply the Transformation on CPU

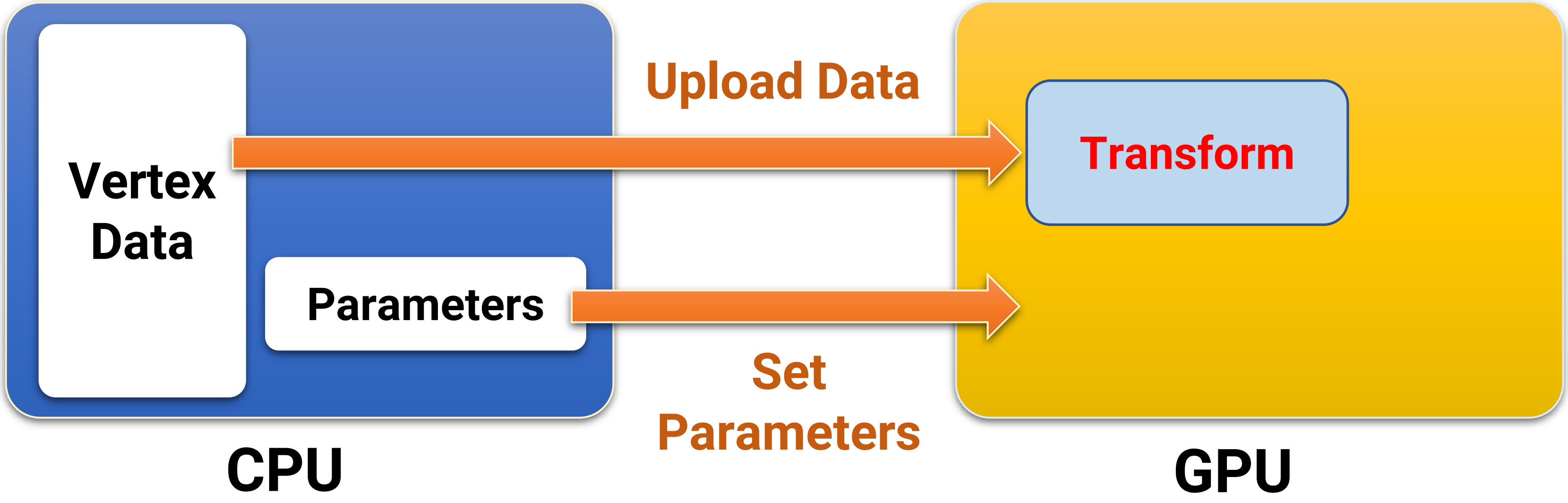
- So far, we have performed the transformation of vertices on the CPU



Vertex attributes
Position
Normal
Texture coordinate

Apply the Transformation on GPU

- However, doing this job on CPU is not cost-effective
  - CPU is good at doing sequential, complex jobs
  - But vertex transform is simple and can be done in parallel
- Next class, we will introduce the **GPU graphics pipeline** and the **vertex shaders** for **parallel** processing




END OF PRESENTATION
ANY QUESTIONS?