Color, Images, and Cameras

Slides by Sarah Jobalia and Kevin Li, with modifications by Kate Baker

Housekeeping

- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**

- **SUBMIT ON GRADESCOPE!!**

Housekeeping

- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**

- **SUBMIT ON GRADESCOPE!!**

- HW2 released

Housekeeping

- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**
- **HW1 GRADING TODAY**

- **SUBMIT ON GRADESCOPE!!**

- HW2 released

- Check out Blender tutorials!

Revisiting Questions

- OpenGL
  - Skimmed last time, too much today to discuss the pipeline in full again
  - Have a look at the linked resources in Tuesday's lecture

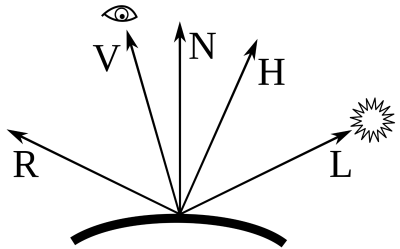
Revisiting Questions

- OpenGL
  - Skimmed last time, too much today to discuss the pipeline in full again
  - Have a look at the linked resources in Tuesday's lecture
- Phong vs Gouraud shading
  - Gouraud shading can and typically uses the Phong reflection model for the colors at its vertices
  - Gouraud can miss highlight entirely inside a triangle, and smears highlights at vertices
  - Normals in Phong shading add more flexibility even if more expensive

Last time…

- Rasterization of triangles to pixels
  - For each pixel, if the center of the pixel is inside the triangle, consider it part of the triangle (and color it with the triangle's color)
- Shading: determining how light and energy bounces off objects

C = ambient + diffuse + specular



Lecture Outline

- In real life
  - The human eye (pupil, cornea, lens)
  - RGB color space
  - The pinhole camera
- In the rasterizer
  - Coordinate frames
  - Perspective & orthographic projection
  - Normalized coordinates & the z-buffer algorithm
- Lens-based cameras
- Beyond rasterization

Lecture Outline

- In real life
  - The human eye (pupil, cornea, lens)
  - RGB color space
  - The pinhole camera
- In the rasterizer
  - Coordinate frames
  - Perspective & orthographic projection
  - Normalized coordinates & the z-buffer algorithm
- Lens-based cameras
- Beyond rasterization

Light and the Human Eye

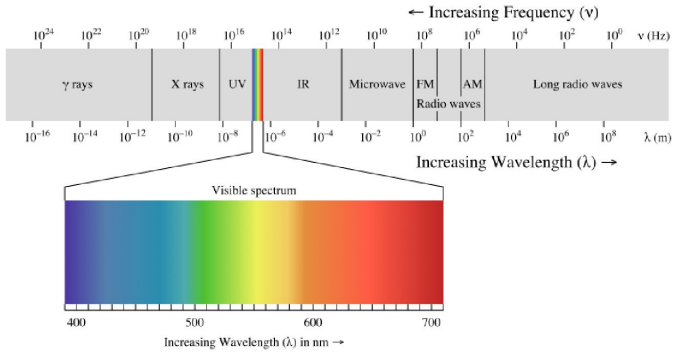
- Light is emitted from a light source
  - E.g. the sun, a light bulb, smartphone

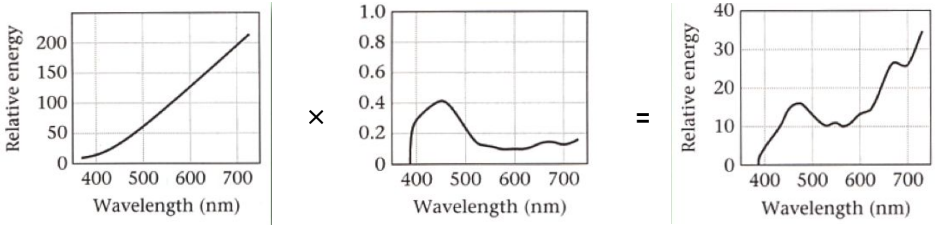
Light and the Human Eye

- Light is emitted from a light source
  - E.g. the sun, a light bulb, smartphone
- Emitted light interacts with the real world
- Eventually, some light may enter our eyes which creates a signal
- Our brain creates an image based on the signals it gets from our eyes

Light and the Human Eye

- Light is emitted from a light source
  - E.g. the sun, a light bulb, smartphone
- Emitted light interacts with the real world
- Eventually, some light may enter our eyes which creates a signal
- Our brain creates an image based on the signals it gets from our eyes





Incoming Energy × Material Reflectance = Reflected Energy

Light and the Human Eye

- **Cone** and **rod** cells absorb light and create a signal
  - **3** types of cone cells, **1** type of rod cell

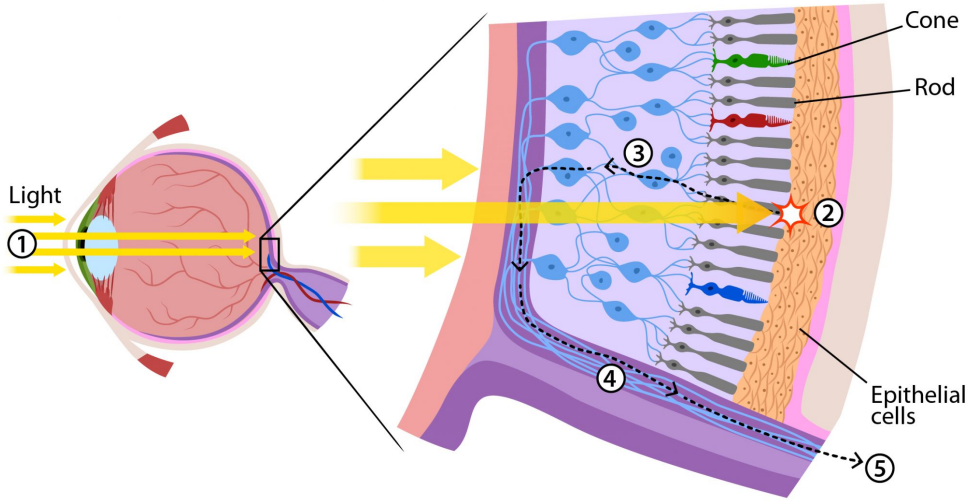
Light and the Human Eye

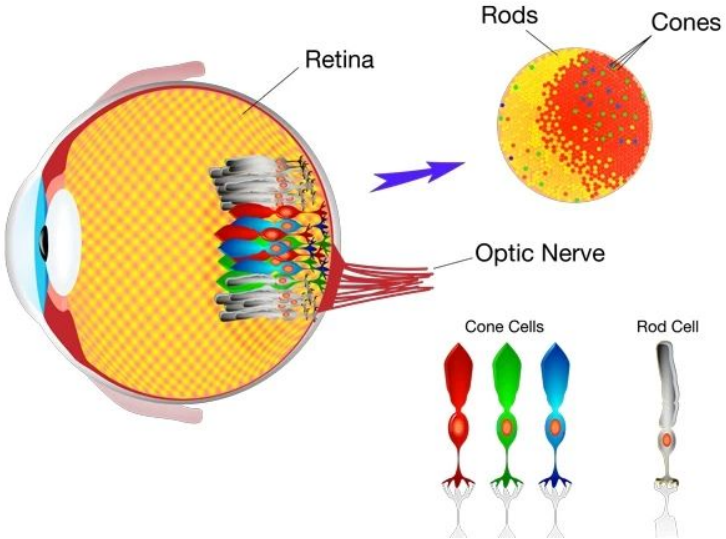
- **Cone** and **rod** cells absorb light and create a signal
  - **3** types of cone cells, **1** type of rod cell
  - Each specialized receptor fires a single signal that can vary in strength
  - Each of the **4** cells are sensitive to different frequencies and intensities of light

Light and the Human Eye

- **Cone** and **rod** cells absorb light and create a signal
  - **3** types of cone cells, **1** type of rod cell
  - Each specialized receptor fires a single signal that can vary in strength
  - Each of the **4** cells are sensitive to different frequencies and intensities of light
- **Cone**: used when there's sufficient light and distinguishes color
- **Rod**: used in low light settings

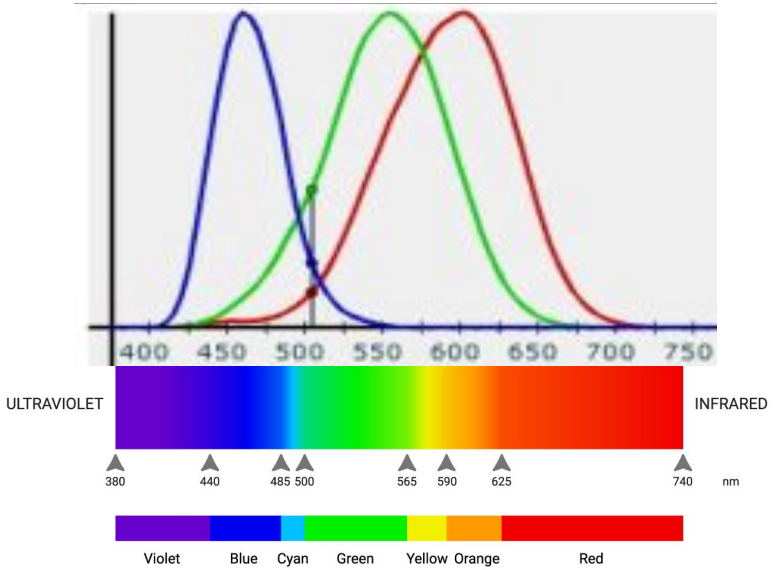
Light and the Human Eye





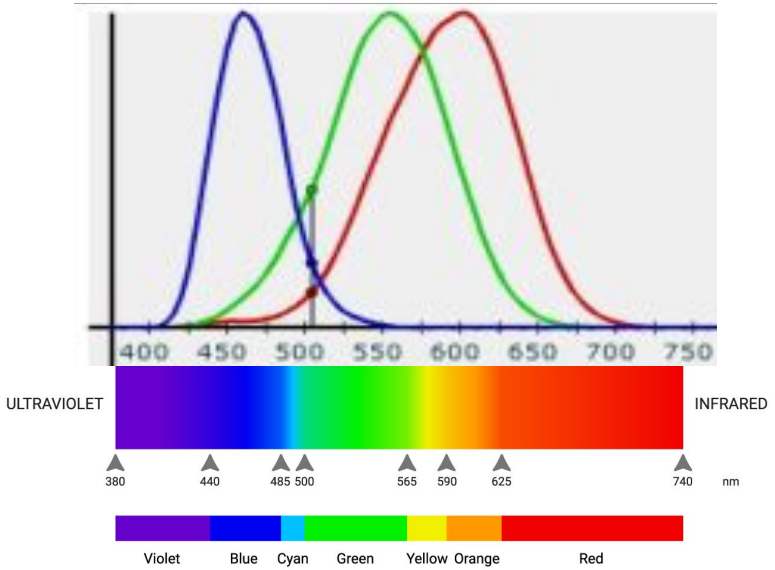
Light and the Human Eye

- Humans have **trichromatic vision**
- Our brains differentiate between colors via the three types of signals we get:
  - L-cones: longer wavelengths (red cones)
  - M-cones: medium wavelengths (green cones)
  - S-cones: shorter wavelengths (blue cones)



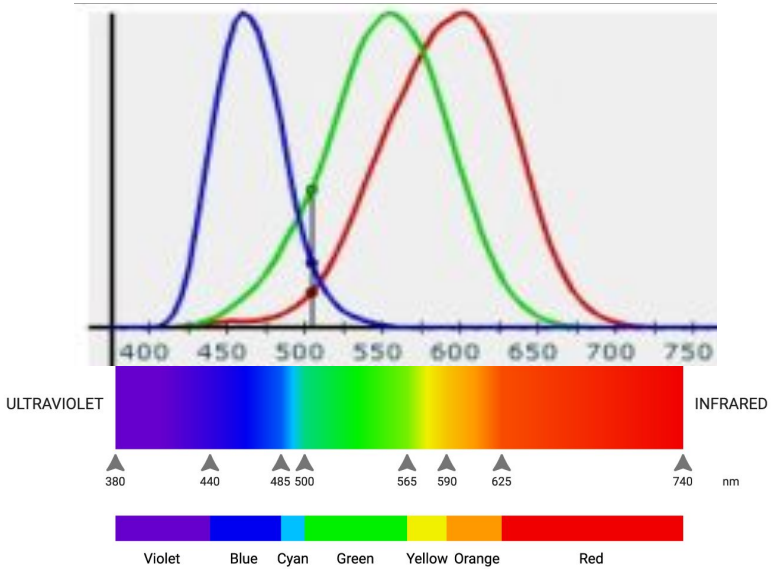
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)



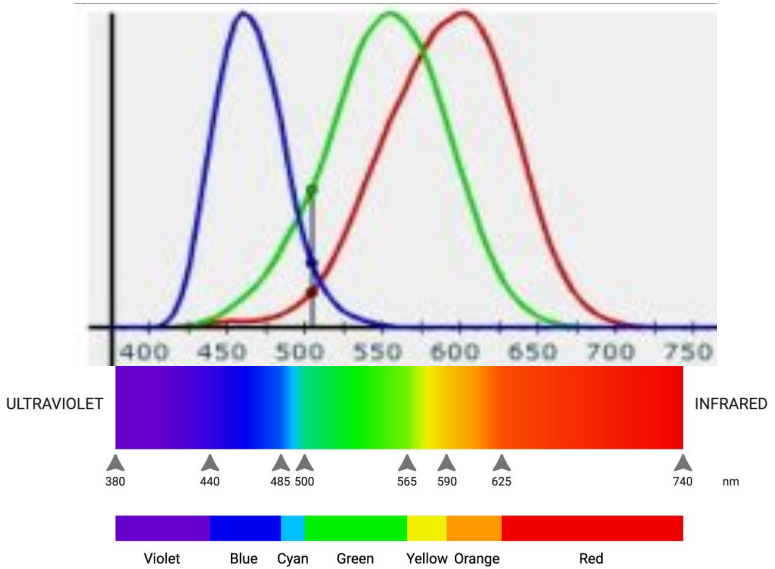
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)
  - L/M cones would fire



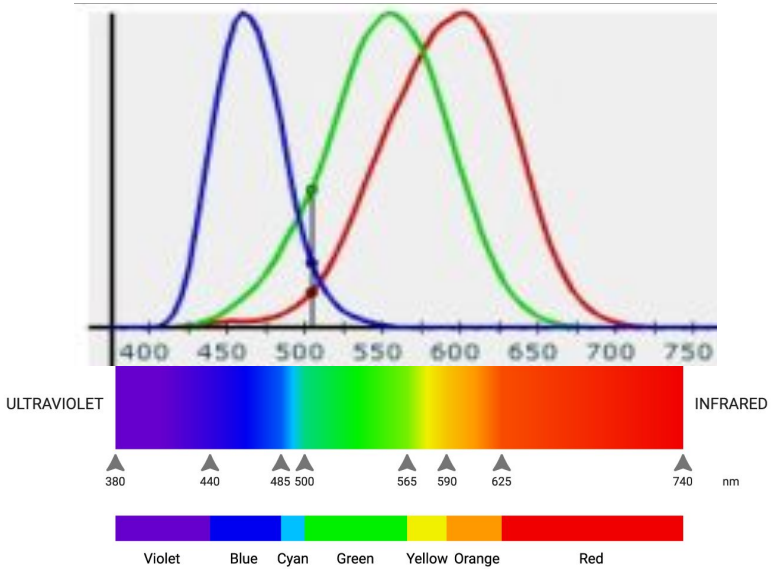
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)
  - L/M cones would fire
- Ex: white light



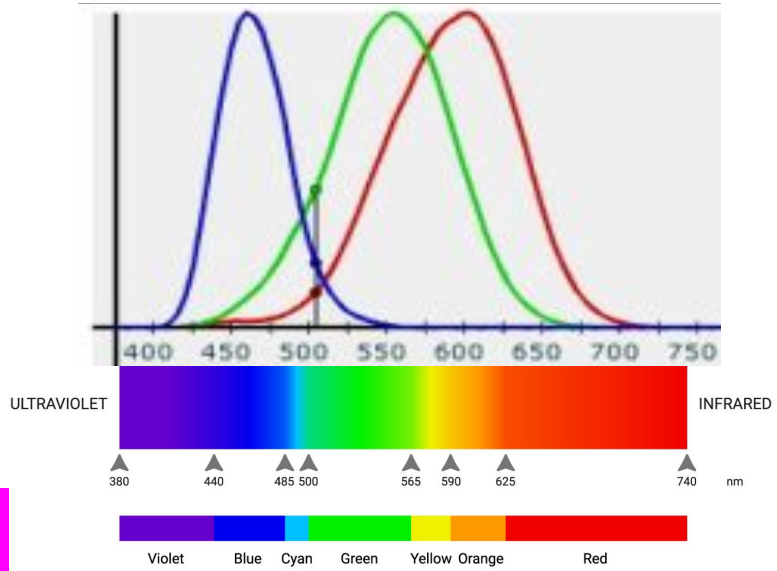
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)
  - L/M cones would fire
- Ex: white light
  - All cones saturated



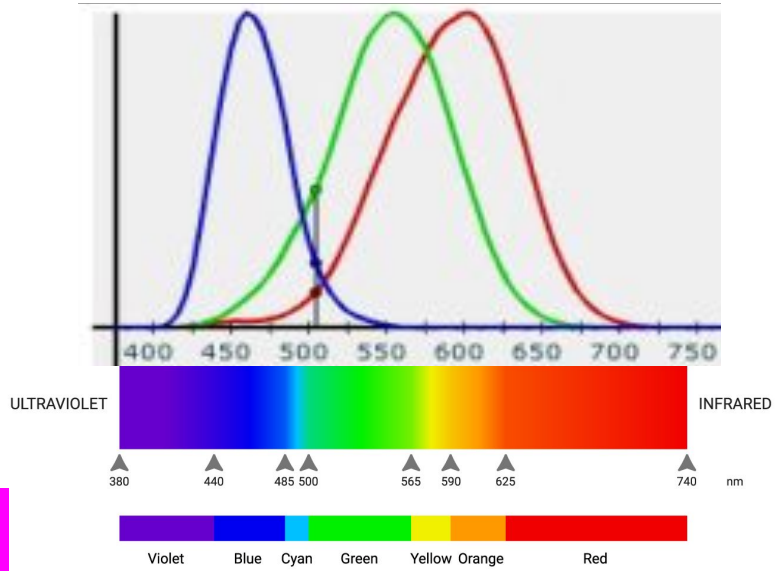
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)
  - L/M cones would fire
- Ex: white light
  - All cones saturated
- Ex: magenta



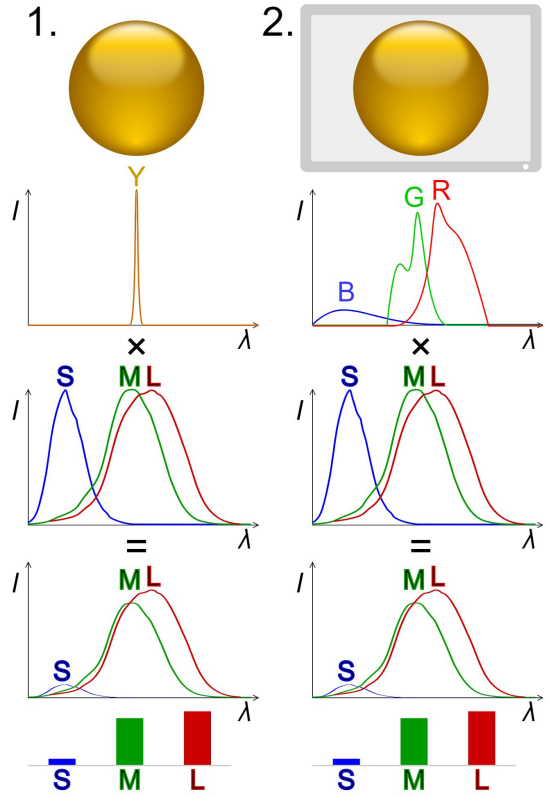
Light and the Human Eye

- Ex: a dandelion (570nm wavelength)
  - L/M cones would fire
- Ex: white light
  - All cones saturated
- Ex: magenta
  - Mix of blue and red light
  - Both blue and red tuned cones would spike



Mixed wavelengths vs single wavelengths

- Given any human perceived “color:”
- Can use **3 single wavelength** lasers to fool a human into mistakenly thinking that the laser combo matches that color
  - Aka turn into a single wavelength
  - Ex lasers: R = 700nm, G = 546nm, B = 435nm
- This is doable because each there are **three** different cones, and each can only send **one** signal



Mixed wavelengths vs single wavelengths

- Ex: magenta
  - Magenta objects reflect long/short wavelengths but absorb mid-length wavelengths
  - Cones don’t measure wavelength and rather how much they’re stimulated

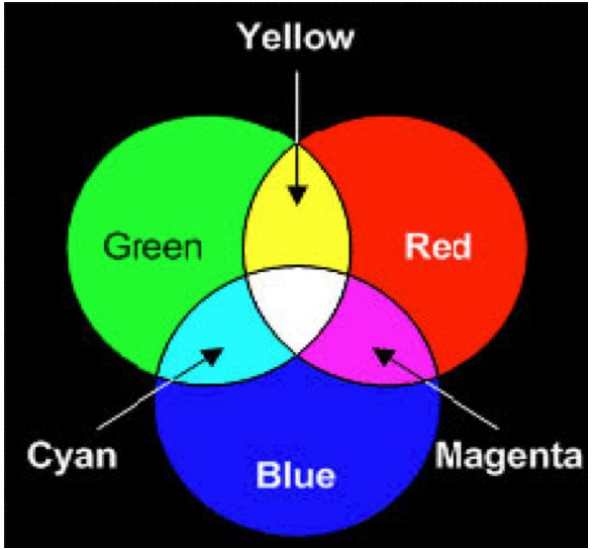


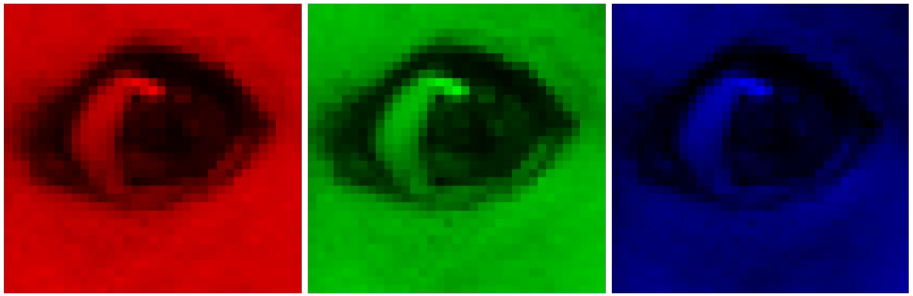
Image Representation

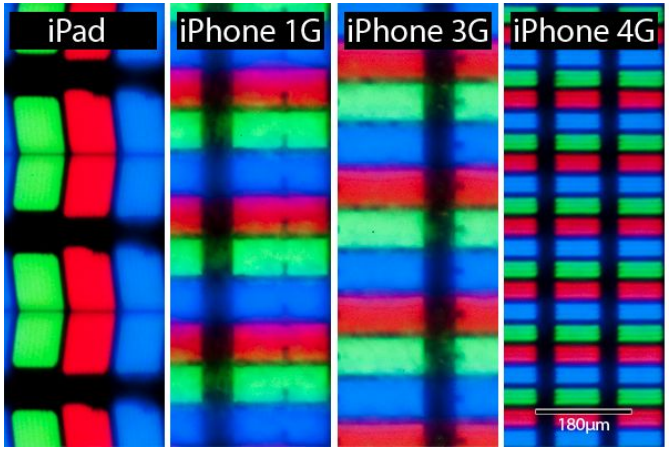
- Graphics: simulating the real world **as the human eye sees it!**

- Algorithms/hardware for displays, cameras, printing, etc. have been optimized for the human eye

Image Representation

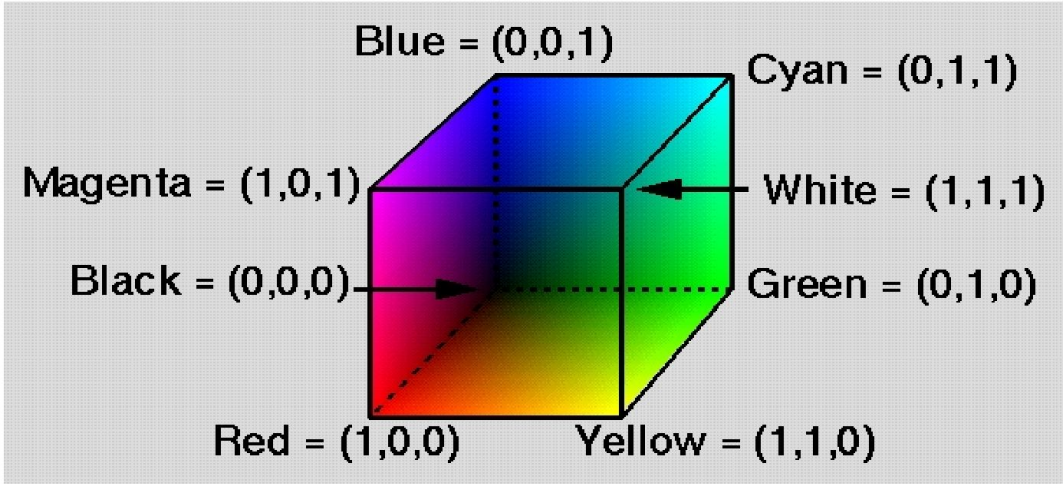
- Images = 2D grid of color (pixels), often in RGB
- Displays = grid of RGB LED lights
- Digital cameras = grid of RGB sensors
- Why?
  - The way our eyes process color!





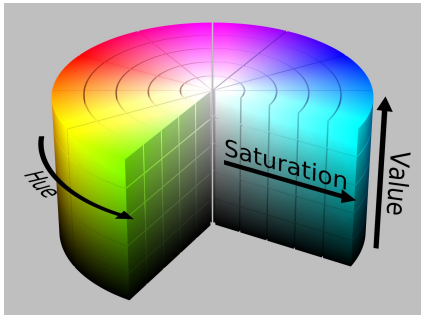
RGB Color Space

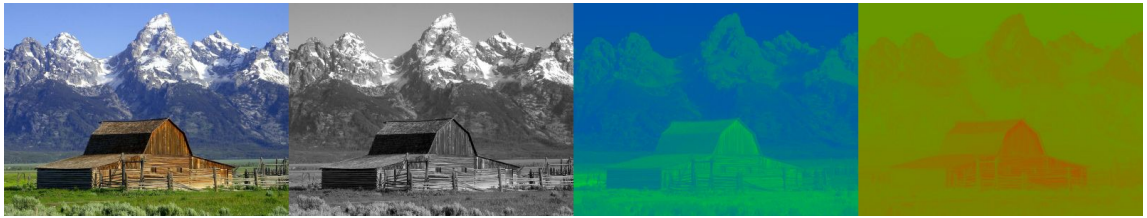
- Each image is a **2D spatial grid of colors** where (0,0,0) = black and (1,1,1) = white



Other Color Spaces

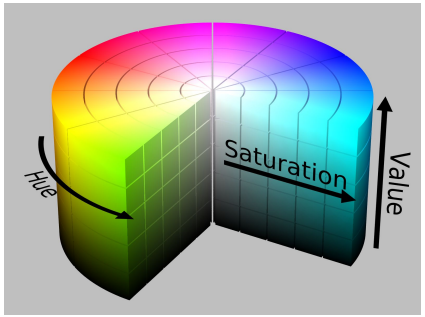
- **HSV** for user interfaces
  - Closer to how we reason about color (purity/intensity)

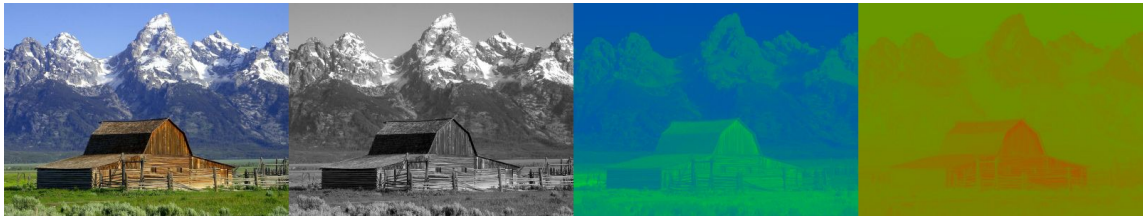




Other Color Spaces

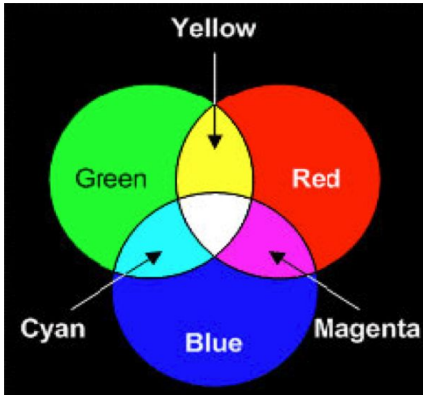
- **HSV** for user interfaces
  - Closer to how we reason about color (purity/intensity)
- **YUV** for video compression
  - Legacy model from black and white TV, but still used to store video
  - Human eyes are more spatially sensitive to intensity than color





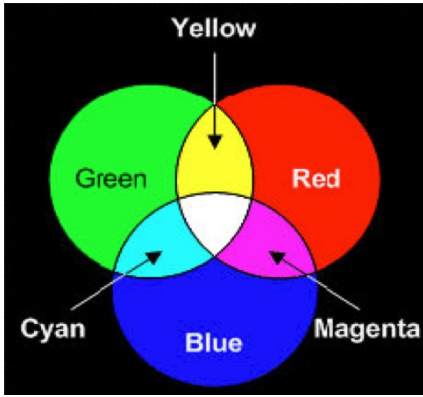
Additive Light vs Subtractive Materials

- Additive color
  - Start with black
  - Every light source you add contributes more weight



Additive Light vs Subtractive Materials

- Additive color
  - Start with black
  - Every light source you add contributes more weight
- Subtractive color
  - Start with white
  - Each ink layer removes certain wavelengths by absorbing them
  - Material colors = defined by what light they absorb
  - Ex: a cyan material absorbs red light

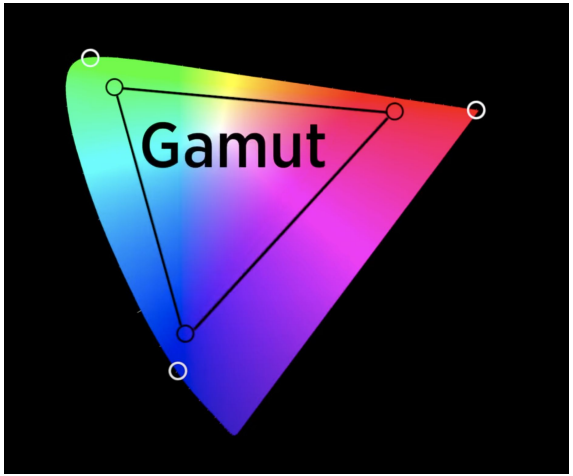


8 bit quantization: 0-255

- Quantization for efficient storage, but need to make it look smooth
- We typically use 256 levels for brightness
  - Store R,G,B each ranging from 0-255
- High Dynamic Range image formats use an even larger range

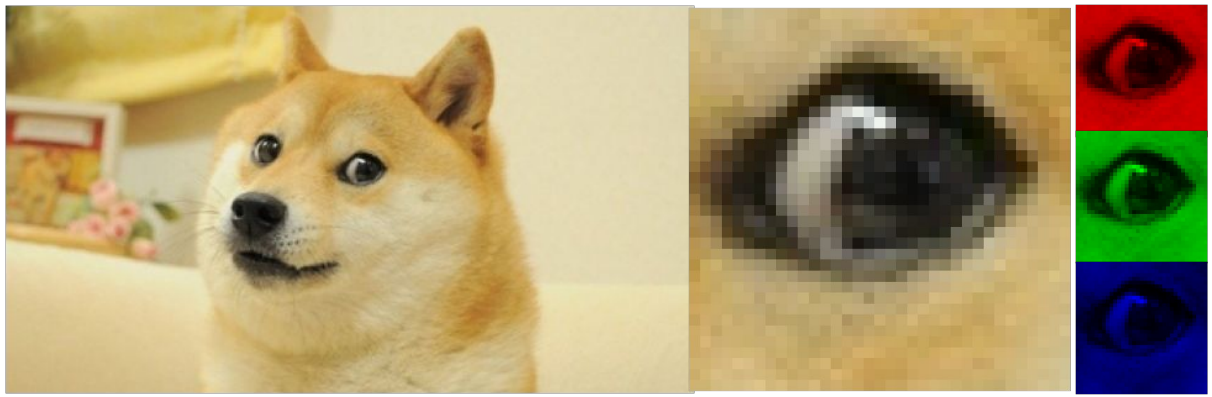
Color Gamut

- A way of converting between storage and display
- Even if a display is composed of three different types of RGB light, it doesn't mean it can represent all the possible colors a human can perceive
- Can lead to differences in color between displays:



Summary so far

- Understanding human perception allows for MANY optimizations in both software and hardware
  - RGB LED displays, camera sensors, video compression, …
- The images we create are not intended to duplicate reality, only to fool humans into believing such!

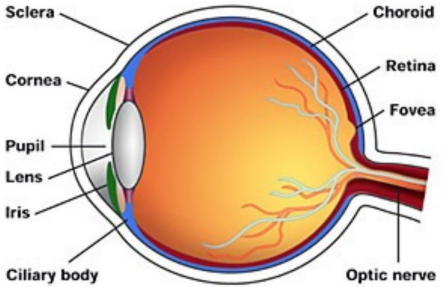


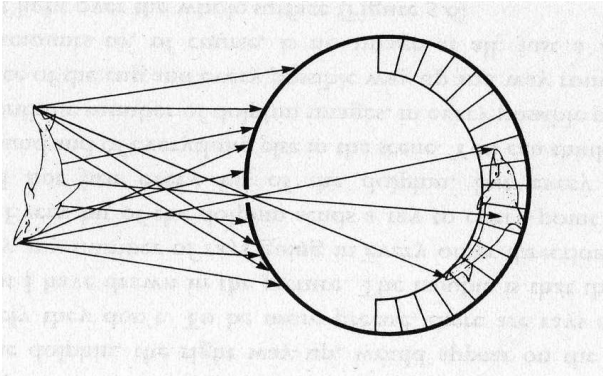
The human eye: part 2

- Light emanates off of every point of an object outwards in every direction

The human eye: part 2

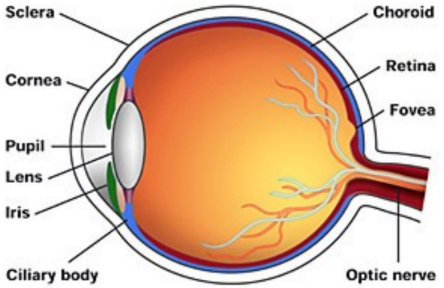
- Light emanates off of every point of an object outwards in every direction
- The **pupil** restricts the entry of light so that each cone only receives light emanating from a small region on the object, giving spatial detail

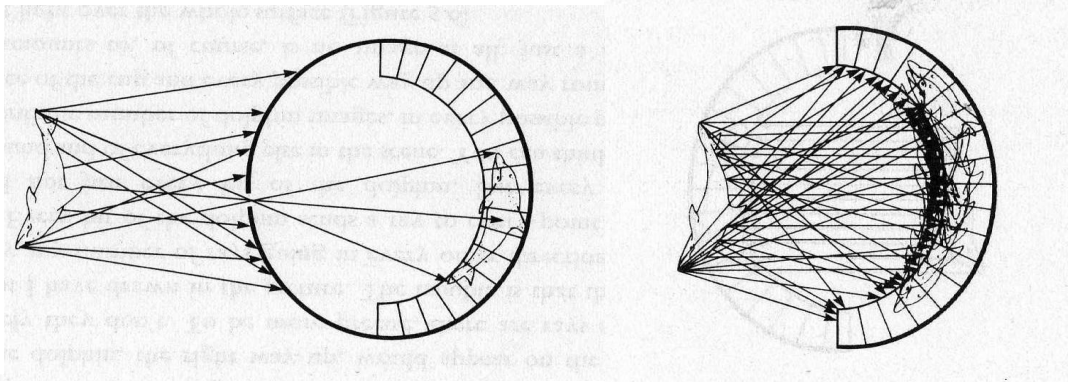




The human eye: part 2

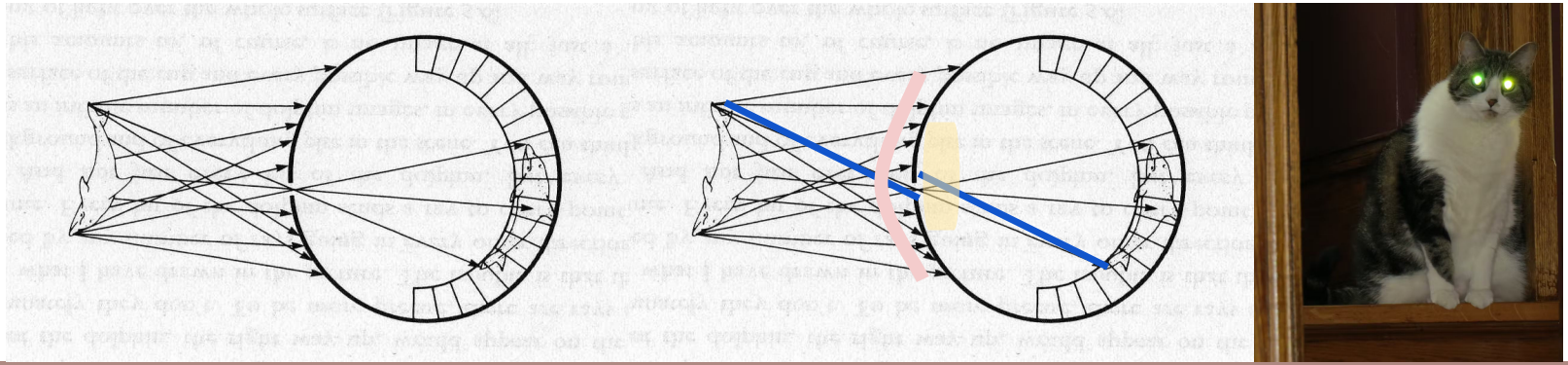
- Light emanates off of every point of an object outwards in every direction
- The **pupil** restricts the entry of light so that each cone only receives light emanating from a small region on the object, giving spatial detail
- Without the pupil, light from every part of an object would hit the same cone on our eye, blurring the image





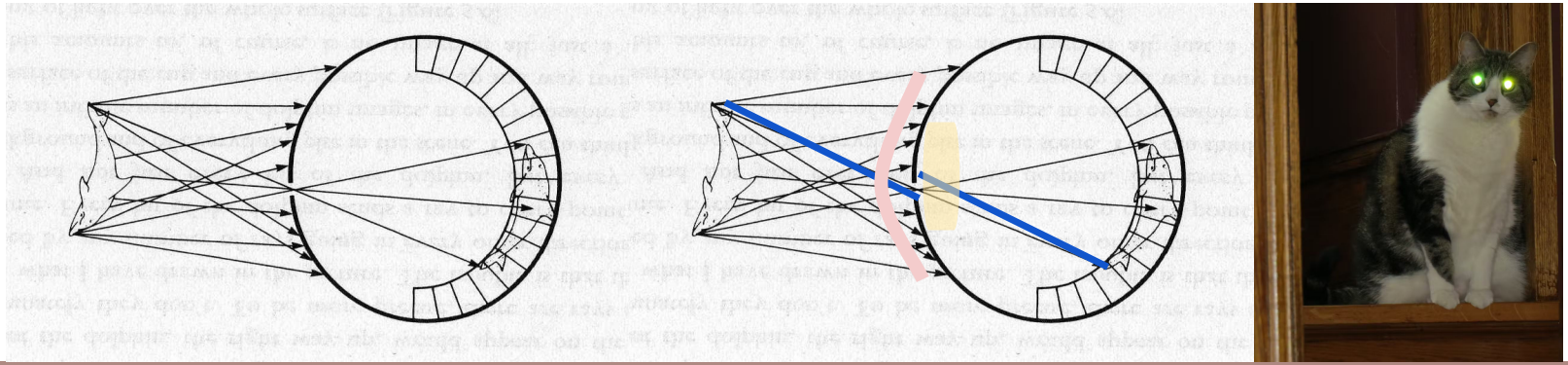
The human eye: part 2

- The pupil restricts the entry of light
  - In the dark, it dilates to allow for more light to enter
  - Cats have wide pupils to let more light in – lots of reflects from the back of their eyes at night!



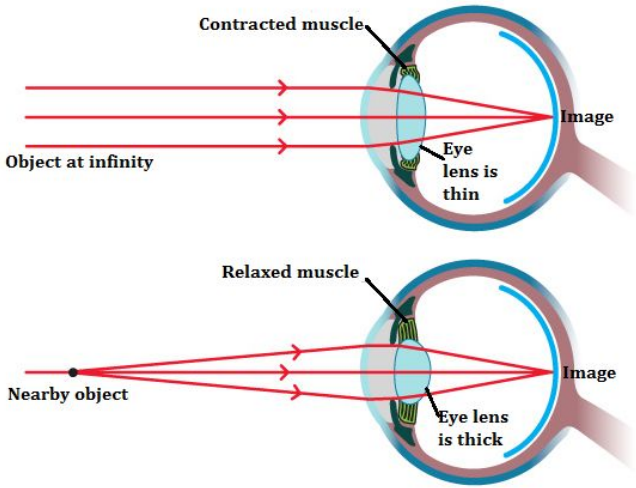
The human eye: part 2

- The pupil restricts the entry of light
  - In the dark, it dilates to allow for more light to enter
  - Cats have wide pupils to let more light in – lots of reflects from the back of their eyes at night!
- We rely on our **cornea** and **lens** to bend light and make everything converge properly



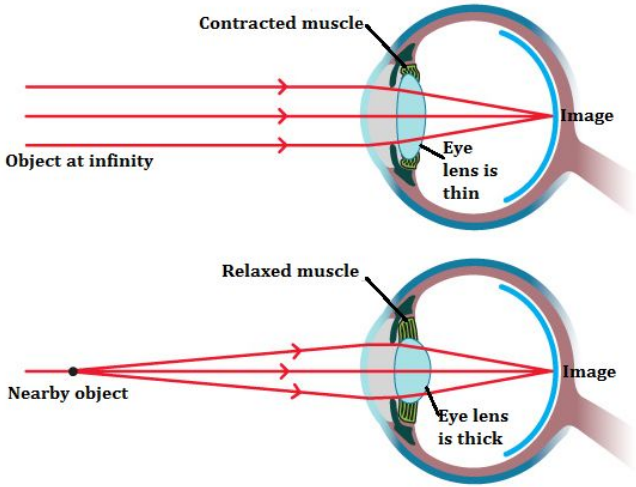
The human eye: part 2

- We rely on our **cornea** and **lens** to bend light and make everything converge properly
- Light travels in straight lines and reflect/refract based on the material



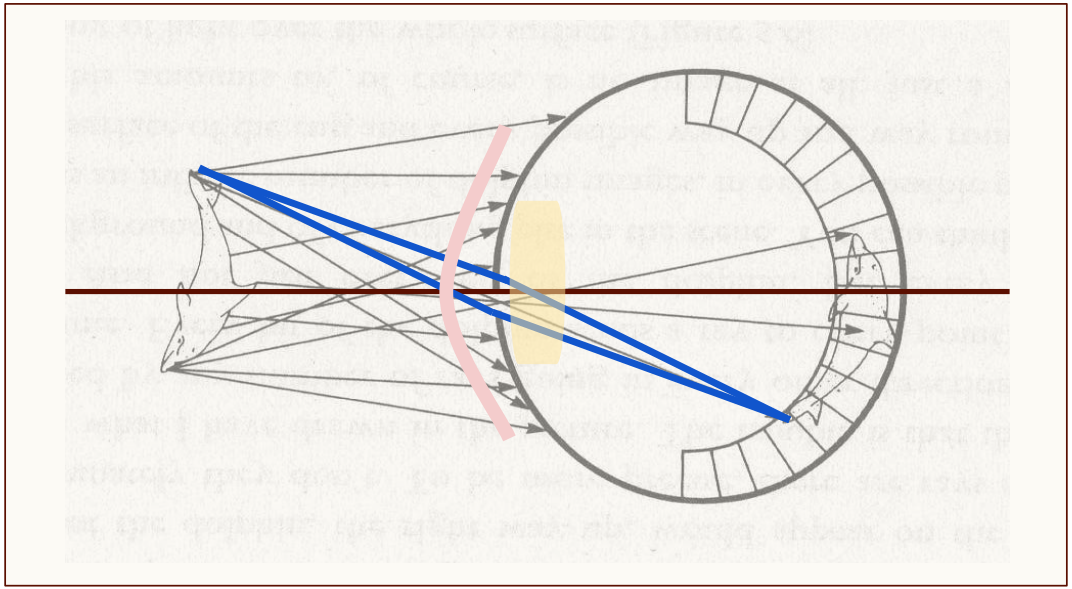
The human eye: part 2

- We rely on our **cornea** and **lens** to bend light and make everything converge properly
- Light travels in straight lines and reflect/refract based on the material
- The **lens** changes its shape so that objects at different depths are in focus
  - Ex: Hold your palm out and stare at it. The projector/laptop screen becomes out of focus



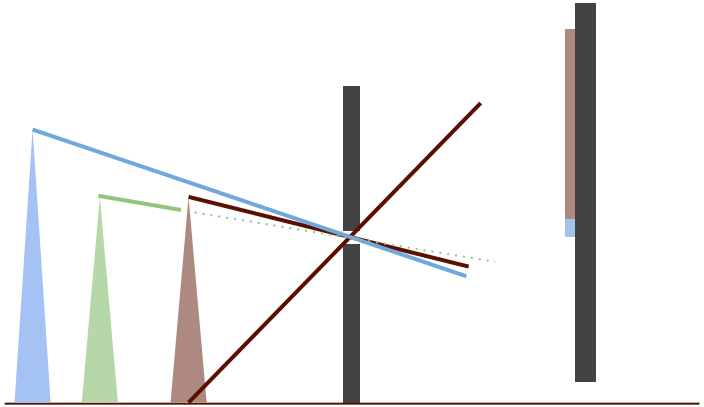
The human eye: part 2

- Note: images as assembled on our retina are “flipped,” so our brains flip them back!



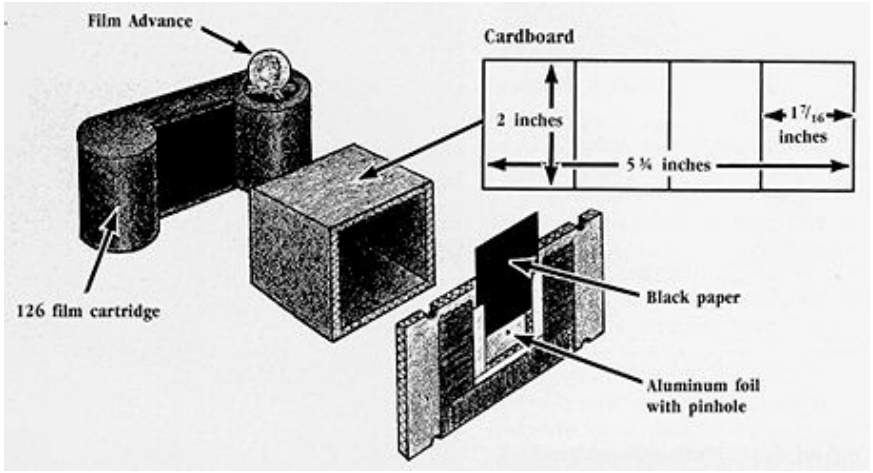
Pinhole Camera

- The first camera (camera obscura effect)!
  - Been around for thousands of years
  - Now used to safely observe solar eclipses
- Make a really small hole on the wall of a very dark room when it's bright out



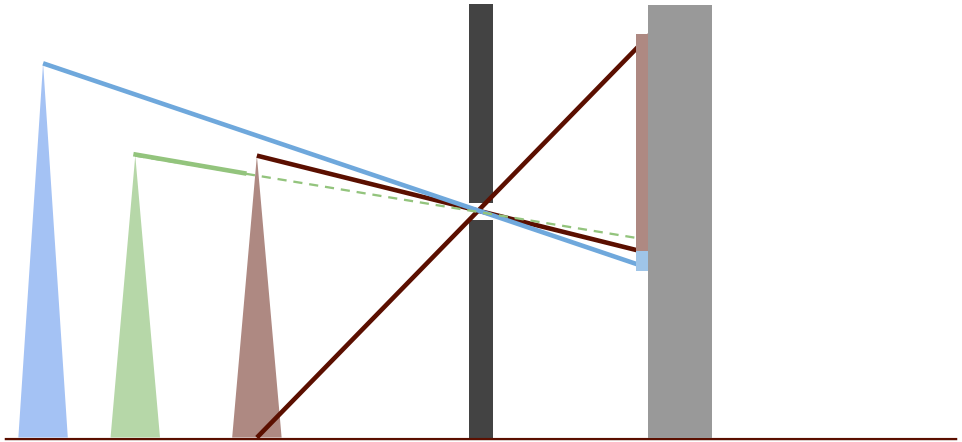
Pinhole Camera

- Can make a simple camera this way!
  - Place film that reacts to light in a dark chamber
  - Cover up the hole once sufficient light has entered and accumulated chemical reactions on the film



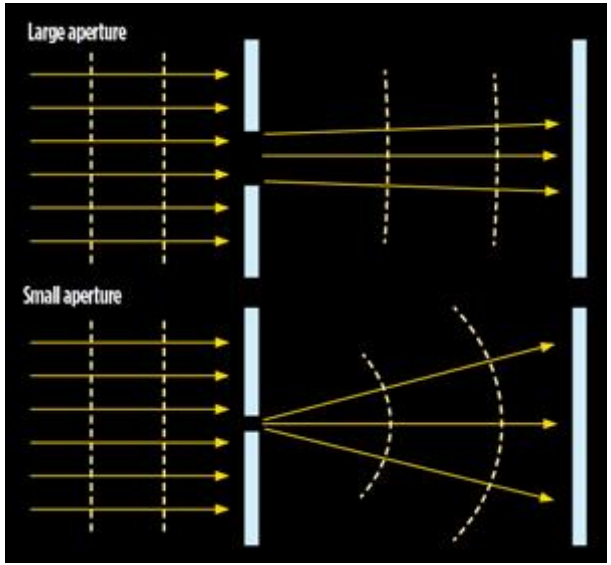
Pinhole Camera

- The pinhole model is a simplification of human perception but…
- It demonstrates why objects get occluded in our vision and why objects that are farther away appear smaller



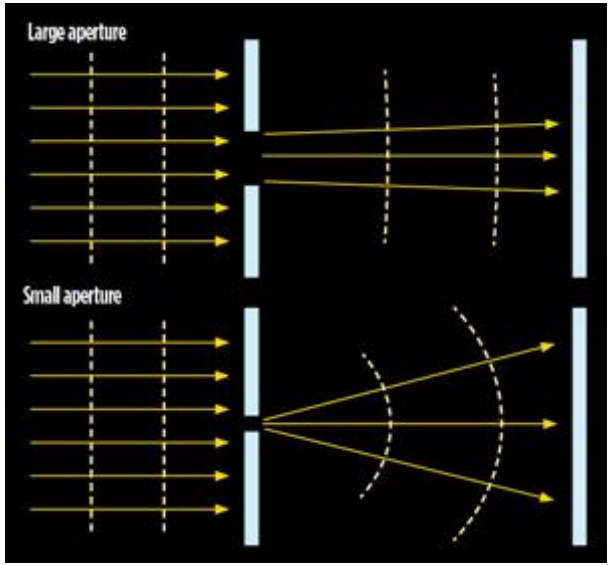
Pinhole Camera

- Disadvantages
  - Very little light comes in, so you need to **expose** for a long time to allow enough light to enter



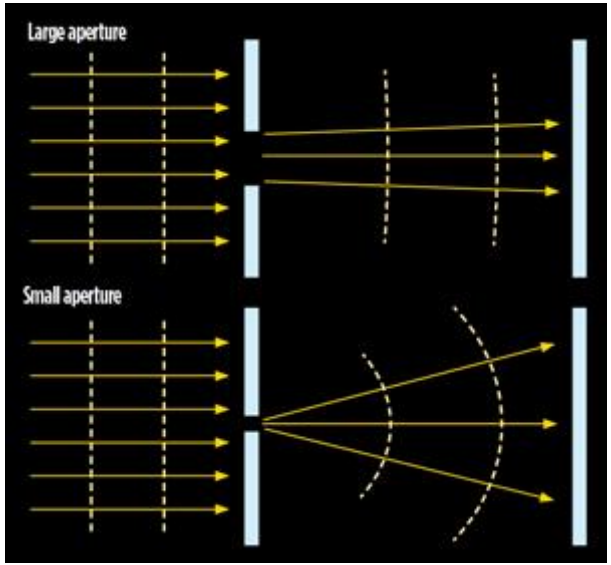
Pinhole Camera

- Disadvantages
  - Very little light comes in, so you need to **expose** for a long time to allow enough light to enter
  - Physical limits to pupil size. Diffraction means we can’t go too small



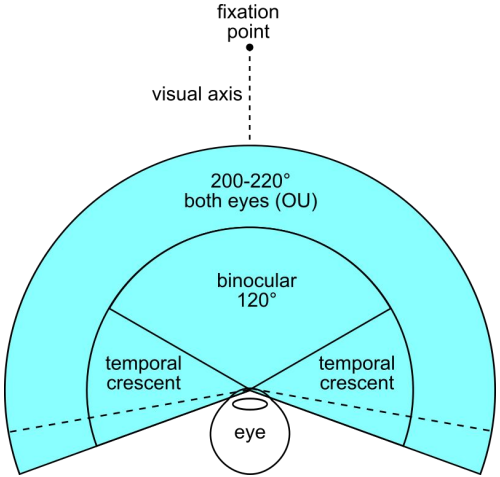
Pinhole Camera

- Disadvantages
  - Very little light comes in, so you need to **expose** for a long time to allow enough light to enter
  - Physical limits to pupil size. Diffraction means we can't go too small
- But we don't have this problem with virtual cameras!



Field of View

- Determines is visible/capturable on the film (retina)
- For pinhole camera: depends mostly on **size of film**
- For eyes / lens-based cameras: depends on the **lens**
  - We will return to lens-based cameras later



Lecture Outline

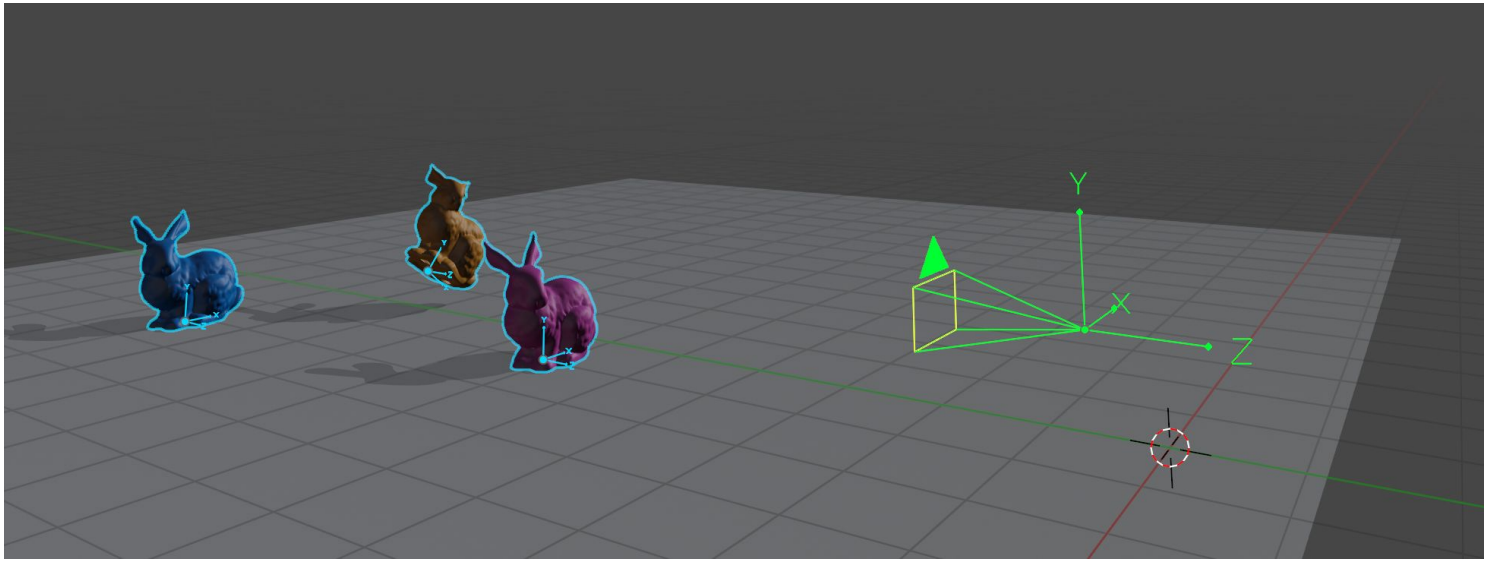
- In real life
  - The human eye (pupil, cornea, lens)
  - RGB color space
  - The pinhole camera
- In the rasterizer
  - Coordinate frames
  - Perspective & orthographic projection
  - Normalized coordinates & the z-buffer algorithm
- Lens-based cameras
- Beyond rasterization

3D → 2D

- There are **objects** in the **world**. We point a **camera** at them and an image appears on **film**
- To model a pinhole camera:
  - Where the camera is and where it's pointing (camera coordinate system)
- To model image creation:
  - Field of view, image resolution (film size)

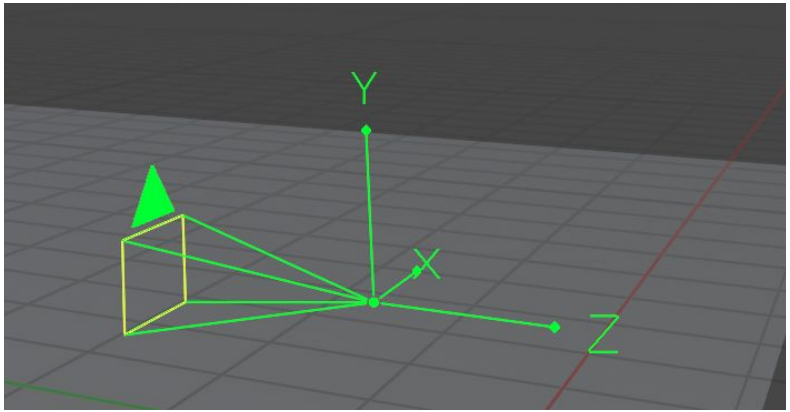
Coordinate Frames

- Object space, world space, **camera space**



Standard Camera Frame

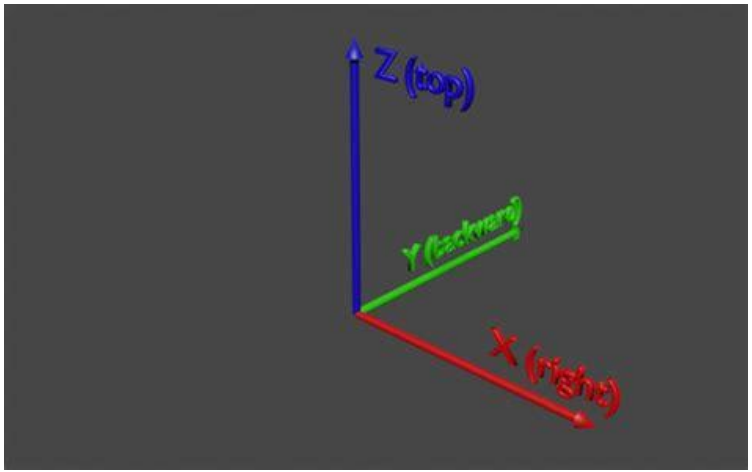
- For rendering: X-right, Y-up, **Z-out**
- In rendering/graphics, we prefer this "Z-out" coordinate
  - This way, x&y correspond to pixel location



Standard Camera Frame

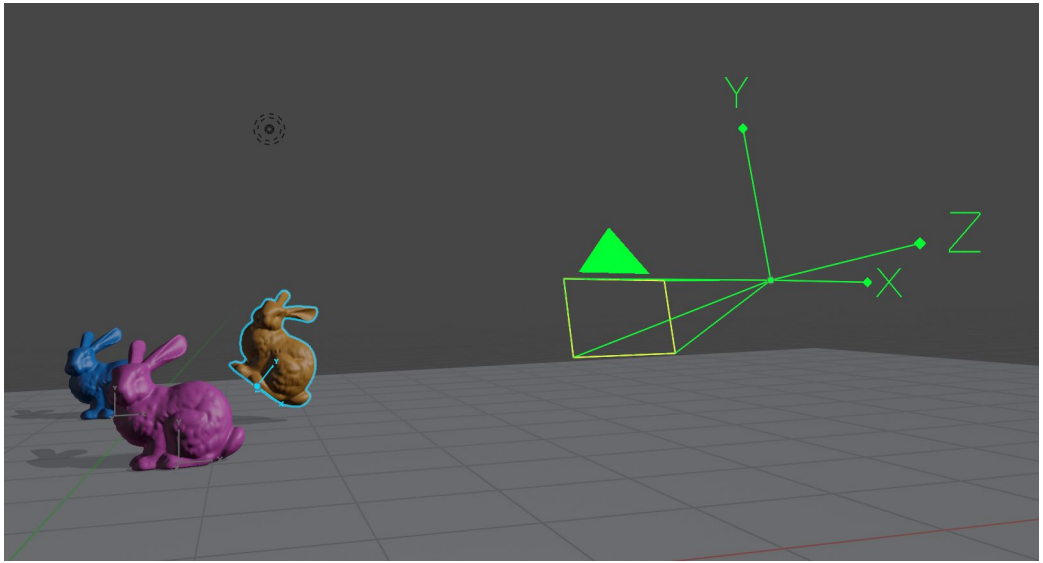
- For rendering: X-right, Y-up, **Z-out**
- In rendering/graphics, we prefer this "Z-out" coordinate
  - This way, x&y correspond to pixel location

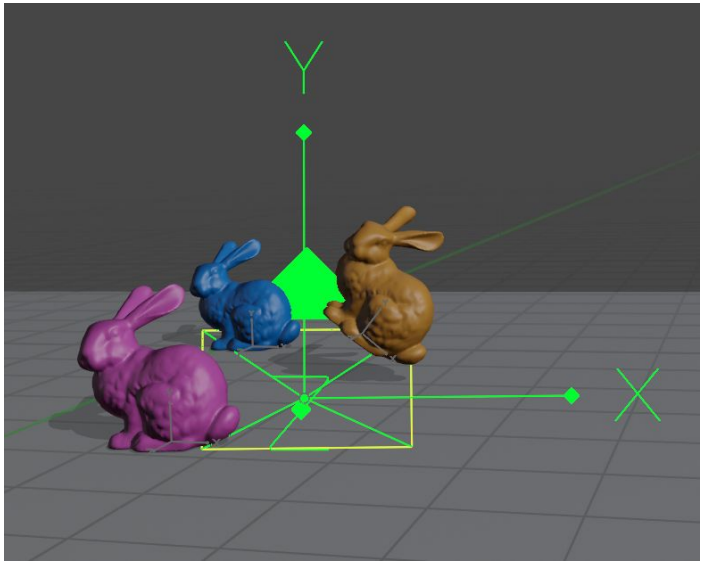
- For modeling: X-right, Y-out, **Z-up**
- In modeling software, they often use "Z-up"
  - This way, x&y is the "flat" ground plane



Camera to Film

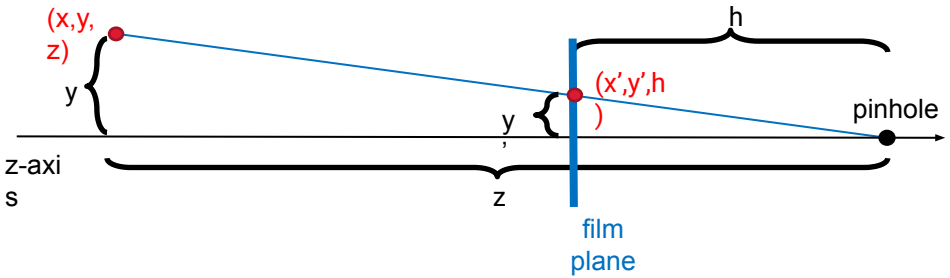
- Once we're in camera space coordinates what do we do?
  - Reminder: origin at (0,0,0), x right, y up

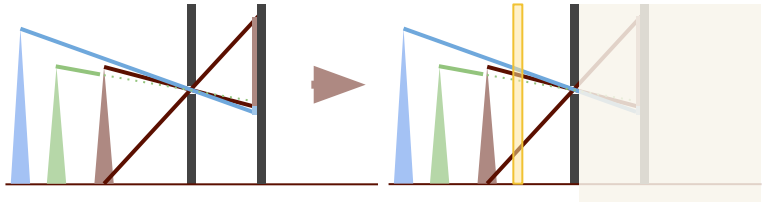




Camera to Film: Modeling Pinhole Camera

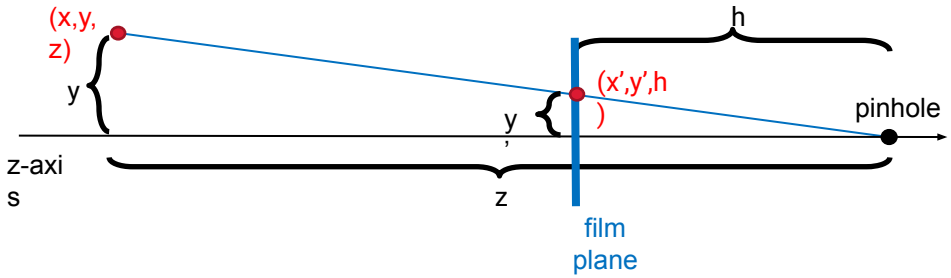
- Move the film **out in front** of the pinhole, so that the image is **not** upside down
- With the pinhole at the origin (0,0,0), points at (x,y,z) get projected to (x',y',h) where h is the distance between the film and the pinhole





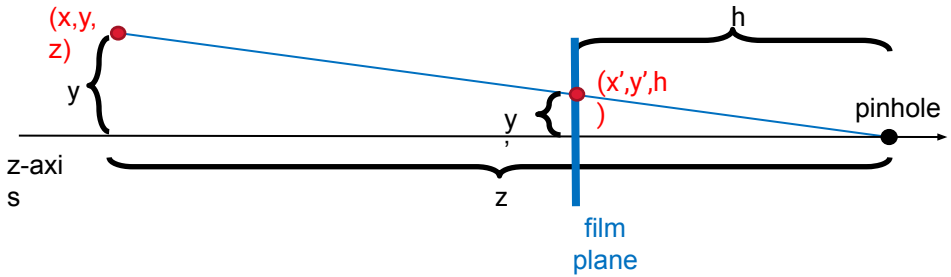
Camera to Film: Modeling Pinhole Camera

- Because the pinhole is at the origin, we can use ratios:
  - The point is at distance **z** from the pinhole
  - The film is at distance **h** from the pinhole



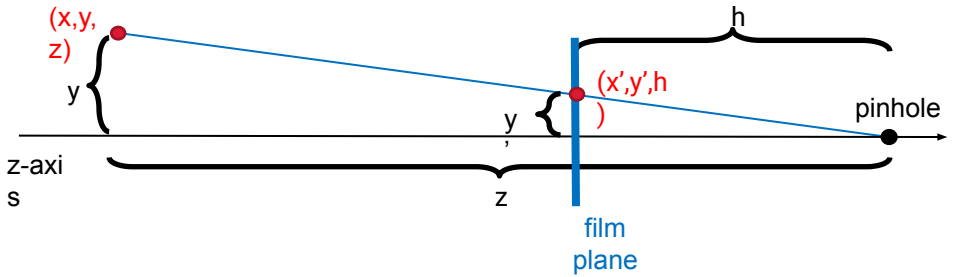
Camera to Film: Modeling Pinhole Camera

- Because the pinhole is at the origin, we can use ratios:
  - The point is at distance **z** from the pinhole
  - The film is at distance **h** from the pinhole
  - So… the projection coordinates scale by **h/z:**
  - **Projection function:** $P(x, y, z) = \left(\frac{h}{z}x,\ \frac{h}{z}y,\ \frac{h}{z}z\right) = \left(\frac{h}{z}x,\ \frac{h}{z}y,\ h\right)$



Camera to Film: Modeling Pinhole Camera

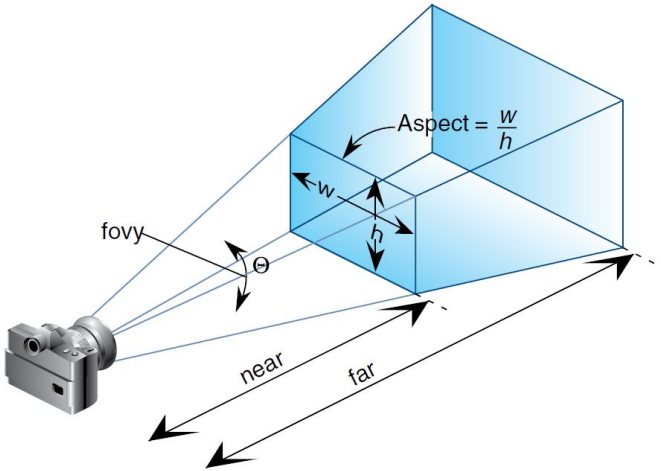
- Because the pinhole is at the origin, we can use ratios:
  - The point is at distance **z** from the pinhole
  - The film is at distance **h** from the pinhole
  - So… the projection coordinates scale by **h/z:**
  - **Projection function:** $P(x, y, z) = \left(\frac{h}{z}x,\ \frac{h}{z}y,\ \frac{h}{z}z\right) = \left(\frac{h}{z}x,\ \frac{h}{z}y,\ h\right)$



- **Normalized pixel coordinates:** $\left(\frac{hx}{z}, \frac{hy}{z}\right)$ Is the normalized pixel coordinate for film at depth h
- This is perspective projection!

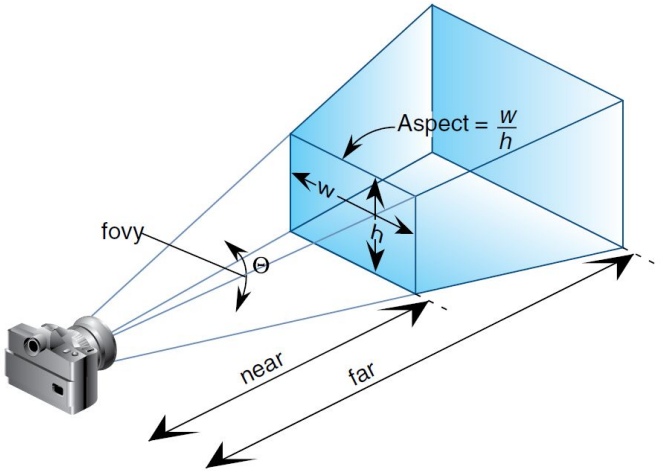
Viewing Frustum

- Only render objects that project to **within the boundaries** of the film



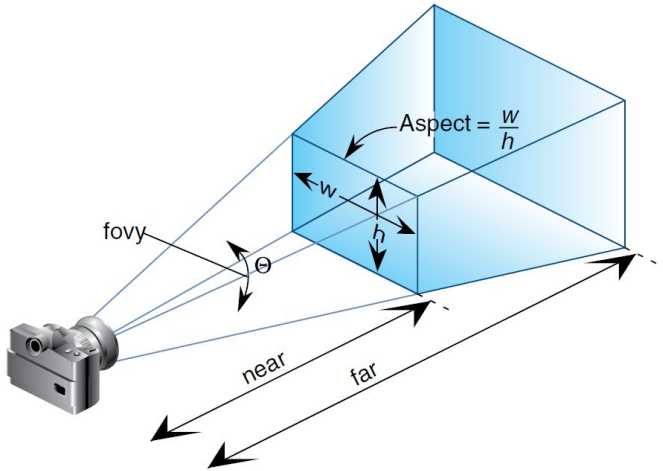
Viewing Frustum

- Only render objects that project to **within the boundaries** of the film
- Only render objects **further away from the camera** than the film



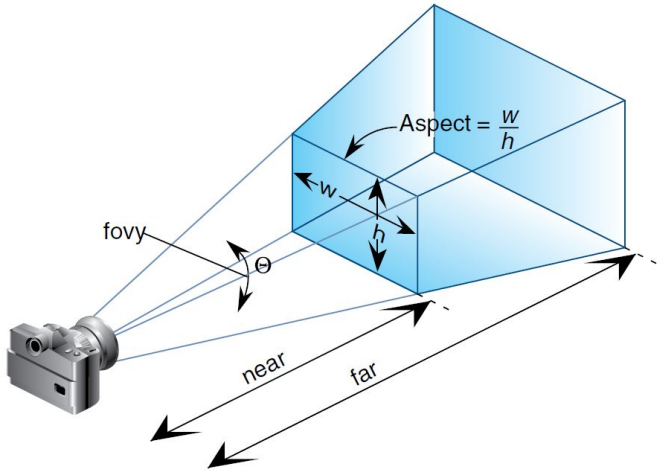
Viewing Frustum

- Only render objects that project to **within the boundaries** of the film
- Only render objects **further away from the camera** than the film
- Film is the **front clipping plane** and you can also add a **back clipping plane** for efficiency
- Volume between **front** and **back** clipping plane is the **<u>viewing frustum</u>** (shown in blue)



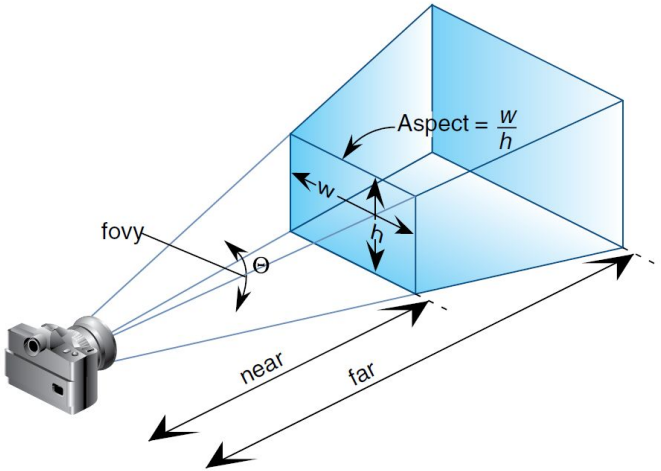
Viewing Frustum

- Only render objects that project to **within the boundaries** of the film
- Only render objects **further away from the camera** than the film
- Film is the **front clipping plane** and you can also add a **back clipping plane** for efficiency
- Volume between **front** and **back** clipping plane is the **<u>viewing frustum</u>** (shown in blue)
- We have a lot of control over the virtual camera!



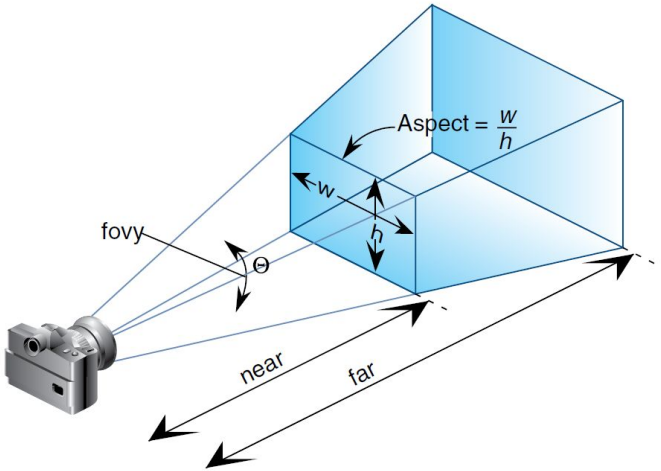
Viewing Frustum

- Ensure near/far clipping planes have enough space between them to contain the scene



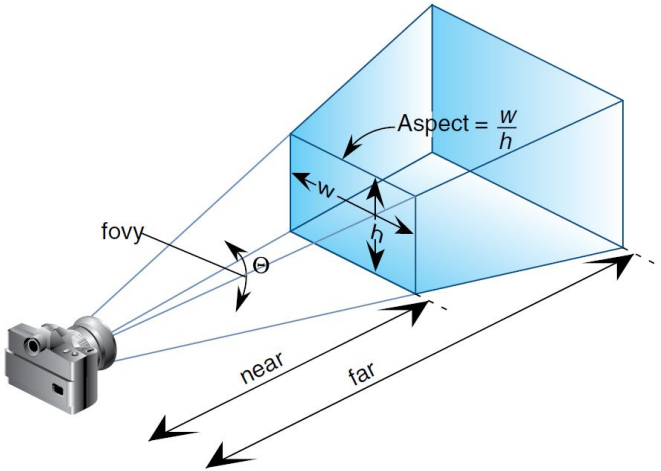
Viewing Frustum

- Ensure near/far clipping planes have enough space between them to contain the scene
- Make sure objects of interest are inside the viewing frustum



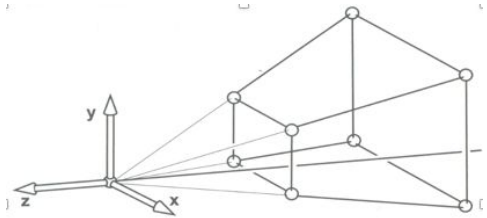
Viewing Frustum

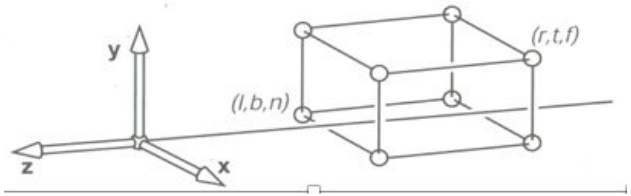
- Ensure near/far clipping planes have enough space between them to contain the scene
- Make sure objects of interest are inside the viewing frustum
- In general, don’t set the near clipping plane at the camera aperture
  - This can cause extreme distortion

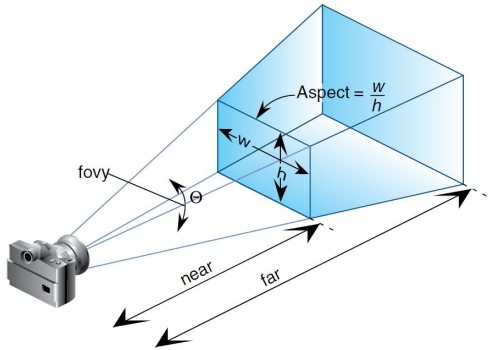


Viewing Frustum

- Bounds on x and y: field of view
- Bounds on z: between the front and back clipping planes

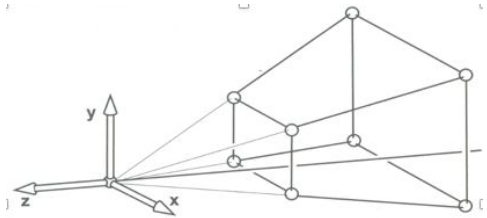


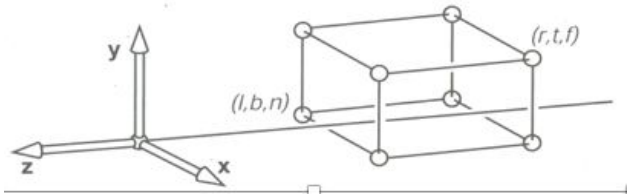


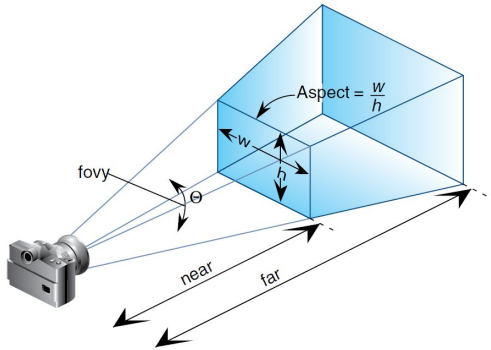


Viewing Frustum

- Bounds on x and y: field of view
- Bounds on z: between the front and back clipping planes
- How do we efficiently convert the frustum into a bounding box?

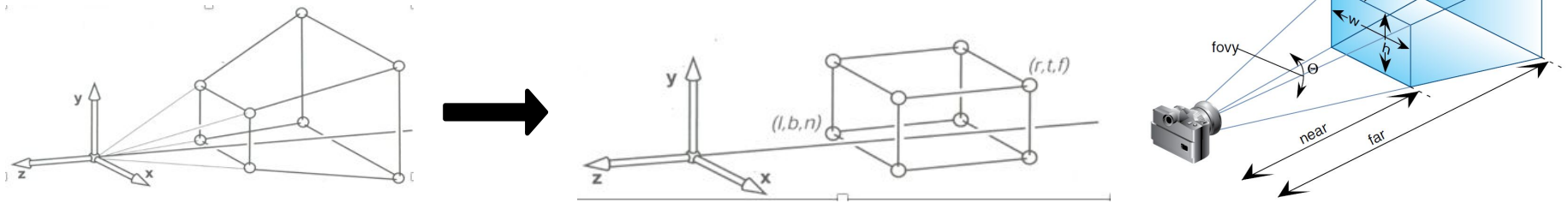






Viewing Frustum

- Bounds on x and y: field of view
- Bounds on z: between the front and back clipping planes
- How do we efficiently convert the frustum into a bounding box?
  - Yay homogeneous matrices again!



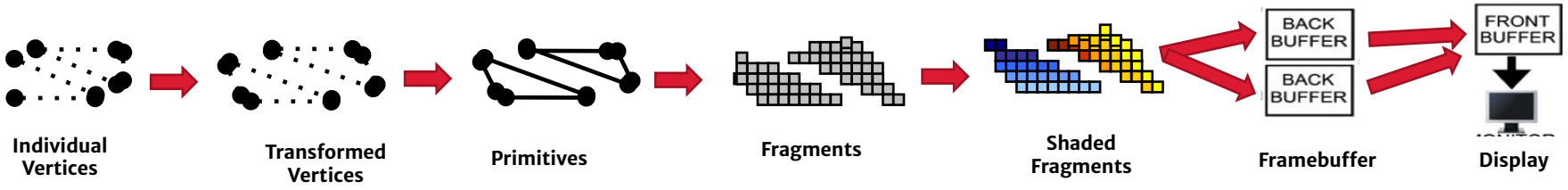
Projection, Field of View, Clipping Plane

- Transform viewing frustum to an orthographic bounding box (1x1x1 cube)
- Instead of projecting everything to the plane at a fixed depth, project to a cube first
- Then clip anything that's not within -1~1
- Note: rx,ry correspond to film boundaries, d0/d1 are clipping planes

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \rightarrow \begin{pmatrix} \frac{1}{r_x} & 0 & 0 & 0 \\ 0 & \frac{1}{r_y} & 0 & 0 \\ 0 & 0 & \frac{d0+d1}{d0-d1} & 2\frac{d0d1}{d0-d1} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

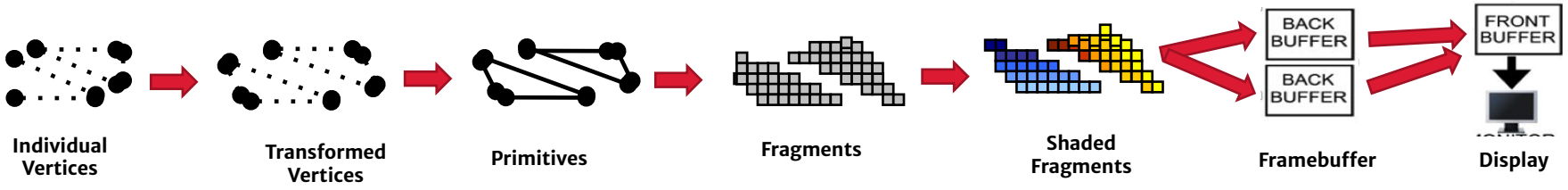
With the Rasterizer…

1. Vertex shader outputs vertex attributes and orthogonal clipping space positions
2. Vertex data assembled into triangles, **data outside the viewing frustum is discarded**



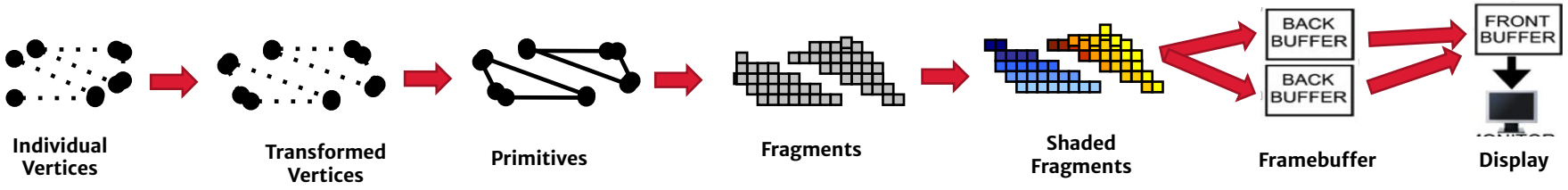
With the Rasterizer…

1. Vertex shader outputs vertex attributes and orthogonal clipping space positions
2. Vertex data assembled into triangles, **data outside the viewing frustum is discarded**
3. For each triangle, rasterize vertex attributes (barycentric interpolation)
4. Fragment shader takes interpolated attributes and computes color



With the Rasterizer…

1. Vertex shader outputs vertex attributes and orthogonal clipping space positions
2. Vertex data assembled into triangles, **data outside the viewing frustum is discarded**
3. For each triangle, rasterize vertex attributes (barycentric interpolation)
4. Fragment shader takes interpolated attributes and computes color
5. Before we get to the final image:
   still need to **determine which triangles occlude other triangles!**



Z-Buffer Algorithm

- We used x&y for image space positions, now use z to determine depth

Z-Buffer Algorithm

- We used x&y for image space positions, now use z to determine depth

Create color buffer C(x,y) to store color and z-buffer Z(x,y) to store depth
Set Z(x,y)= - infinity for all x,y

Z-Buffer Algorithm

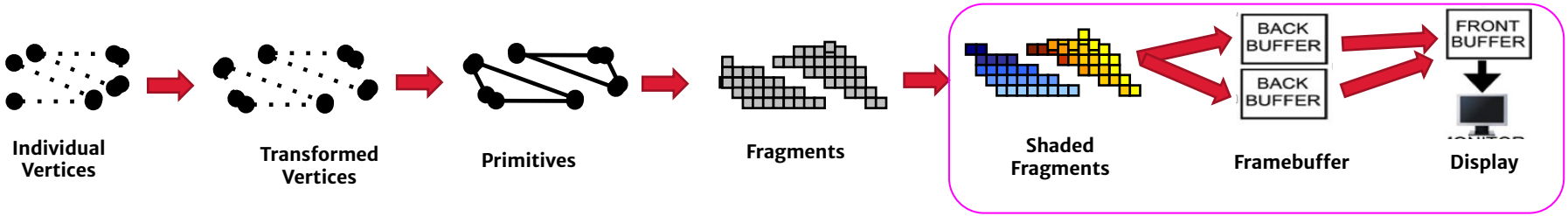
- We used x&y for image space positions, now use z to determine depth

```
Create color buffer C(x,y) to store color and z-buffer Z(x,y) to store depth
Set Z(x,y)= - infinity for all x,y
For each fragment:
    For each coordinate (x,y,z) in the fragment and its corresponding
    color c:
        If Z(x,y) < z:
            C(x,y)=c
            Z(x,y)=z
```

We've Completed the Rasterization Pipeline!

- Vertex shader outputs vertex attributes and orthogonal clipping space positions
- Vertex data assembled into triangles, data outside the viewing frustum is discarded
- For each triangle, rasterize vertex attributes (barycentric interpolation)
- Fragment shader takes interpolated attributes and computes color
- Use the **z-buffer** to help assemble pixel colors into the full image

- (There's more functionality to OpenGL, but we've walked through one instance of the entire pipeline!)

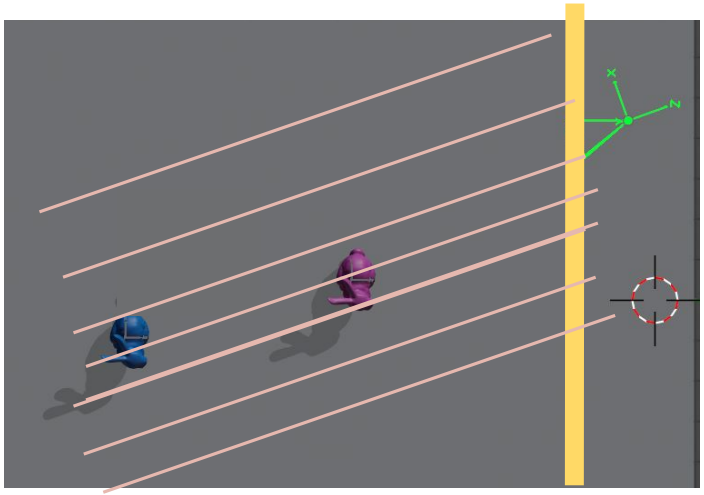


Lecture Outline

- In real life
  - The human eye (pupil, cornea, lens)
  - RGB color space
  - The pinhole camera
- In the rasterizer
  - Coordinate frames
  - Perspective & orthographic projection
  - Normalized coordinates & the z-buffer algorithm
- Lens-based cameras
- Beyond rasterization

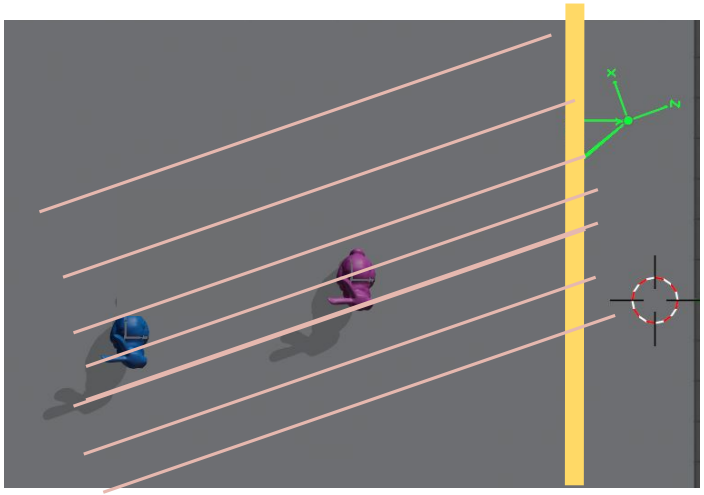
Perspective vs Orthographic Projection

- Orthographic projection = using a 4x4 identity matrix for camera projection
  - Aka all projection lines are orthogonal to the film plane



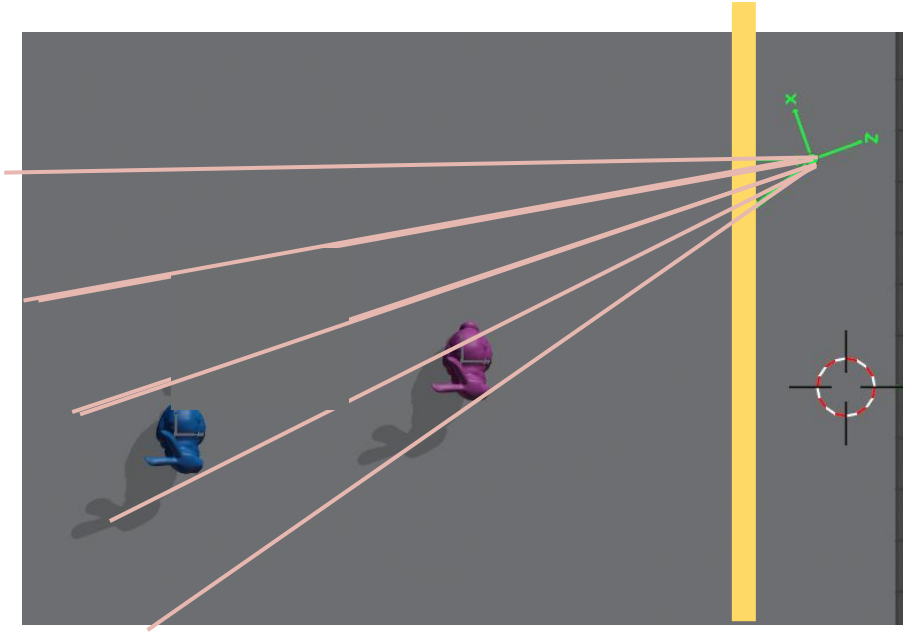
Perspective vs Orthographic Projection

- Orthographic projection = using a 4x4 identity matrix for camera projection
  - Aka all projection lines are orthogonal to the film plane



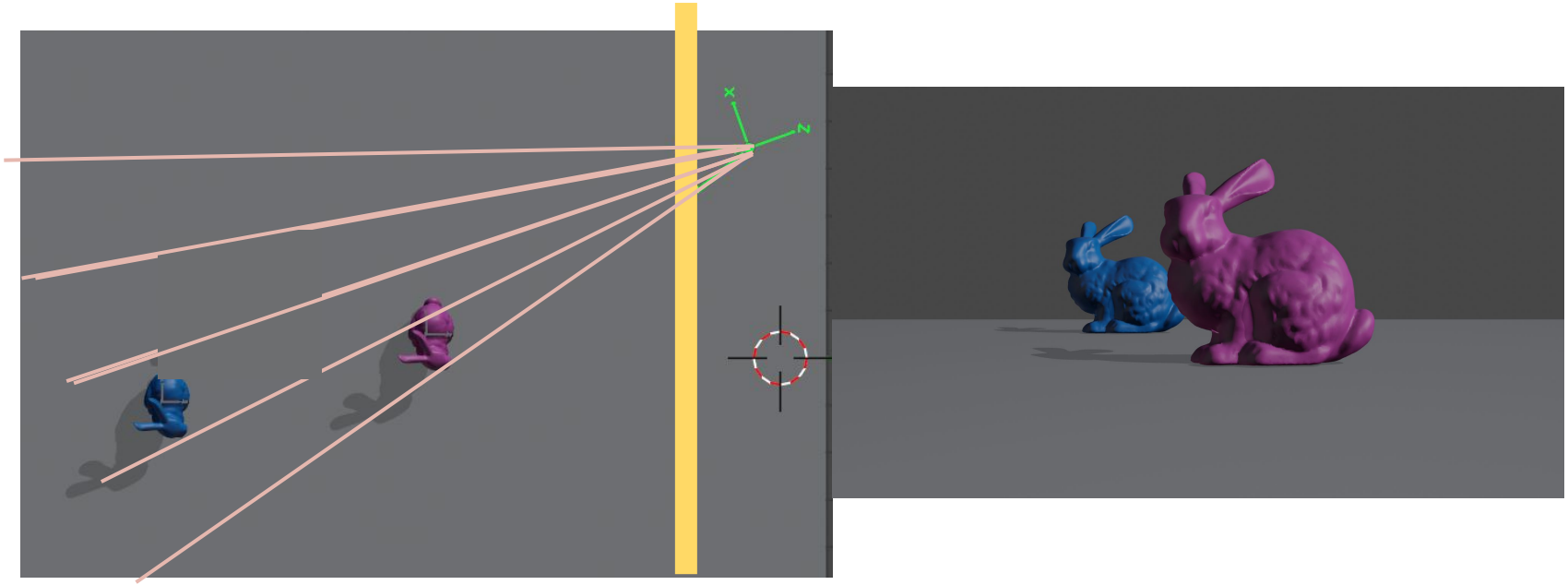
Perspective vs Orthographic Projection

- Versus perspective projection…



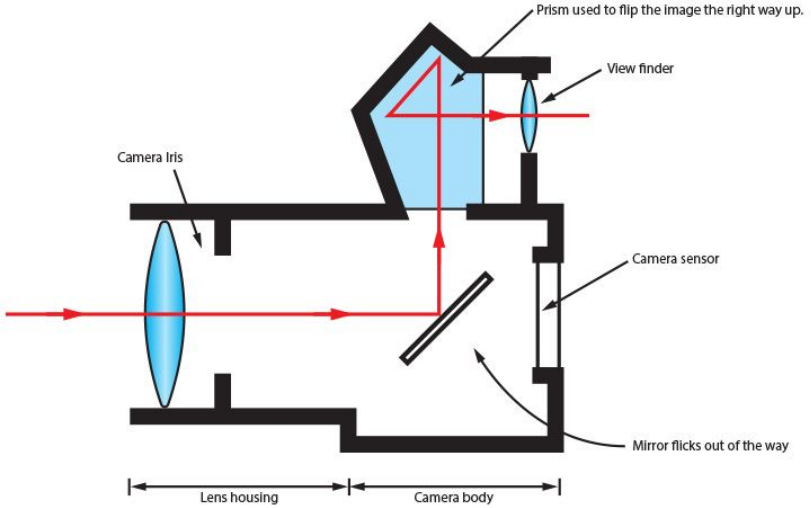
Perspective vs Orthographic Projection

- Versus perspective projection…



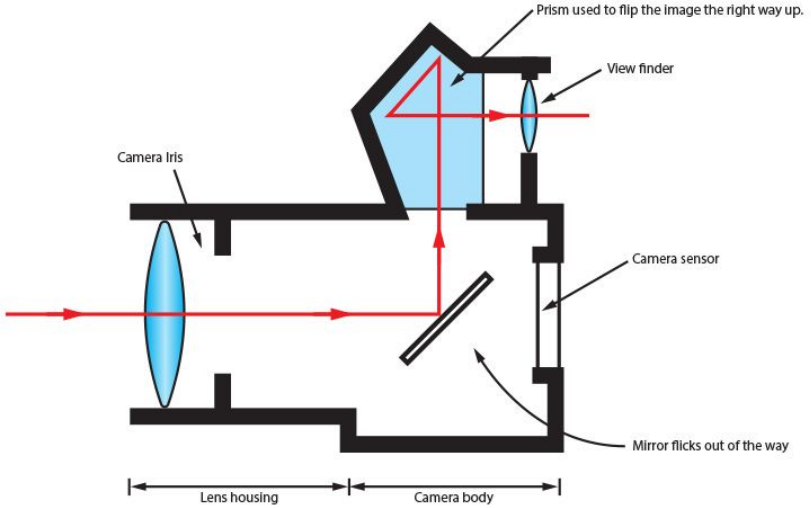
Lens-Based Cameras

- Problem with pinhole camera: long exposure time



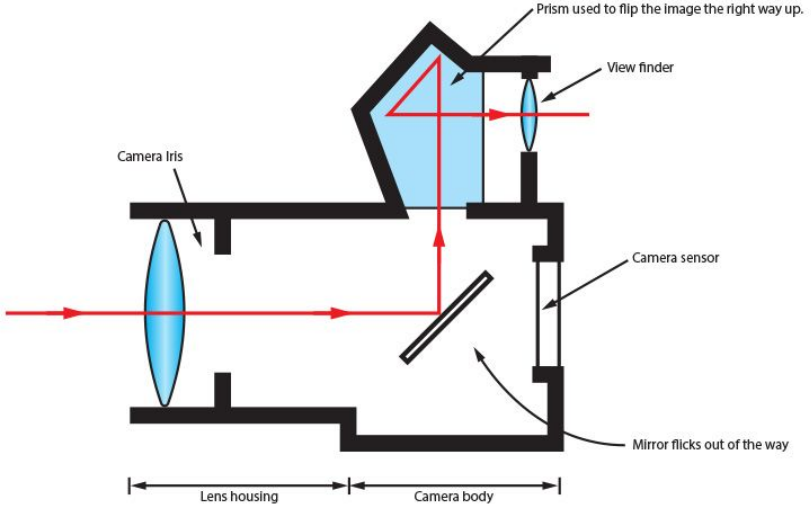
Lens-Based Cameras

- Problem with pinhole camera: long exposure time
- In contrast, lens-based can provide shorter exposure time and objects in focus



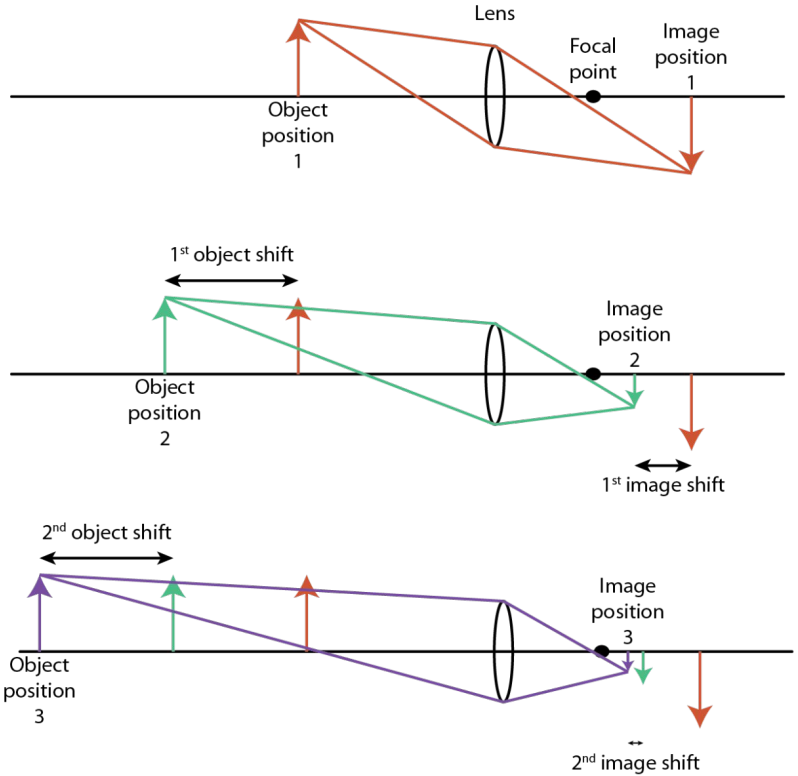
Lens-Based Cameras

- Problem with pinhole camera: long exposure time
- In contrast, lens-based can provide shorter exposure time and objects in focus
- Convex lens
  - Bends incoming light raws so that parallel rays converge at a single point (more light faster)
- Aperture / depth of field trade off



Focus Distance

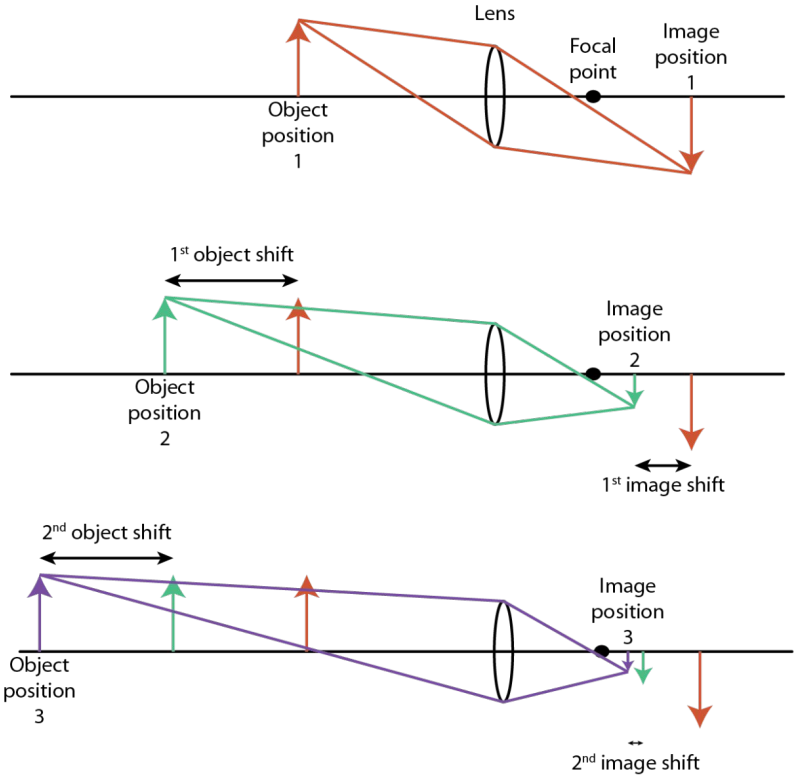
- How far away the object has to be in order to be in focus
- Determined by **focal length** and **distance of the lens to the sensor**
- Adjust focus distance by moving the lens.



https://physicssoup.wordpress.com/2015/04/26/how-do-object-distance-and-focal-length-affect-depth-of-field/

Focal Length

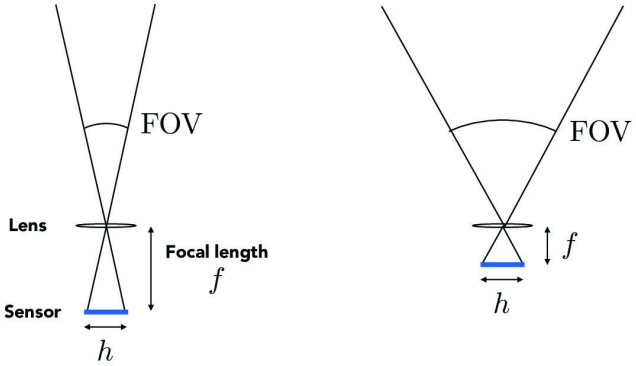
- The distance from the lens where parallel light beams converge
- Depends on the design of the lens
- Weaker lens have longer focal length
- **Field of view, focus distance, depth of field**
- Note: there are fixed lenses and zoom lenses



https://physicssoup.wordpress.com/2015/04/26/how-do-object-distance-and-focal-length-affect-depth-of-field/

Field of View

- Field of view depends on sensor size and focal length
- Wide angle lens have a wider field of view
  - Corresponds to a shorter focal length



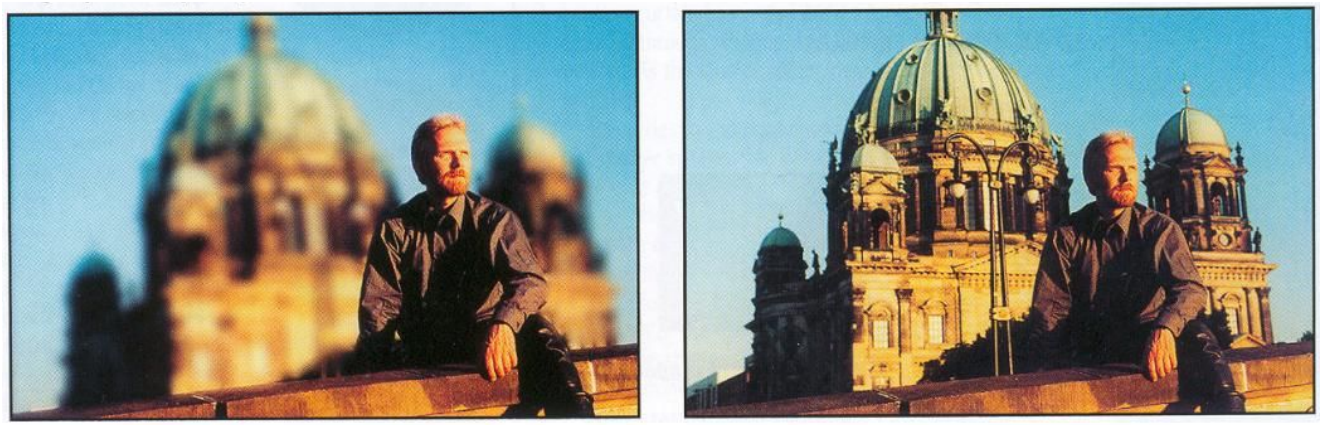
For a fixed sensor size, decreasing the focal length increases the field of view.

$$\text{FOV} = 2 \arctan\left(\frac{h}{2f}\right)$$

Berkeley CS 184

Depth of Field

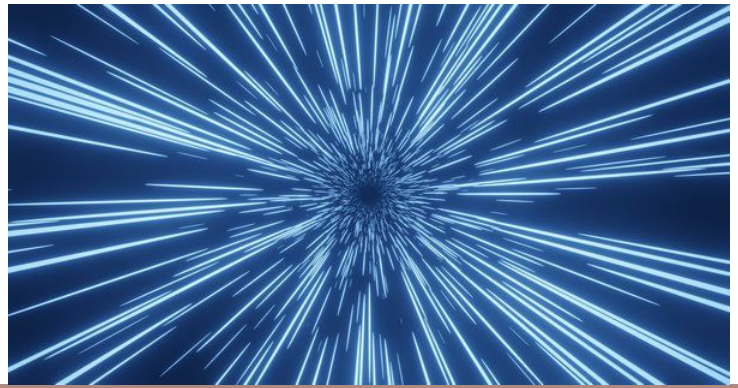
- Blurring of objects not in focus
- Depends on the focus distance and aperture size
- Larger aperture gives you **less** depth of field



From Photography, London et al.

Exposure Time

- Instead of 20 mins (pinhole camera), we can work with fractions of a second
- We can play with the exposure time on this level by controlling how long the mirror in the camera stays up (in DSLR and Film cameras)
  - Or by how long the sensor record data
- For graphics: motion blur!



Imperfections

- Oftentimes, we want to make things imperfect!
  - E.g. motion blur, depth of field, lens flare
  - Where we intentionally make things out of focus/blurry
- Overly sharp details can make things look fake

Lecture Outline

- In real life
  - The human eye (pupil, cornea, lens)
  - RGB color space
  - The pinhole camera
- In the rasterizer
  - Coordinate frames
  - Perspective & orthographic projection
  - Normalized coordinates & the z-buffer algorithm
- Lens-based cameras
- Beyond rasterization

Limitations of Rasterization

- Hacks
  - Ex: depth of field. If things are far away enough, we can “fake” it with blurred images instead of geometry (e.g. the sky!)

Limitations of Rasterization

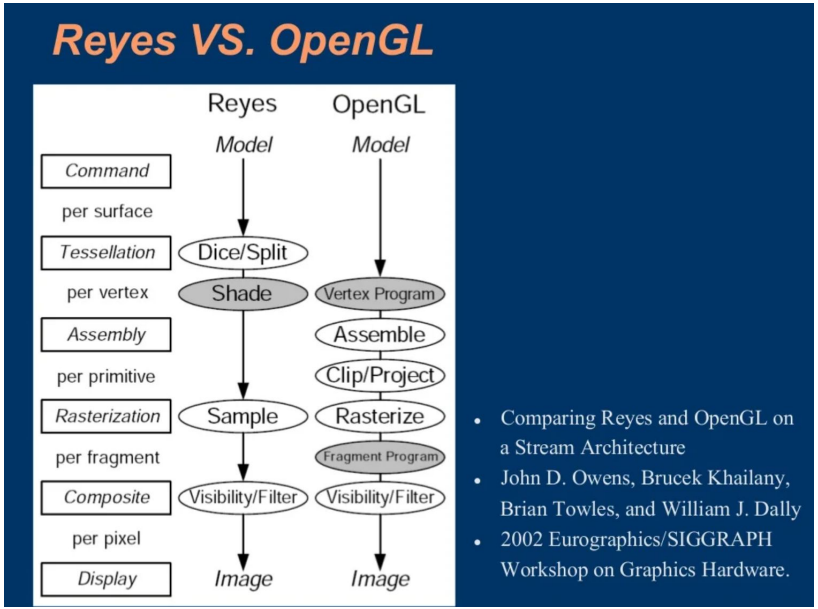
- Hacks
  - Ex: depth of field. If things are far away enough, we can “fake” it with blurred images instead of geometry (e.g. the sky!)
- Can “hack” motion blur, depth of field, etc. with various techniques, but it’s slower to compute
  - Ex: blurring based on z-buffer values, doing many renders, then smoothing out

Limitations of Rasterization

- Hacks
  - Ex: depth of field. If things are far away enough, we can “fake” it with blurred images instead of geometry (e.g. the sky!)
- Can “hack” motion blur, depth of field, etc. with various techniques, but it’s slower to compute
  - Ex: blurring based on z-buffer values, doing many renders, then smoothing out
- If compute time and compute resources are not issues, we generally prefer more physically based image generation techniques over rasterization

Rasterization Still Used!

- REYES (Star Trek II, Toy Story…)
- Different type of rasterization pipeline
- But still rasterization!
- (Works on quads, not triangles)



Beyond Rasterization…

- **Rasterization**:
  - for every triangle in the environment:
    compute color **c** based on the lighting and material models.
  - for every projected point **(x,y,z)** and corresponding color **c**:
    set film position (x,y) to **c** (subject to depth test on z)

Beyond Rasterization…

- **Rasterization**:
  - for every triangle in the environment:
    compute color **c** based on the lighting and material models.
  - for every projected point **(x,y,z)** and corresponding color **c**:
    set film position (x,y) to **c** (subject to depth test on z)

- **Raytracing**:
  - for every point **(x,y)** on the camera film:
    backwards trace out possible trajectories of light rays hitting **(x,y)**, and check what the color of that light ray might be based on the objects and lights in the environment

Additional Resources + Exploring!

- Pixar's REYES rendering architecture
- List of color spaces and their uses
- Muybridge's *The Horse in Motion*
  - Great Stanford history here!
- Camera Obscura (Pinhole Camera)
- Lens flares: from accidental to artistic

- Fundamentals of Computer Graphics, Steve Marschner and Peter Shirley
  - Preview, Book