Lecture 17:
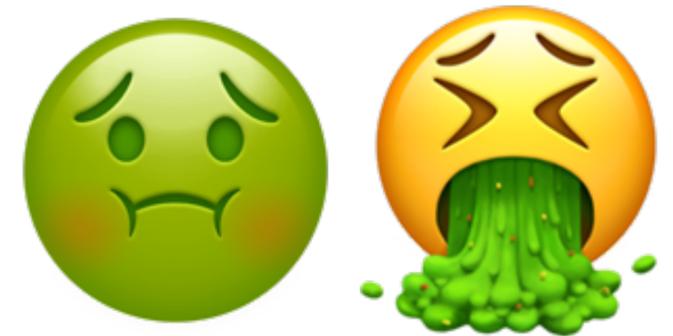
Rendering for Virtual Reality

How not to:

Computer Graphics: Rendering, Geometry, and Image Manipulation
Stanford CS248A, Winter 2026

VR headsets

Meta Quest 3

Apple Vision Pro

Sony Morpheus

Google Daydream

Google Cardboard

Valve Index

VR gaming

Eleven: Table Tennis (Fun Labs)

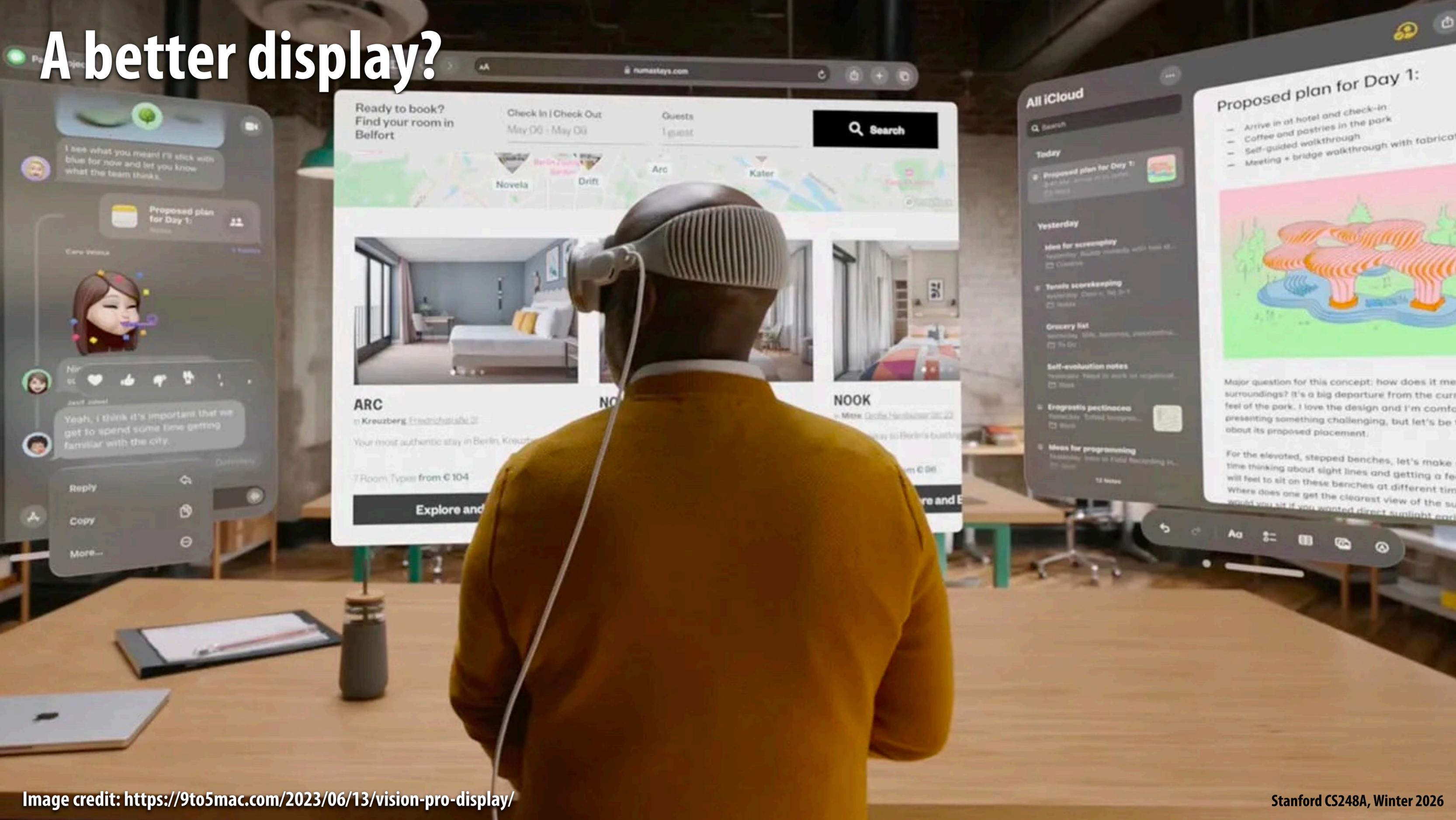VR teleconference / video chat
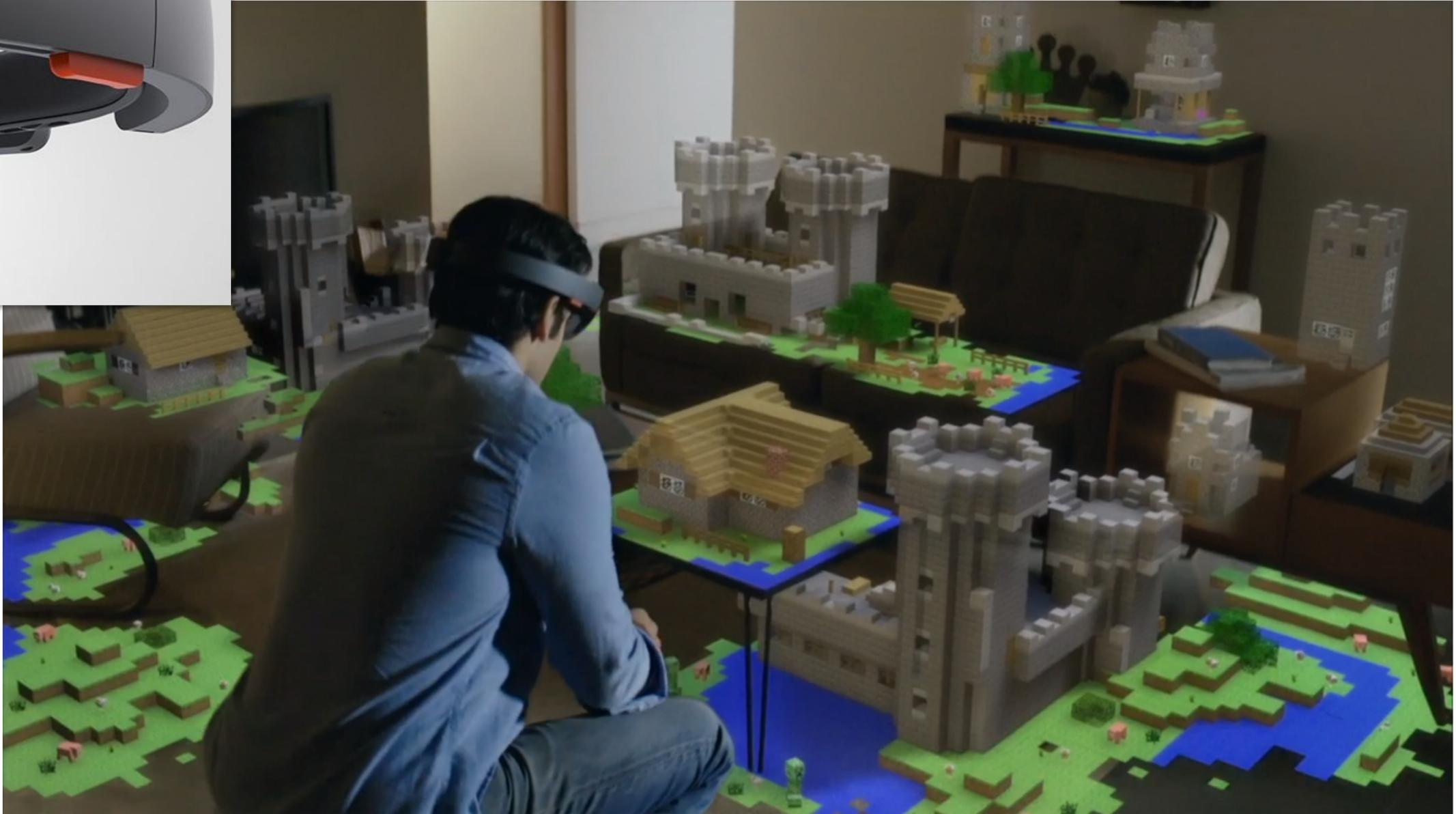
Image credit: Spatial

VR video

Vaunt VR
Paul McCartney concert)

A better display?

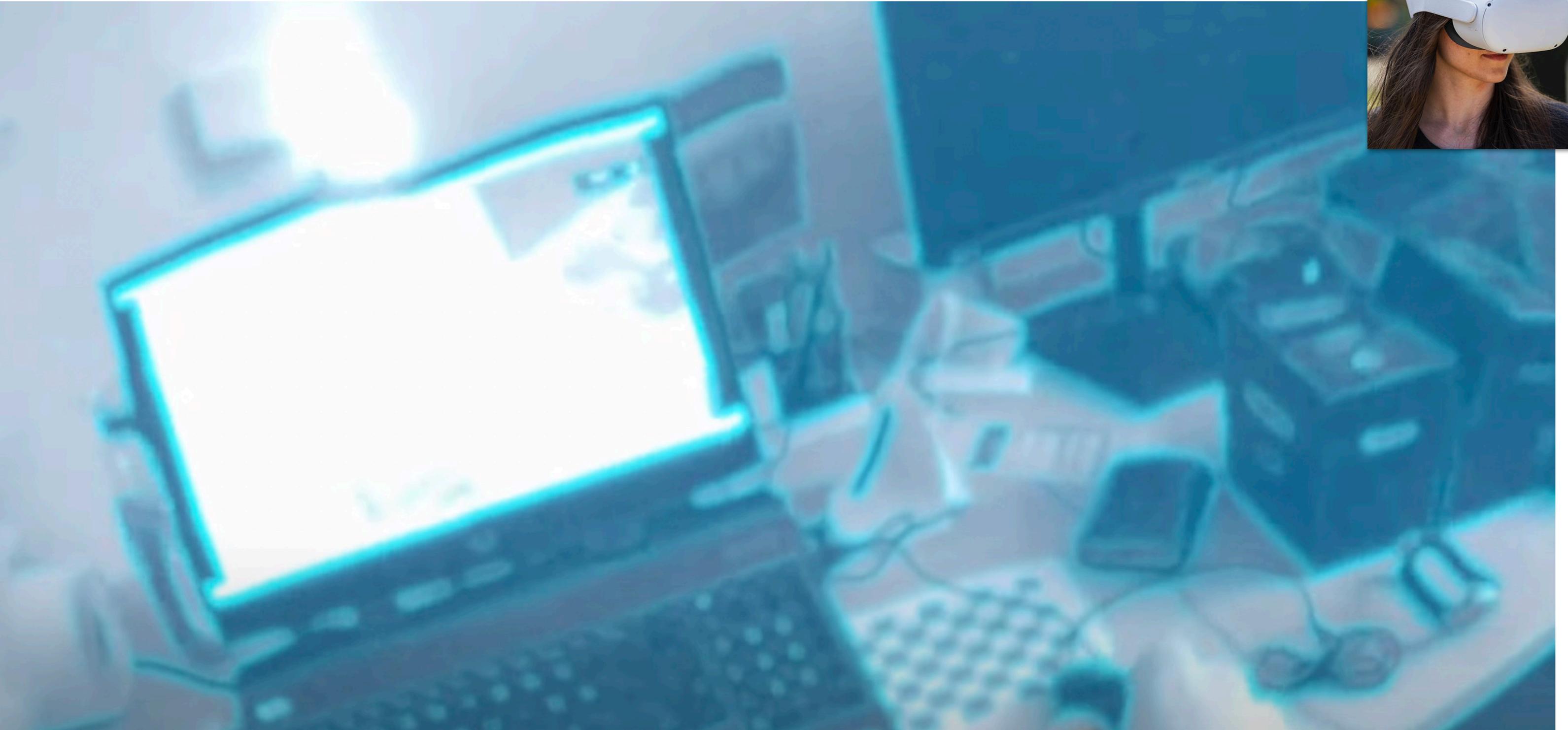# AR headset: Microsoft Hololens

Snap Spectacles

AR on a mobile device

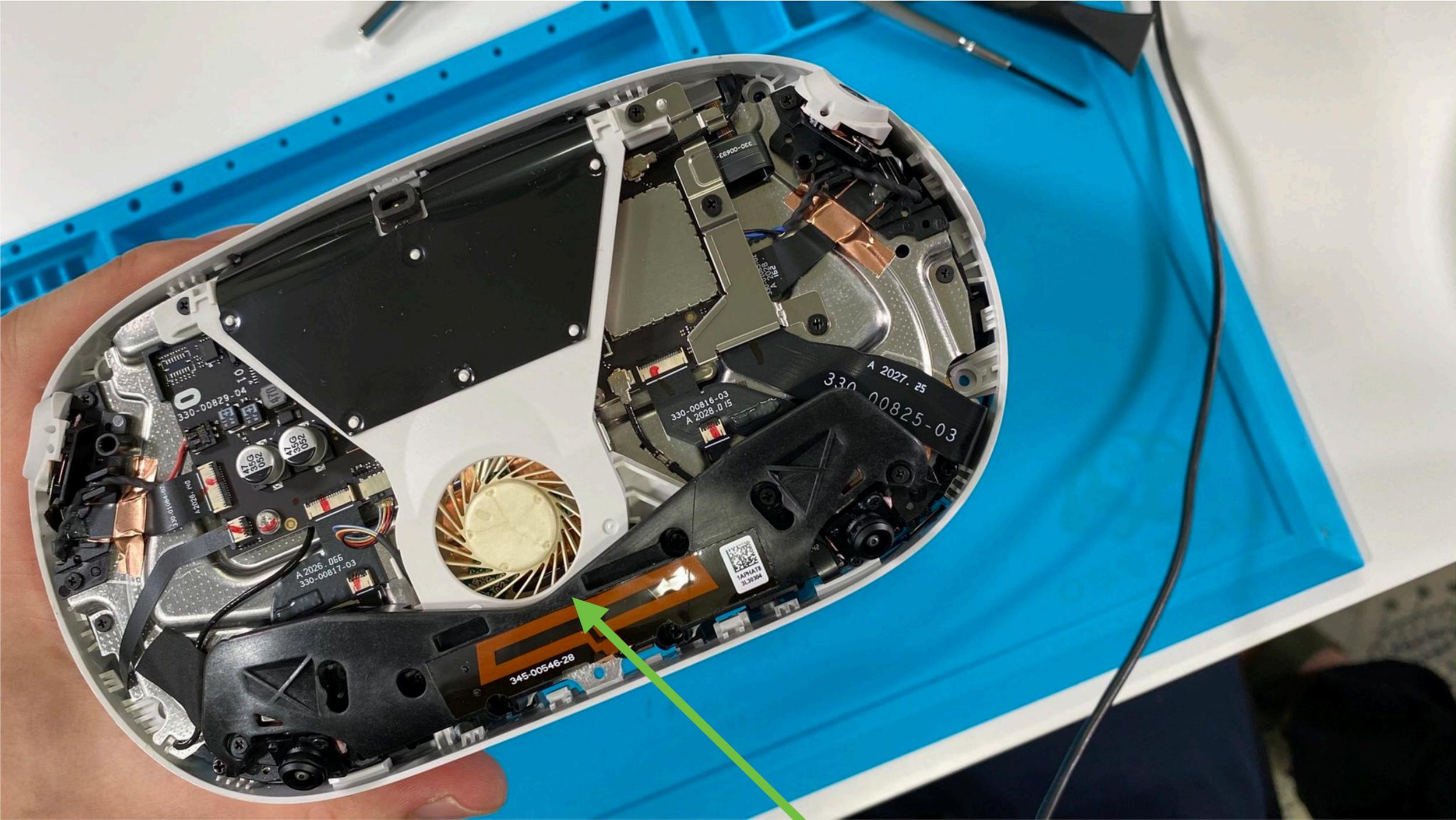# AR "passthrough" on VR headset

Oculus Quest 2 headset (2020)

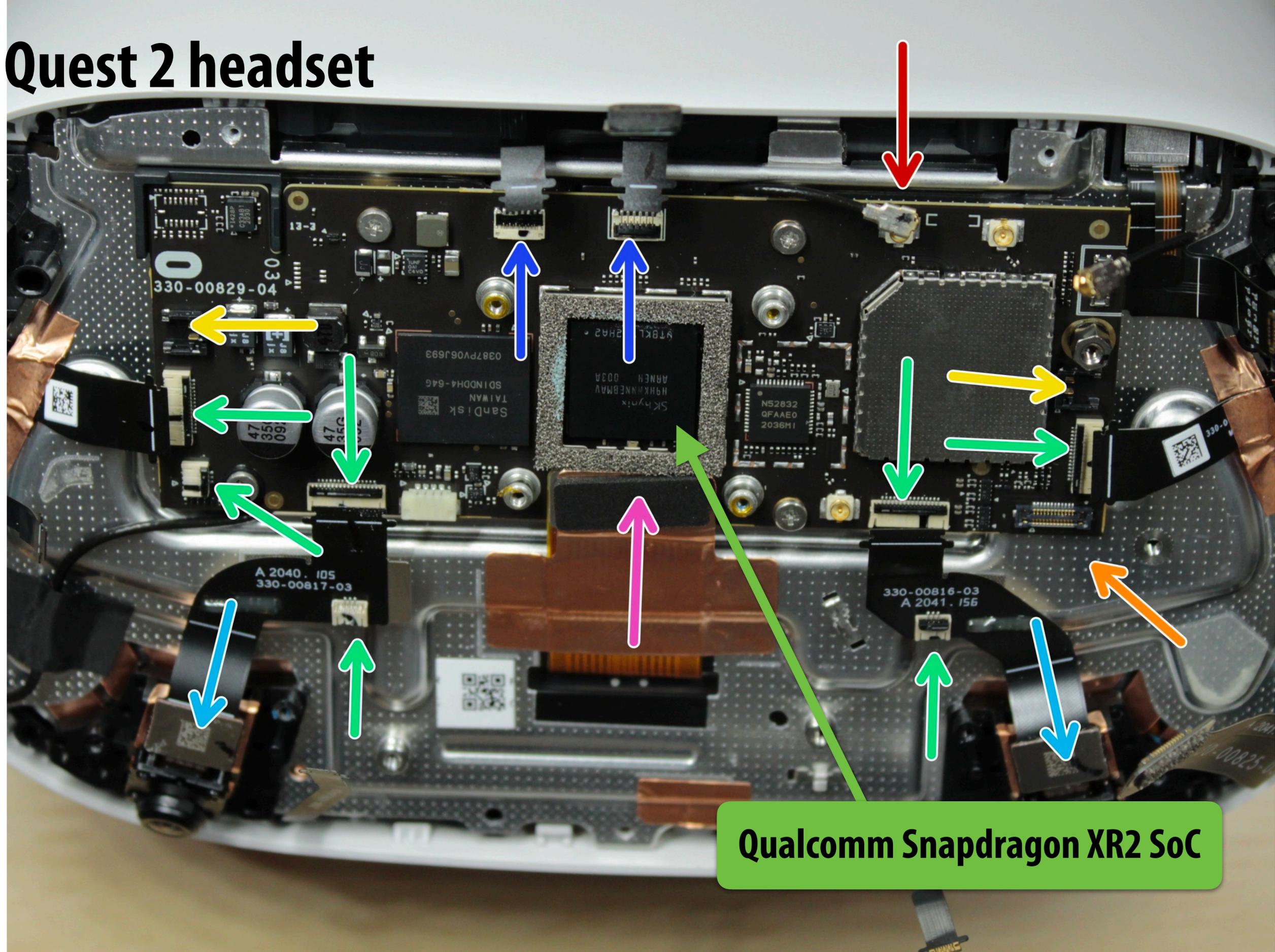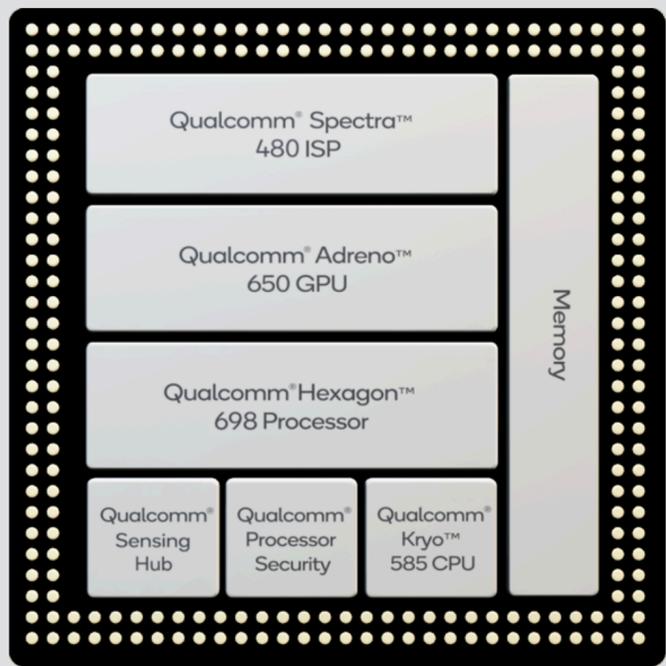# Oculus Quest 2 headset (lens side view)

Oculus Quest 2 headset

Fan

Oculus Quest 2 headset

Qualcomm Snapdragon XR2 SoC

Oculus Quest 2 headset (Snapdragon SoC)

Qualcomm snapdragon

XR2 5G Platform

Qualcomm® Spectra™ 480 ISP	
Qualcomm® Adreno™ 650 GPU	Memory
Qualcomm® Hexagon™ 698 Processor	
Qualcomm® Sensing Hub — Qualcomm® Processor Security — Qualcomm® Kryo™ 585 CPU	

This diagram is from Snapdragon 865
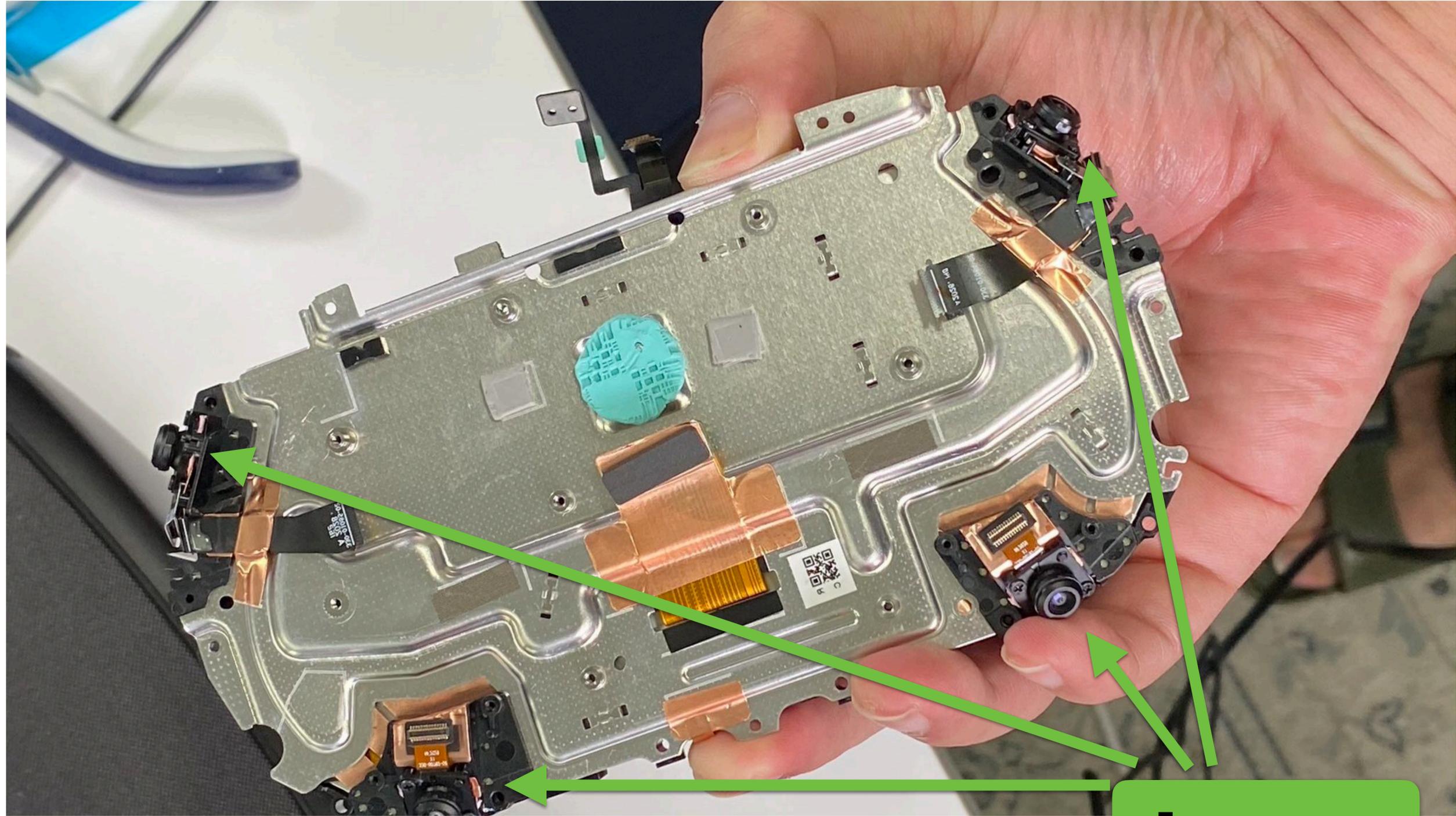
4 high-performance cores
4 low-performance (low energy) cores
Image processor + DSP
Multi-core graphics processor (GPU) — up to 3000 x 3000 display @ 90 Hz
Additional processor for sensors (IMU etc)
Can process inputs from up to seven simultaneous video camera streams
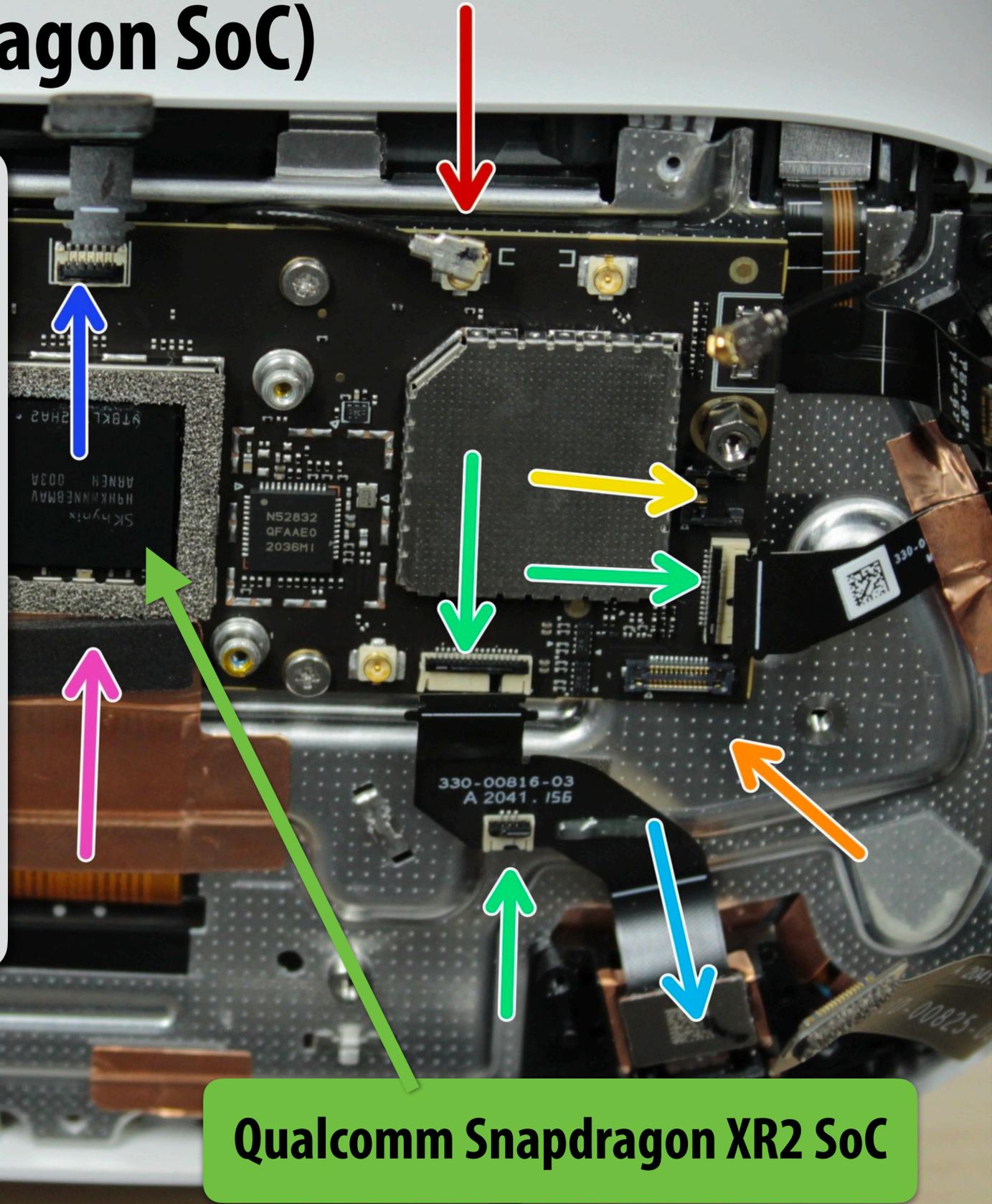
Qualcomm Snapdragon XR2 SoC

Oculus Quest 2 headset

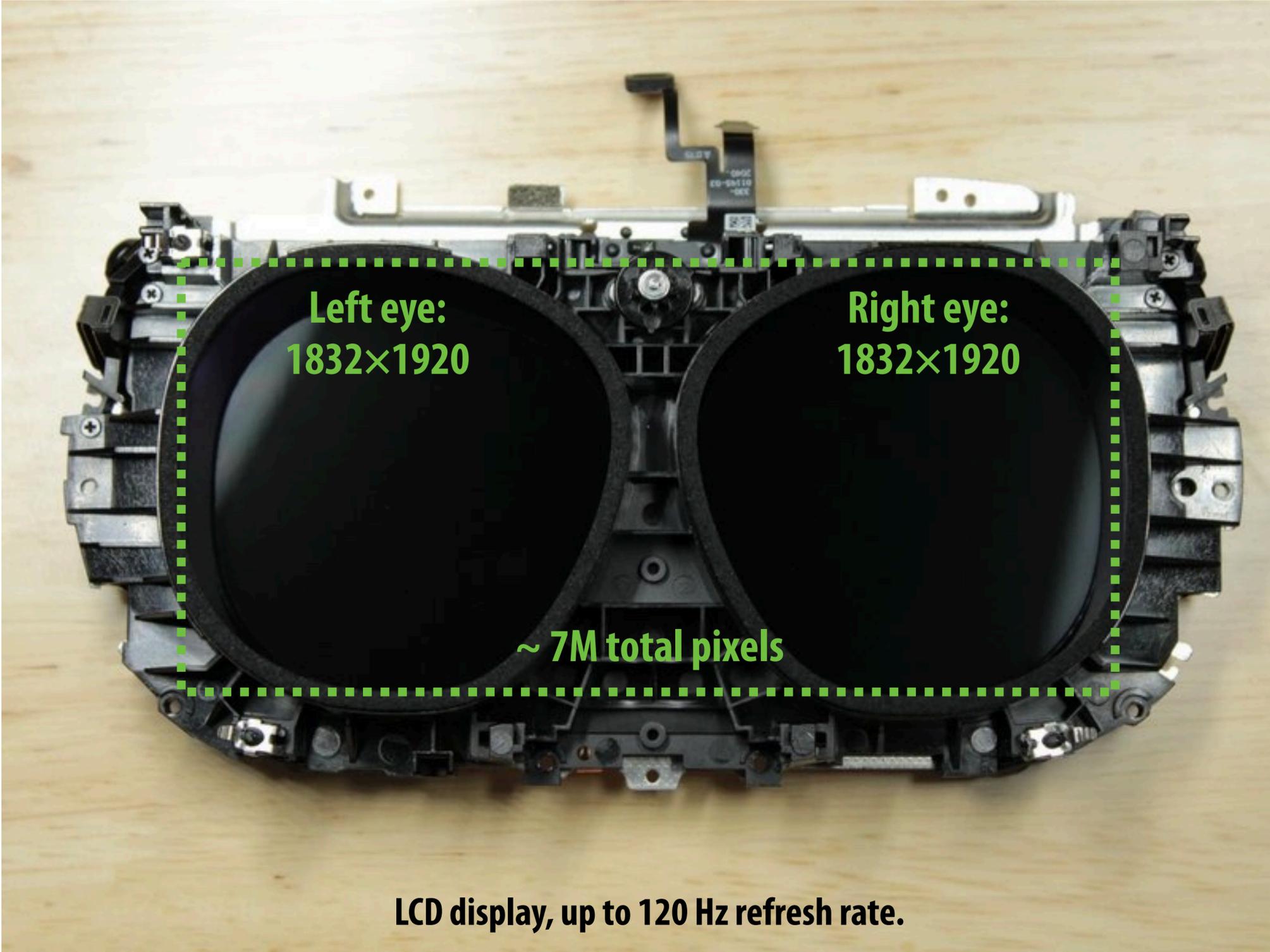**Four cameras**

Oculus Quest 2 headset (lens assembly)

Oculus Quest 2 display + lens assembly

Left eye: 1832×1920

Right eye: 1832×1920

~ 7M total pixels

LCD display, up to 120 Hz refresh rate.

Review from last class:
What does a lens do?

What does a lens do?

A lens refracts light.

Camera with lens: every pixel accumulates all rays of light that pass through lens aperture and refract toward that pixel

In-focus: all rays of light from a point in the scene arrive at a point on sensor plane

Pixel P1

Pixel P2

Sensor plane: (X,Y)

Lens

Scene object 1

Scene object 2

Scene focal plane

Camera's field of view
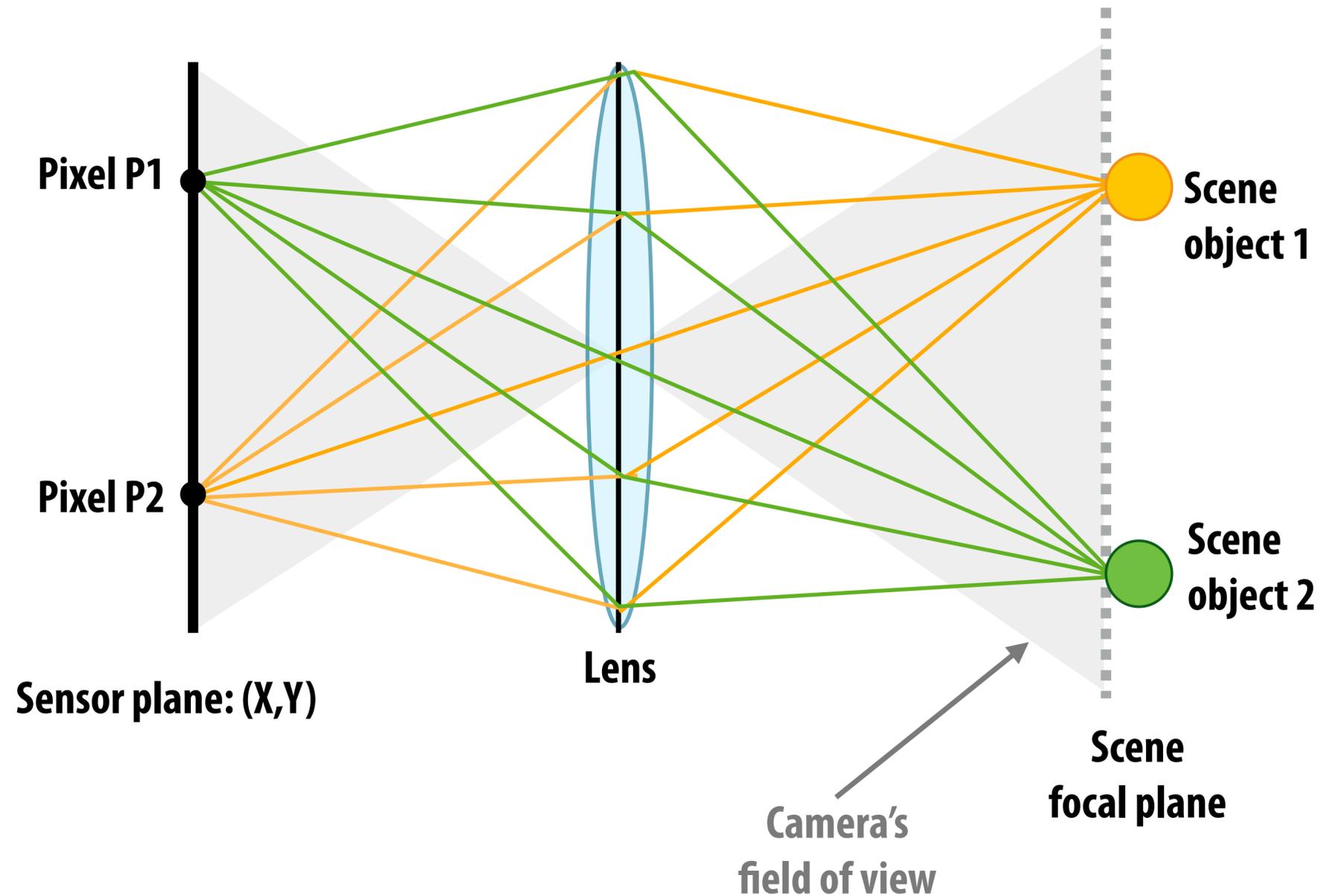
The eye
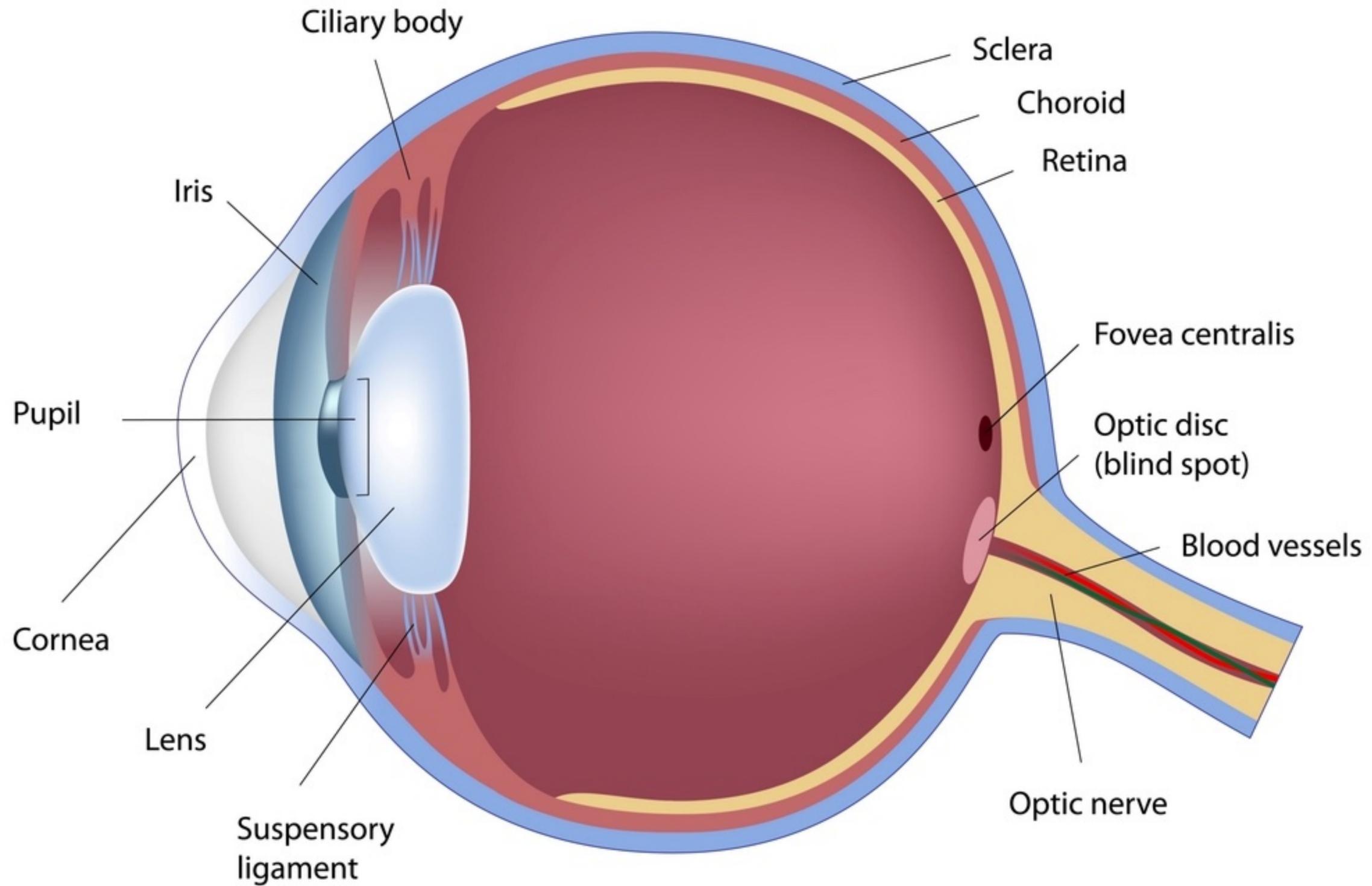
Consider projection of scene object on retina

In diagram: the red dot in scene projects onto point X on back of eye (retina)

Retina

Pupil

Lens

X

Eye focused at a distance

Plane of focus

Red and yellow cups = in focus

Teal cup = out of focus
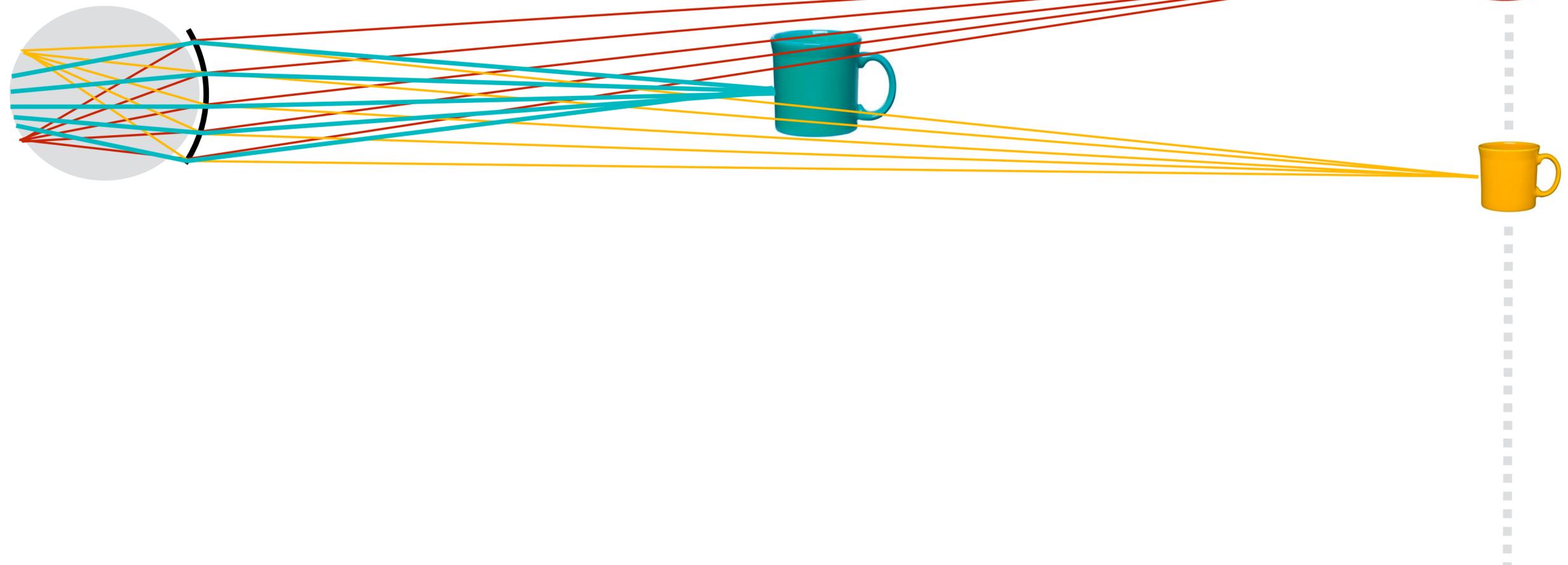
Eye focused at a distance

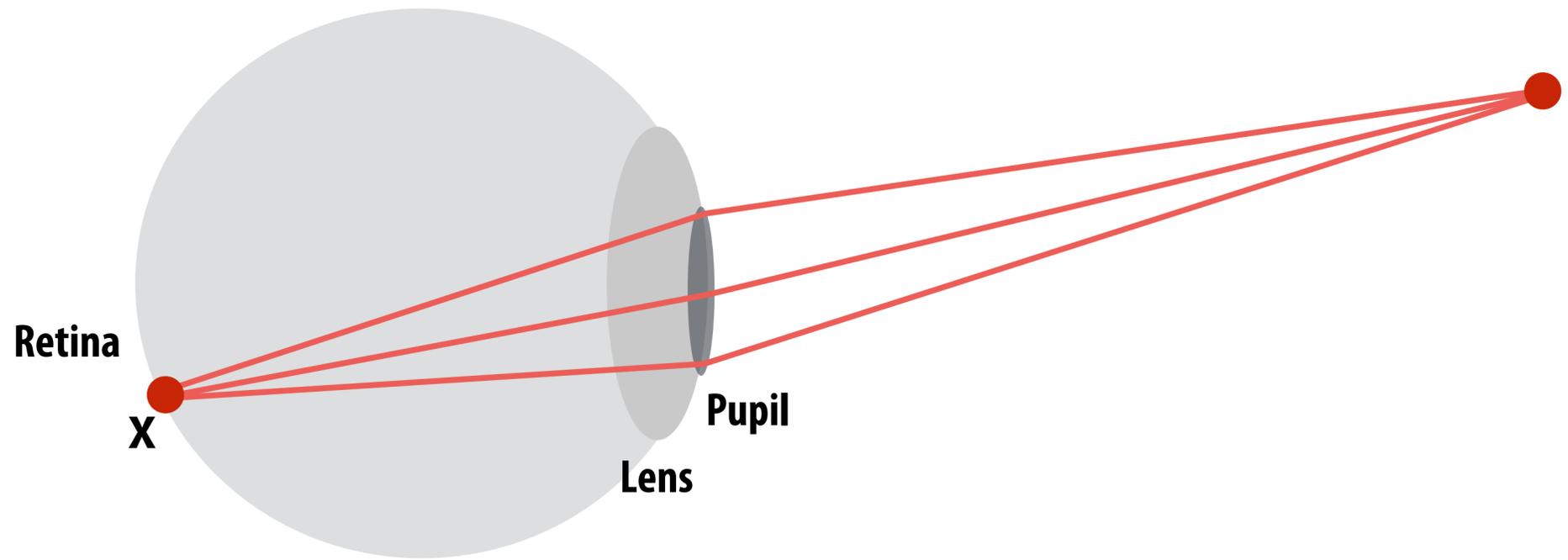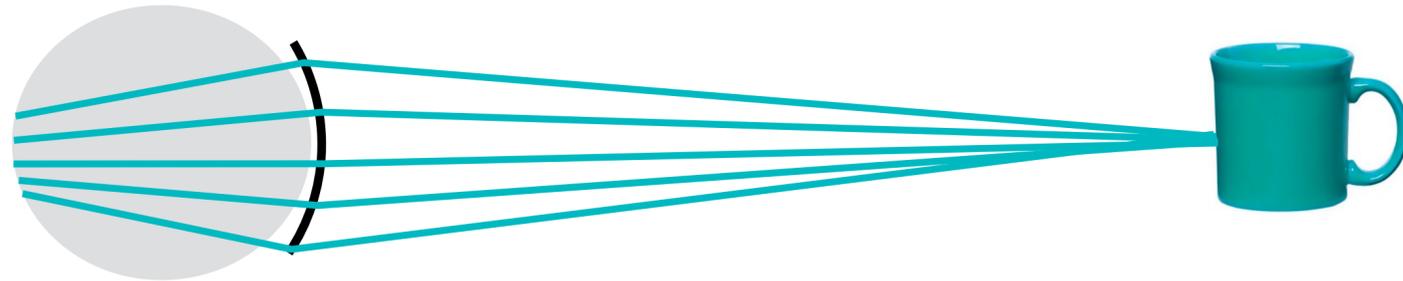## Teal cup = out of focus

Eye focused up close

Teal cup = in focus

Plane of focus
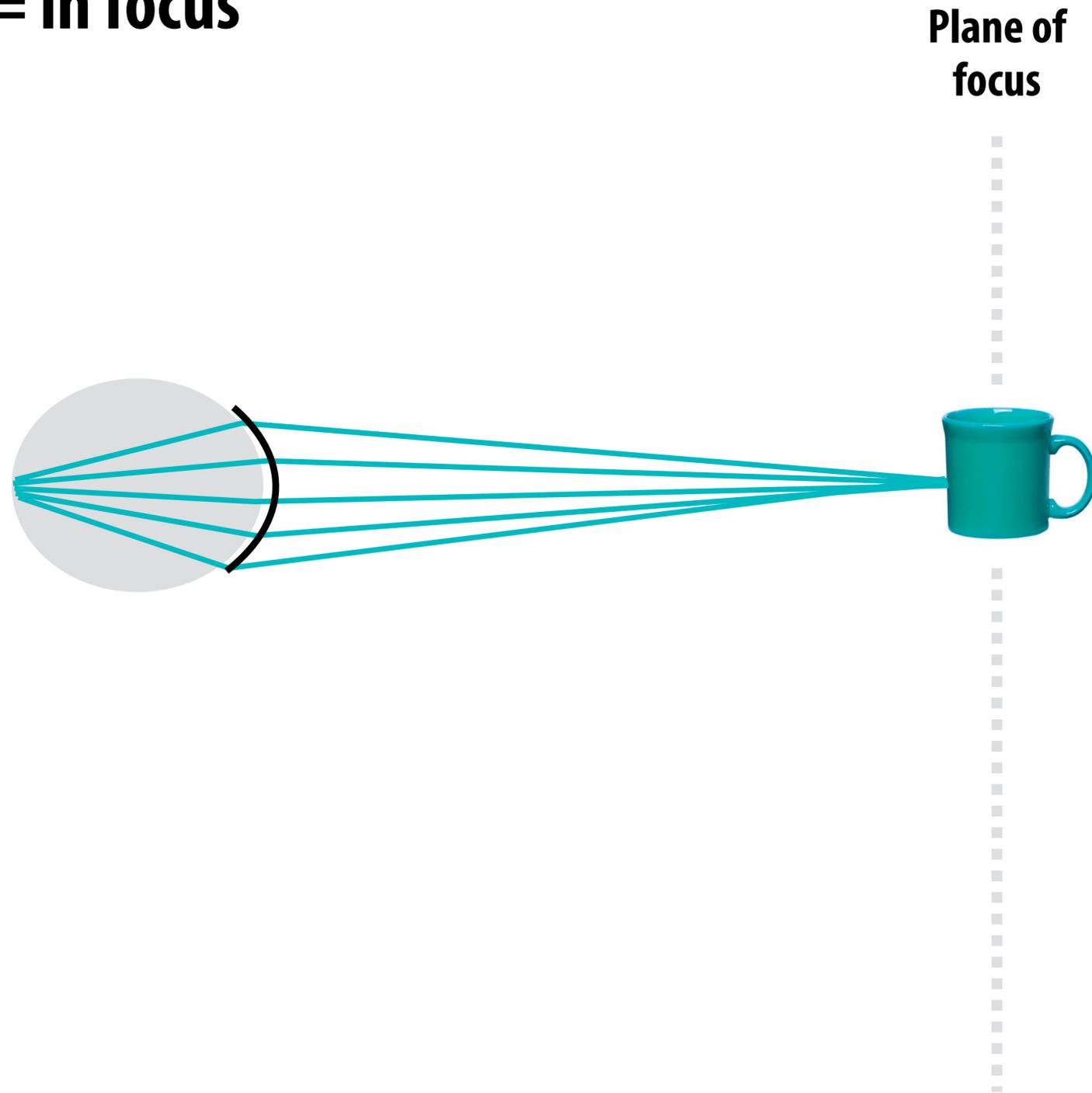
Accommodation and vergence

Accommodation: changing the focal length of the eye to focus at different distances

Eye accommodated
at far distance

Eye accommodated
at near distance
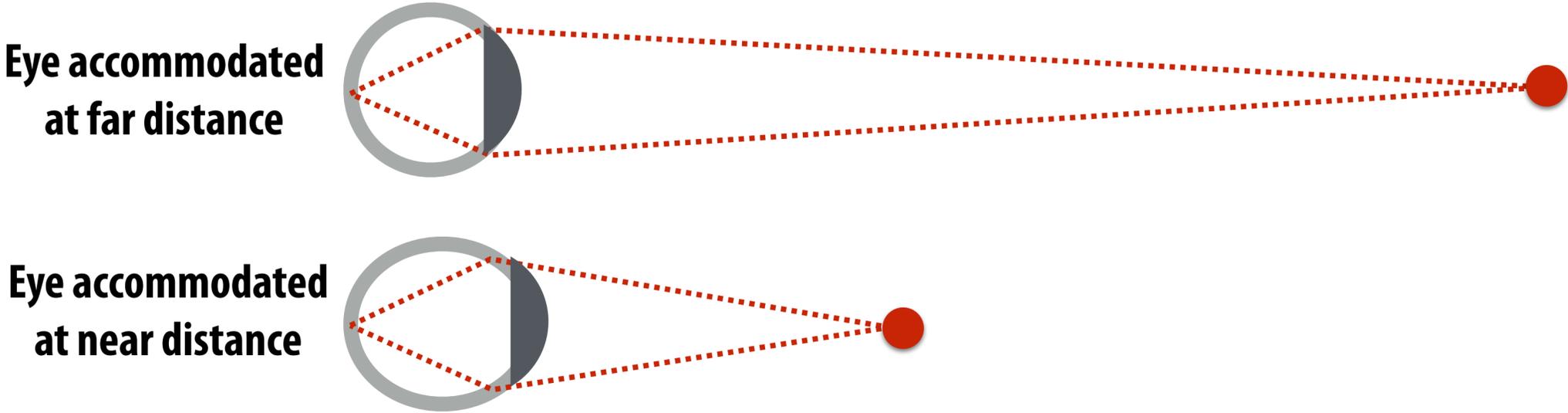
Vergence: rotation of eye to ensure projection of object falls in center of retina
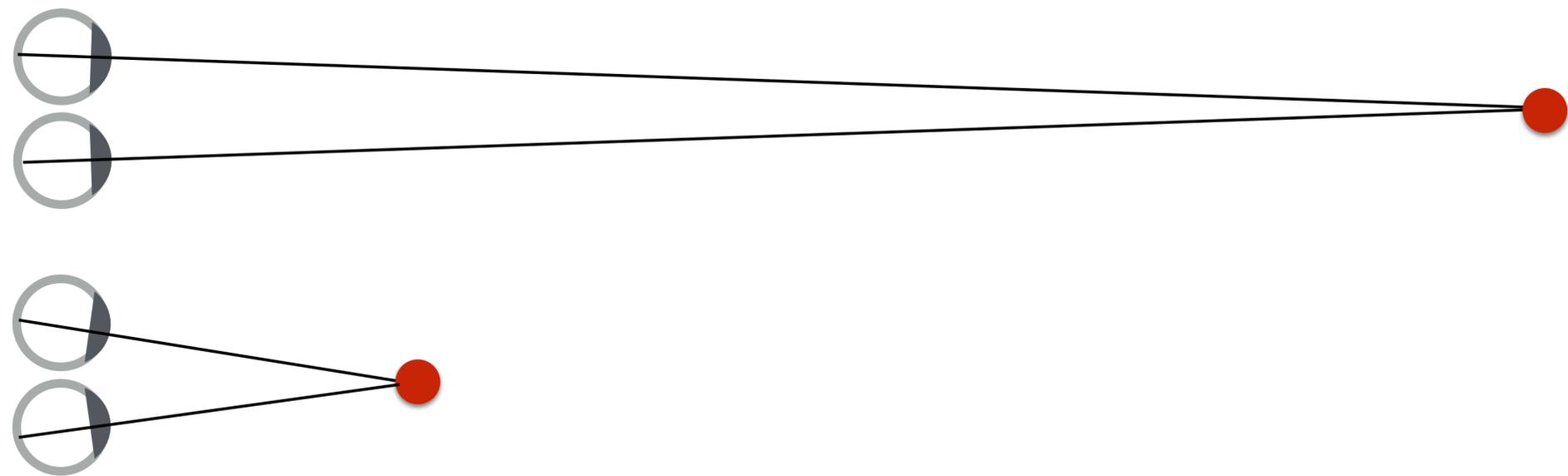
Screen up close
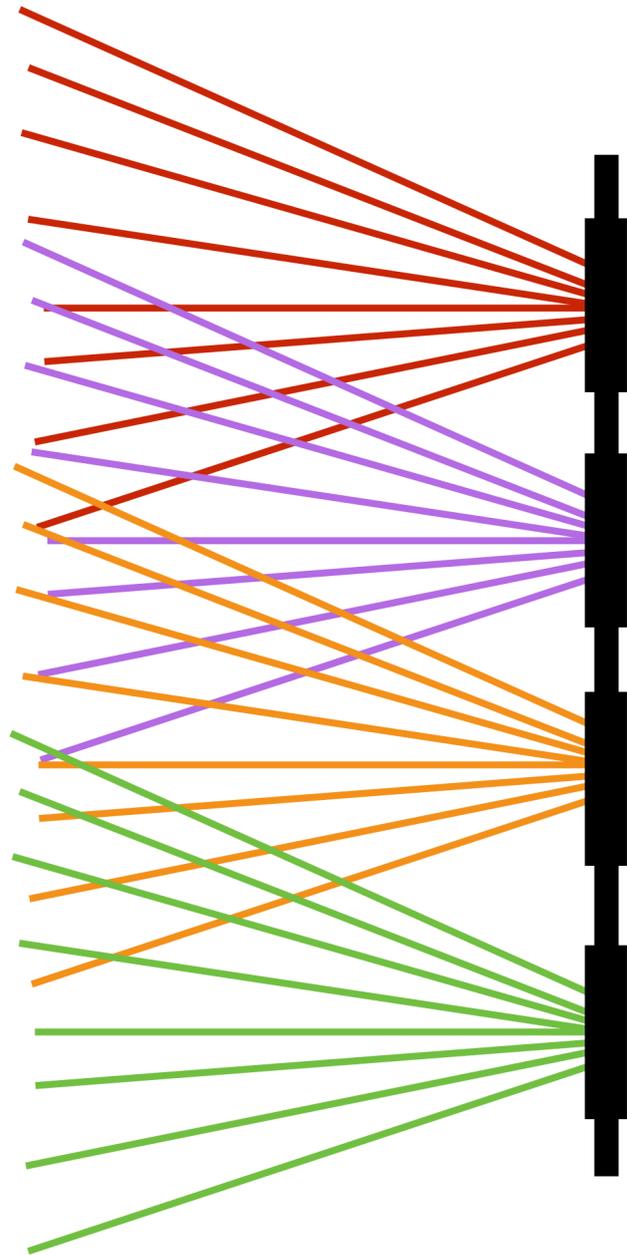
Display pixel on my laptop

(close up photo)

Each pixel emits light
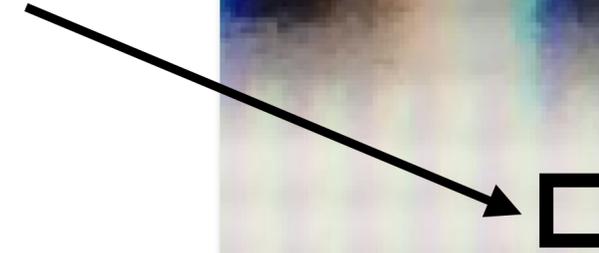
Role of lenses in VR headset

1. **Create wide field of view**

2. **Place focal plane at several meters away from eye (close to infinity)**

Note: parallel lines reaching eye converge to a single point on display (eye accommodates to plane near infinity)

eye

Color LCD display
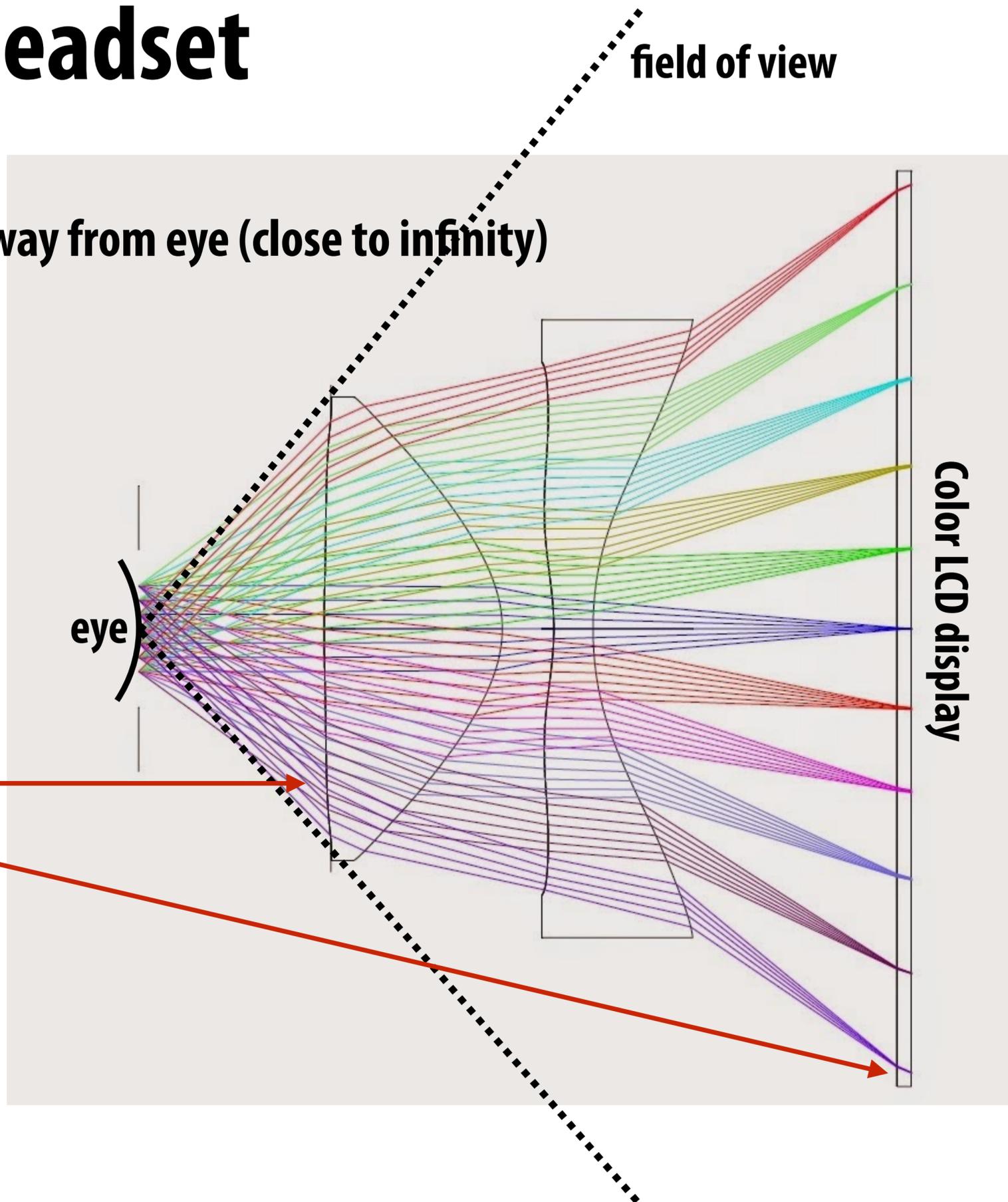
Lens diagram from Open Source VR Project (OSVR) (Not the lens system from the Oculus Quest 2) http://www.osvr.org/

Accommodation/vergence conflict

■ **Given design of current VR displays, consider what happens when objects are up-close to eye in a virtual 3D scene**

- Eyes must remain accommodated to near infinity (otherwise image on screen won't be in focus)

- But eyes must converge in attempt to fuse stereoscopic images of object up close

- Brain receives conflicting depth clues… (discomfort, fatigue, nausea)
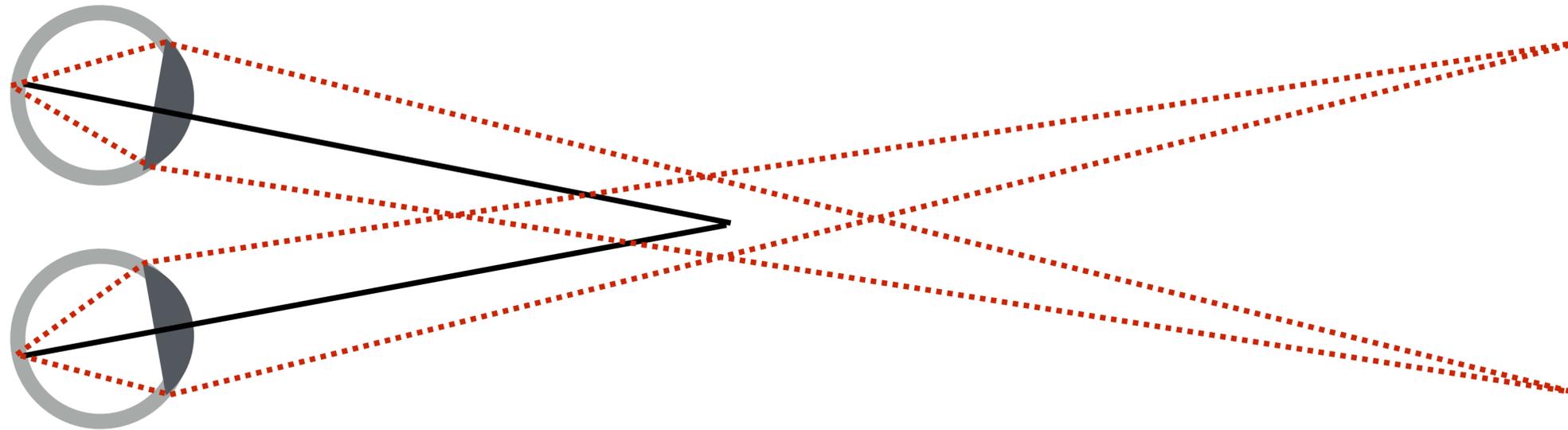
This problem stems from nature of display design. If you could just make a display that emits the same rays of light that would be produced by a virtual scene, then you could avoid the accommodation - vergence conflict…

A better (future) display
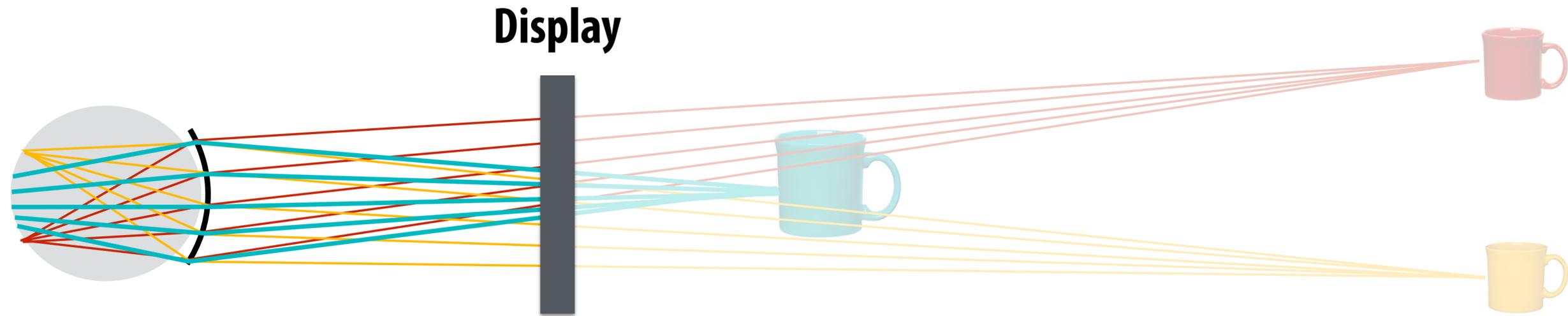
Display

Note how this hypothetical display creates the same rays of light as what would be seen in the real environment.

The *same position* on the display emits light with *different colors* in *different directions*. (Current LCD displays emit same color in all directions from each pixel)

The display generates the same "light field" in front of the eye as present in the real scene.

Need for high resolution

Resolution: Oculus Quest 2 display

Left eye:
1832×1920

Right eye:
1832×1920

~ 7M total pixels

LCD display, up to 120 Hz refresh rate.

Need for high resolution

(Note: this does not account for eye's ability to rotate in socket)

160°

~5°

Future "retina" VR display:
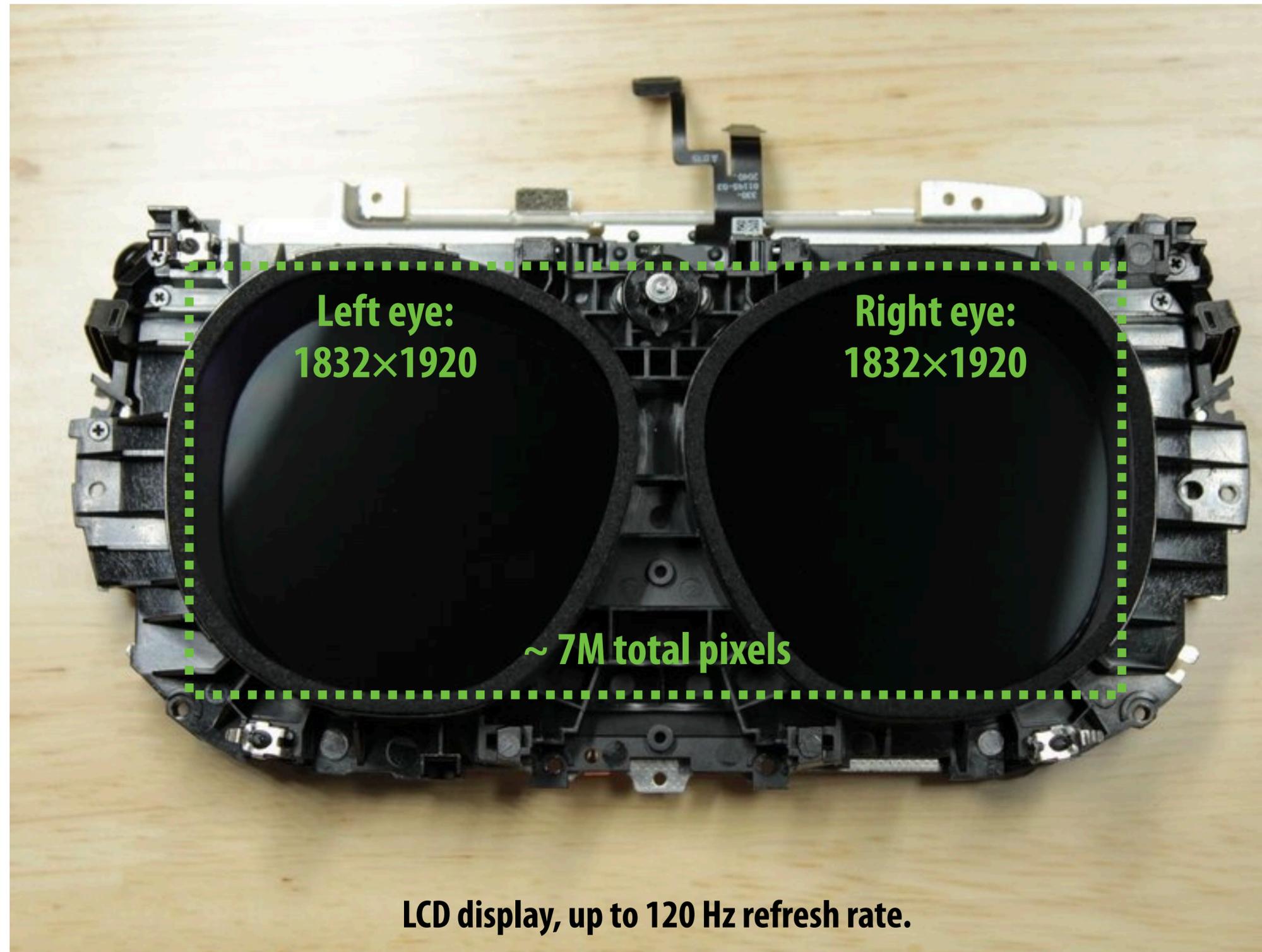
~ 8K x 8K display per eye
= 128 MPixel
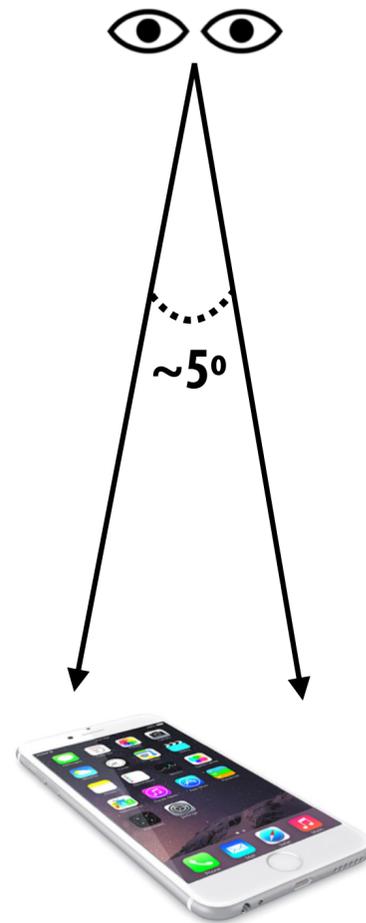
iPhone 15 Pro: 4.7 in "retina" display:
1179 x 2556 (3 MPixel)
442 ppi → 88 ppd

Apple Vision Pro

- ~11.4M visible pixels per panel

- 3,386 PPI

- 100 degree field of view

- ~33 PPD

16K TVs!!!

15,360 x 8,640 resolution…

~ 132 Mpixel 🤯🤯🤯🤯

Forget 8K, Sony's New 63-Foot 16K Crystal LED TV Is Now Available—for a Few Million

The ballpark figure is $5 million.

By RACHEL CORMACK ✚

f 🐦 in ✉ +

Courtesy of Sony

When a new gogglebox drops, it's always the same drill: The screen gets bigger, the resolution gets better and the design gets bolder. Indeed, it's difficult for a brand to stand out. Unless you're Sony and the new TV your peddling is the size of a New York City public bus and also happens to boasts an unheard-of 16K screen.

Earlier this year when Sony unveiled the colossal 63-foot TV—the biggest 16K screen of its kind —it had commercial cinemas in its sights. But, hey, why should theaters have all the fun?

Consider bandwidth cost of getting pixels to display

- **132 Mpixel @ 120 Hz x 24 bpp = 354 Gbits/s**

- **Note: modern display compression technologies (such as Display Stream Compression — DSC 1.2a) provide ~ 3:1 compression**
  - **Reduces need to 118 Gbits/s bandwidth**

- **Now consider *energy cost* of *transmitting pixels* to display at this rate**
  - **Rough estimate: ~ 100 pico-Joules per bit transferred ***
  - **100 Pj/bit x 118 Gbit/s = 11.8 J/s = 11.8 W**
  - **Snapdragon SoC in Oculus Quest 2 designed for TDP of ~ 5W**

* Signaling technologies undergo rapid improvement, feasible to see 1pJ/bit in the next decade

Display stream compression (DSC)

- **Goal: high compression ratio but, but cheap to encode so compression can be performed at high data rate.**

- **Example modes:**

  - **MMAP (Modified median-adaptive prediction)**

$$P0 = CLAMP(a + \tilde{b} - \tilde{c}, MIN(a, \tilde{b}), MAX(a, \tilde{b}))$$

$$P1 = CLAMP(a + \tilde{d} - \tilde{c} + R0, MIN(a, \tilde{b}, \tilde{d}), MAX(a, \tilde{b}, \tilde{d}))$$

$$P2 = CLAMP(a + \tilde{e} - \tilde{c} + R0 + R1,$$
$$MIN(a, \tilde{b}, \tilde{d}, \tilde{e}), MAX(a, \tilde{b}, \tilde{d}, \tilde{e})) \qquad (2)$$

Where $\tilde{b}$, $\tilde{c}$, $\tilde{d}$, and $\tilde{e}$ are the results of the blending operation for the pixels in the previous line, R0 and R1 are the inverse quantized residuals of the first two pixels in the group,

c	b	d	e	f
a	P0	P1	P2	

  - **ICH (indexed history mode): Retain buffer of the last 32 pixels and encoding is index into that buffer.**

- **Encoding performed on luma/chroma representation**

  - **YCgCo-R (see next slide)**

YCgCo

- Luma, chrominance green, chrominance orange
- Conversation from RGB to luma/chroma representation with minimal hardware (only additions and shifts)
- YCgCo-R is a slight modification that supports lossless (bit precise) conversation from and back to RGB

$$
\begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 0 & -\frac{1}{2} \\ -\frac{1}{4} & \frac{1}{2} & -\frac{1}{4} \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}
$$

$$
\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix}
$$

RGB

Y

Cg
("chrominance green")

Co
("chrominance orange")
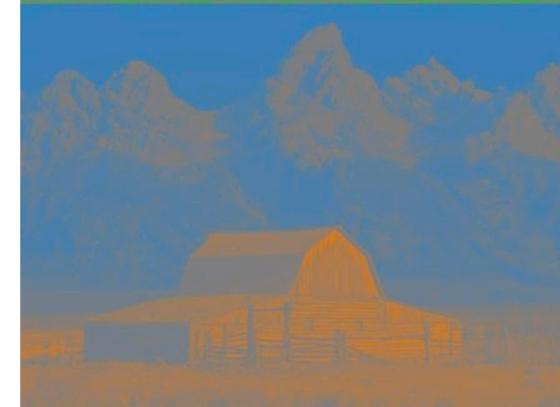
The eye

Density of rod and cone cells in the retina
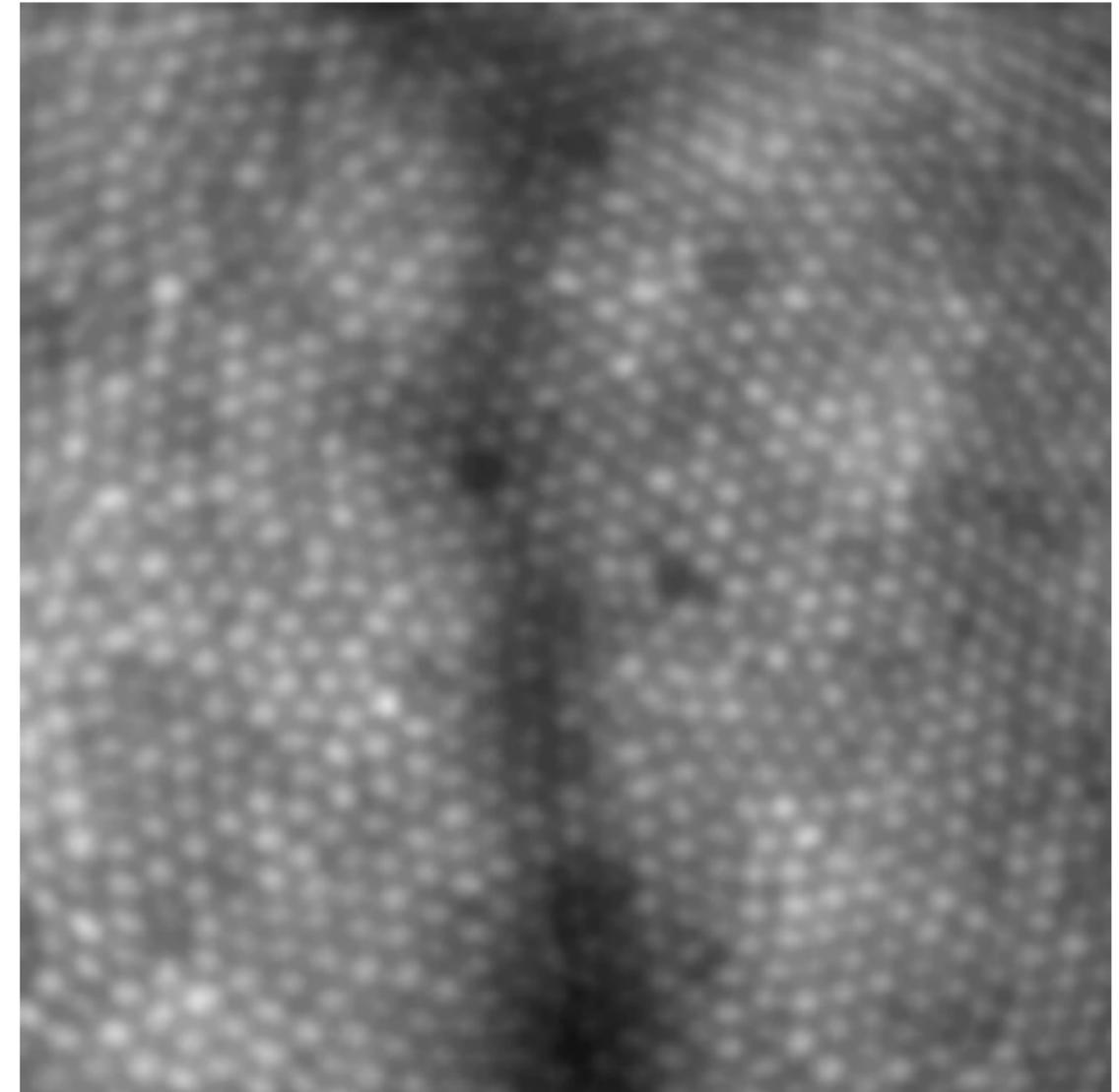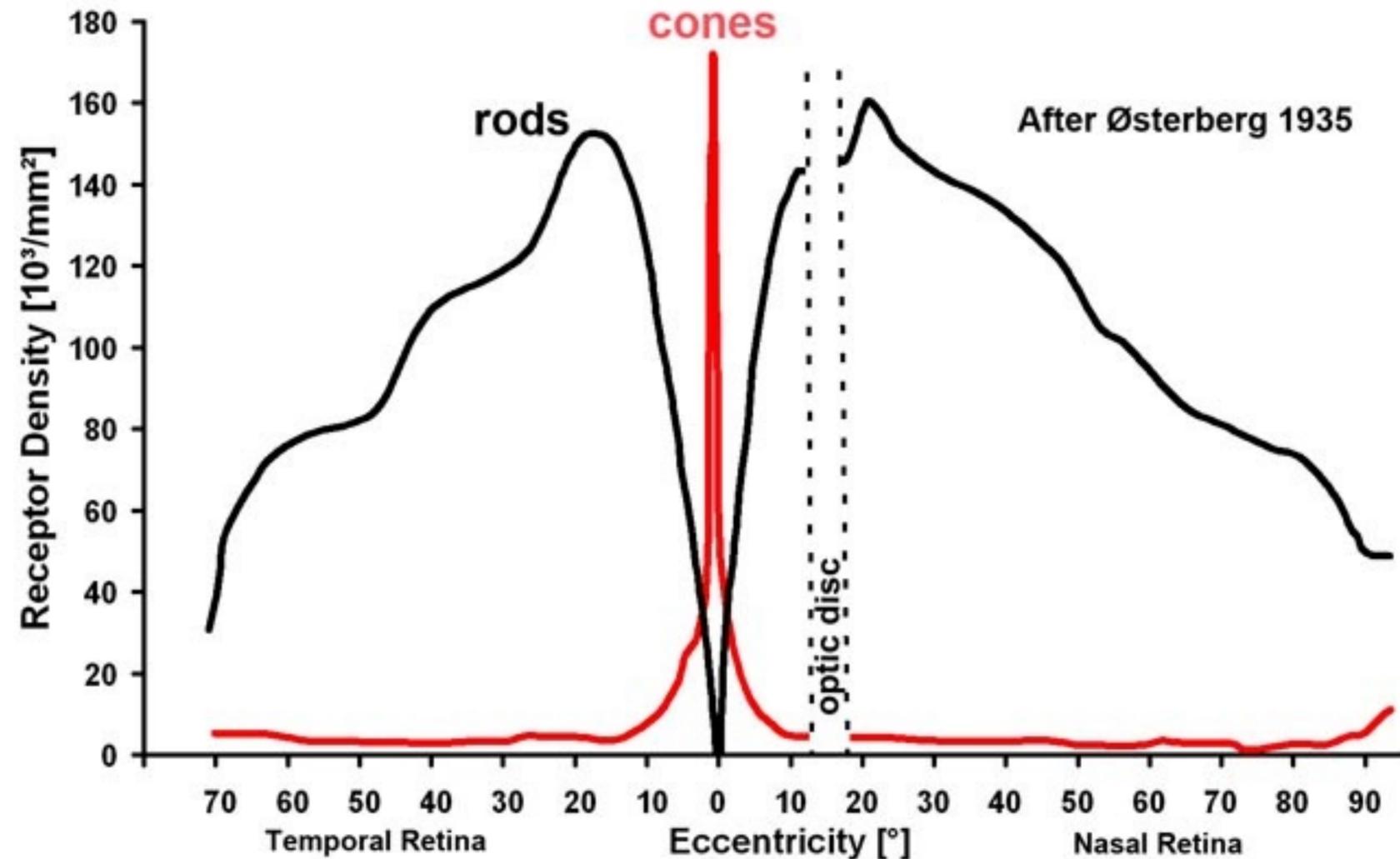
[Roorda 1999]

- Cones are color receptive cells
- Highest density of cones is in fovea
  (best color vision at center of where human is looking)

- Implication: human eye has low spatial resolution away from fovea *(opportunity to reduce computation by computing less in these areas)*

Reducing rendering cost via foveated rendering

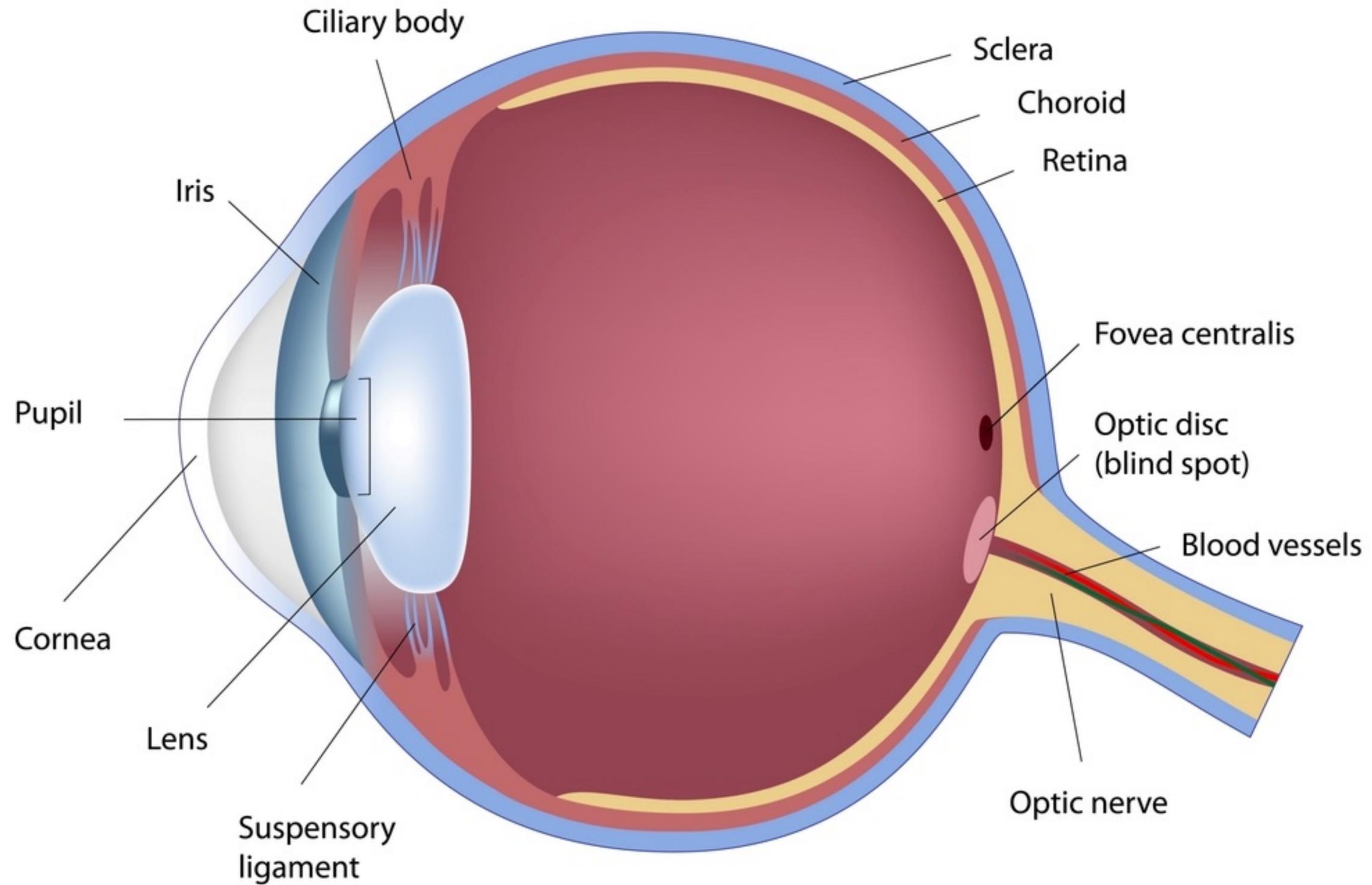**Idea: track user's gaze using an eye tracker, render with increasingly lower resolution farther away from gaze point**

VR headset with eye tracker:
HTC Vive Pro Eye

high-res image

med-res image

low-res image

Three images blended into one for display
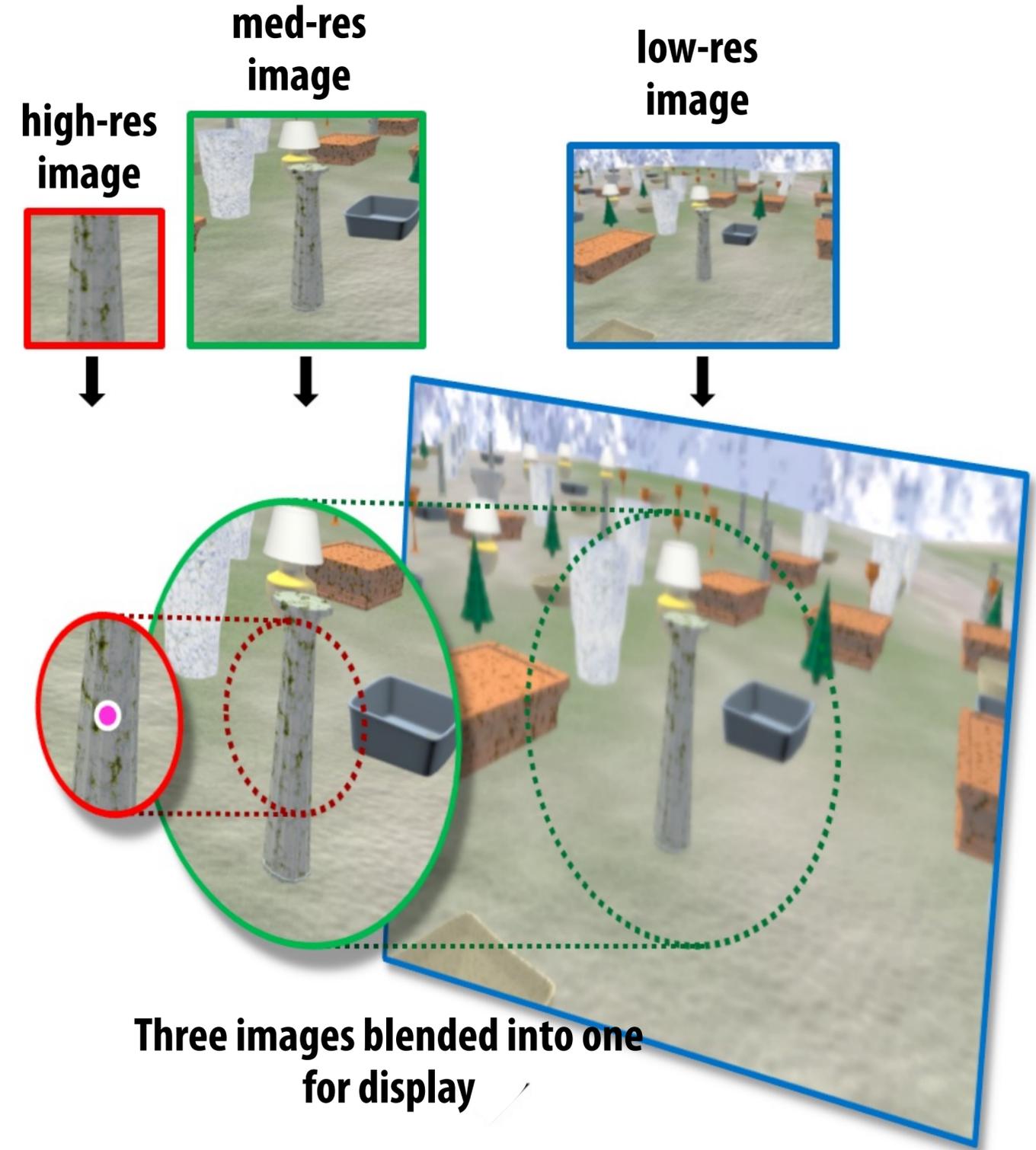
Eye tracking based solutions

- **Given gaze information, there are many rendering cost reducing strategies**
  - **Use low resolution rendering away from point of gaze**
  - **More practical: perform part of the rendering computation at lower frequency (lower-rate shading, reduce texture LOD etc.) ***

- **Fundamental problem: accurate low-latency eye tracking is challenging**
  - **Abnormal eyes, etc.**

* We'll come back to this in a second when we talk about lens matched shading

Accounting for distortion due to design of head-mounted display

Lenses introduce distortion

Lenses introduce distortion

- **Pincushion distortion**

- **Chromatic aberration (different wavelengths of light refract by different amount)**

View of checkerboard through Oculus Rift lens
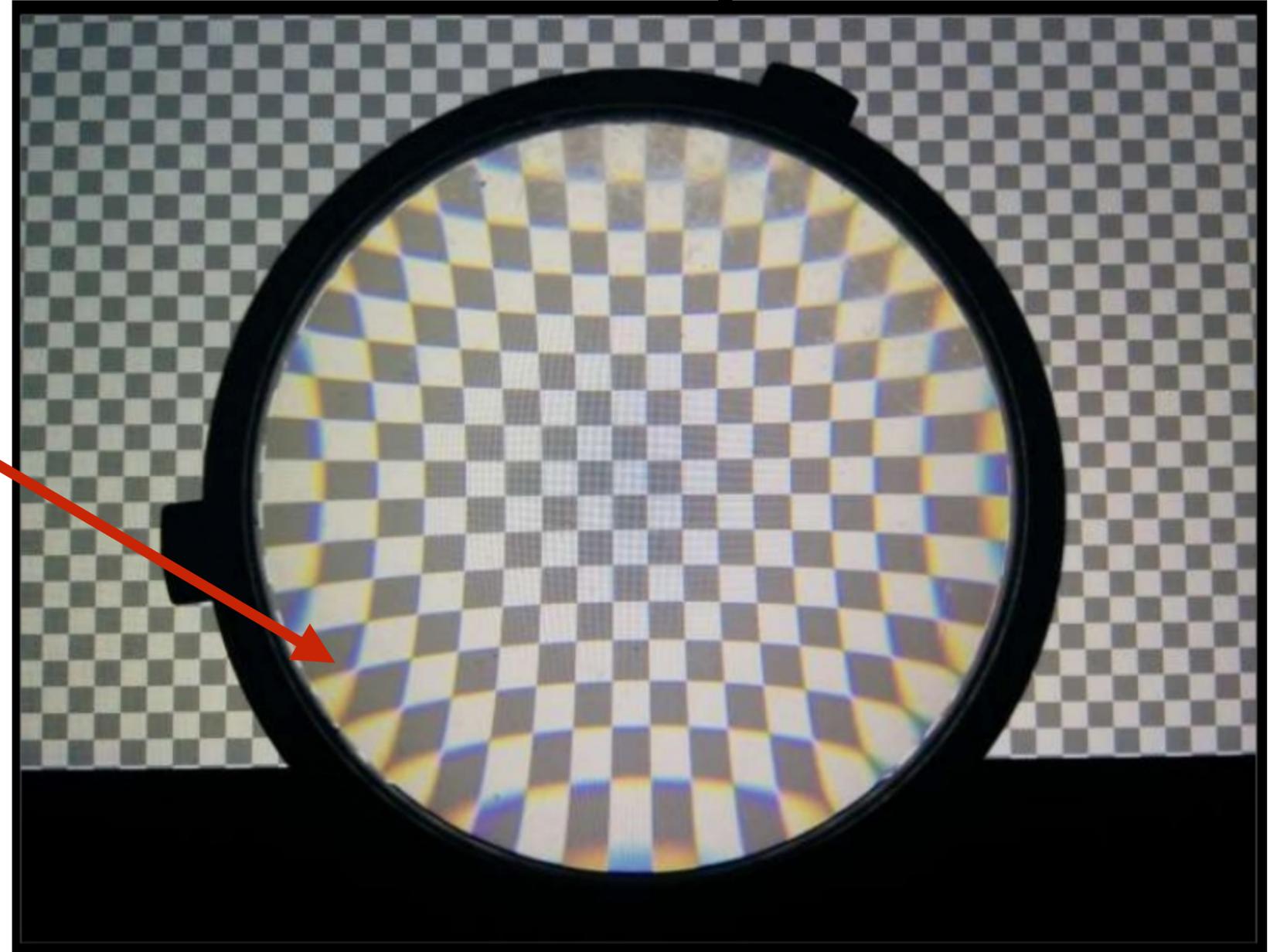
Rendered output must compensate for distortion of lens in front of display

Step 1: render scene using traditional graphics pipeline at full resolution for each eye

Step 2: warp images so rendering is viewed correctly when screen viewed under lens distortion

    (Can apply different distortion to R, G, B to approximate correction for chromatic aberration)

Press F9 for Full-Screen on Rift

Problem: rendering at higher resolution than needed at periphery
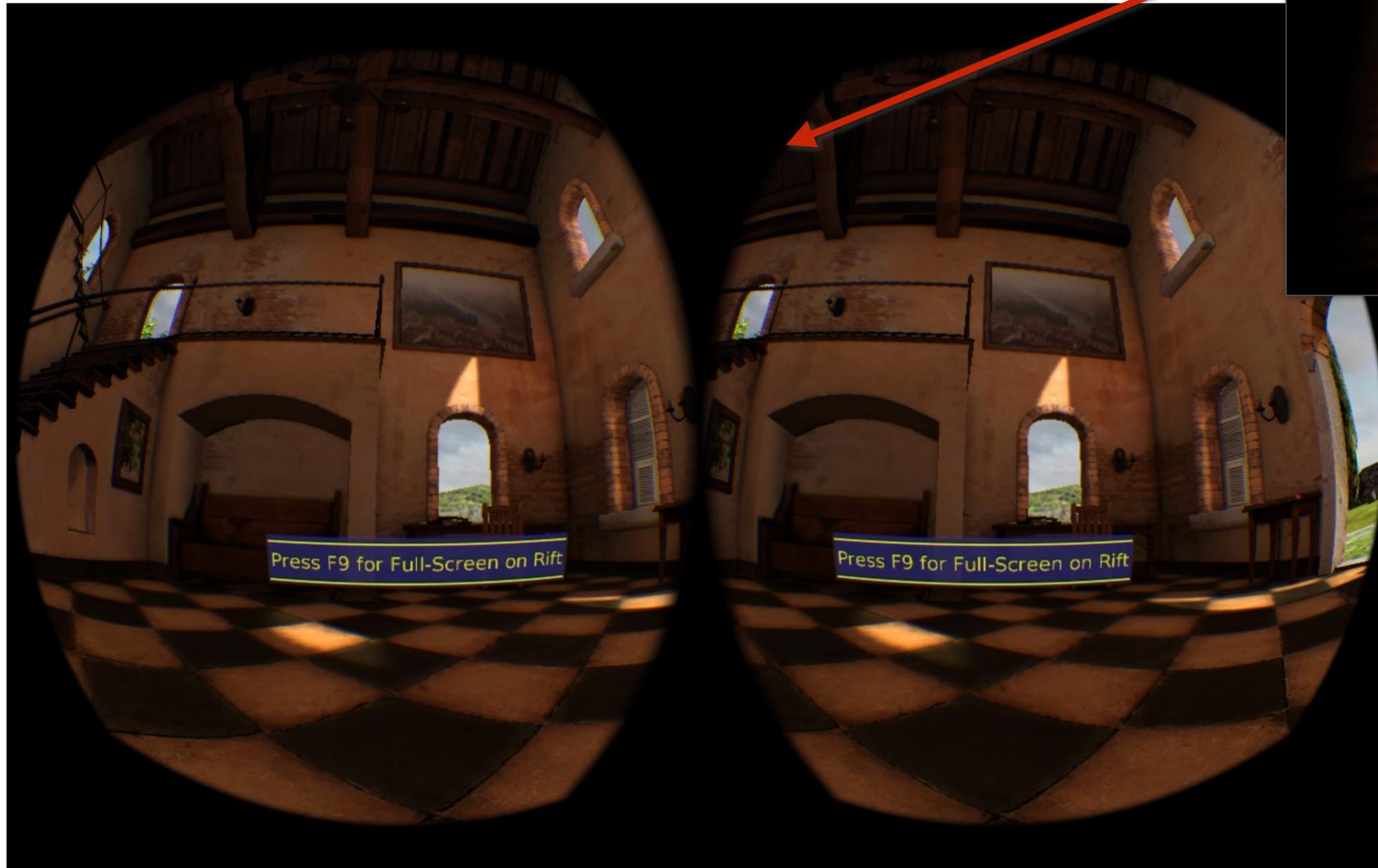

Rendered Image

Warped Image

Shading Rate After Lens Warp
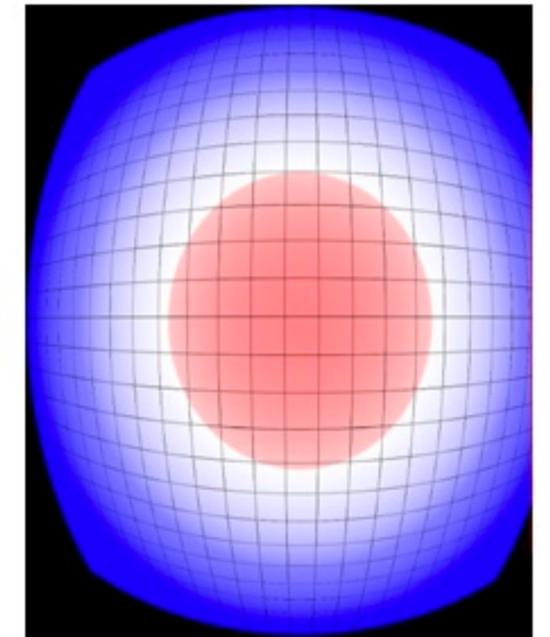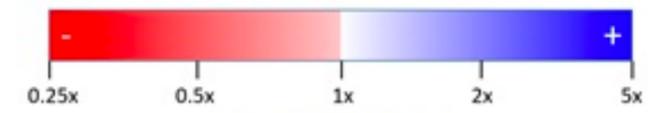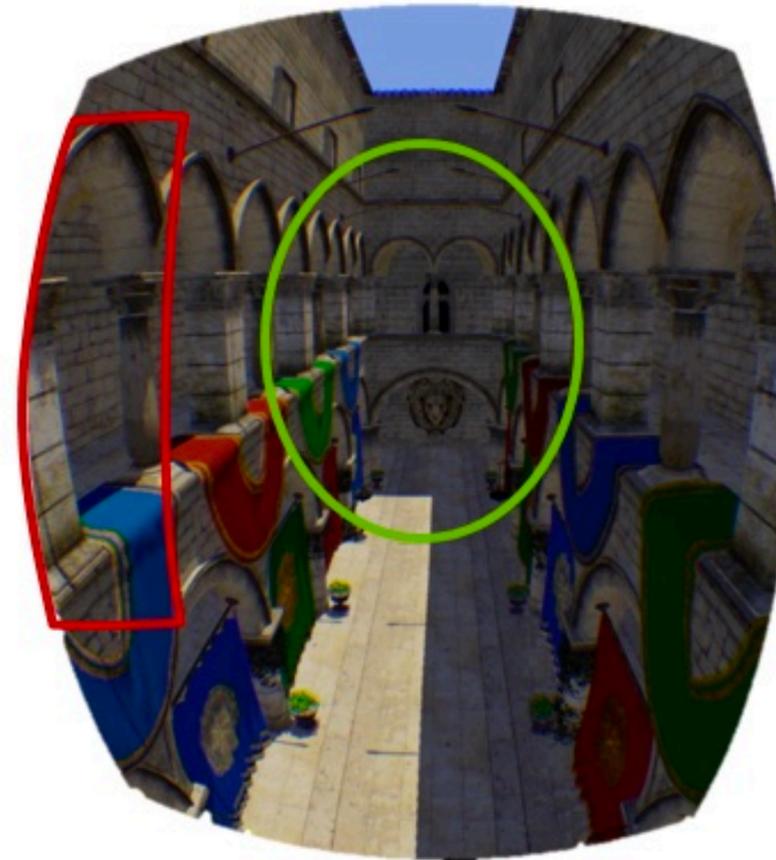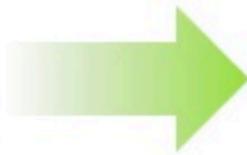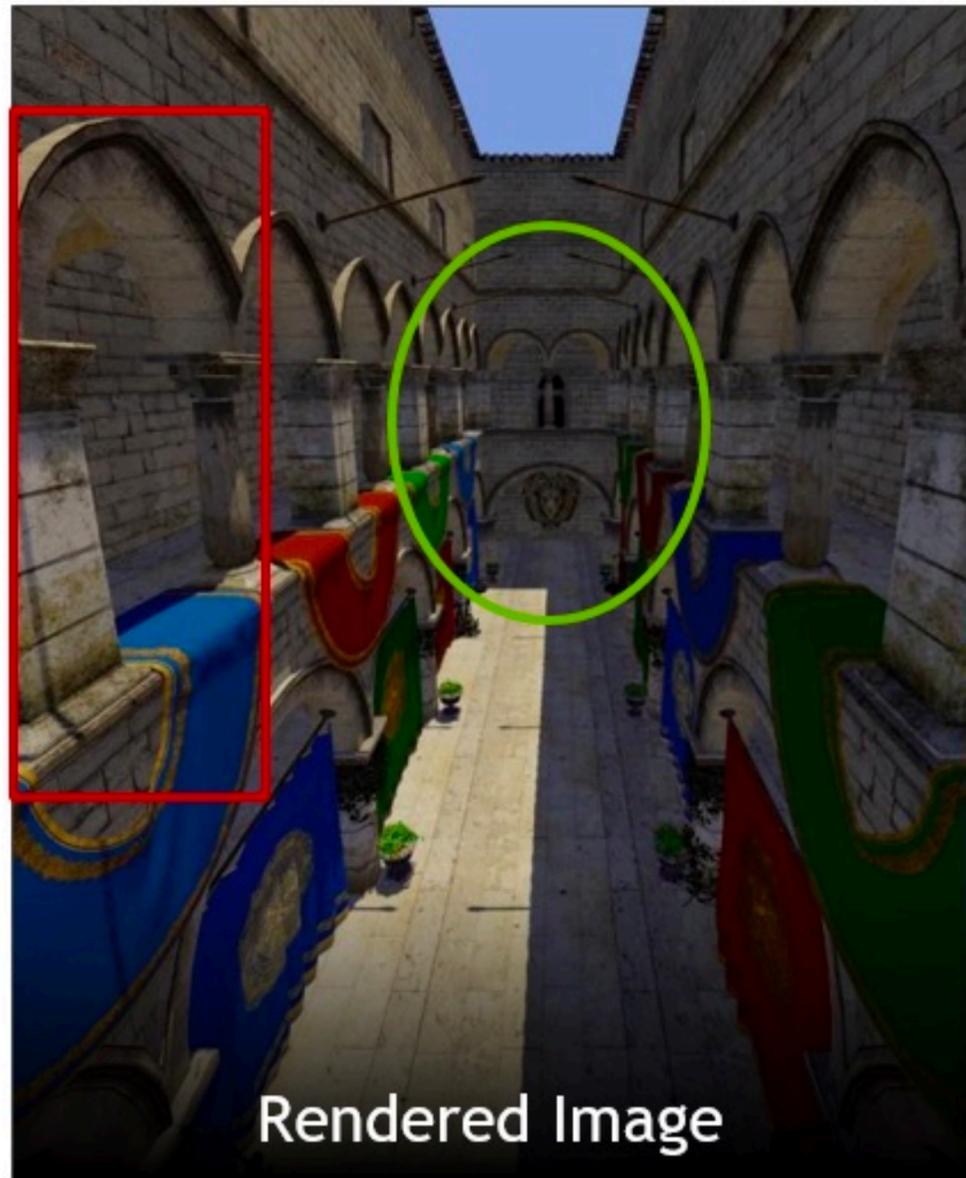
Performing unnecessary rendering work in the periphery due to:

1. Warp to reduce optical distortion (result: sample shading more densely in the periphery than in center of screen)
2. Eye has less spatial resolution in periphery (assuming viewer's gaze is toward center of screen)

Modern solution: lens matched shading

- **Render scene with four viewports, each has different projection matrix**
- **"Compresses" scene in the periphery (fewer samples), while not affecting scene near center of field of view**

Original Image

LMS Image
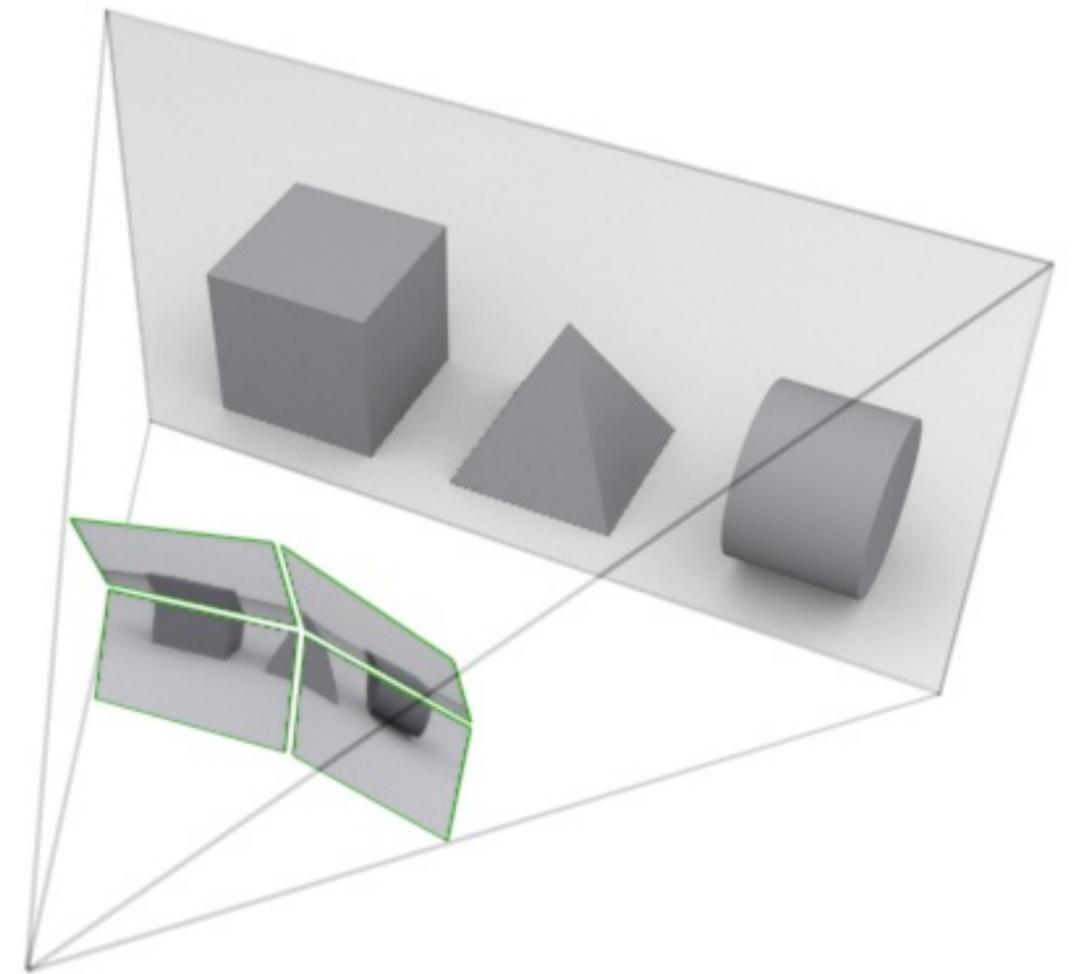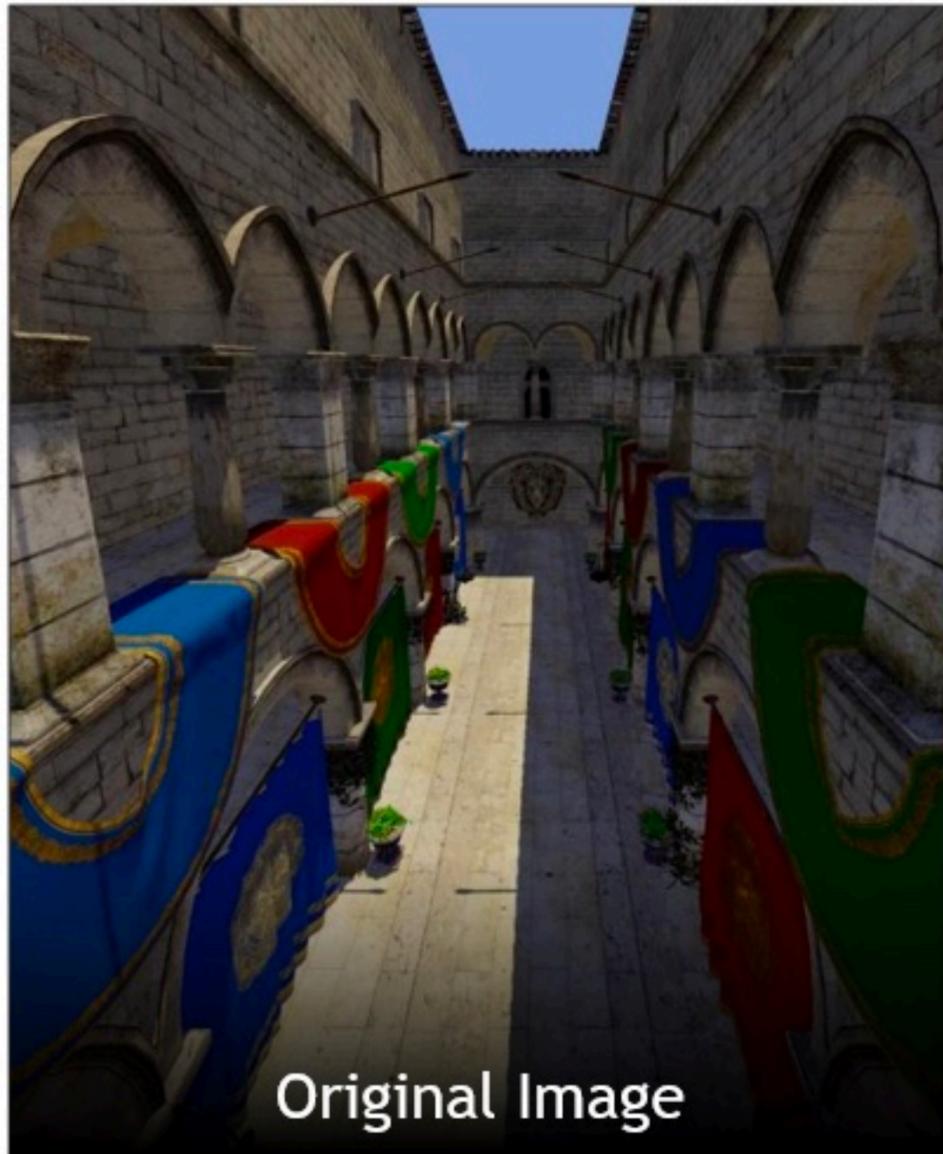
Need for low latency
(End-to-end head motion to photon latency)

How not to:

🤢🤮
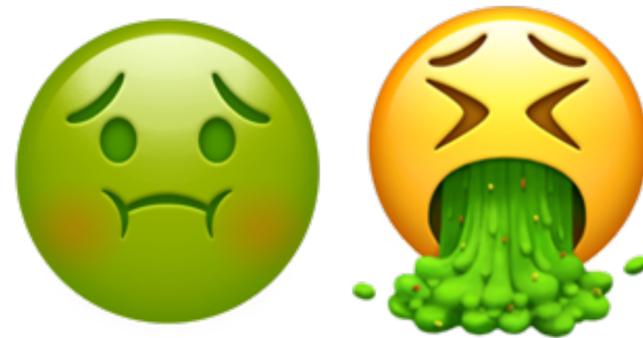
Need for low latency

■ **The goal of a VR graphics system is to achieve "presence", tricking the brain into thinking what it is seeing is real**

■ **Achieving presence requires an exceptionally low-latency system**
  - **What you see must change when you move your head!**
  - **End-to-end latency: time from moving your head to the time new photons from the display hit your eyes**
    - **Measure user's head movement**
    - **Update scene/camera position**
    - **Render new image**
    - **Perform any distortion corrections**
    - **Transfer image to display in headset**
    - **Actually emit light from display (photons hit user's eyes)**
  - **Latency goal of VR: 10-25 ms**
    - **Requires exceptionally low-latency head tracking**
    - **Requires exceptionally low-latency rendering and display**

Thought experiment: effect of latency

- **Consider a 1,000 x 1,000 display spanning 100° field of view**
  - **10 pixels per degree**

- **Assume:**
  - **You move your head 90° in 1 second (only modest speed)**
  - **End-to-end latency of graphics system is 33 ms (1/30 sec)**
    - In other words, the time from you moving you head to the display emitting light for a frame that reflects that movement.

- **Therefore:**
  - **Displayed pixels are off by 3° ~ 30 pixels from where they would be in an ideal system with 0 latency**

"Outside in" tracking: Oculus CV1 IR camera and IR LEDs (Early headset technology)

Headset contains:
IR LEDs (tracked by camera)
Gyro + accelerometer (1000Hz). (rapid relative positioning)

60Hz IR Camera
(measures absolute position
of headset 60 times a second)

Most modern systems use "inside out" tracking

- **Wide-angle cameras look outward from headset**

- **Use computer vision (SLAM) to estimate 3D structure of world and position/orientation of camera in the world**

- **These cameras also track the position/orientation of the controllers**
    - **Quest 2 controllers have 15 infrared LEDs to aid tracking**
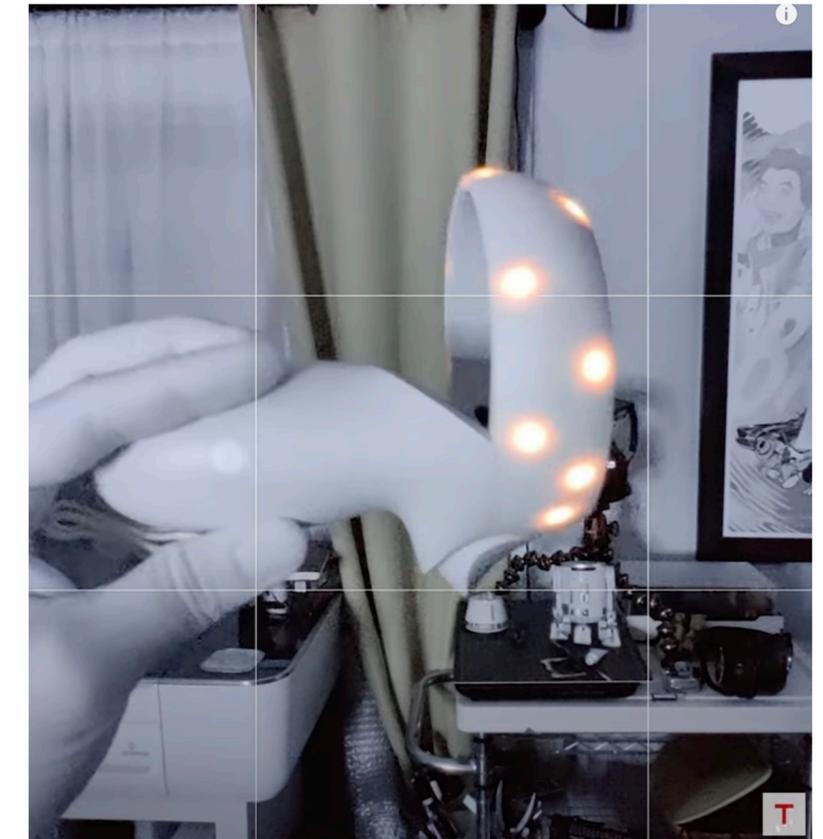
**View of controller through
infrared camera
(credit Adam Savage's Testbed)**

Frame life cycle

- **Goal: maintain as low latency as possible under challenging rendering conditions:**

  - **Battery-powered device (not a high-end desktop CPU/GPU)**

  - **High-resolution outputs (+ both left and right eye views)**

  - **Implication: can take awhile to render a frame** 🤮

Update game state (animation, physics, AI, etc)

Issue GPU
rendering commands

Main thread (frame 0)

Render Thread (frame 0)
(send GPU command)

GPU processing (frame 0)

Compositor
(Frame 0)

vsync   vsync   vsync

Frame life cycle

- **Goal: maintain as low latency as possible under challenging rendering conditions:**

  - **Battery-powered device (not a high-end desktop CPU/GPU)**

  - **High-resolution outputs (+ both left and right eye views)**

  - **Implication: can take awhile to render a frame 🤮**

Pipelined view:

Assume vsync fires at 90 Hz
Frame throughput = 90 fps, frame latency > 1/90th of a second
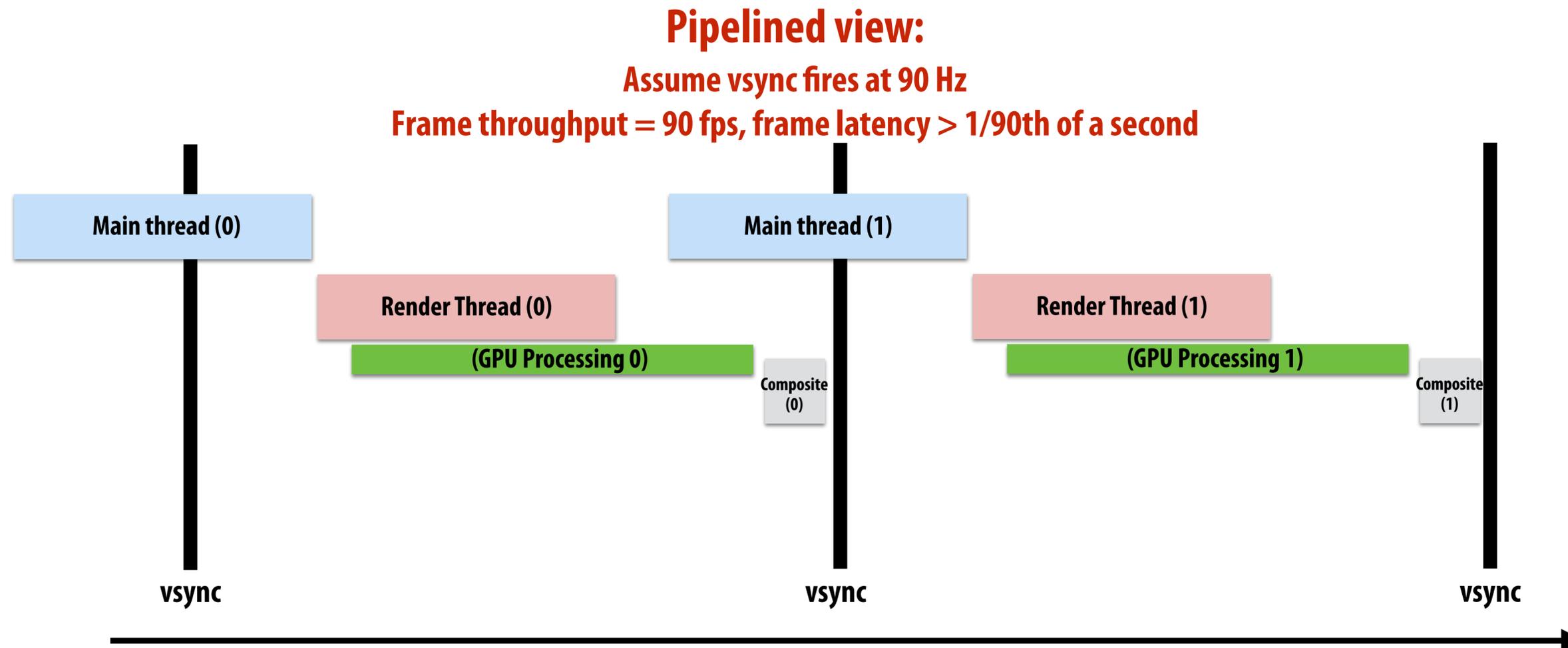
Main thread (0)		Main thread (1)	
	Render Thread (0)		Render Thread (1)
	(GPU Processing 0)		(GPU Processing 1)
	Composite (0)		Composite (1)

vsync   vsync   vsync
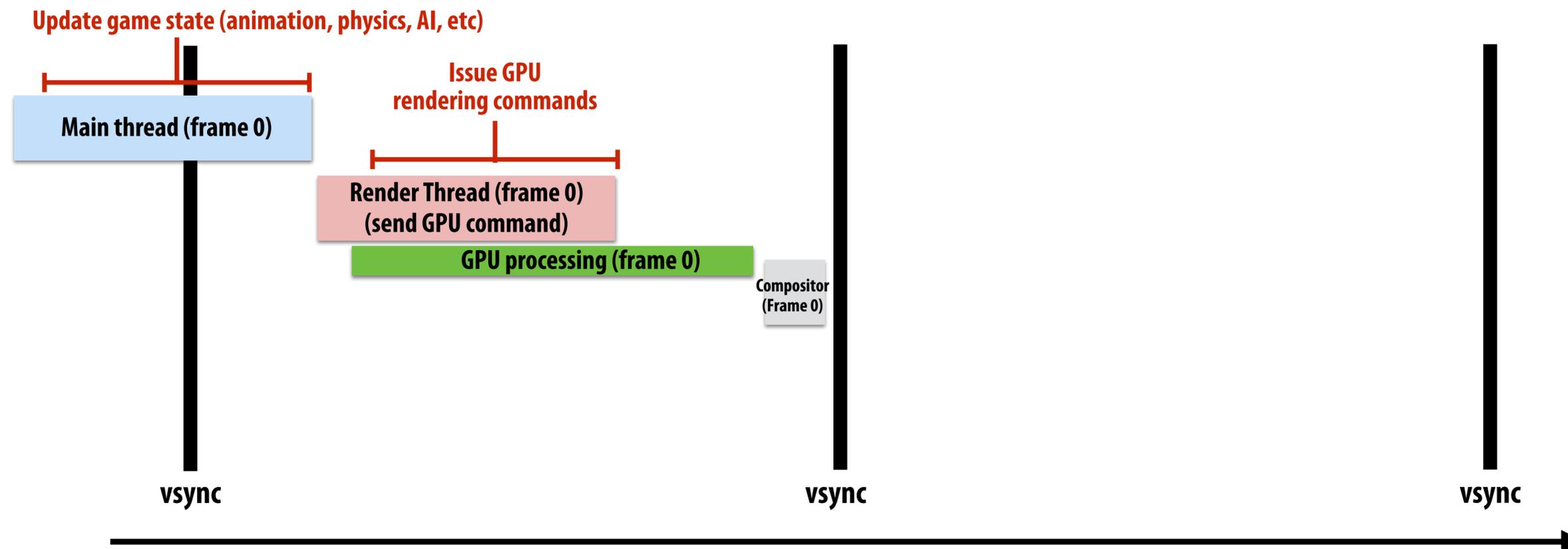
Frame life cycle
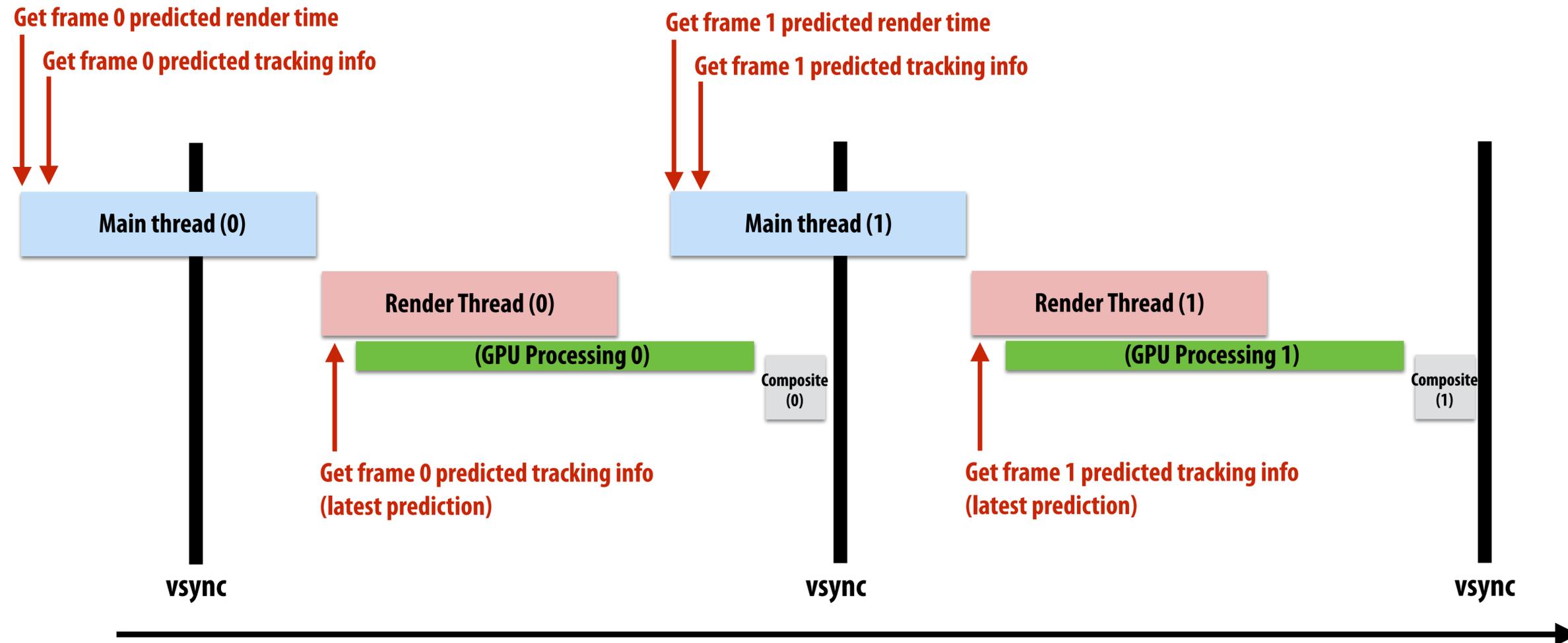
- **Key ideas:**
  - Game state updated on "predicted" tracking info
  - Re-update head/controller tracking predictions right before drawing
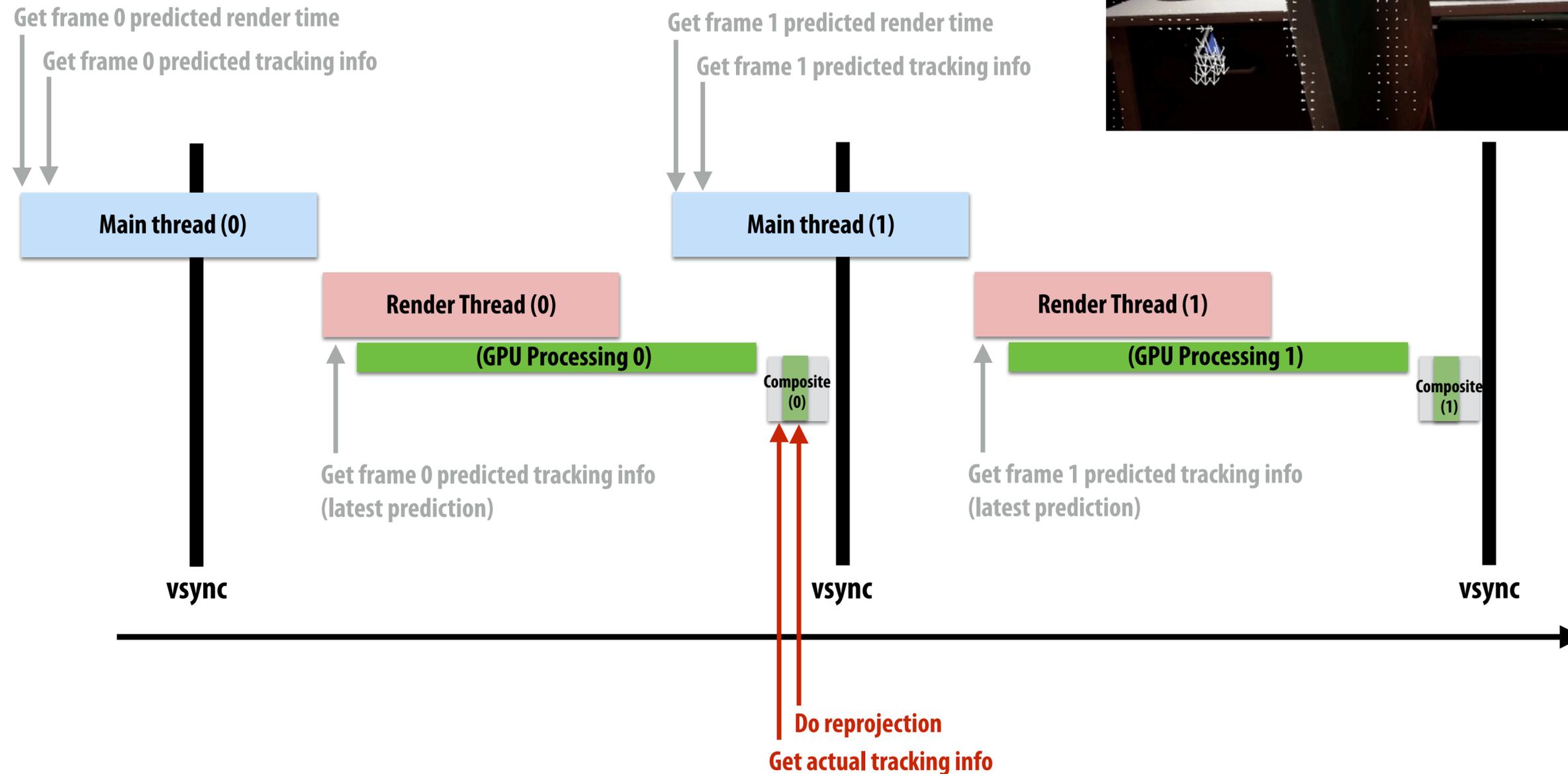 - Start next frame (frame 1 in this example) at last possible moment that gives it time to finish before target display time

Reducing latency via reprojection

Get frame 0 predicted render time

Get frame 0 predicted tracking info

Get frame 1 predicted render time

Get frame 1 predicted tracking info

ASW 2.0

Main thread (0)

Main thread (1)

Render Thread (0)

(GPU Processing 0)

Render Thread (1)

(GPU Processing 1)

Composite (0)

Composite (1)

Get frame 0 predicted tracking info
(latest prediction)

Get frame 1 predicted tracking info
(latest prediction)

vsync

vsync

vsync

Do reprojection

Get actual tracking info

- **Key idea ("time warp"): after rendering is complete, re-project rendered image to produce view of scene from most recent head position**

- **Accurate re-projection requires both rendered image and its depth map**

Reprojection

ASW 2.0

Increasing *frame rate* via reprojection

Example: app with higher cost rendering
Per-frame GPU rendering time ~ 1.2x of time between display frames
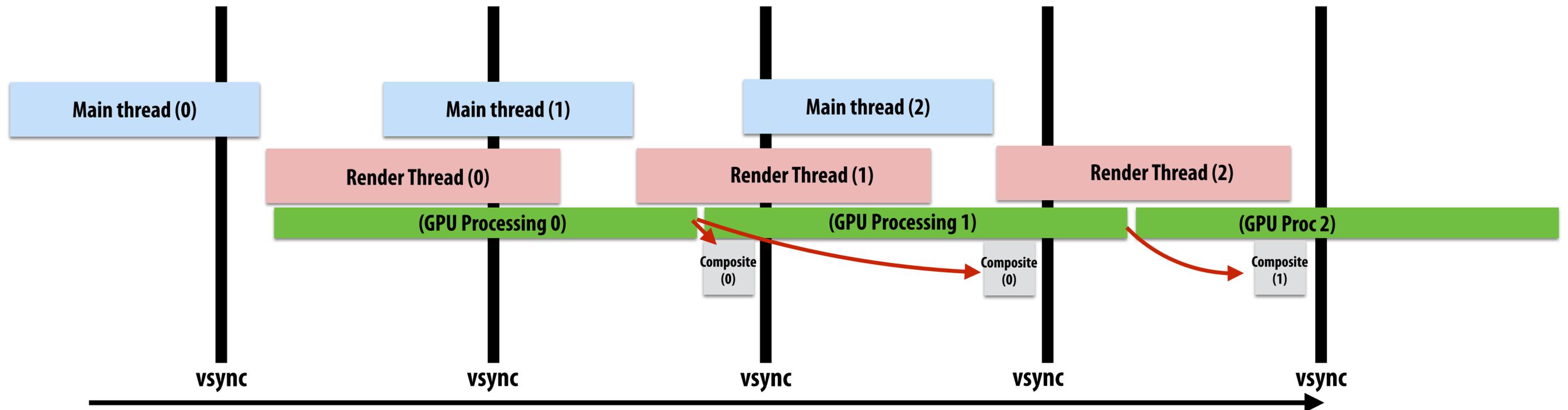
- **Store last rendered frame**
- **If new frame not ready at time of next display, warp that last completed frame**

Accounting for interaction of:
display update +
display attached to head

# Consider projection of scene object on retina

Retina

X

Pupil

Lens
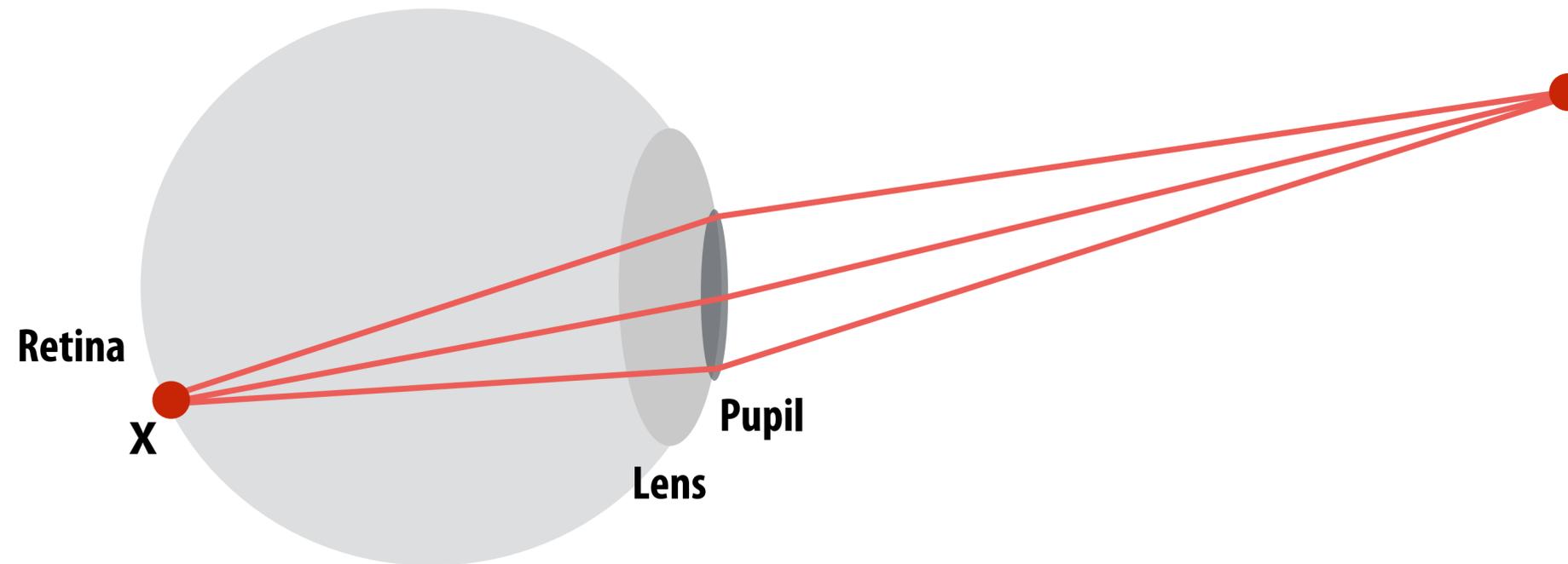
Here: object projects onto point X on back of eye (retina)

Consider object position relative to eye

X

(position of object relative to eye)

time

X

(position of object relative to eye)

time
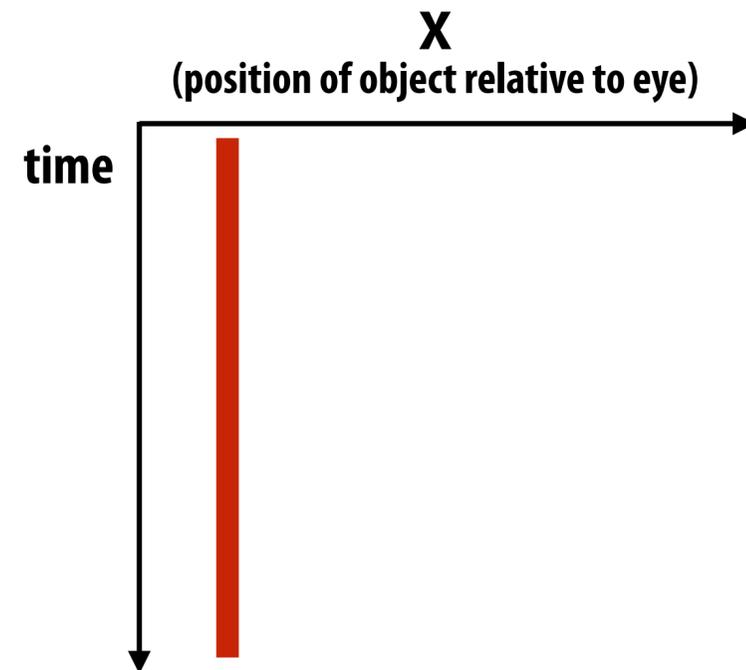
Case 1: object stationary relative to eye:
eye still and red object still
OR
red object moving left-to-right and
eye rotating to track object
OR
red object stationary in world but head moving
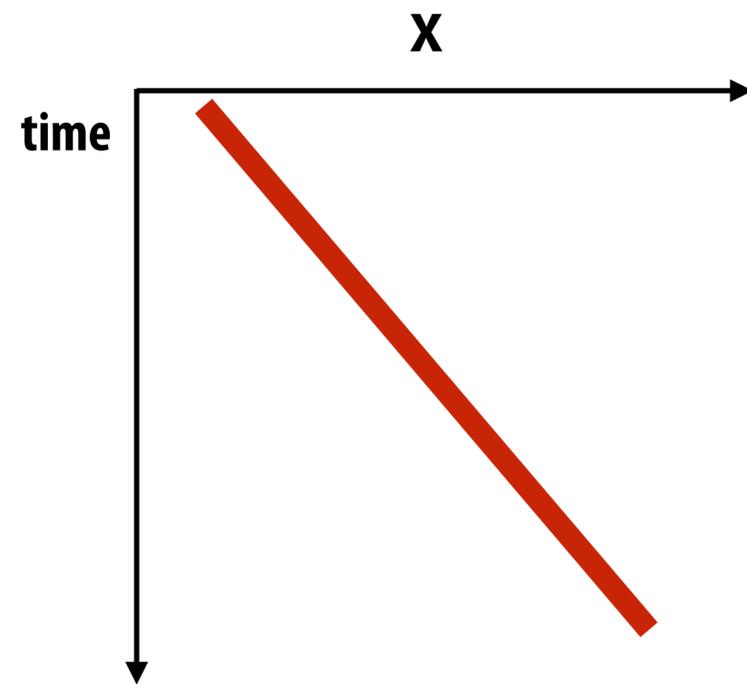and eye rotating to track object
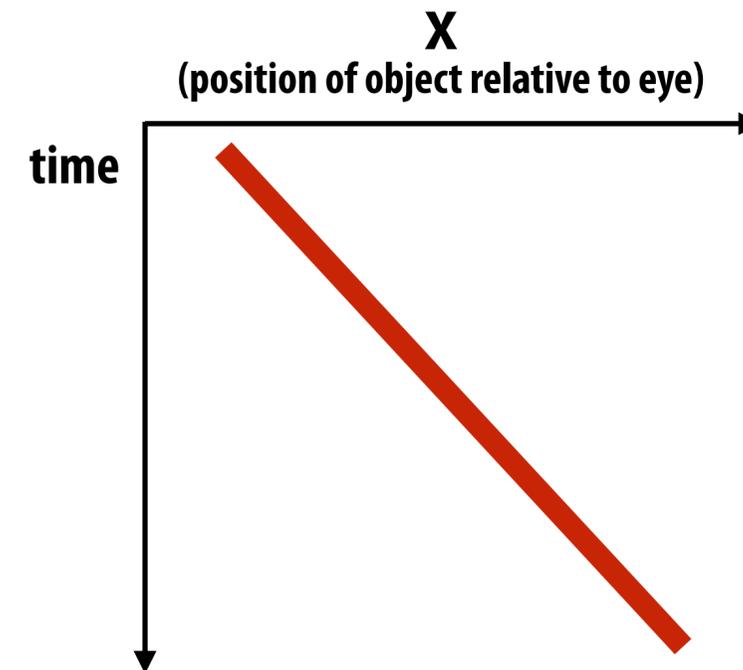
Case 2: object moving relative to eye:
(red object moving from left to right but
eye stationary, i.e., it's focused on a different
stationary point in world)

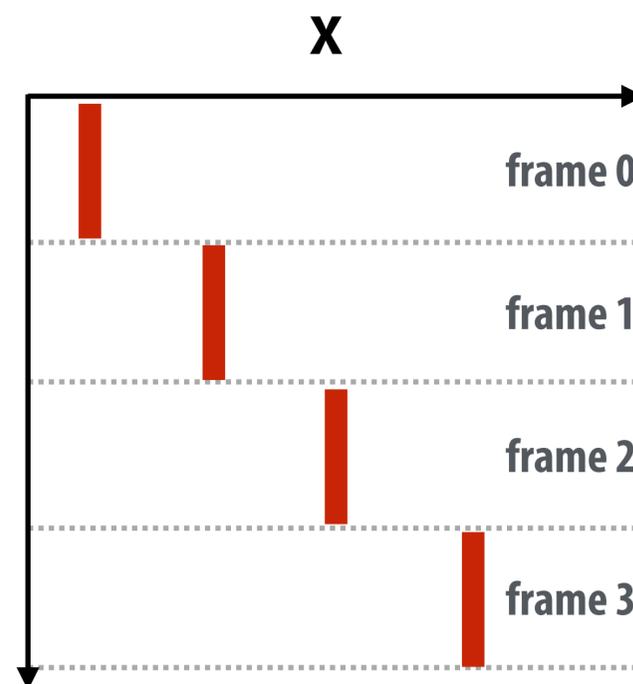NOTE: THESE GRAPHS PLOT <u>OBJECT POSITION</u> RELATIVE TO EYE

RAPID HEAD MOTION WITH EYES TRACKING A MOVING OBJECT IS A FORM OF CASE 1!!!
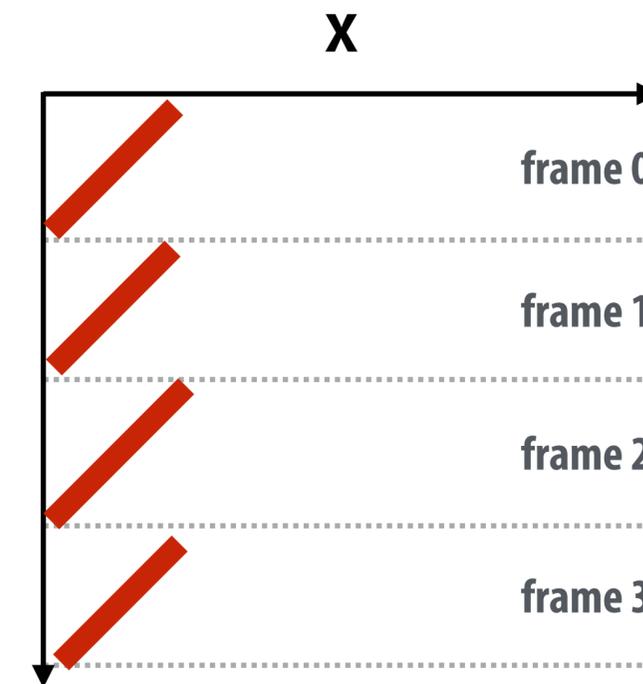
Effect of latency: judder

x

time

x

frame 0

frame 1

frame 2

frame 3

x

frame 0

frame 1

frame 2

frame 3

Case 2: object moving from left to right, eye stationary
(eye stationary with respect to display)

Continuous representation.

Case 2: object moving from left to right, eye stationary
(eye stationary with respect to display)

Light from display
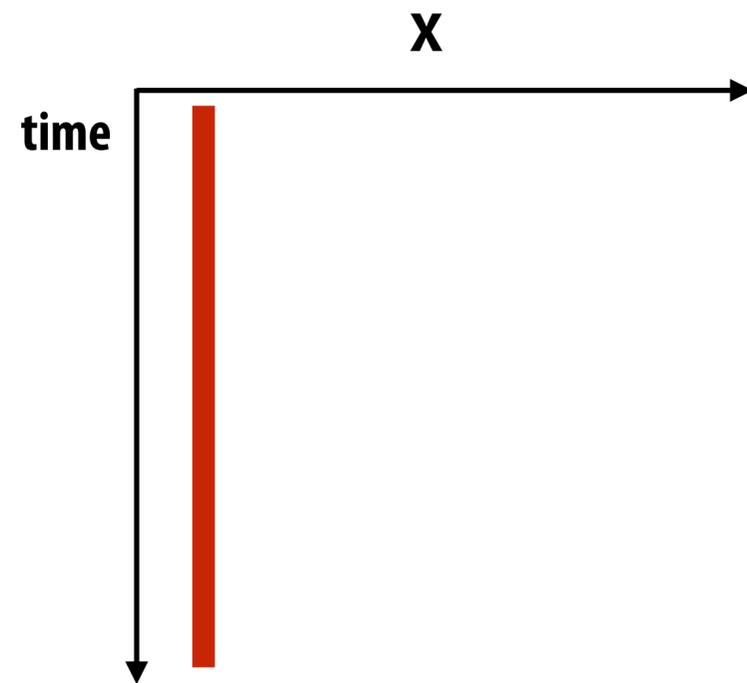**(image is updated each frame)**

Case 1: object moving from left to right, eye moving _continuously_ to track object
(eye moving relative to display!)

Light from display
(image is updated each frame)

Case 1 explanation: since eye is moving, object's position is relatively constant relative to eye (as it should be since the eye is tracking it). But due discrete frame rate, object falls behind eye, causing a smearing/strobing effect ("choppy" motion blur). Recall from earlier slide: 90 degree motion, with 50 ms latency results in 4.5 degree smear

Reducing judder: increase frame rate
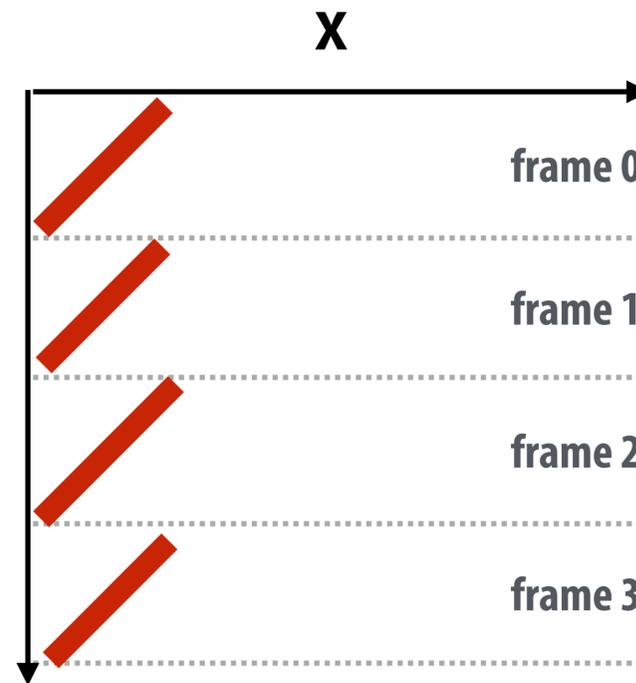

X

time

Case 1: continuous ground truth

red object moving left-to-right and
eye moving to track object
OR
red object stationary but head moving
and eye moving to track object

X

frame 0

frame 1

frame 2

frame 3

Light from display
(image is updated each frame)

X

frame 0
frame 1
frame 2
frame 3
frame 4
frame 5
frame 6
frame 7

Light from display
(image is updated each frame)

Higher frame rate results in closer
approximation to ground truth

Reducing judder: low persistence display

X   X   X

time

frame 0

frame 1

frame 2

frame 3

frame 0

frame 1

frame 2

frame 3

Case 1: continuous ground truth

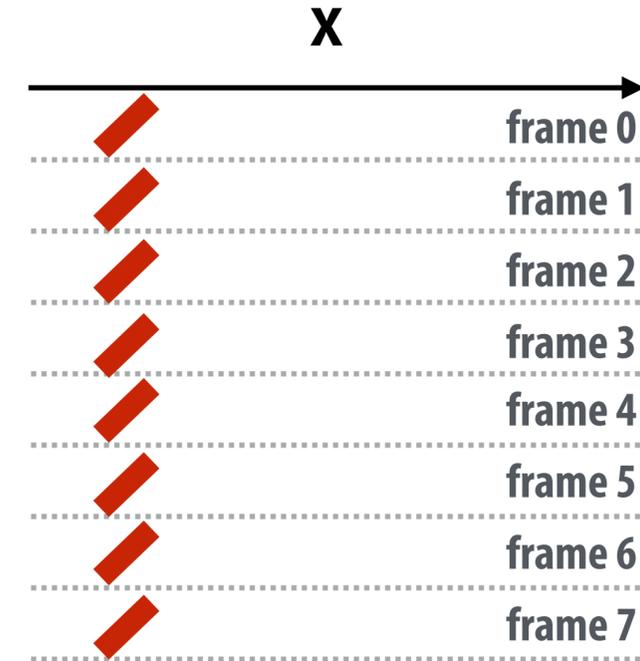**Light from full-persistence display**

Light from low-persistence display

**red object moving left-to-right and
eye moving to track object
OR
red object stationary but head moving
and eye moving to track object**

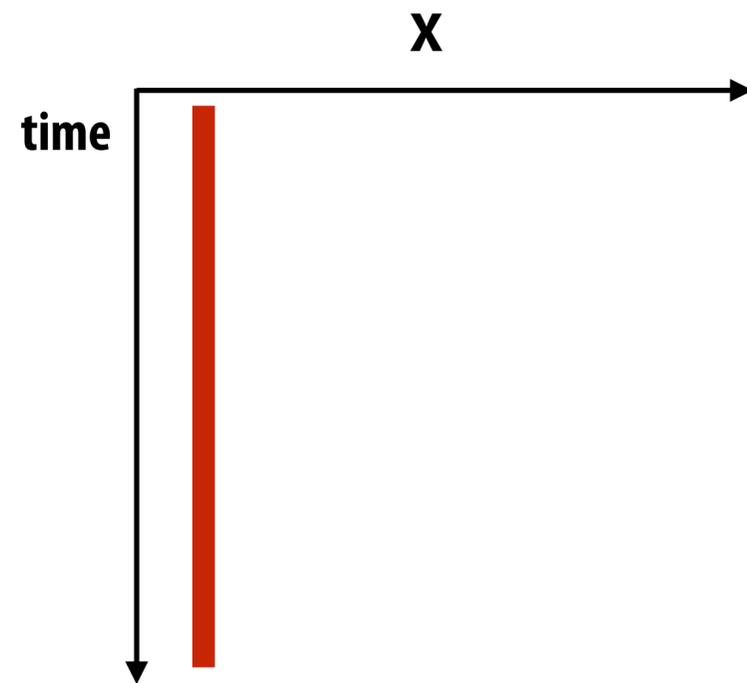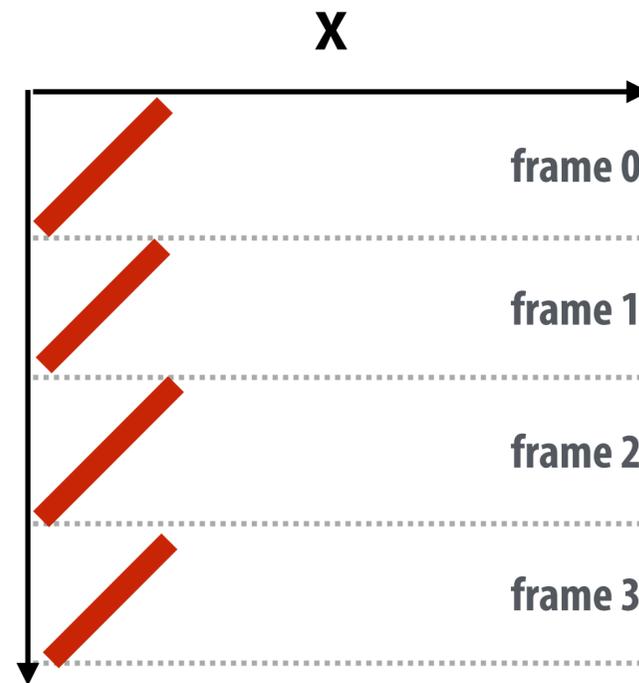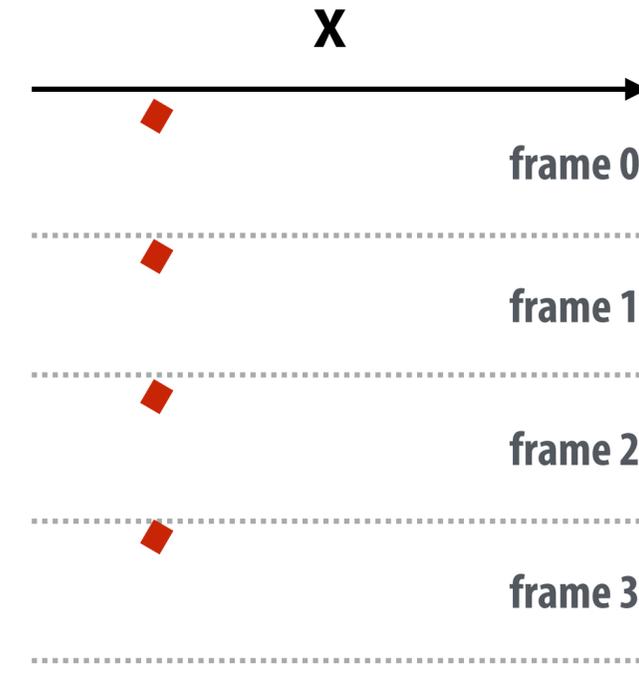**Full-persistence display: pixels emit light for entire frame**

Low-persistence display: pixels emit light for small fraction of frame

Oculus Rift CV1 low-persistence display

- **90 Hz frame rate (~11 ms per frame)**
- **Pixel persistence = 2-3ms**

Artifacts due to rolling backlight

- **Image rendered based on scene state at time t_0**

- **Image sent to display, ready for output at time $t_0 + \Delta t$**

- **"Rolling backlight" OLED display lights up rows of pixels in sequence**
  - **Let r be amount of time to "scan out" a row**
  - **Row 0 photons hit eye at $t_0 + \Delta t$**
  - **Row 1 photos hit eye at $t_0 + \Delta t + r$**
  - **Row 2 photos hit eye at $t_0 + \Delta t + 2r$**

- **Implication: photons emitted from bottom rows of display are "more stale" than photos from the top!**

- **Consider <u>eye moving horizontally relative to display</u> (e.g., due to head movement while tracking square object that is stationary in world)**

Result: perceived shear!
Similar to rolling shutter effects on modern digital cameras.

X
(position of object relative to eye)

Y
display pixel row
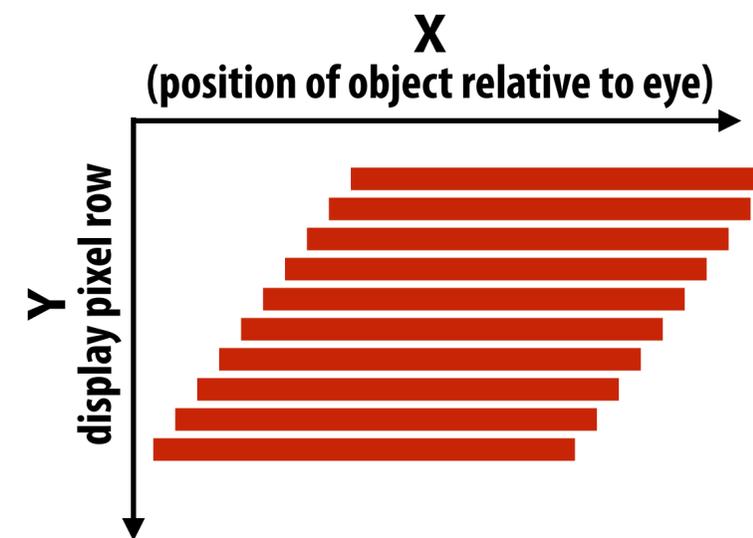
What you want to see

What you do see (sheared image)…
lower rows of image are increasingly "stale"

(User is rotating head to left — so scene should be shifting to right…
but bottom of screen is "lagging" and not updated to be consistent with head movement)

Compensating for rolling backlight

- **Perform post-process shear on rendered image**

  - **Similar to previously discussed barrel distortion and chromatic warps**

  - **Predict head motion, assume fixation on static object in scene**

    - **Only compensates for shear due to head motion, not object motion**

- **Render each row of image at a different time (the predicted time photons will hit eye)**

  - **Suggests exploration of different rendering algorithms that are more amenable to fine-grained temporal sampling, e.g., ray tracing? (each row of camera rays samples scene at a different time)**

Reducing bulky form factor + AR

Glasses form factor (for AR applications)
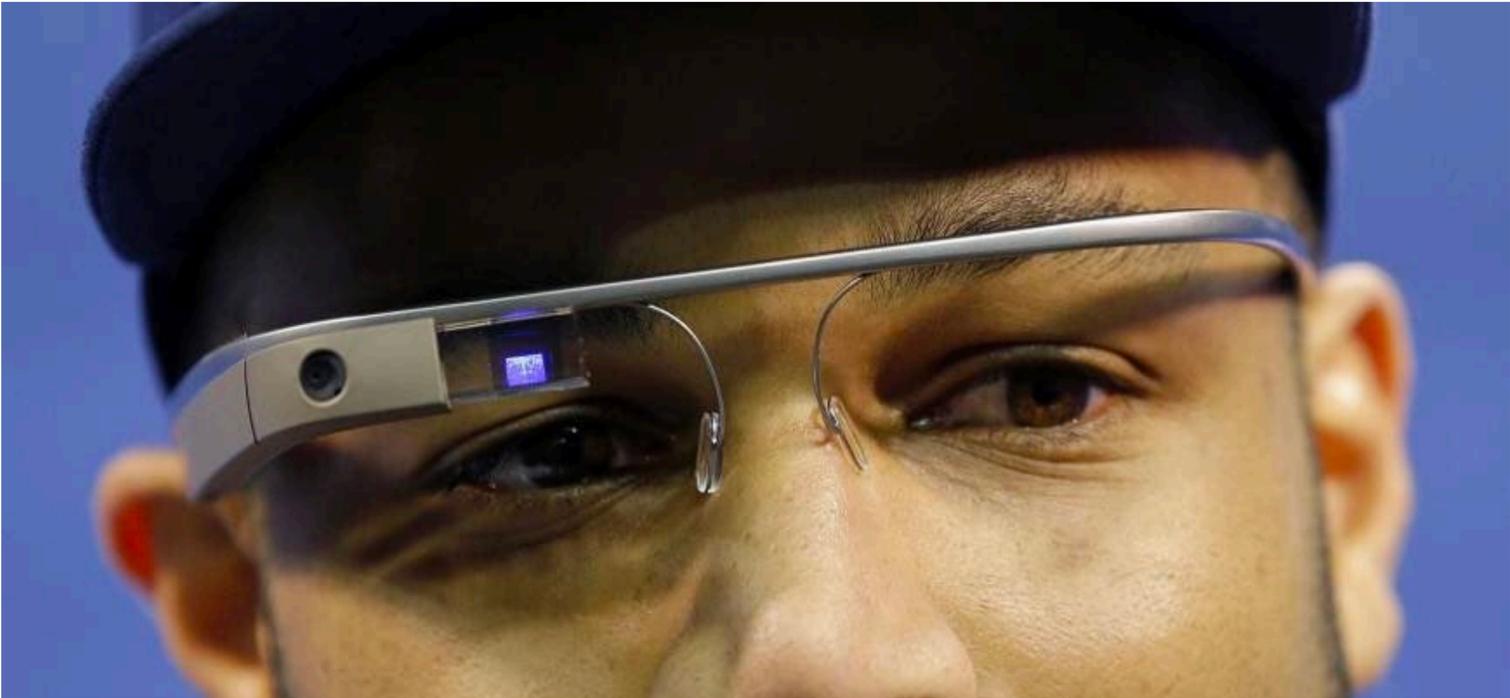
Google Glass (2013)

Qualcomm "AR smartviewer" glasses
(tethered to smartphone)

Snap Spectacles v4 (2021)   (Snap reports 30 minute battery life)

Ray-ban Meta "AI glasses"

- **NOT an AR device (no display)**
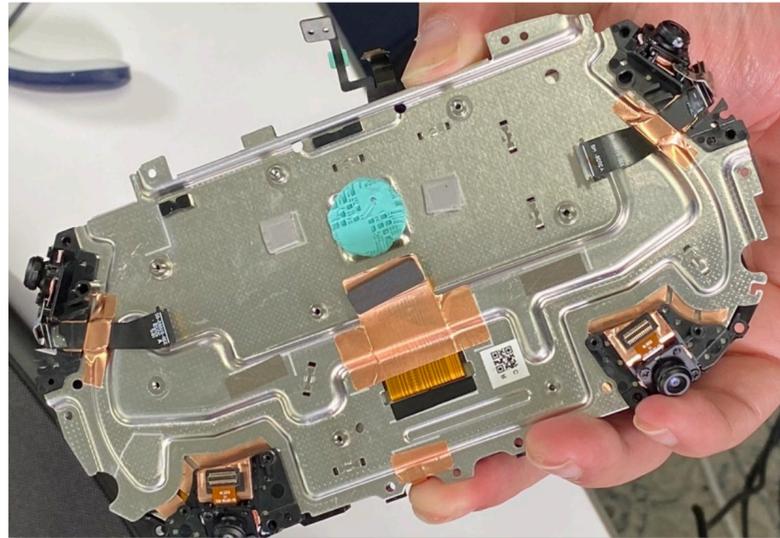- **Contains camera, mic, speakers**

Meta Ray-ban display

- **Has an 600x600 AR display**
- **20 degree FOV**
- **EMG band used to control device using subtle hand movements**
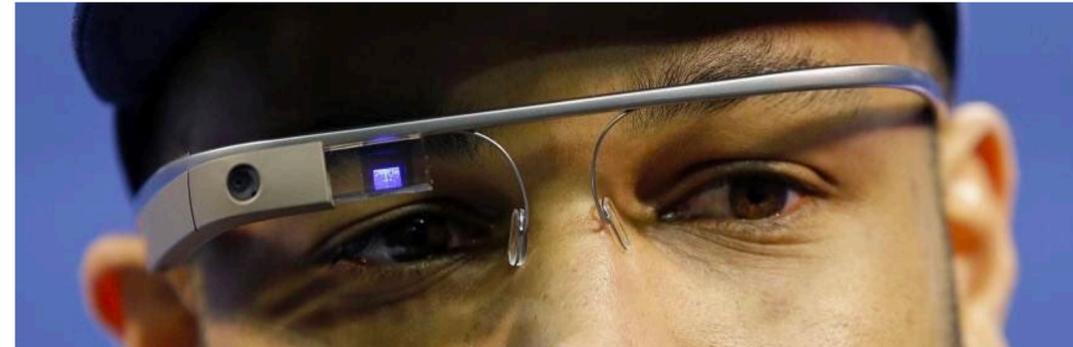
AR / VR summary

- **Difficult technical challenge**
- **Interest in glasses form factor will place considerably more pressure on system efficiency**

VS.

- **Many new challenges of AR:**
  - Rendering to a display that "overlays" on the real world (how to draw black?)
  - Intelligently interpreting the world to know what content to put on the display
  - Ethical/privacy questions about applications