The Graphics Pipeline and OpenGL IV:
Stereo Rendering, Depth of Field Rendering, Multi-pass Rendering

Gordon Wetzstein

Stanford University

EE 267 Virtual Reality

Lecture 6

stanford.edu/class/ee267/

Lecture Overview

- overview of glasses-based stereo
- stereo rendering with OpenGL
  - projection matrix
  - view matrix
- offscreen frame buffers and multi-render passes
- anaglyph stereo rendering with GLSL
- depth of field rendering

Glasses-based Stereo

1. Anaglyph

2. Polarization

3. Shutter Glasses

4. Chromatic Filters (e.g., Dolby)

Glasses-based Stereo

2. Polarization

Glasses-based Stereo

- passive glasses
- active LC element on projector or interlaced rows/columns on monitor (resolution loss)

2. Polarization

- e.g. RealD – most 3D cinemas use this
- circular polarization to allow for head roll
- inexpensive glasses, little crosstalk
- need polarization-preserving screen!

Glasses-based Stereo

3. Shutter Glasses

Glasses-based Stereo

- active glasses, temporally-multiplexed display

- e.g. StereoGraphics

- somewhat expensive glasses, little crosstalk

- need fast display (at least 120 Hz)

- sync monitor update with glasses

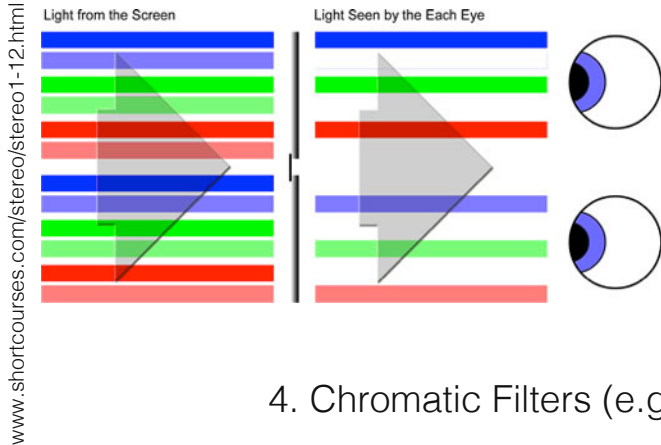3. Shutter Glasses

Glasses-based Stereo

4. Chromatic Filters (e.g., Dolby)

Glasses-based Stereo

- passive glasses, usually two projectors with passive color filters

- somewhat expensive glasses (not as widespread in cinemas)

- full color!

www.shortcourses.com/stereo/stereo1-12.html

4. Chromatic Filters (e.g., Dolby)

Glasses-based Stereo

1. Anaglyph

Glasses-based Stereo

1. Anaglyph

- passive, inexpensive glasses (least expensive overall)
- no modifications to display necessary – just render stereo images in different colors
- cannot reproduce correct colors! but not as bad as it sounds

Put on Your 3D Glasses Now!

Anaglyph Stereo - Monochrome

- render L & R images, convert to grayscale
- merge into red-cyan anaglyph by assigning I(r)=L, I(g,b)=R (I is anaglyph)

from movie "Bick Buck Bunny"

Anaglyph Stereo – Full Color

- render L & R images, do not convert to grayscale
- merge into red-cyan anaglyph by assigning I(r)=L(r), I(g,b)=R(g,b) (I is anaglyph)

from movie "Bick Buck Bunny"

Anaglyph Stereo - Dubois

- paper: Eric Dubois "A Projection Method to Generate Anaglyph Stereo Images", ICASSP 2001

- optimize color management in CIE XYZ space

- requires spectral transmission of glasses & spectral emission curves of display primaries

- great course project - see previous course projects …

Open Source Movie: Big Buck Bunny
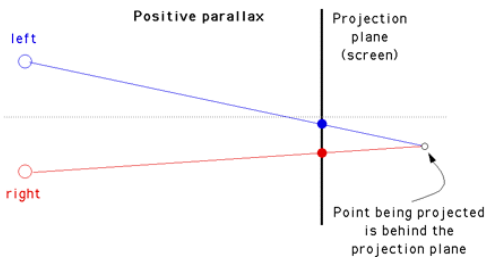
Rendered with Blender (Open Source 3D Modeling Program)

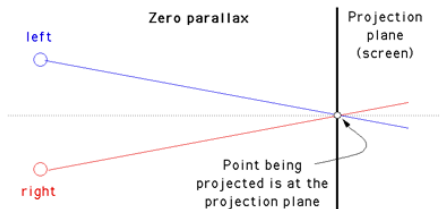http://bbb3d.renderfarming.net/download.html

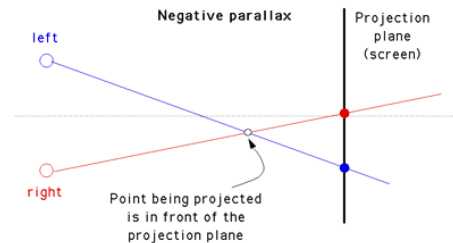# Anaglyph Comparison

… show video clips …

Parallax

- parallax is the relative distance of a 3D point projected into the 2 stereo images
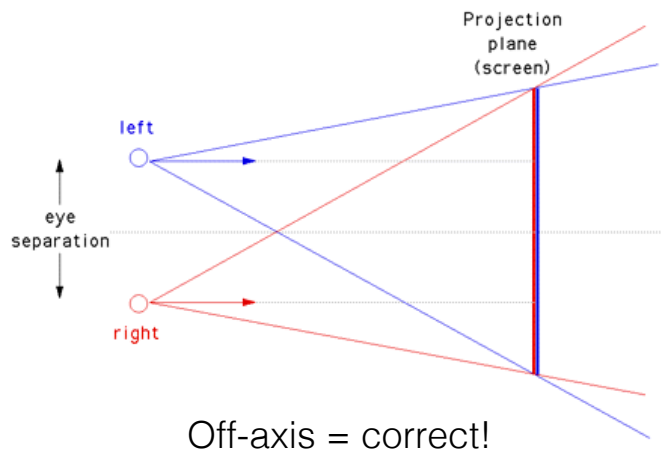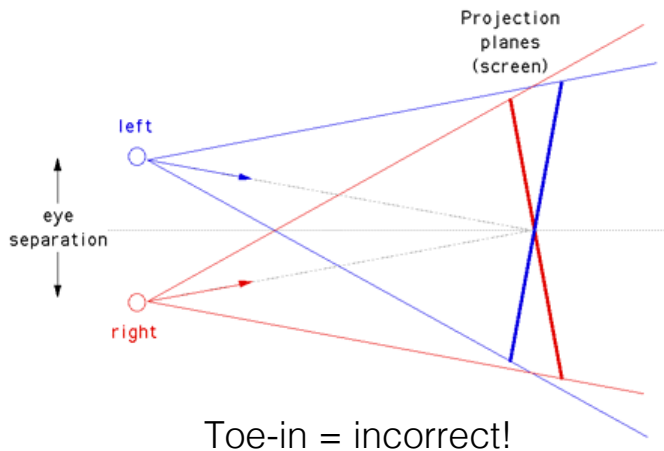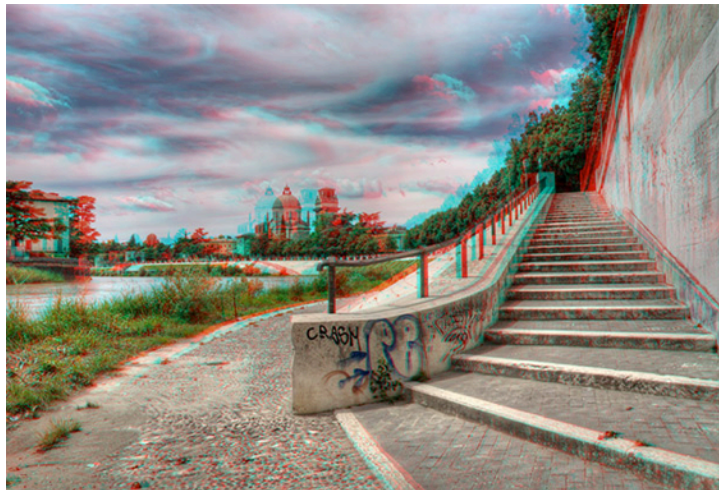

case 1

case 2

case 3

Parallax

- visual system only uses horizontal parallax, no vertical parallax!
- naïve toe-in method creates vertical parallax → visual discomfort

Toe-in = incorrect!   Off-axis = correct!

Parallax – well done

Parallax – well done

1862
"Tending wounded Union soldiers at Savage's Station, Virginia, during the Peninsular Campaign",
Library of Congress Prints and Photographs Division

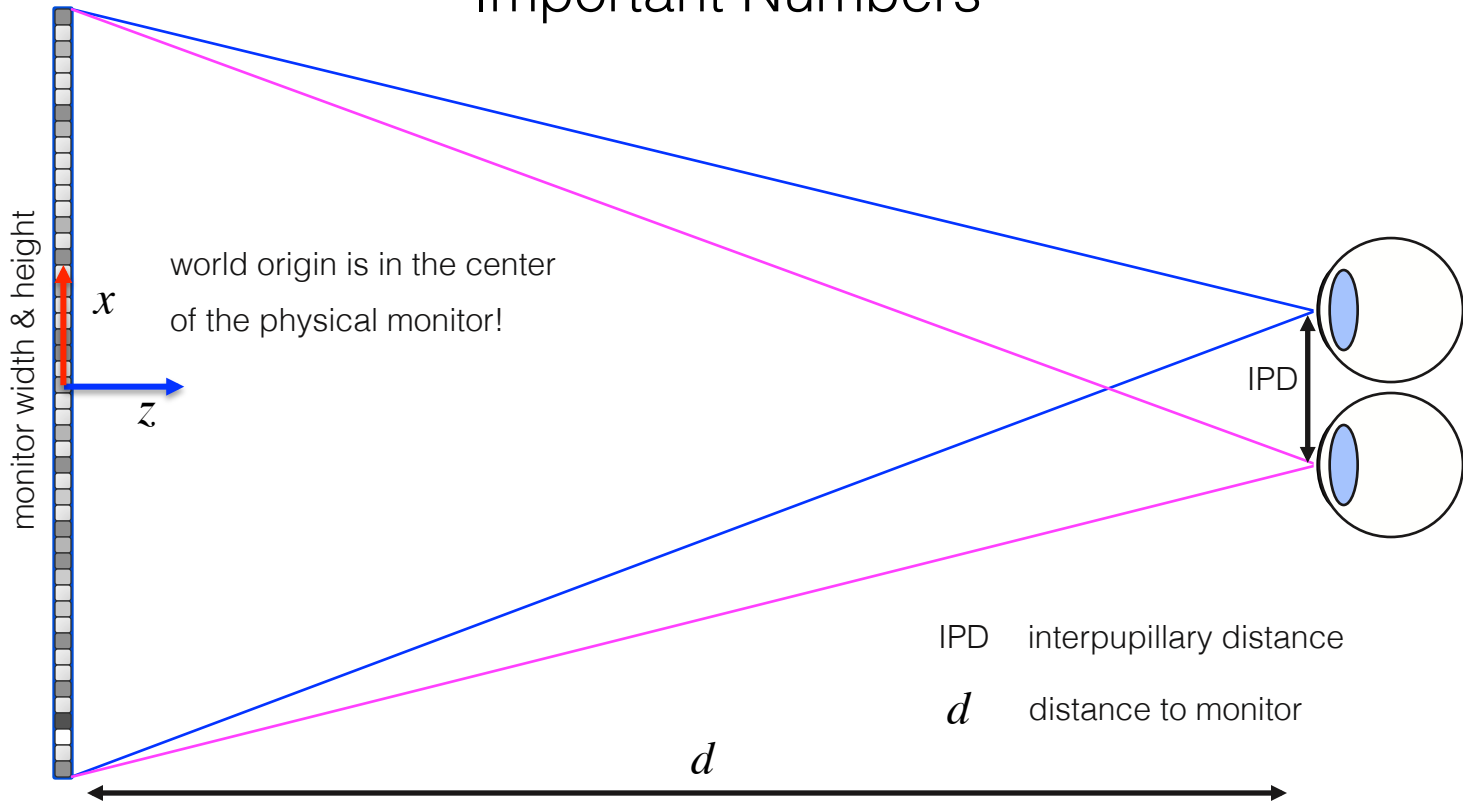# Parallax – not well done (vertical parallax = unnatural)
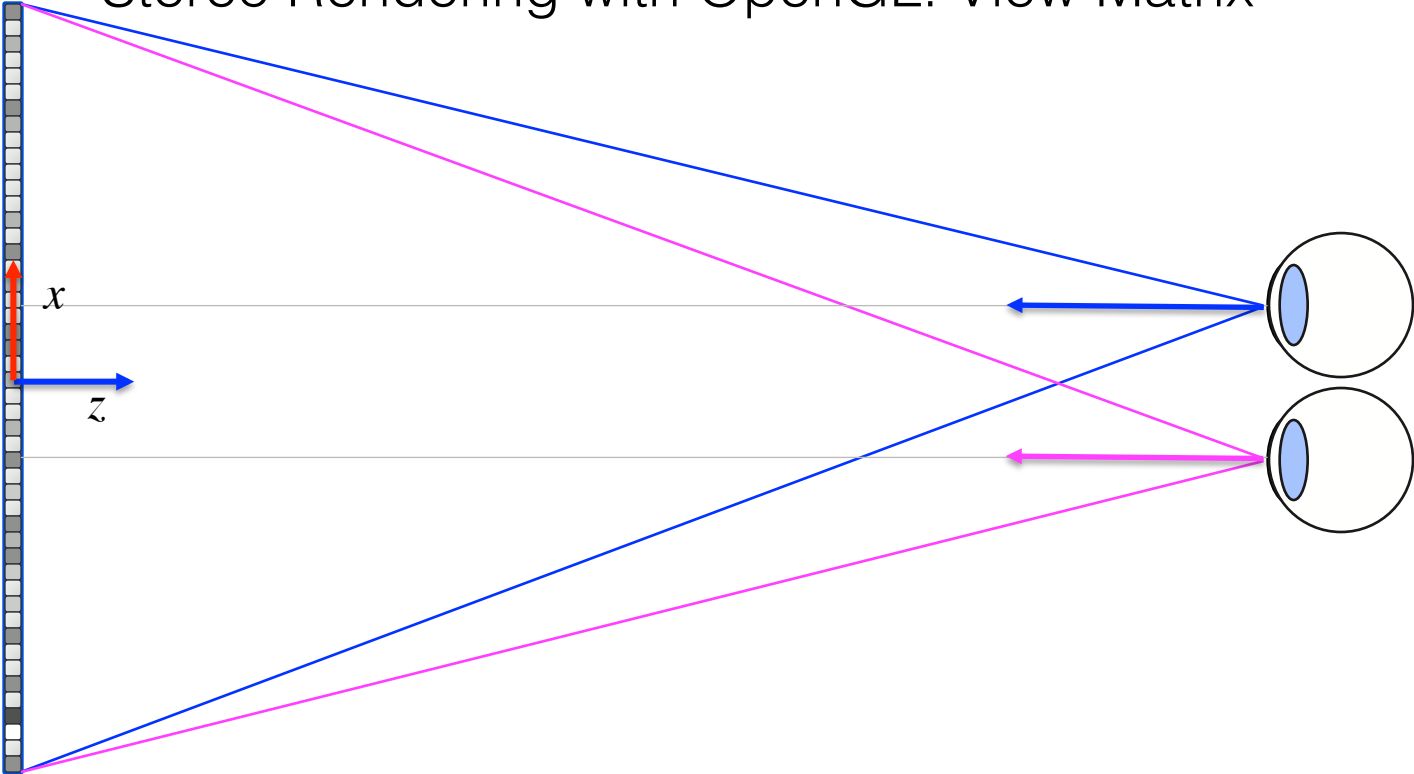
Take Off Your 3D Glasses Now!

Stereo Rendering with OpenGL/WebGL: View Matrix

- need to modify view matrix and projection matrix
- rendering pipeline does not change – only those two matrices

- however: need to render two images in sequence (more details later)

- look at view matrix first: write your own `lookAt` function that uses rotation & translation matrix to generate view matrix from `eye,center,up` parameters
- do not use `THREE.Matrix4().lookAt()` function – this does not work properly!

Important Numbers

monitor width & height

x

z

world origin is in the center of the physical monitor!

IPD

d

IPD interpupillary distance

d distance to monitor

Stereo Rendering with OpenGL: View Matrix

Stereo Rendering with OpenGL: View Matrix

center points for lookat!

Stereo Rendering with OpenGL: View Matrix

lookAt(vec3(ipd/2,0,d),
               vec3(ipd/2,0,0),
               vec3(0,1,0));

lookAt(vec3(-ipd/2,0,d),
               vec3(-ipd/2,0,0),
               vec3(0,1,0));
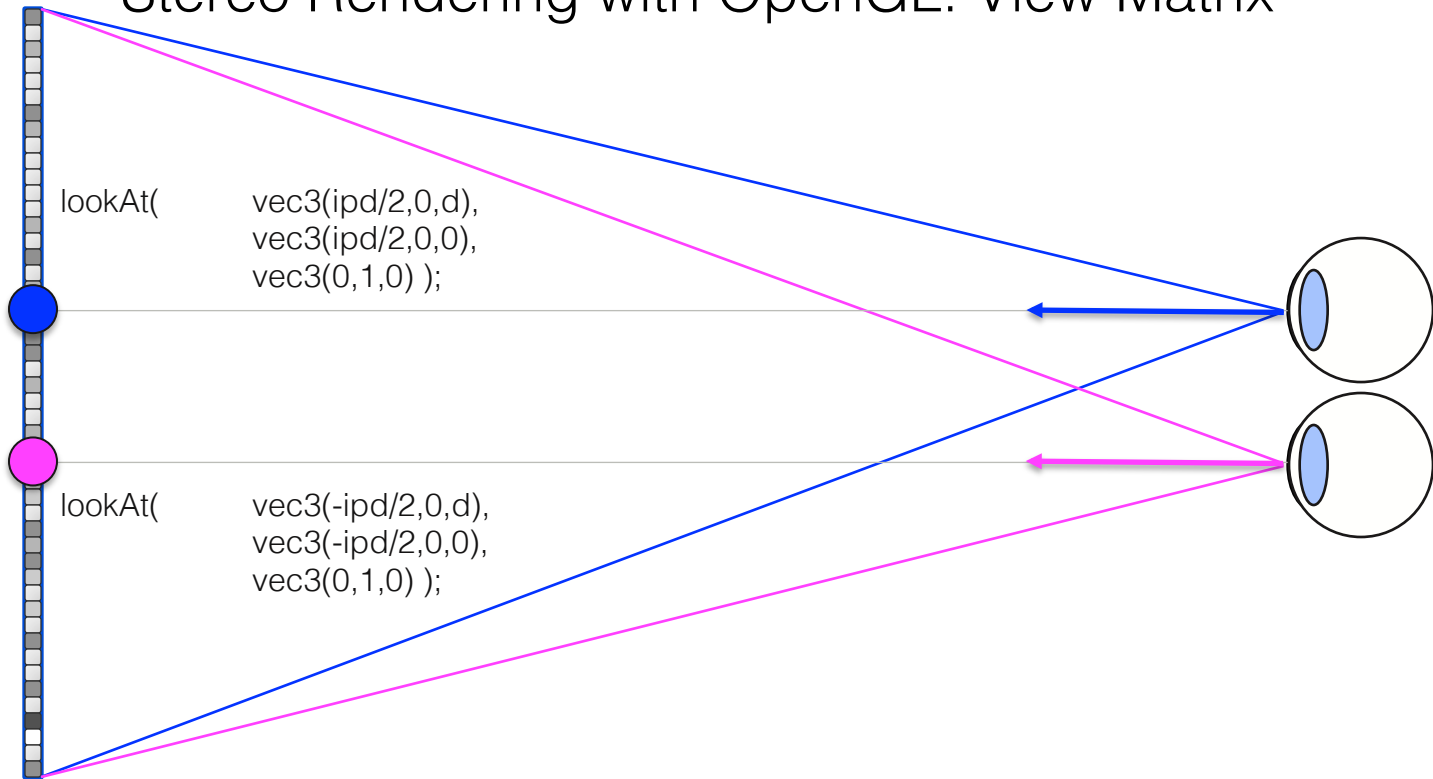
Stereo Rendering with OpenGL: Projection Matrix

- perspective projection we have discussed so far is on-axis=symmetric

- we need a different way to set up the asymmetric, off-axis frustum

- use `THREE.Matrix4().makePerspective(left,right,top,bottom,znear,zfar)`

On-axis Frustum

monitor

view frustum
__symmetric__

y

z

h

d

Side View

On-axis Frustum

monitor

view frustum
<u>symmetric</u>

near
clipping
plane

z_{near}

y

z

h

d

Side View

On-axis Frustum

monitor

view frustum
<u>symmetric</u>

near
clipping
plane

z_{near}

y

z

top

bottom

h

d

Side View

On-axis Frustum

monitor

view frustum
<u>symmetric</u>

near
clipping
plane

z_{near}

<u>similar triangles:</u>

$$top = z_{near} \frac{h}{2d}$$

top

bottom

$$bottom = -z_{near} \frac{h}{2d}$$

h

d

Side View

Off-axis Frustum

monitor

view frustum
asymmetric

near
clipping
plane

z_{near}

w

x

z

ipd/2

d

Top View

Off-axis Frustum

virtual image

view frustum
__asymmetric__

near
clipping
plane

z_{near}

x

z

w

right

ipd/2

left

d

Top View

Off-axis Frustum

virtual image

view frustum __asymmetric__

near clipping plane

z_{near}

similar triangles:

$$right = z_{near} \frac{w + ipd}{2d}$$

$$left = -z_{near} \frac{w - ipd}{2d}$$

w

right

left

ipd/2

d

Top View

Off-axis Frustum

virtual image

view frustum
asymmetric

similar triangles:

$$right = z_{near} \frac{w - ipd}{2d}$$

$$left = -z_{near} \frac{w + ipd}{2d}$$

z_{near}

right

left

w

d

Top View

Anaglyph with OpenGL

- most efficient way:
  1. clear color and depth buffer
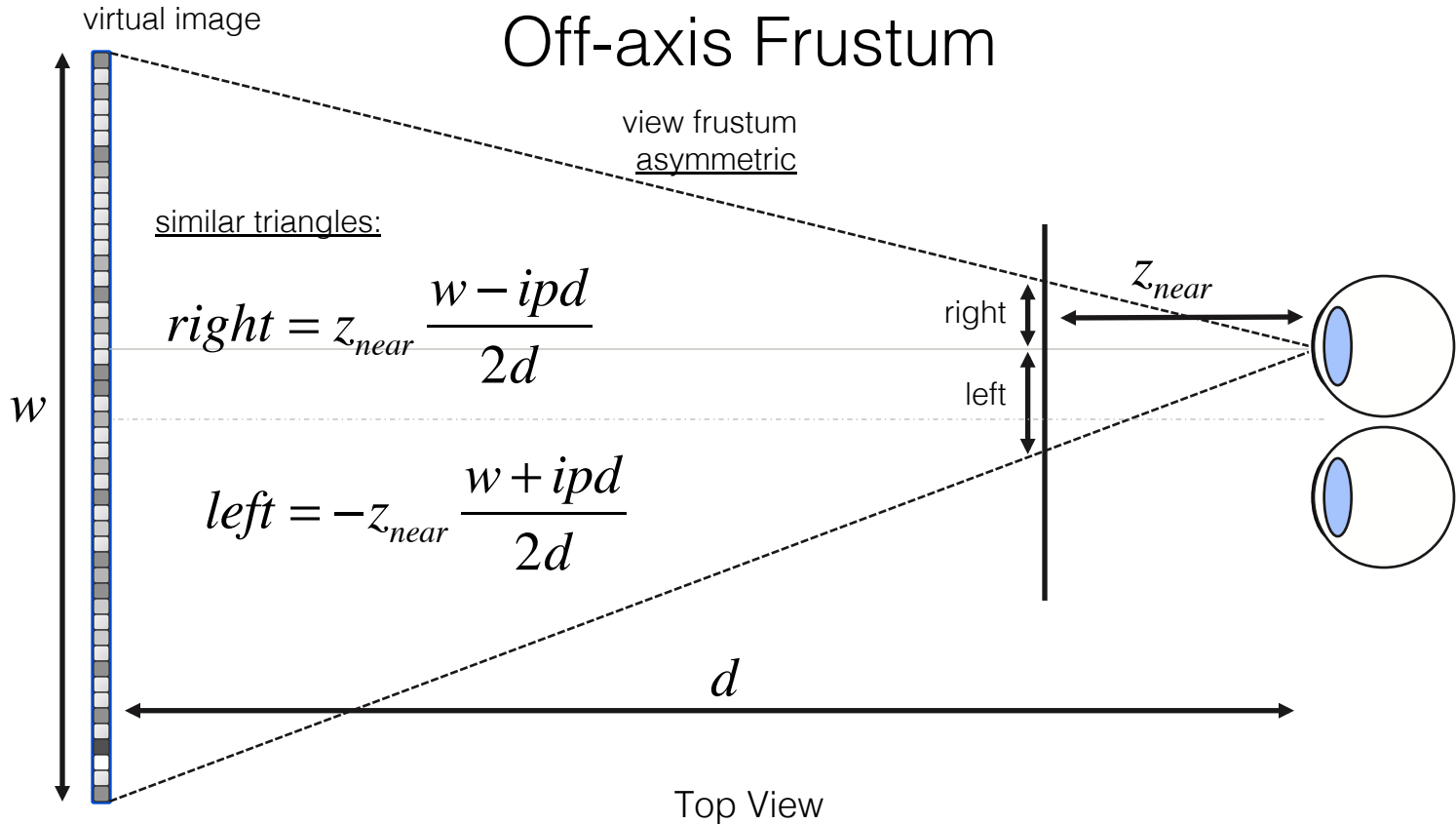 2. set left modelview and project matrix, render scene <u>only</u> into red channel
  3. clear depth buffer
  4. set right modelview and project matrix, render scene <u>only</u> into green & blue channels

- we'll do it in a slightly more complicated way (need for other tasks anyway):
  - multiple render passes
  - render into offscreen (frame) buffers
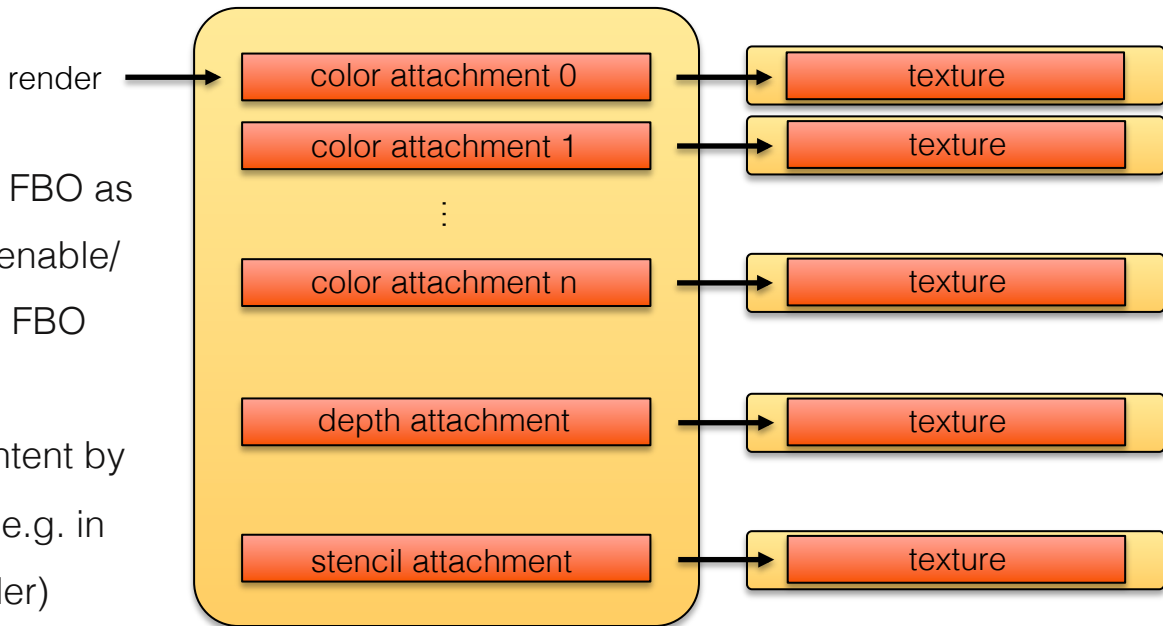
OpenGL Frame Buffers

- usually (frame) buffers are provided by the window manager (i.e., your browser)

- for most mono applications, two (double) buffers: back buffer and front buffer
  → render into back buffer; swap buffers when done (WebGL does this for you!)
- advantage: rendering takes time, you don't want the user to see how triangles get drawn onto the screen; only show final image

- in many stereo applications, 4 (quad) buffers: front/back left and right buffer
- render left and right images into back buffers, then swap both together

OpenGL Frame Buffers

- more generic model: offscreen buffer

- most common form of offscreen buffer in OpenGL: framebuffer object

- concept of "render-to-texture" but with multiple "attachments" for color, depth, and other important per-fragment information

- as many framebuffer objects as desired, they all "live" on the GPU (no memory transfer)

- bit depth per color: 8 bits, 16 bits, 32 bits for color attachments; 24 bits for depth

OpenGL Frame Buffers

FrameBuffer Object (FBO)

- render into FBO as usual, just enable/disable the FBO

- access content by texture ID (e.g. in GLSL shader)

OpenGL Frame Buffers

- FBOs are crucial for multiple render passes!

- 1st pass: render color and depth into FBO
- 2nd pass: render textured rectangle – access FBO in fragment shader

- we'll provide a simple-to-use interface that shields you from the details of FBOs

- in JavaScript FBOs are wrapped by *WebGLRenderTarget* in Three.js

- more details in lab / homework starter code on Friday …

Anaglyph Rendering with OpenGL & GLSL

1. activate FBO1
2. set *left* modelview & projection matrix
3. render scene
4. deactivate FBO1

render pass 1

5. activate FBO2
6. set *right* modelview & projection matrix
7. render scene
8. deactivate FBO2

render pass 2

9. render rectangle, pass FBO1 and FBO2 into fragment shader as textures
10. merge stereo images in fragment shader

render pass 3

Retinal Blur / Depth of Field Rendering
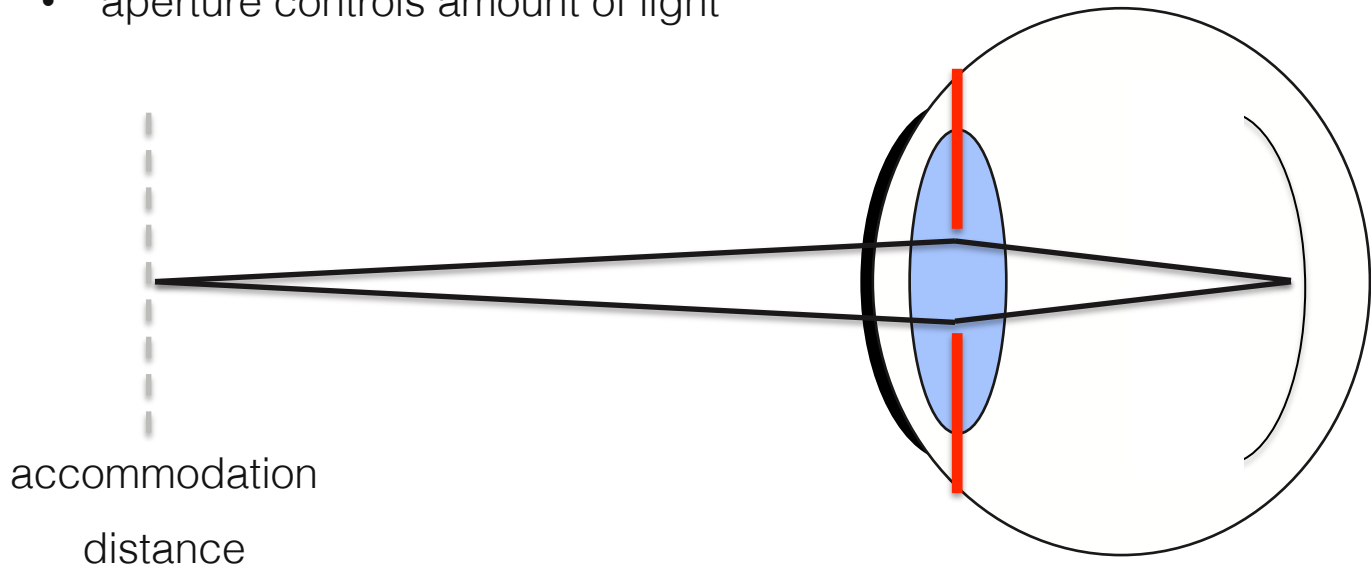
- aperture controls amount of light
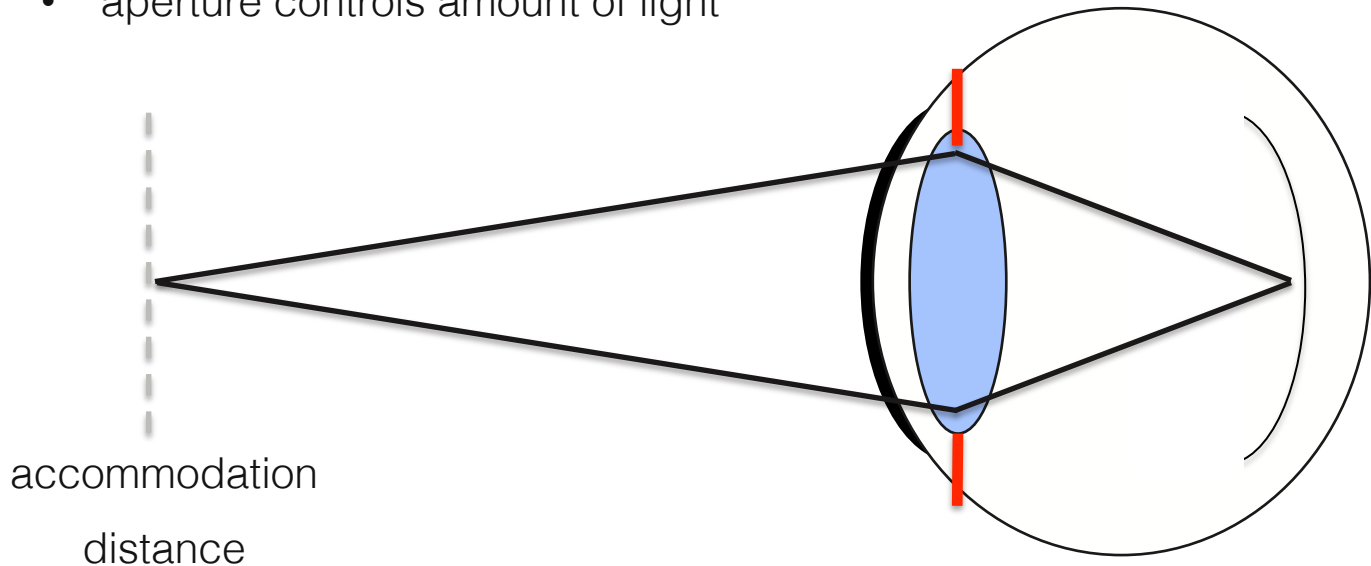

accommodation

distance

Retinal Blur / Depth of Field Rendering

- aperture controls amount of light

accommodation

distance

Retinal Blur / Depth of Field Rendering

- out of focus blur

accommodation

distance
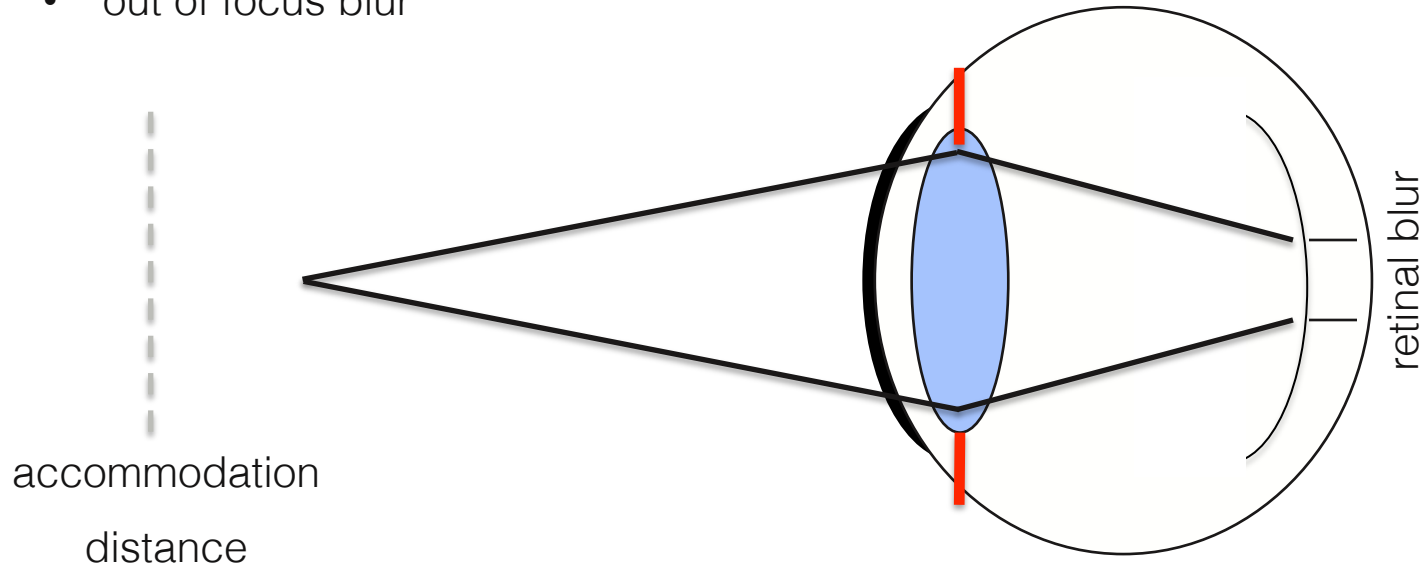
retinal blur

Retinal Blur / Depth of Field Rendering

- out of focus blur
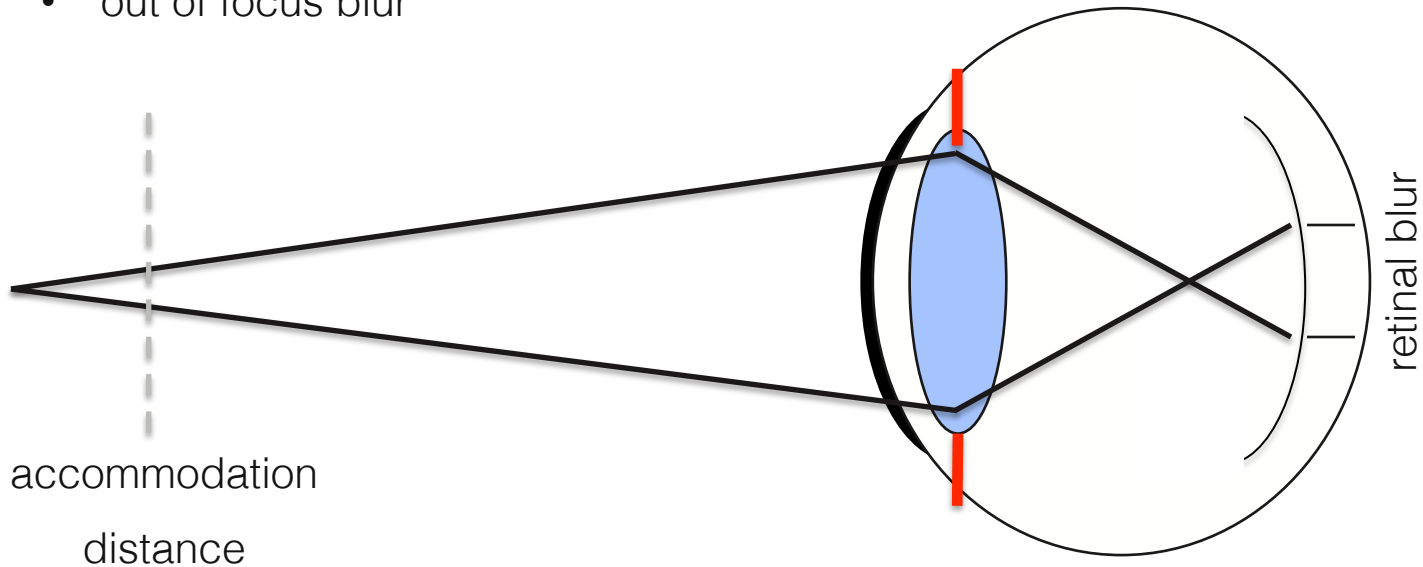
accommodation

distance

retinal blur

Retinal Blur / Depth of Field Rendering

$$c = M \cdot D \cdot \frac{|S - S_1|}{S}$$

$$M = \frac{f}{S_1 - f}$$

$$f = 17\, mm$$

D

retinal blur diameter c

accommodation distance: S_1

S
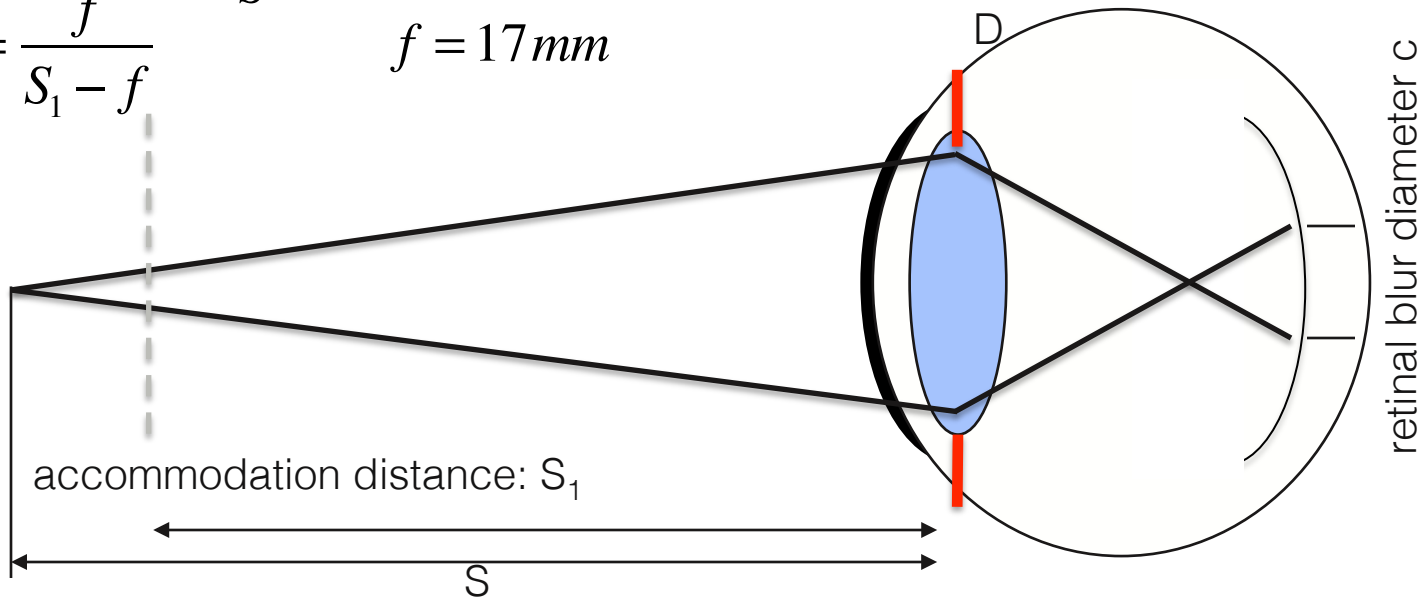
Circle of Confusion

$$c = M \cdot D \cdot \frac{|S - S_1|}{S}$$
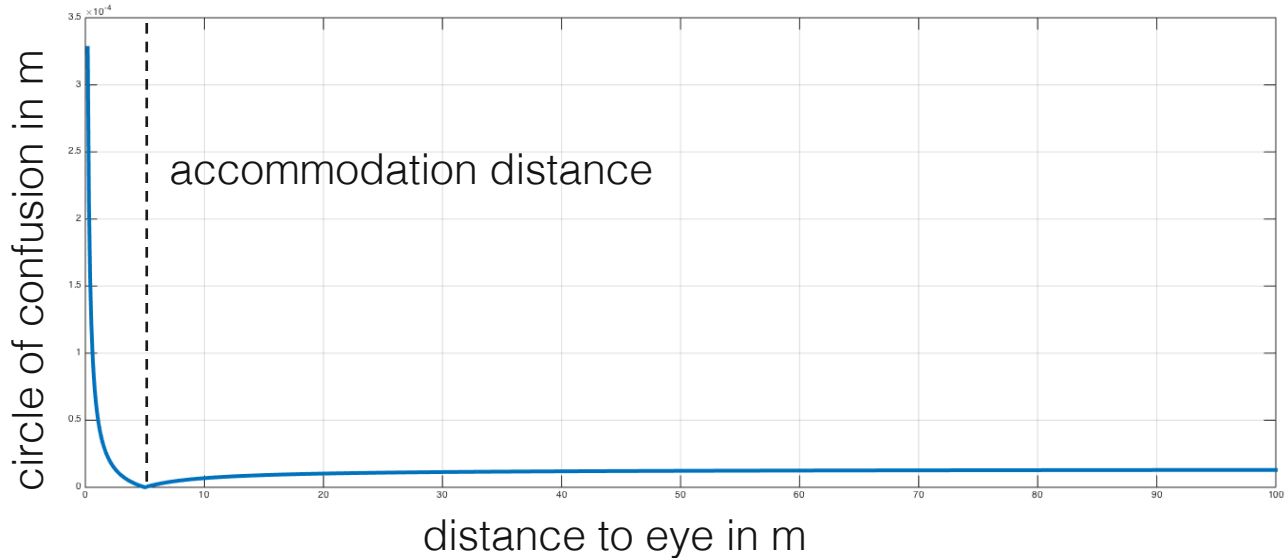
Retinal Blur / Depth of Field Rendering

depth of field blur on screen
(in mm) via similar triangles:

$$b = \frac{|S - S_1| \, D}{S}$$

b

D

accommodation dist = dist to screen: S_1

S

Depth of Field with OpenGL/GLSL

- two rendering passes:
  1. render image and depth map into FBO
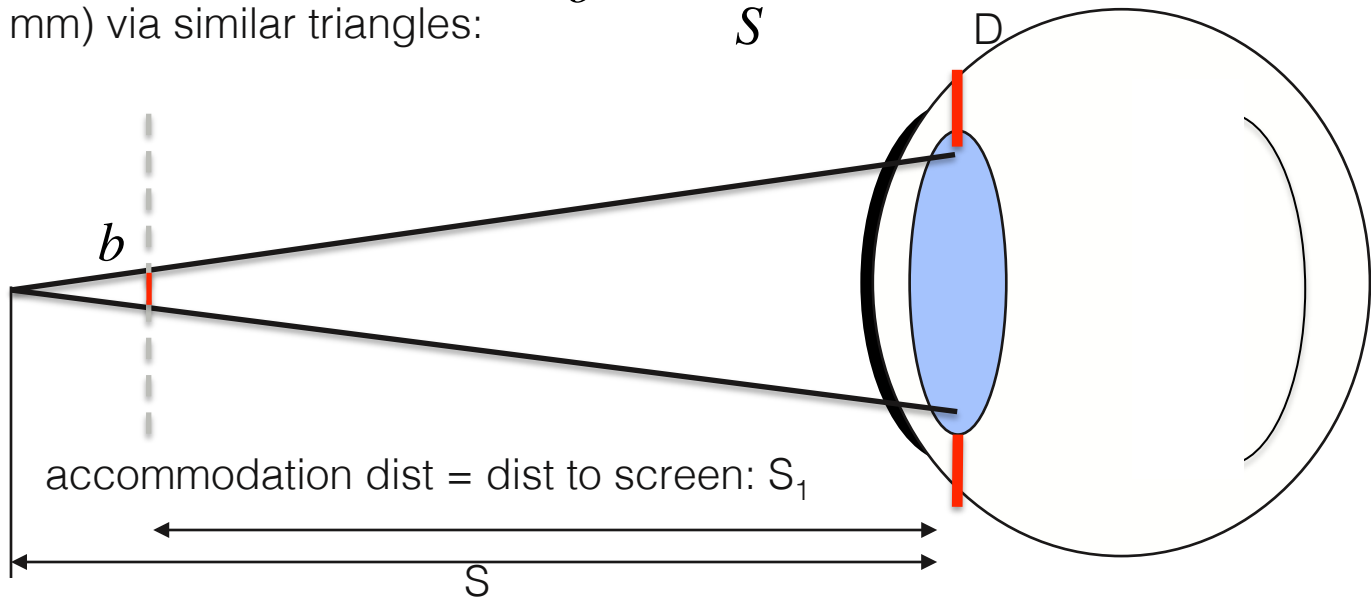 2. render quad textured with image + depth
     - vertex shader is pass-through (just transforms, pass on texture coordinates, no lighting)
     - in fragment shader:
       - calculate depth for each fragment in mm (given in clip coords)
       - calculate retinal blur size in pixels given depth & pupil diameter
       - apply blur via convolution with double for loop over neighboring color values in the texture

Depth of Field with OpenGL/GLSL

- how to get metric depth of a fragment?

- in fragment shader we provide depth map `z` as uniform texture in window coordinates (range [0,1]) along with x,y fragment position in window coordinates

- need to convert x,y,z_{window} to view/camera coordinates x,y,z_{view} and then calculate distance as $$dist = \sqrt{x_{view}^2 + y_{view}^2 + z_{view}^2}$$

How to get Metric Depth of Fragment

1. convert window coordinates to clip coordinates (see course notes on graphics pipeline for derivation)

$$M_{proj} = \begin{pmatrix} \dfrac{2n}{r-l} & 0 & \dfrac{r+l}{r-l} & 0 \\[2ex] 0 & \dfrac{2n}{t-b} & \dfrac{t+b}{t-b} & 0 \\[2ex] 0 & 0 & -\dfrac{f+n}{f-n} & \dfrac{-2 \cdot f \cdot n}{f-n} \\[2ex] 0 & 0 & -1 & 0 \end{pmatrix}$$

$$z_{clip} = -\frac{f+n}{f-n} z_{view} - \frac{2fn}{f-n}$$

$$w_{clip} = -z_{view}$$

$$\boxed{z_{view} = \frac{2fn}{f-n} \cdot \frac{1}{z_{NDC} - \dfrac{f+n}{f-n}}}$$

$$\begin{pmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \\ 1 \end{pmatrix} = \begin{pmatrix} x_{clip} / w_{clip} \\ y_{clip} / w_{clip} \\ z_{clip} / w_{clip} \\ 1 \end{pmatrix}$$

$$\underbrace{\begin{pmatrix} x_{NDC} \\ y_{NDC} \\ z_{NDC} \\ 1 \end{pmatrix}}_{v_{NDC}}$$

$$\underbrace{\begin{pmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{pmatrix}}_{v_{clip}} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot \underbrace{\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}}_{v} = M_{proj} \cdot \underbrace{\begin{pmatrix} x_{view} \\ y_{view} \\ \boxed{z_{view}} \\ 1 \end{pmatrix}}_{v_{view}}$$

How to get Metric Depth of Fragment

1. convert window coordinates to clip coordinates (see course notes on graphics pipeline for derivation)

$$M_{proj} = \begin{pmatrix} \dfrac{2n}{r-l} & 0 & \dfrac{r+l}{r-l} & 0 \\[2mm] 0 & \dfrac{2n}{t-b} & \dfrac{t+b}{t-b} & 0 \\[2mm] 0 & 0 & -\dfrac{f+n}{f-n} & \dfrac{-2 \cdot f \cdot n}{f-n} \\[2mm] 0 & 0 & -1 & 0 \end{pmatrix}$$

$$x_{clip} = \frac{2n}{r-l}x_{view} + \frac{r+l}{r-l}z_{view}$$

$$y_{clip} = \frac{2n}{t-b}y_{view} + \frac{t+b}{t-b}z_{view}$$

$$x_{view} = \frac{x_{clip} - \dfrac{r+l}{r-l}z_{view}}{\dfrac{2n}{r-l}}, \qquad y_{view} = \frac{y_{clip} - \dfrac{t+b}{t-b}z_{view}}{\dfrac{2n}{t-b}}$$

How to get Metric Depth of Fragment

2. now compute distance (see course notes on graphics pipeline for derivation)

$$dist = \sqrt{x_{view}^2 + y_{view}^2 + z_{view}^2}$$
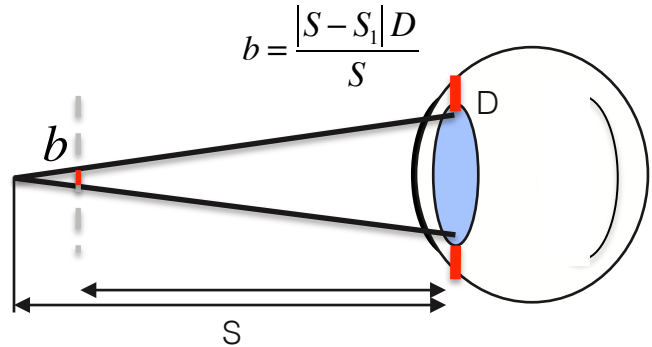
Depth of Field with OpenGL/GLSL

- <u>how to compute retinal blur size and convert to pixels</u> ?

$$pixel\_size_{x/y} = \frac{screen\_size_{x/y}}{screen\_resolution_{x/y}}$$

$$blur\_diameter\_px = \frac{b}{pixel\_size}$$

$$b = \frac{|S - S_1| D}{S}$$

screen_size is *either* screen width *or* height (same units as other distances)

screen_resolution is *either* number of horizontal pixels *or* vertical pixels of the screen

Depth of Field with OpenGL/GLSL

1. activate FBO

2. set modelview & projection matrix

3. render 3D scene

4. deactivate FBO

render pass 1

5. render rectangle, pass FBO with image & depth map into
   fragment shader as textures

6. execute depth of field fragment shader

render pass 2

Depth of Field with OpenGL/GLSL

- <u>putting it all together</u> – this is just a general overview, do not use this exact code

```
uniform sampler2D image;    // RGB image was written in the first rendering pass
uniform sampler2D depthMap; // depth map was written in the first rendering pass
uniform float    znear;
uniform float    zfar;
uniform float    pupilDiameter;
varying vec2     textureCoords;

void main () // fragment shader
{
  // get fragment z in NDC
  float zNDC = 2*sampler2D( depthMap, textureCoords ).r – 1;

  // get z in view coordinates (metric depth of current fragment)
  float distanceToFragement = …

  // compute retinal blur radius in pixels
  float blurRadius = …
  int blurRadiusInt = round(blurRadius);

  // set output color by averaging neighboring pixels in the color image (i.e., convolution)
  gl_FragColor.rgb = 0;
  for (int i=–blurRadiusInt; i<blurRadiusInt; i++)
    for (int j=–blurRadiusInt; j<blurRadiusInt; j++)
      if (float(i*i+j*j) <= blurRadius*blurRadius)
        gl_FragColor.rgb += … texture lookup in neighboring pixels

  // normalize color
  …
}
```

Summary

- many different technologies for glasses-based stereo
- we'll work with anaglyph for this lab + homework
- color management is important for anaglyph
- getting the view and projection matrices right is important (otherwise headaches)

- may need multiple render passes (all wrapped in the starter code)

- depth of field rendering may add more realism

Next Lecture: HMD Optics and Microdisplays

- magnifiers
- VR & AR optics
- microdisplays
- stereo rendering for HMDs
- lens distortion / undistortion
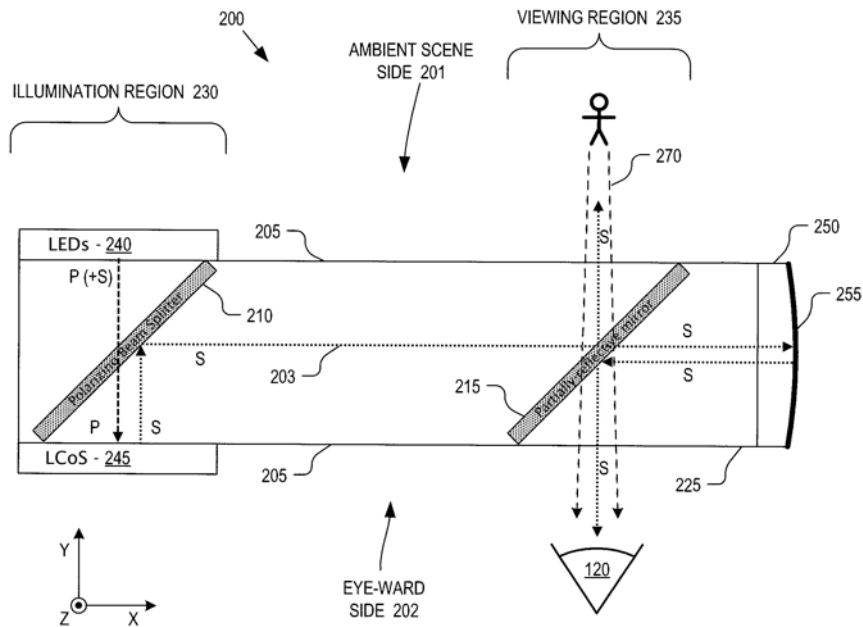
drawing from Google Glass patent

Further Reading

- http://paulbourke.net/stereographics/stereorender/

- Eric Dubois, "A Projection Method to Generate Anaglyph Stereo Images", ICASSP 2001

- Library of Congress, Stereoscopic Cards:
    http://www.loc.gov/pictures/search/?st=grid&co=stereo