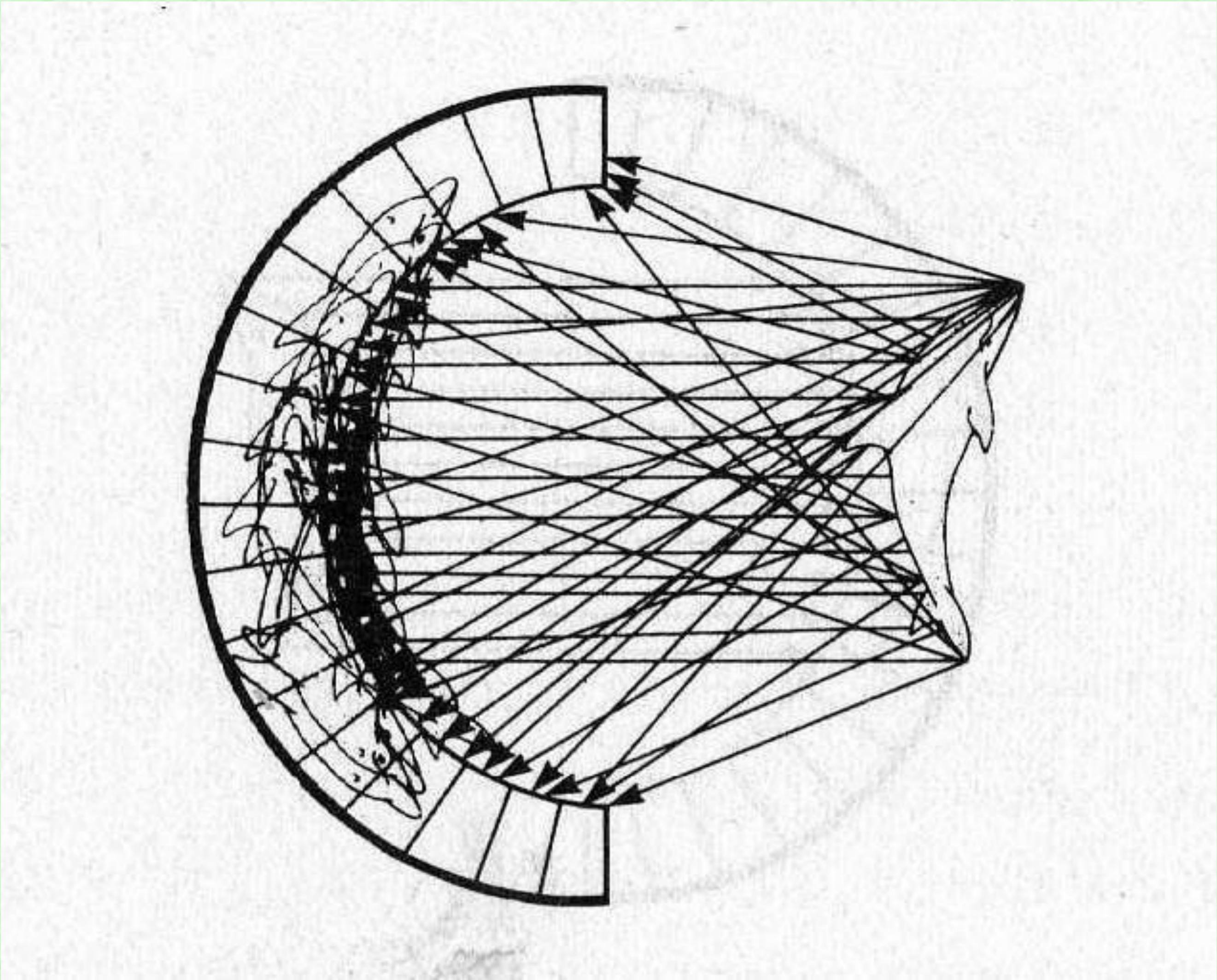
The Virtual World

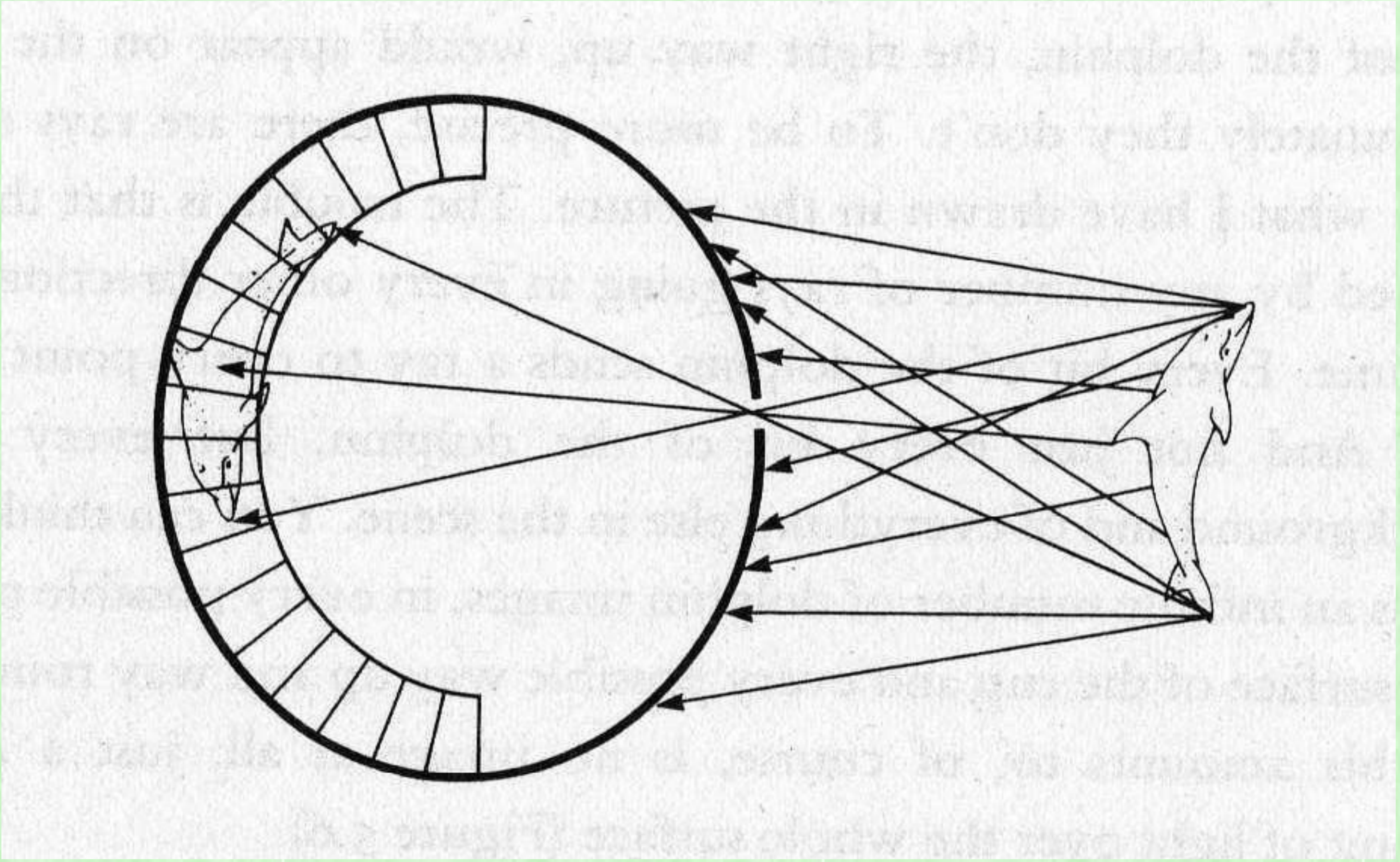
Building a Virtual World

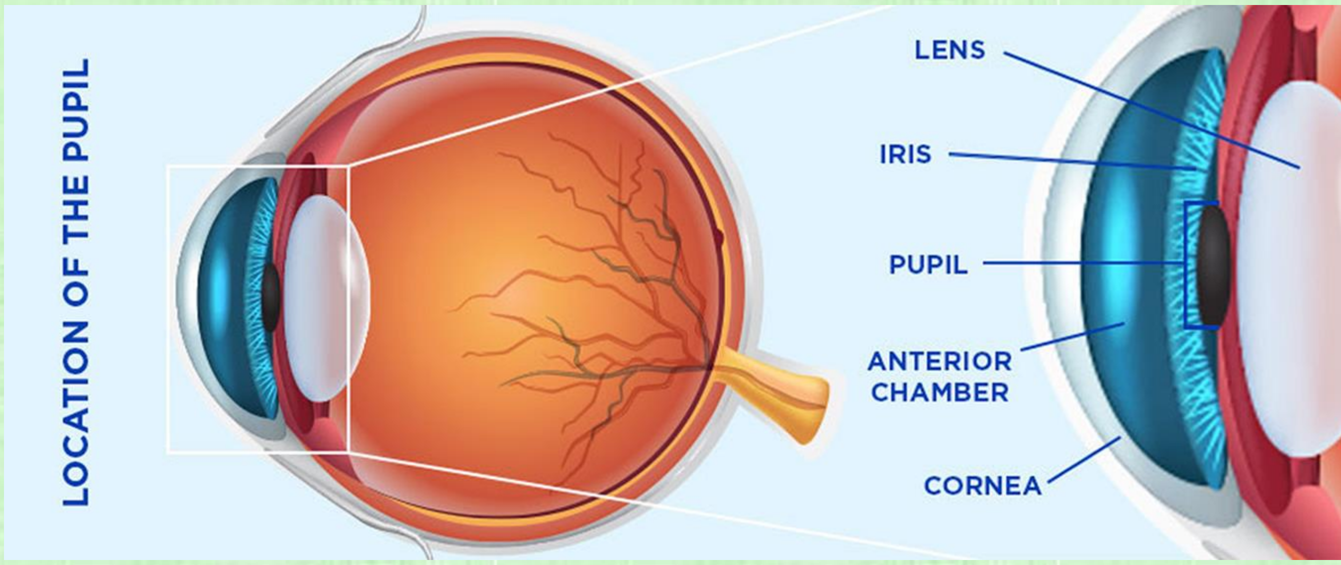
- Goal: mimic human vision in a virtual world (with a computer)
  - Cheat for efficiency, using knowledge about light and the human eye/perception (e.g. from the last lecture)
- Create a virtual camera: place it somewhere and point it at something
- Put film (containing pixels, each with RGB values ranging from 0-255) into the camera
- Place objects into the world, including a floor/ground, walls, ceiling/sky, etc.
  - Two step process: (1) make objects, (2) place objects (transformations)
  - Making objects is itself a two-step process: (1) build geometry (geometric modeling), (2) paint geometry (texture mapping)
- Put lights into the scene (so that it's not completely dark)
- Finally, <u>snap the picture</u>:
  - "Code" emits light from (virtual) light sources, bounces that light off of (virtual) geometry, and follows that bounced light into the (virtual) camera and onto the (virtual) film
  - Taking a picture creates film data as the final image
  - We will consider 2 methods (<u>scanline rendering</u> and <u>ray tracing</u>) for the taking this picture

Pupil

- Light emanates off of every point of an object outwards in every direction
  - That's why we can all see the same spot on the same object
  - Light leaving that spot/point (on the object) is entering each of our eyes
- Without a pupil, light from every point on an object would hit the same cone on our eye, averaging/blurring the light information
- The (small) pupil restricts the entry of light so that each cone only receives light from a small region on the object, giving interpretable spatial detail

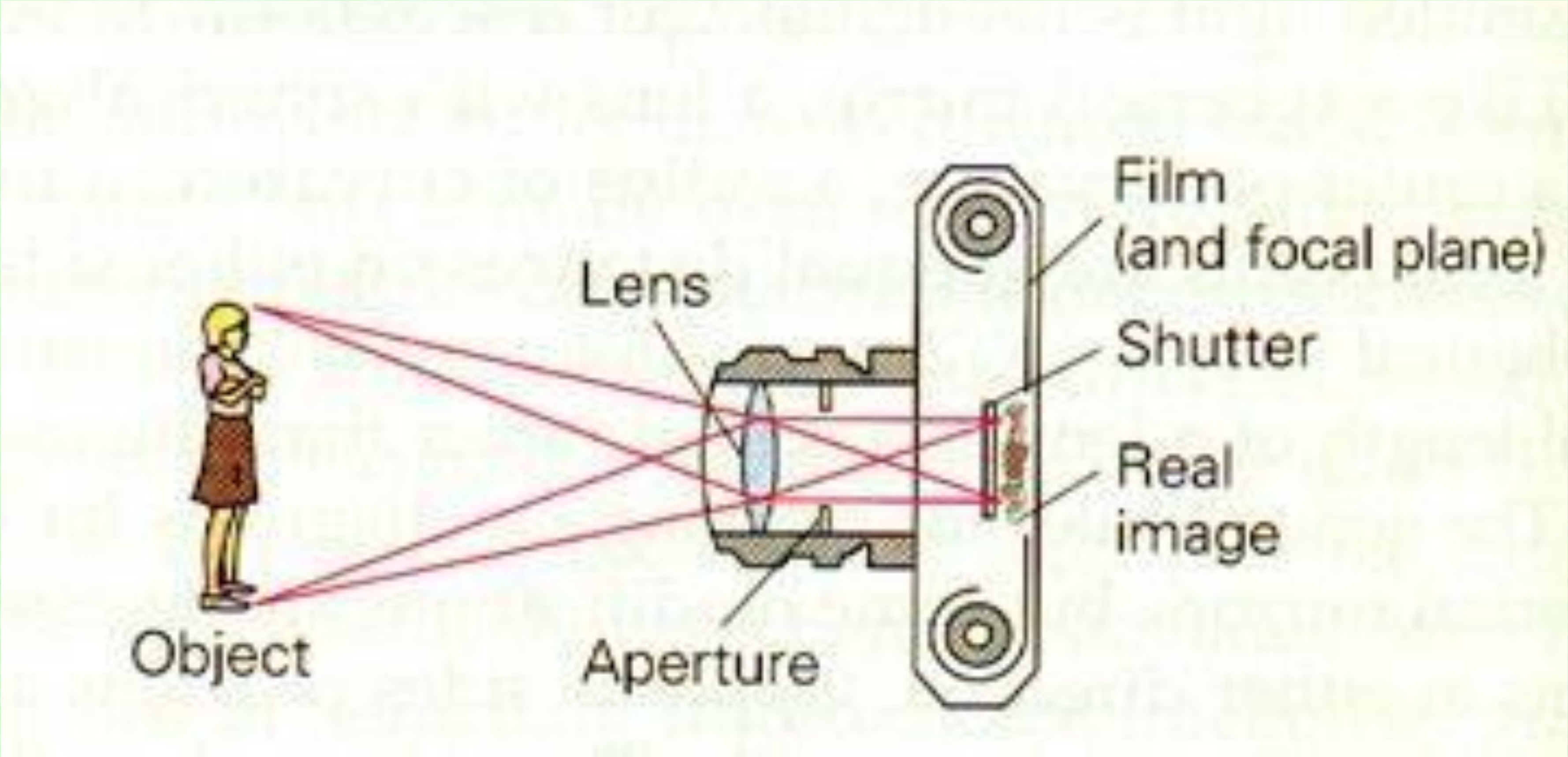


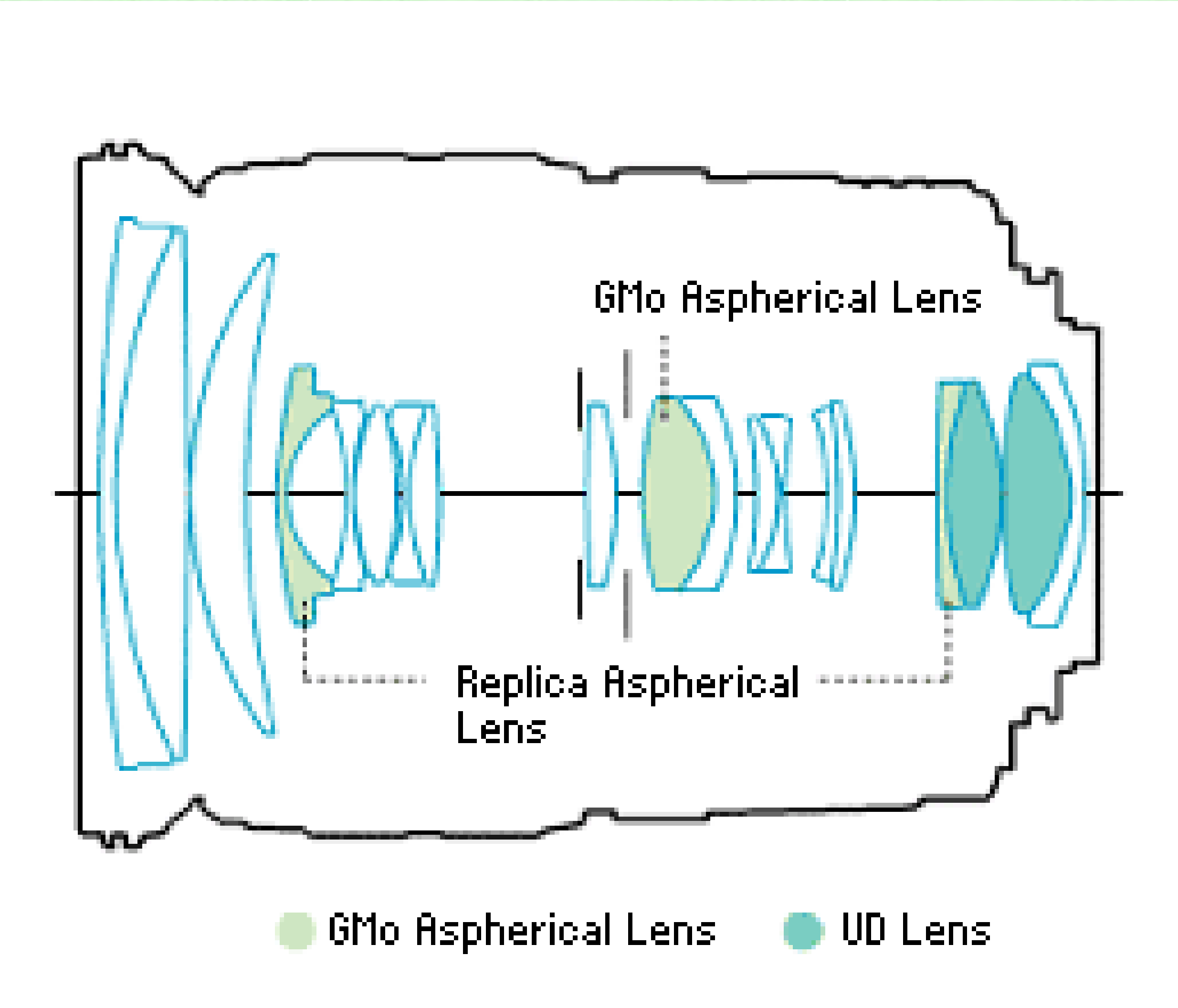




Aperture

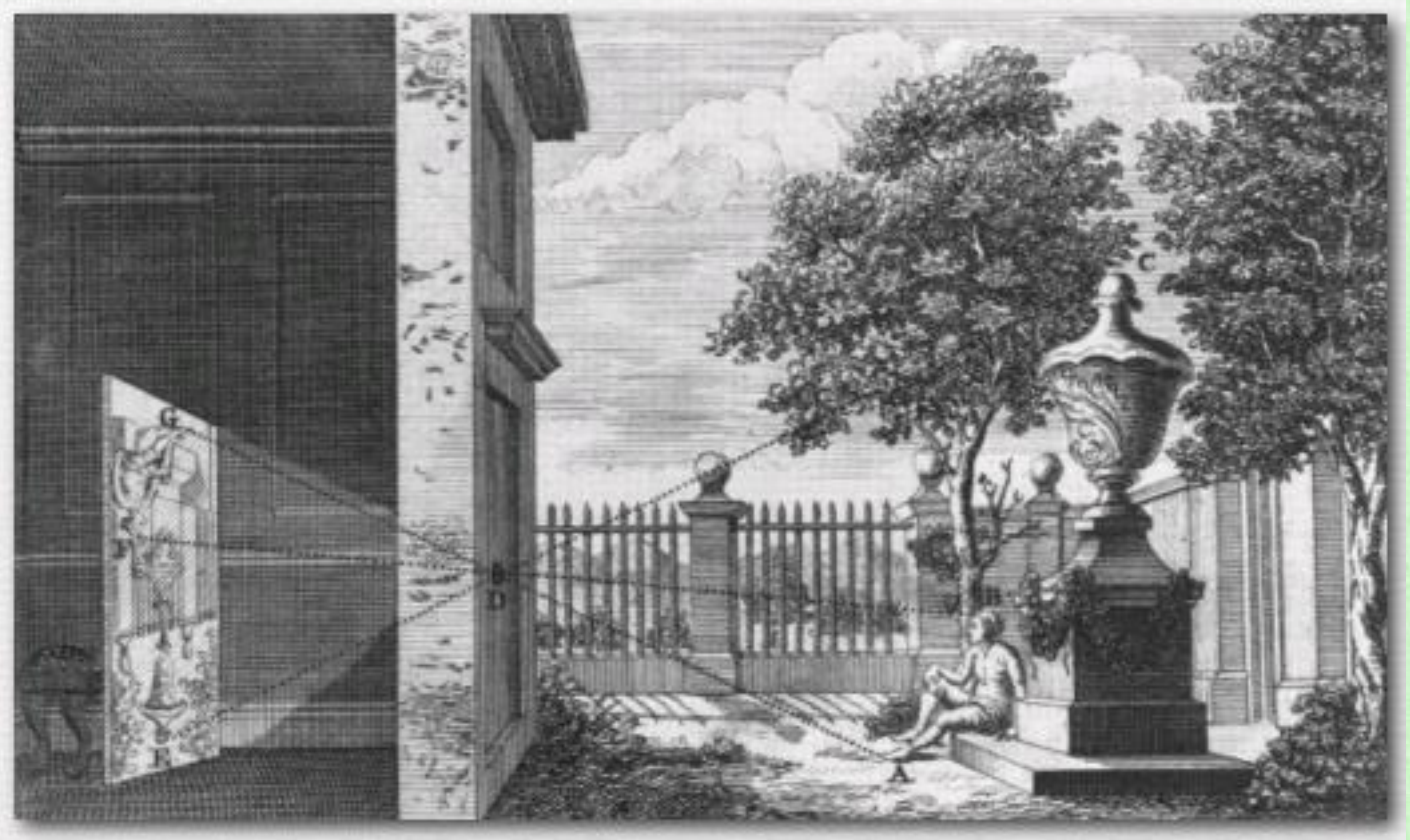
- Cameras are similar to the eye (with mechanical as opposed to biological components)
- Instead of cones, the camera has mechanical pixels
- Instead of a pupil, the camera has a small (adjustable) aperture for light to pass through
- Cameras also typically have a hefty/complex lens system





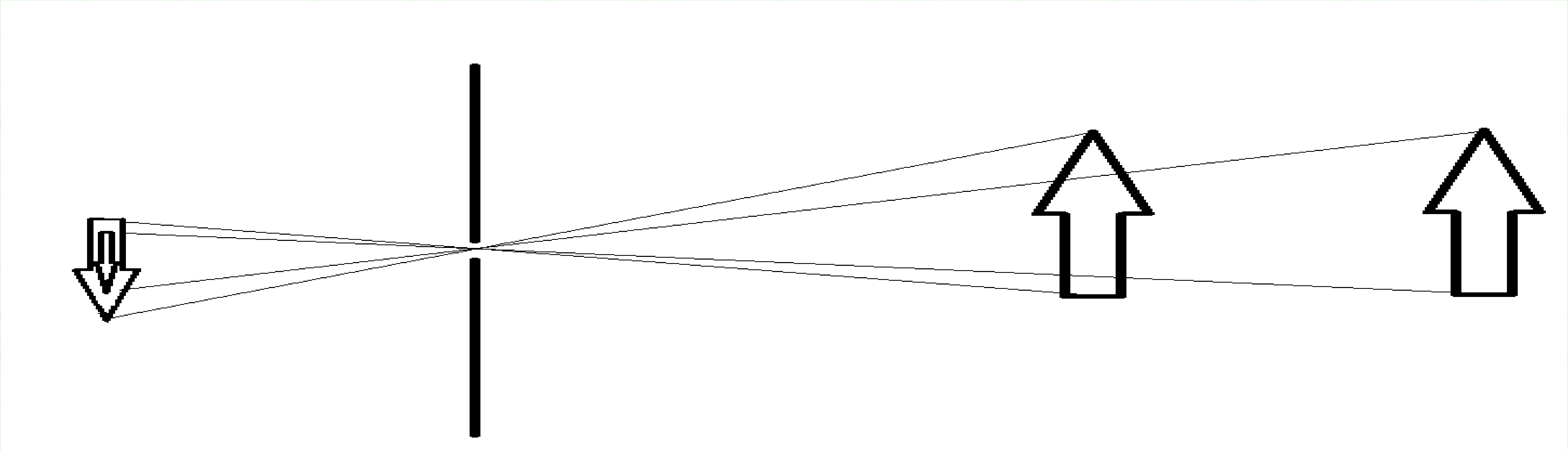
Pinhole Camera

- The pupil/aperture has to have a finite size in order for light to be able to pass through it
- When too small, not enough light enters and the image is too dark/noisy to interpret
  - In addition, light can diffract (instead of traveling in straight lines) distorting the image
- When too large, light from a large area of an object hits the same cone (causing blurring)
- Luckily, a virtual camera can use a single point for the aperture (without worrying about dark or distorted images)



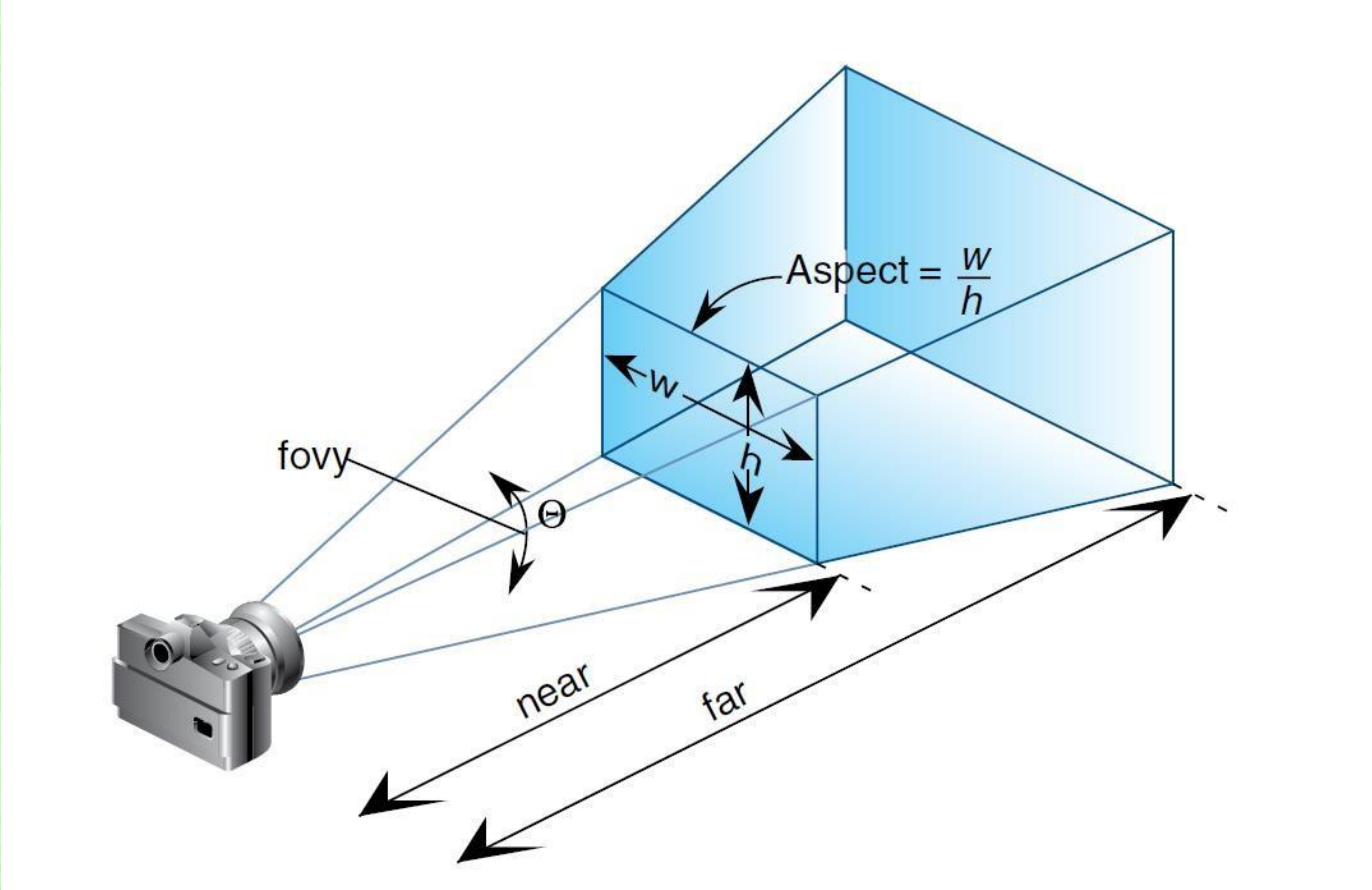
Pinhole Camera (a theoretical approximation)

- Light leaving any point travels in straight lines
- We only care about the lines that hit the pinhole (a single point)
  - Using a single point gives infinite depth of field (everything is in focus, no blurring)
- An upside-down image is formed by the intersection of these lines with an image plane
- More distant objects subtend smaller visual angles and appear smaller
- Objects occlude objects behind them



Virtual Camera

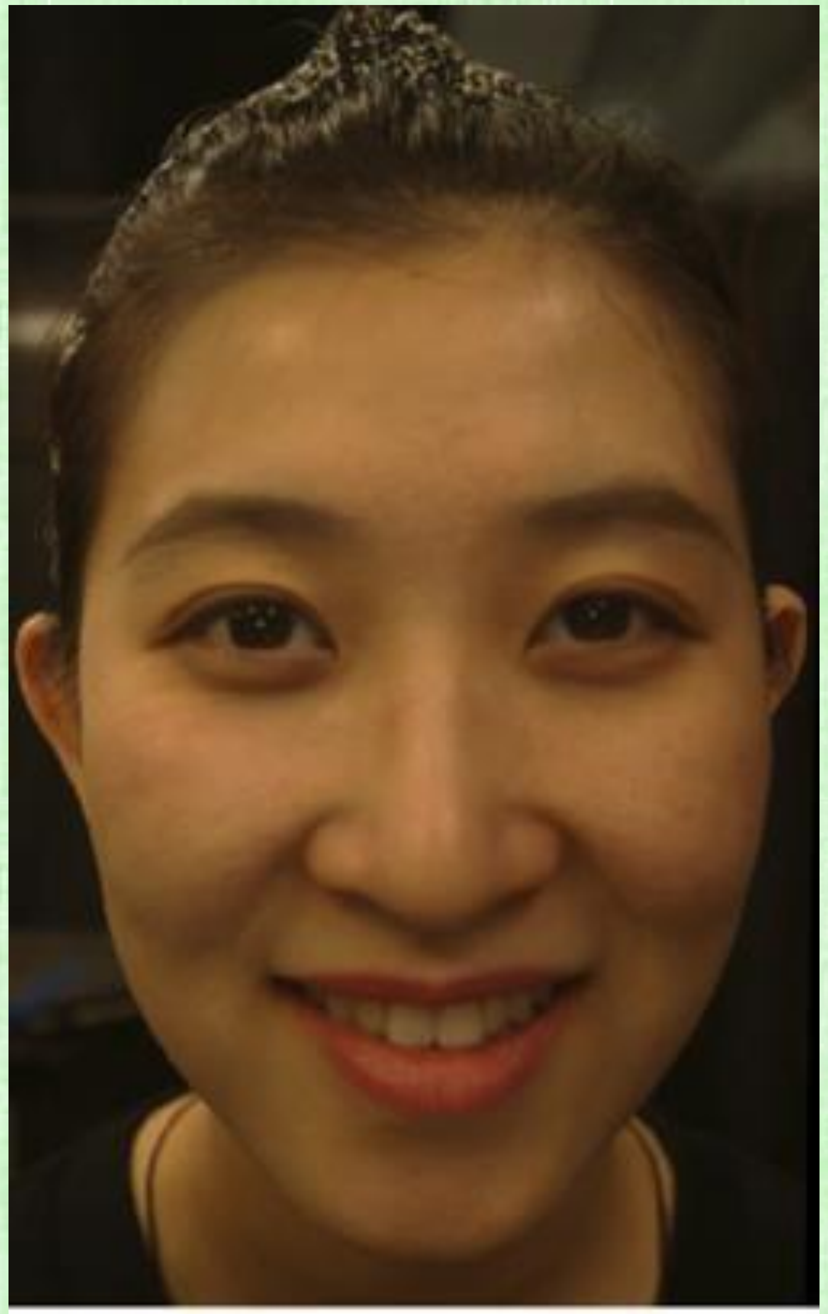
- Trick: Move the film out in front of the pinhole, so that the image is not upside down
- Only render (compute an image for) objects further away from the camera than the film plane
- Add a back clipping plane for efficiency
- The volume between the film (front clipping plane) and the back clipping plane is called the viewing frustum (shown in blue)
  - Make sure that the near/far clipping planes have enough space between them to contain the scene
  - Make sure objects are inside the viewing frustum
  - Do not set the near clipping plane to be at the camera aperture!



Camera Distortion depends on Distance

- Do not put the camera too close to objects of interest!
  - Significant/severe deductions for poor camera placement, fisheye, etc. (because the distortion looks terrible)
- Set up the scene like a real-world scene!
- Get very familiar with the virtual camera!

@160CM



@25CM

ACTIVITY: modeling and drawing a cube

- Goal: generate a realistic drawing of a cube
- Key questions:

  - *Modeling:* how do we describe the cube?

  – *Rendering:* how do we then visualize this model?

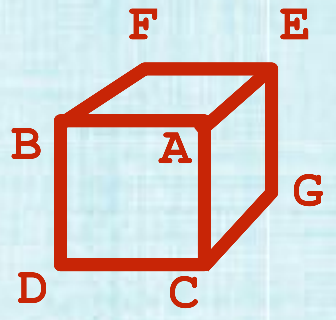
ACTIVITY: modeling the cube

- Suppose our cube is...
  - centered at the origin (0,0,0)
  - has dimensions 2 x 2 x 2
- QUESTION: What are the coordinates of the cube vertices?

```
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )
D: (-1,-1, 1 )    H: (-1,-1,-1 )
```

- QUESTION: What about the edges?

```
AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```



ACTIVITY: drawing the cube

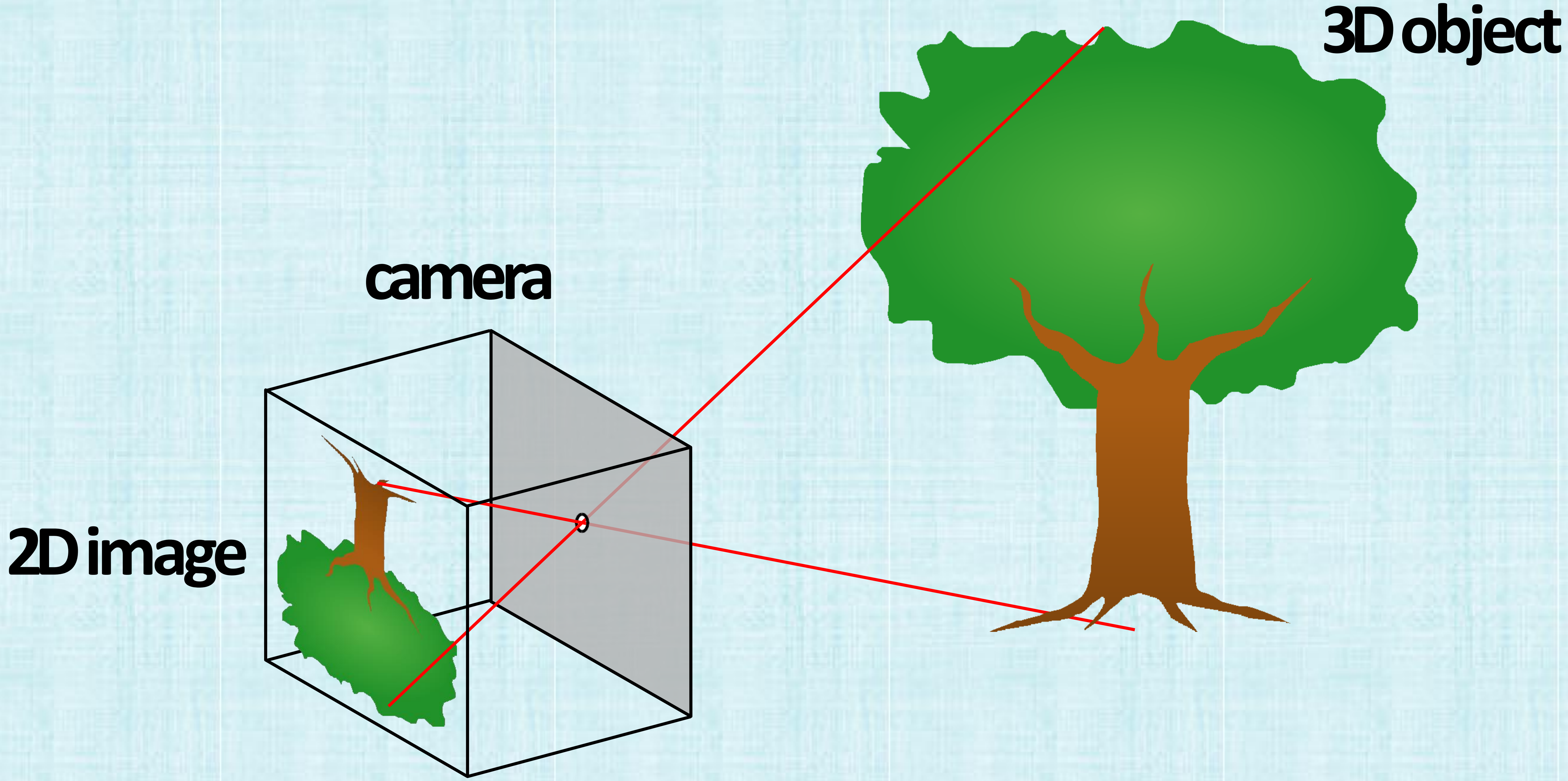
- We now have a digital description of the geometry of the cube:

```
VERTICES                                      EDGES
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )        AB, CD, EF, GH,
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )        AC, BD, EG, FH,
D: (-1,-1, 1 )    H: (-1,-1,-1 )        AE, CG, BF, DH
```

- How do we draw this 3D cube as a 2D (flat) image?

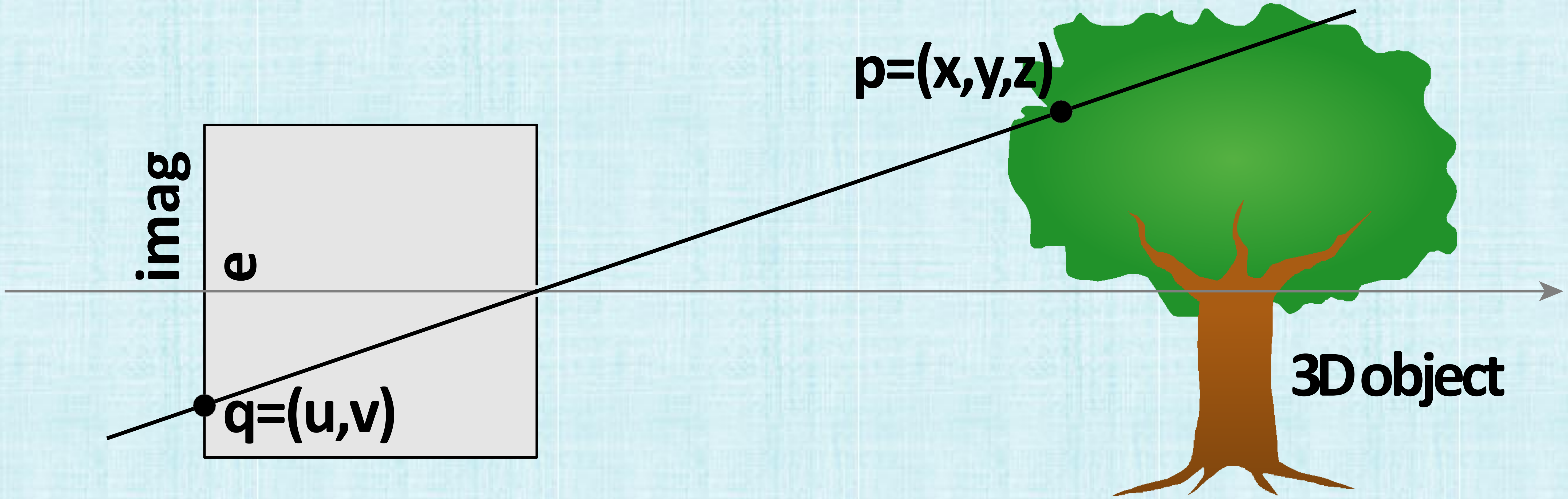
Perspective projection

- Objects look smaller as they get further away ("perspective")
- Why does this happen?
- Consider simple ("pinhole") model of a camera:



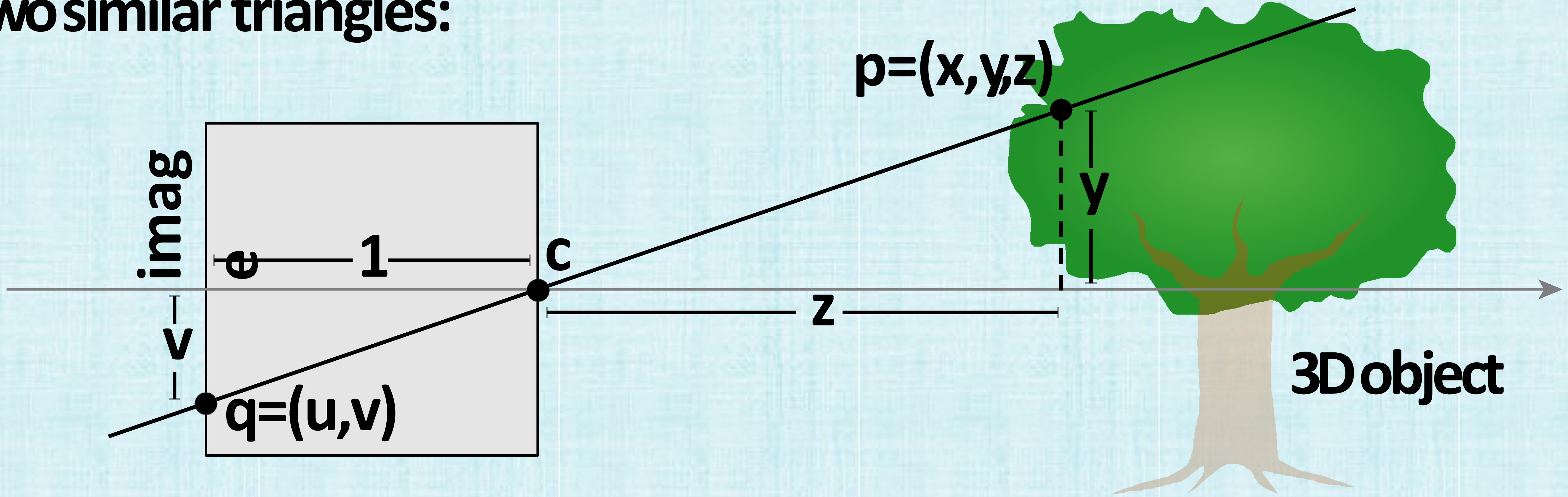
Perspective projection: side view

- Where exactly does a point p = (x,y,z) on the tree end up on the image?
- Let's call the image point q=(u,v)



Perspective projection: side view

- Where exactly does a point p = (x,y,z) on the tree end up on the image?
- Let's call the image point q=(u,v)
- Notice two similar triangles:

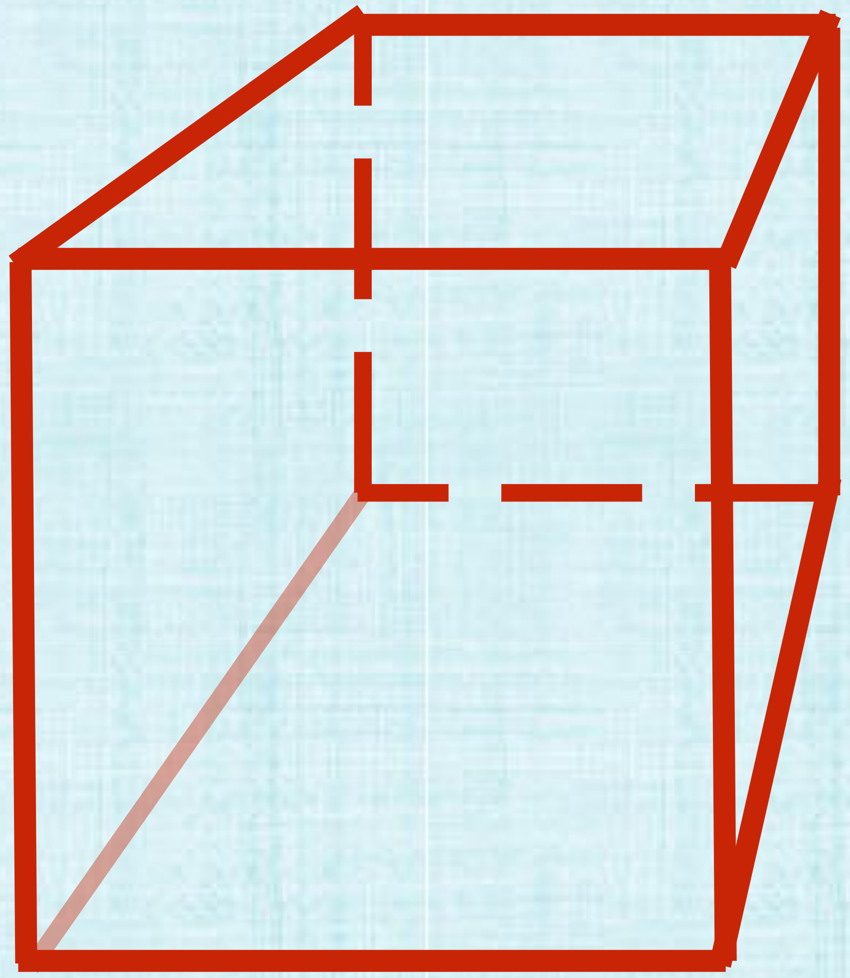


- Assume camera has unit size, coordinates relative to pinhole c
- Then v/1 = y/z… v = y/z
- Likewise, horizontal offset u = x/z

Can you visualize what it should look like?

- Consider a cube with these vertices:

```
VERTICES
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )
D: (-1,-1, 1 )    H: (-1,-1,-1 )

EDGES

AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```



Self-check

- Now imagine a camera positioned at (2,3,5) looking at the cube… can you picture what it should look like?

ACTIVITY: draw image made by pinhole camera

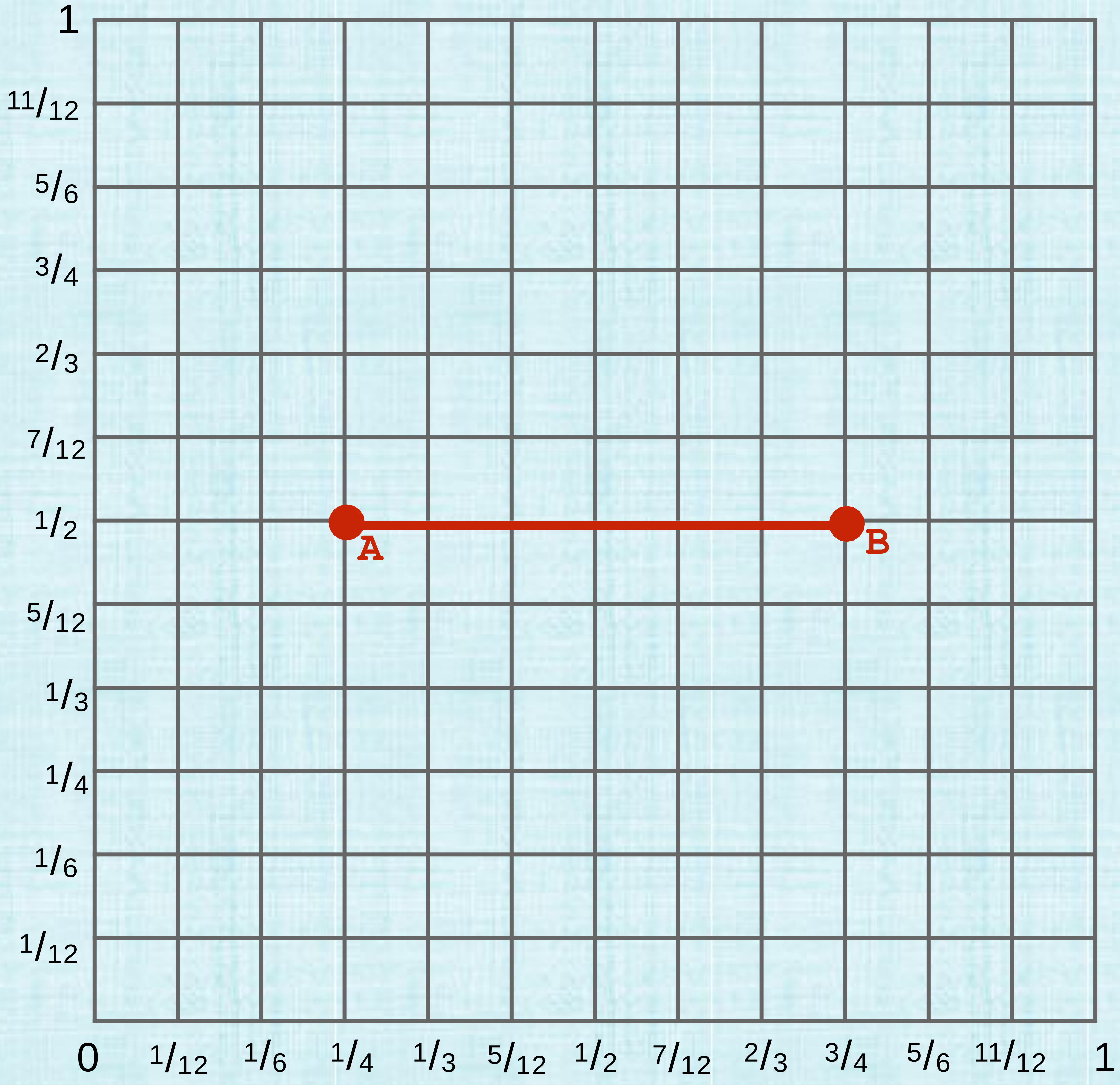
- Pick two vertices that share an edge and do it yourself!
  - Let's assume camera is at point c=(2,3,5)
  - Convert (X,Y,Z) of both endpoints of cube edge to screen point (u,v):
    1. Subtract camera point c from vertex (X,Y,Z) to get (x,y,z)
    2. Divide x and y by z to get (u,v)—*write as a fraction*
  - Then draw a line between (u1,v1) and (u2,v2) for all edges

Vertex position in absolute world coordinates

Vertex position relative to camera

```
VERTICES                                  EDGES
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )          AB, CD, EF, GH,
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )          AC, BD, EG, FH,
D: (-1,-1, 1 )    H: (-1,-1,-1 )          AE, CG, BF, DH
```

Render a cube!



- Assume camera is at point c=(2,3,5)
- Convert (X,Y,Z) of both endpoints of edge to (u,v):
  1. Subtract camera c from vertex (X,Y,Z) to get (x,y,z)
  2. Divide x and y by z to get (u,v)
- Draw line between (u1,v1) and (u2,v2)

VERTICES

```
A: ( 1, 1, 1 )    E: ( 1, 1,-1 )
B: (-1, 1, 1 )    F: (-1, 1,-1 )
C: ( 1,-1, 1 )    G: ( 1,-1,-1 )
D: (-1,-1, 1 )    H: (-1,-1,-1 )
```

EDGES

```
AB, CD, EF, GH,
AC, BD, EG, FH,
AE, CG, BF, DH
```

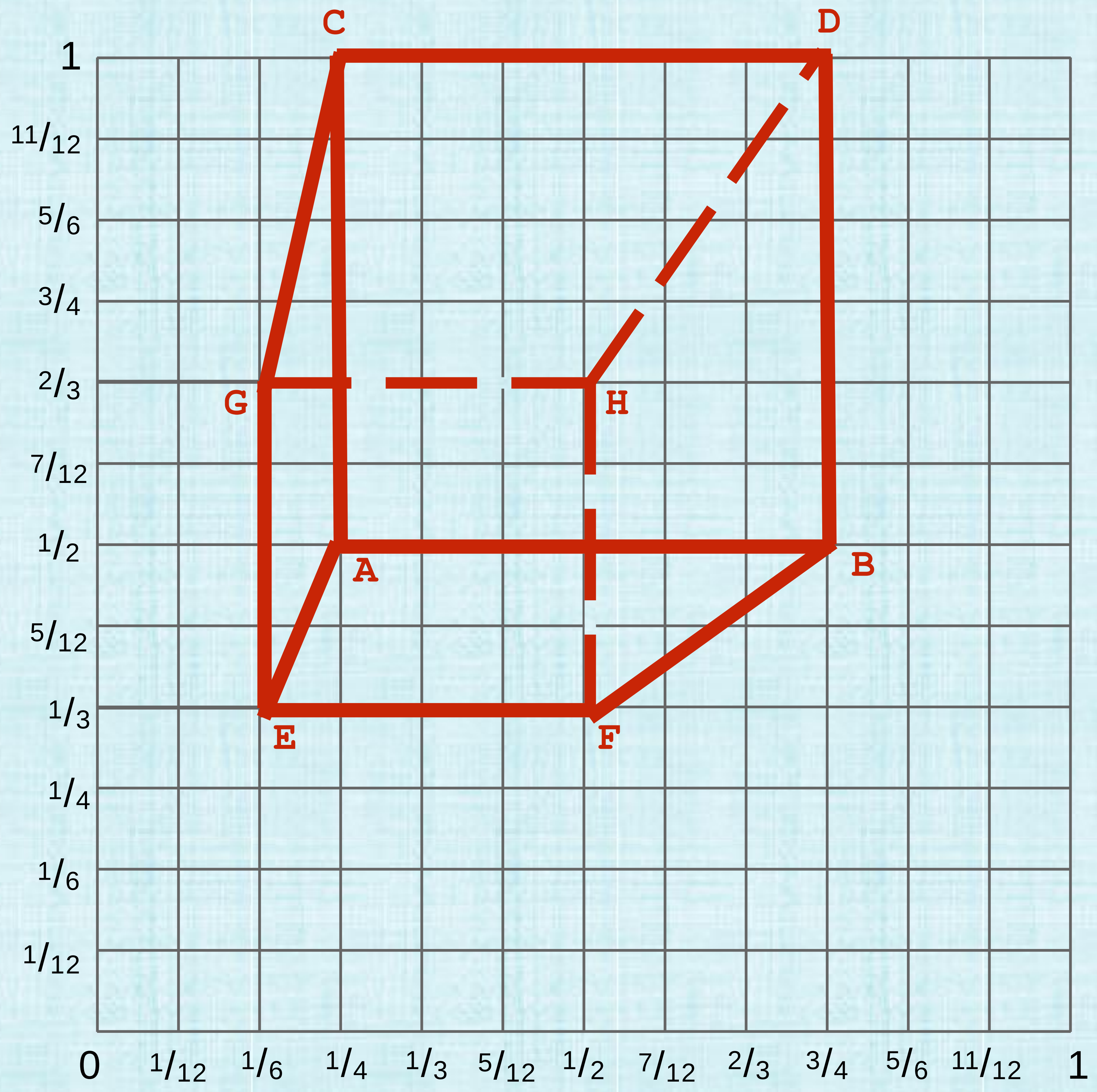
Projected coordinates:

A: (1/4, 1/2)
B: (3/4, 1/2)

Vertex	World (x,y,z)	Camera Space (x_c,y_c,z_c)	2D Projection (x′,y′)
A	(1,1,1)	(1−2,1−3,1−5)=(−1,−2,−4)	(−1/−4,−2/−4)=(1/4,1/2)
B	(−1,1,1)	(−1−2,1−3,1−5)=(−3,−2,−4)	(−3/−4,−2/−4)=(3/4,1/2)
C	(1,−1,1)	(1−2,−1−3,1−5)=(−1,−4,−4)	(−1/−4,−4/−4)=(1/4,1)
D	(−1,−1,1)	(−1−2,−1−3,1−5)=(−3,−4,−4)	(−3/−4,−4/−4)=(3/4,1)
E	(1,1,−1)	(1−2,1−3,−1−5)=(−1,−2,−6)	(−1/−6,−2/−6)=(1/6,1/3)
F	(−1,1,−1)	(−1−2,1−3,−1−6)=(−3,−2,−6)	(−3/−6,−2/−6)=(1/2,1/3)
G	(1,−1,−1)	(1−2,−1−3,−1−5)=(−1,−4,−6)	(−1/−6,−4/−6)=(1/6,2/3)
H	(−1,−1,−1)	(−1−2,−1−3,−1−5)=(−3,−4,−6)	(−3/−6,−4/−6)=(1/2,2/3)

How did we do?

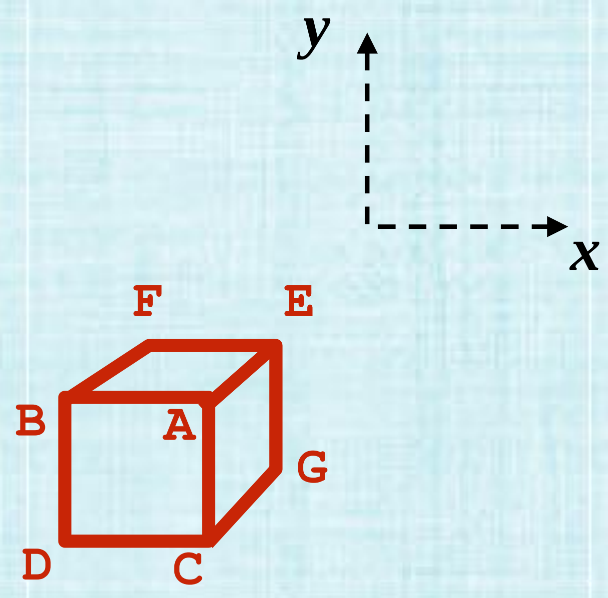
Recall: camera at (2,3,5), looking in -Z direction, cube centered at origin



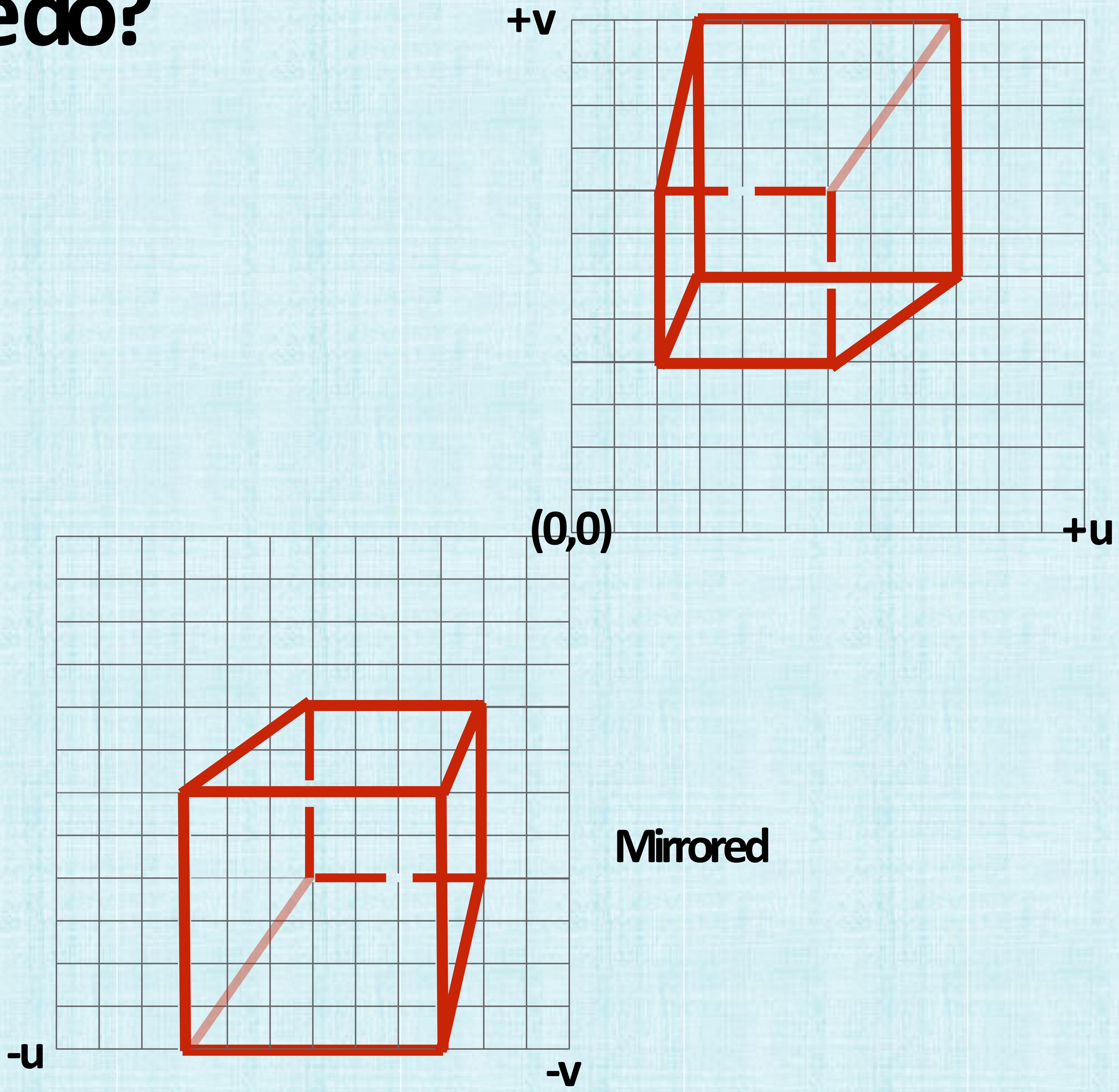
2D coordinates (after projection):

```
A:  (1/4, 1/2)
B:  (3/4, 1/2)

C:  (1/4, 1)
D:  (3/4, 1)

E:  (1/6, 1/3)
F:  (1/2, 1/3)

G:  (1/6, 2/3)
H:  (1/2, 2/3)
```

Keep in mind, this image is mirrored since it is a pinhole projection. Mirror the result about the origin (0,0) and you get…



How did we do?



Mirrored