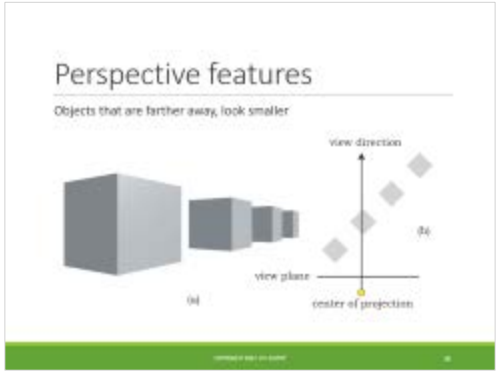
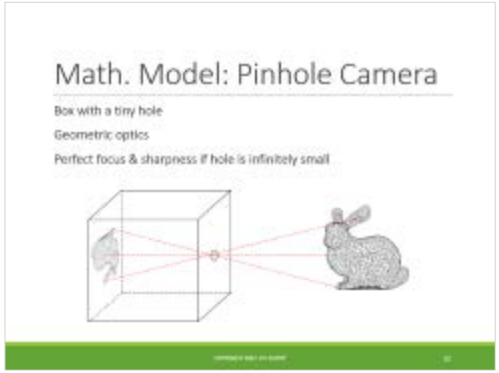
# Fundamentals of Computer Graphics

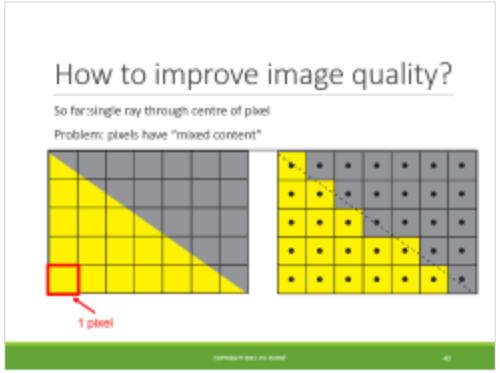
PHILIP DUTRÉ
DEPARTMENT OF COMPUTER SCIENCE

# Lecture overview





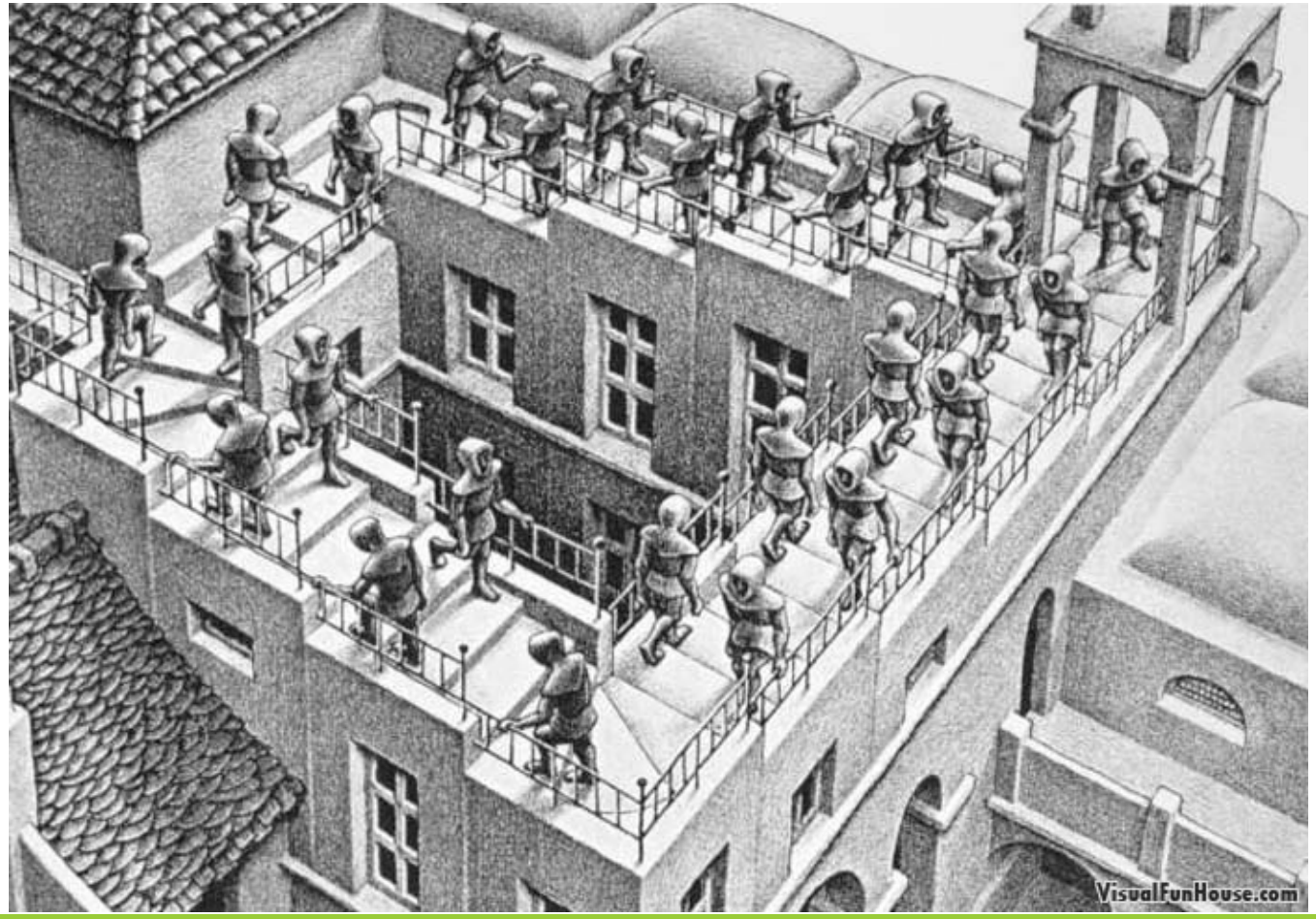


Relevant sections in book: Chapter 4, 5.3, 8, 9 and 10. Chapter 11 interesting to read/
(Illustrations from *Ray Tracing From The Ground Up*, *Physically-Based Rendering*, *Fundamentals of Computer Graphics*)
(Page numbering might skip some slides due to 'hidden' slides in my presentation.)

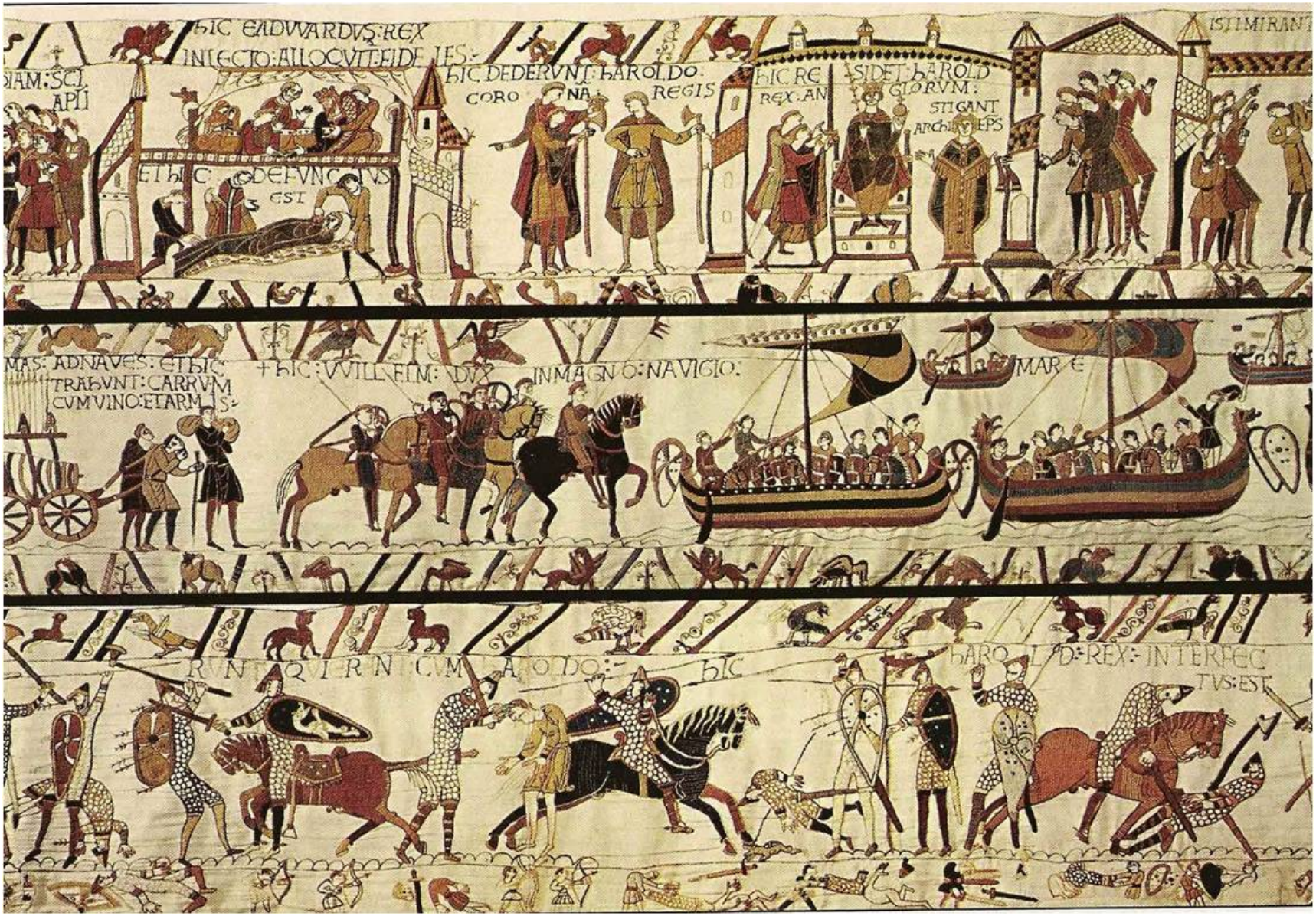
# Perspective

# Perspective

# Perspective

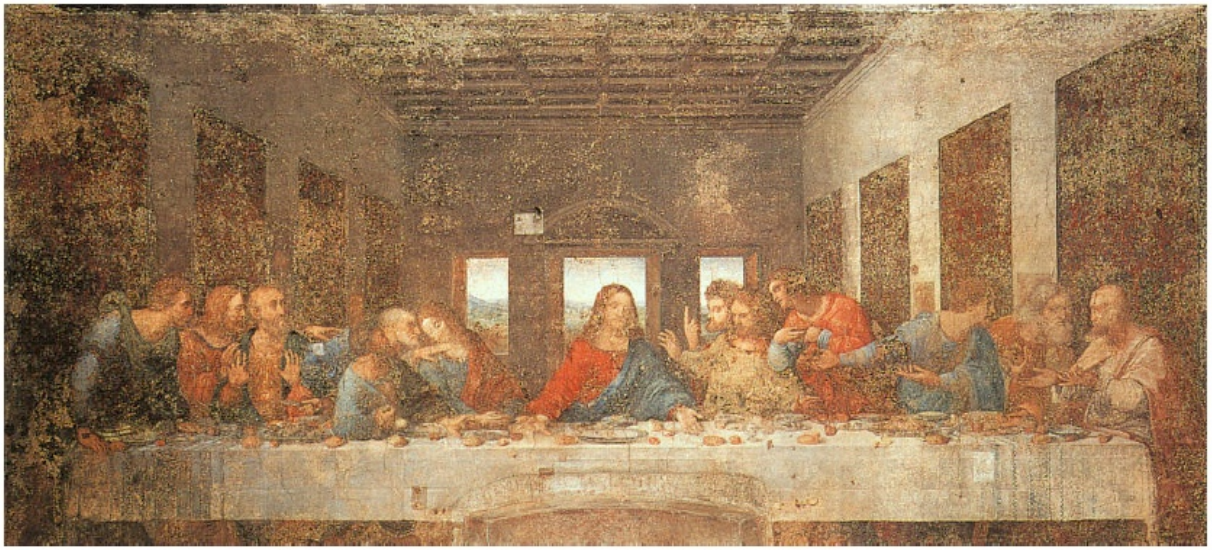


Lascaux Cave paintings, ~17000 BP




Bayeux Tapestry, ~1070

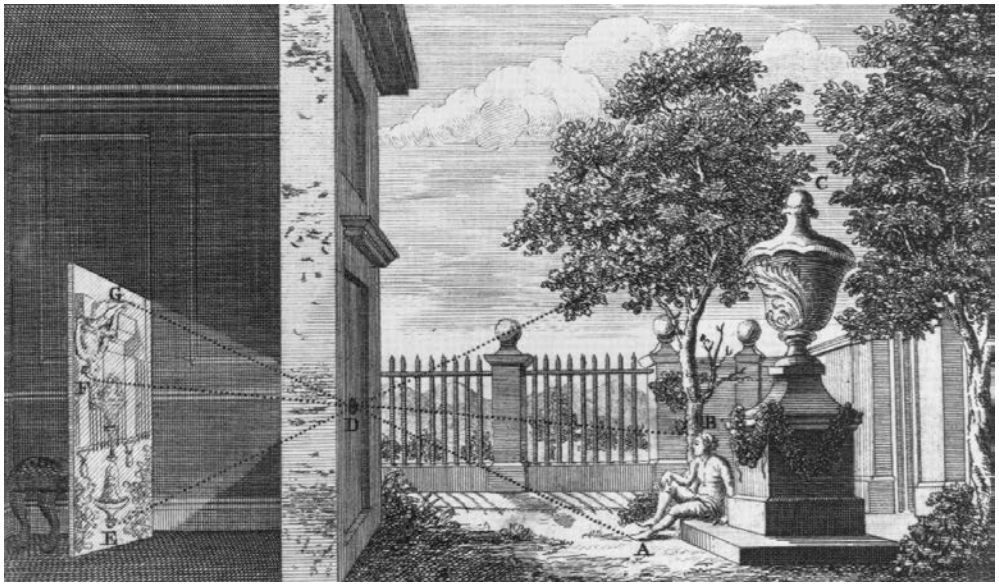
Duccio c. 1308

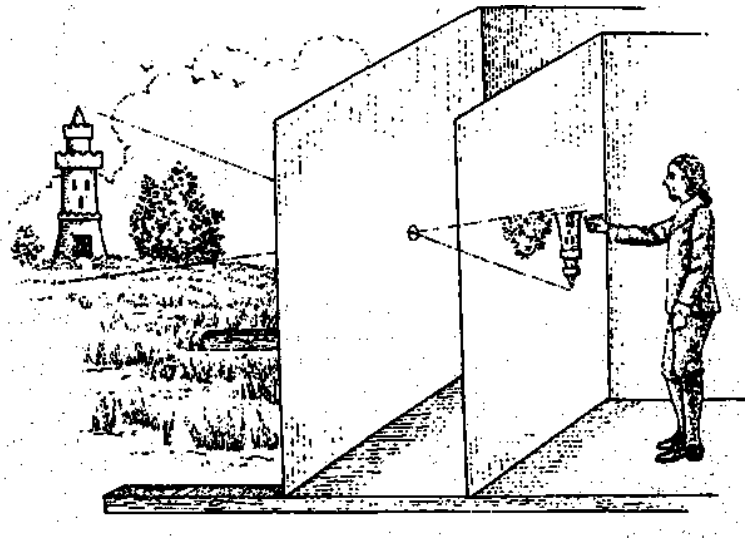


da Vinci c. 1498

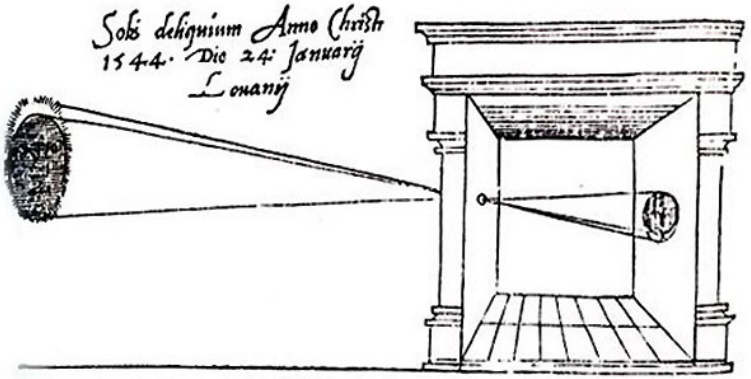
The Music Lesson,  Vermeer ~1662

# Camera Obscura
# Camera Lucida



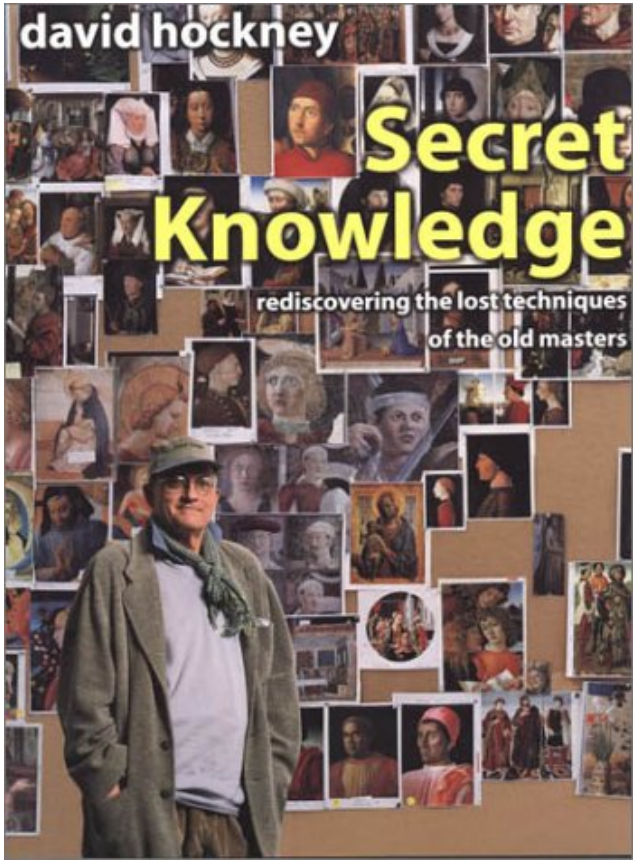



illum in tabula per radios Solis, quàm in cœlo contingit: hoc est, si in cœlo superior pars deliquiũ patiatur, in radiis apparebit inferior deficere, vt ratio exigit optica.

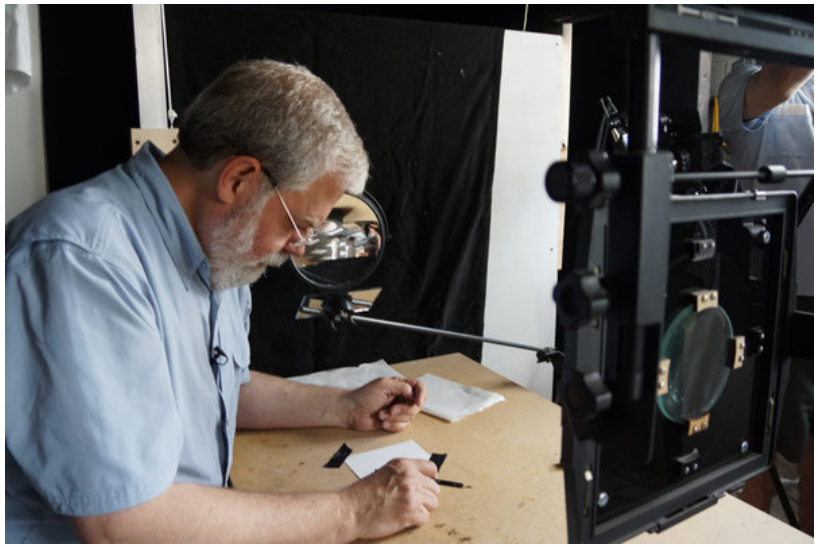


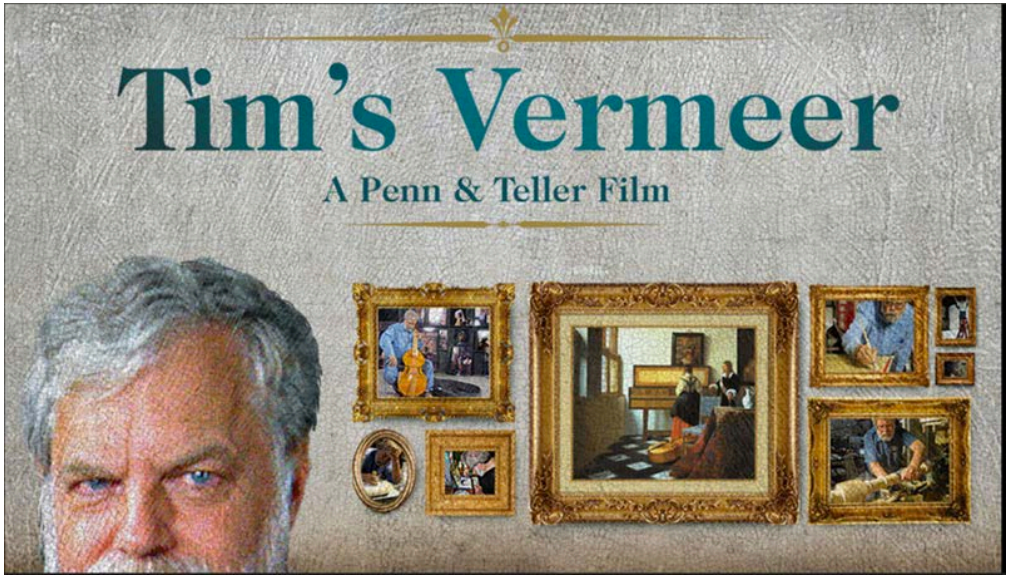

Sic nos exactè Anno .1544. Louanii eclipsim Solis obseruauimus, inuenimusq; deficere paulò plus q̃ dextantem, hoc est. 10. vncias siue digitos vt nostri loquun-

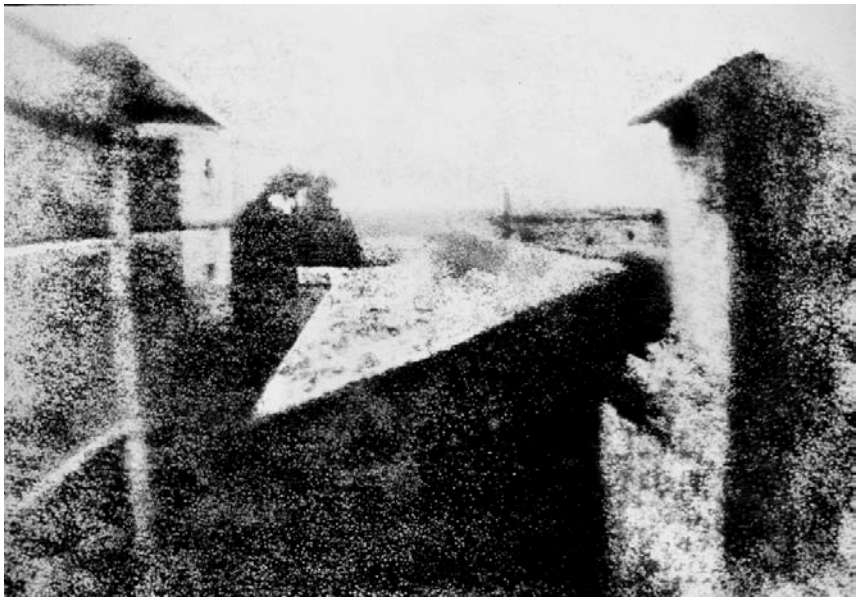
# Camera Obscura in Art




(Arnolfini Wedding, Jan Van Eyck, 1434)

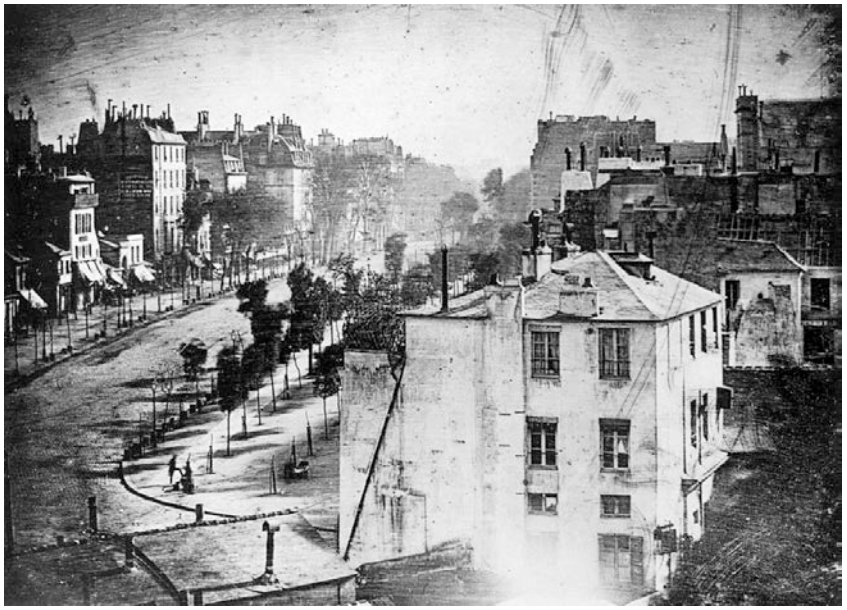


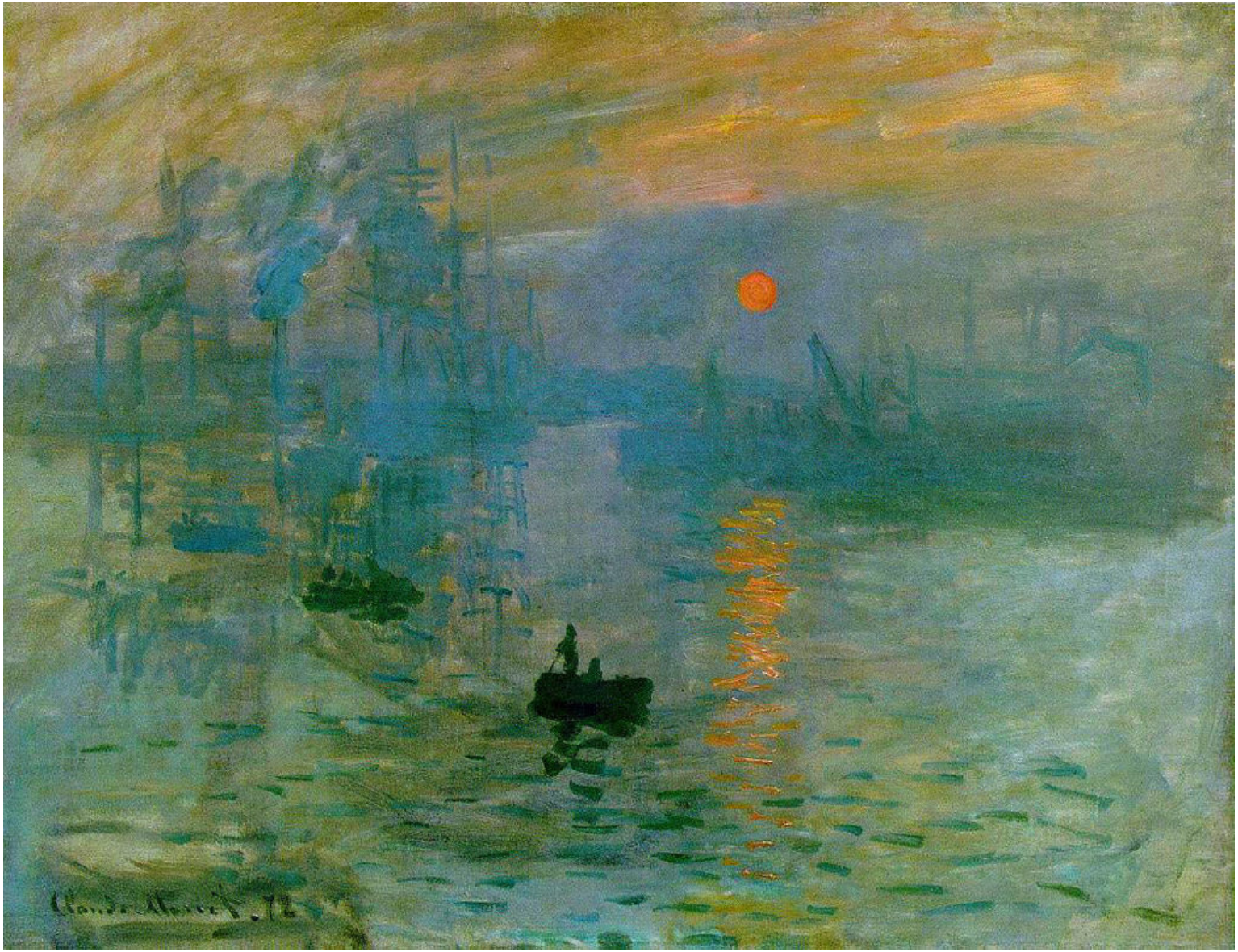

Tim's Vermeer
A Penn & Teller Film



~1826 *Point de vue du Gras*

~1838 *Boulevard du Temple*

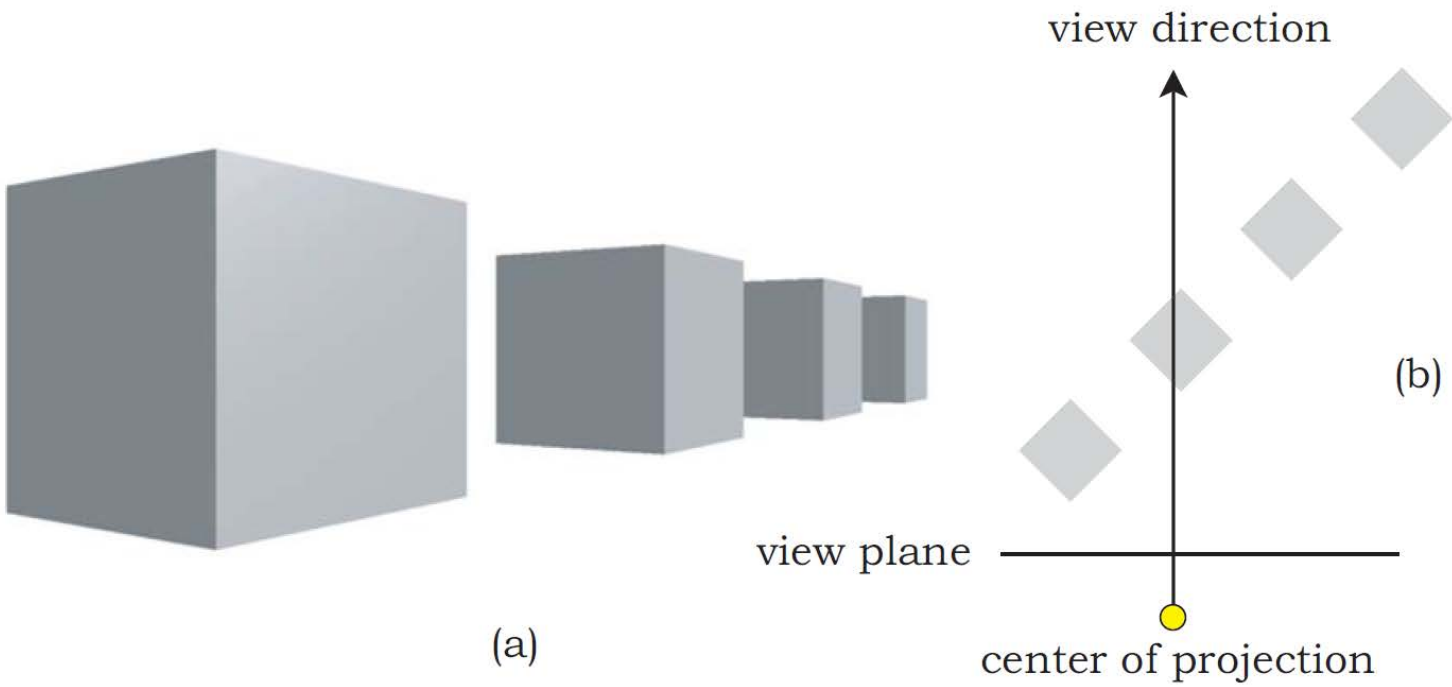




*Impression soleil levant*, Claude Monet, 1872

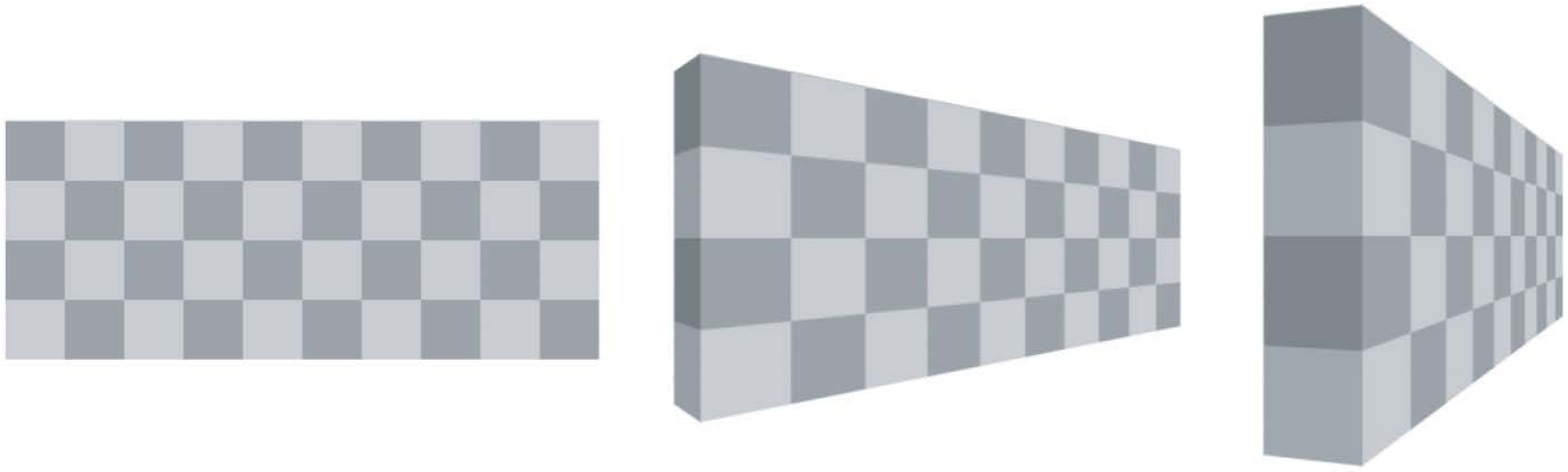
# Perspective features

Objects that are farther away, look smaller



# Perspective features

Foreshortening: rotated objects become “shorter”



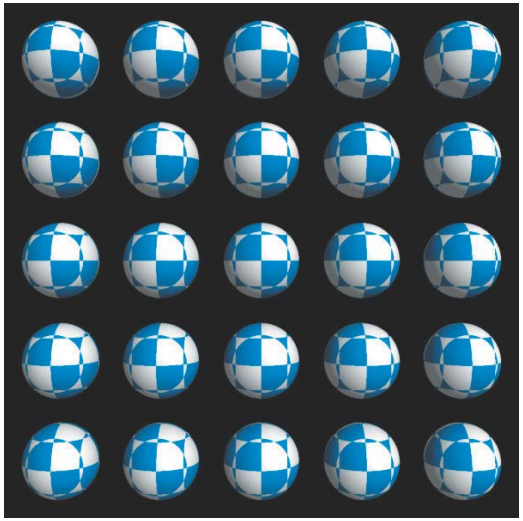
(a) (b) (c)

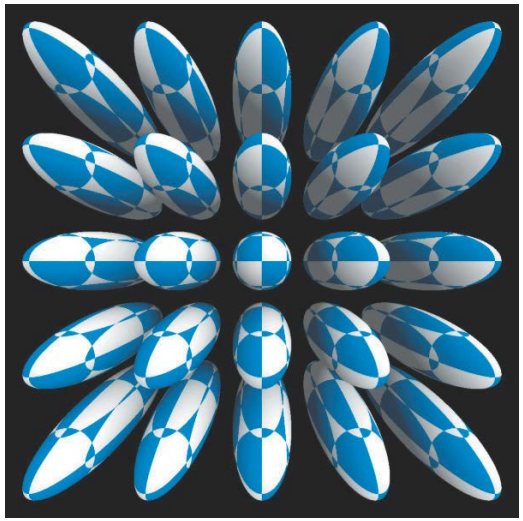
# Perspective features

Straight line ➔ straight line after projection

- Triangle in 3D is transformed into a  triangle in 2D after projection
  - (Perspective of a triangle can be computed by considering only the vertices)
  - (this simplifies some of the math in all sorts of graphics algorithms …)

Note: Spheres are not circles after perspective projection!)



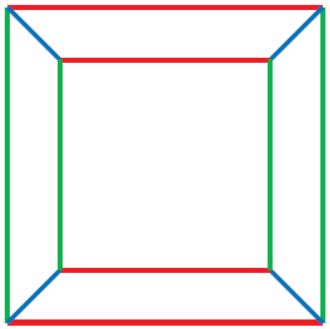


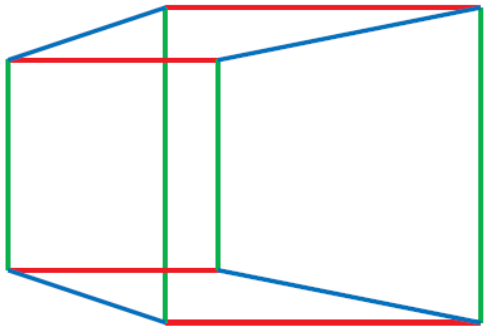
# Perspective features

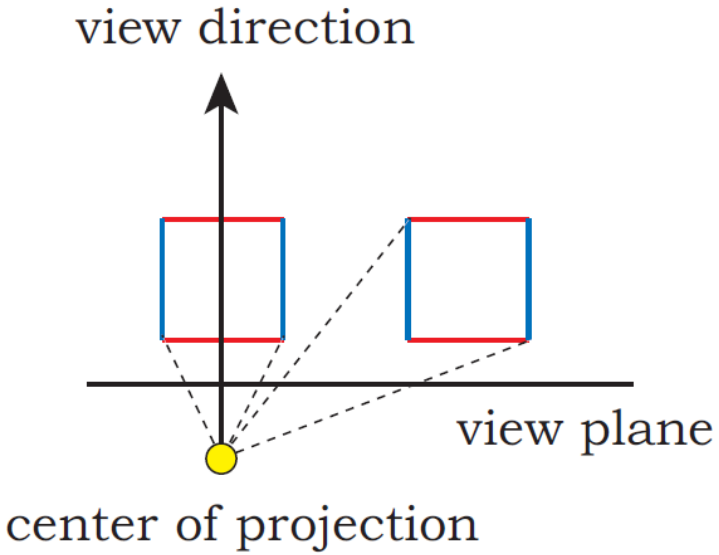
Parallel lines which are parallel to the projection plane remain parallel

Parallel lines not parallel to projection plane ➔ vanishing point

➔ this often forms the basis for manual constructions of perspective

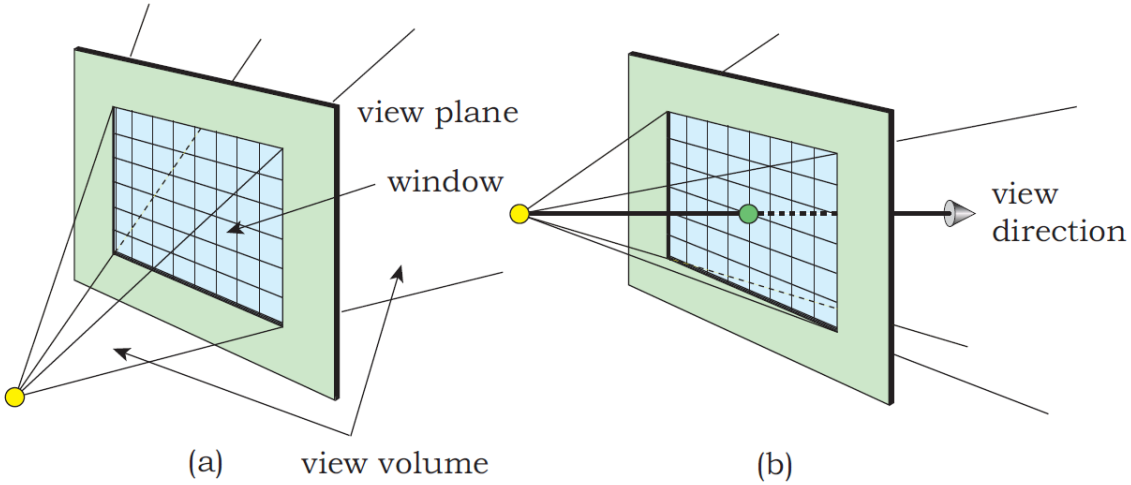






# Perspective features

View plane, view direction, image window …



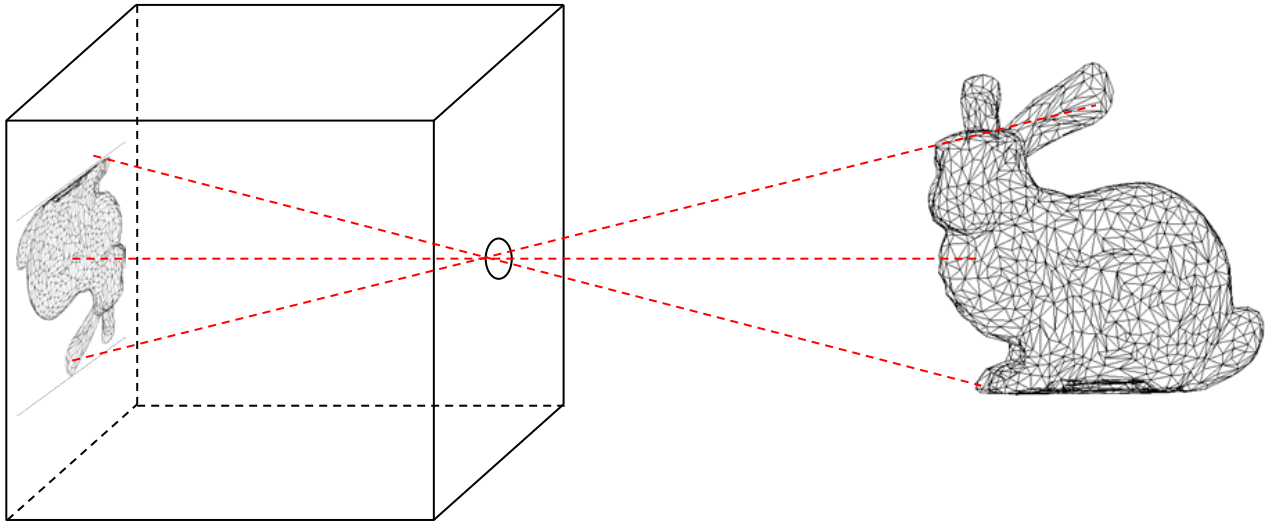

(Steve Marschner, Cornell University)

# Math. Model: Pinhole Camera

Box with a tiny hole

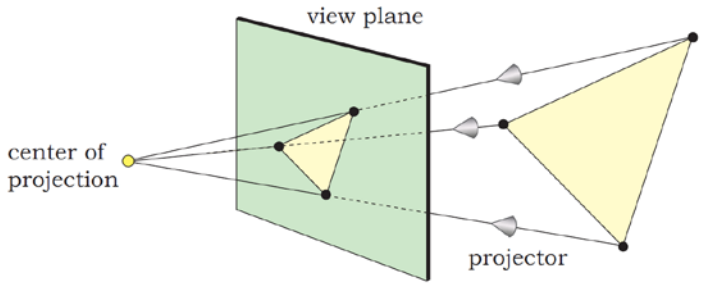
Geometric optics

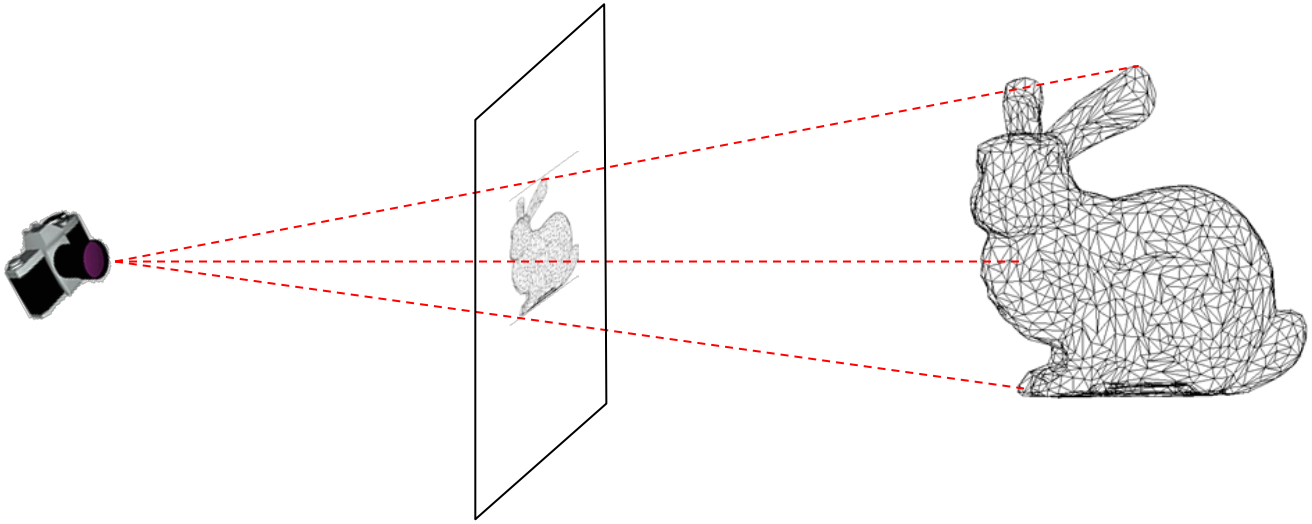
Perfect focus & sharpness if hole is infinitely small



# Math. Model: Pinhole Camera

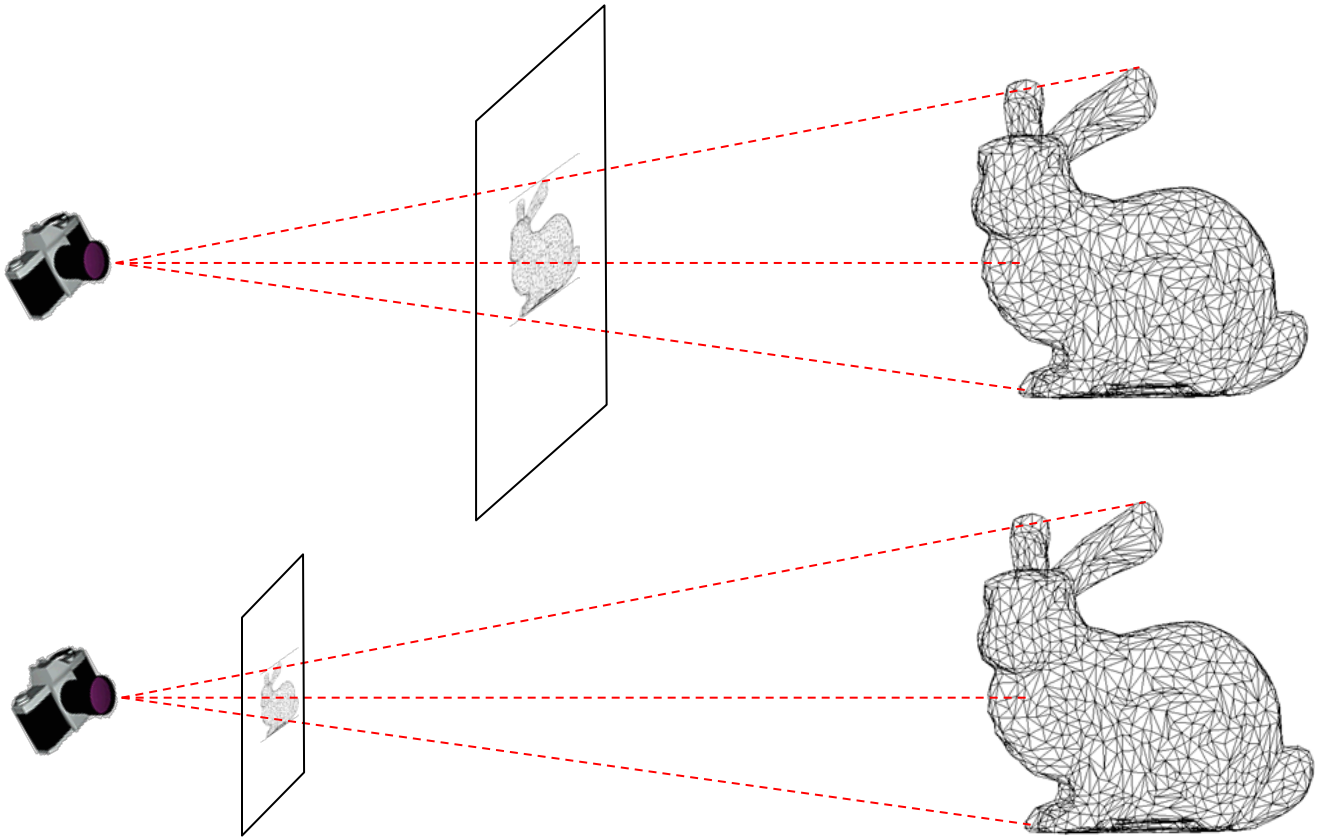
"Eye-image pyramid"



# Math. Model: Pinhole Camera

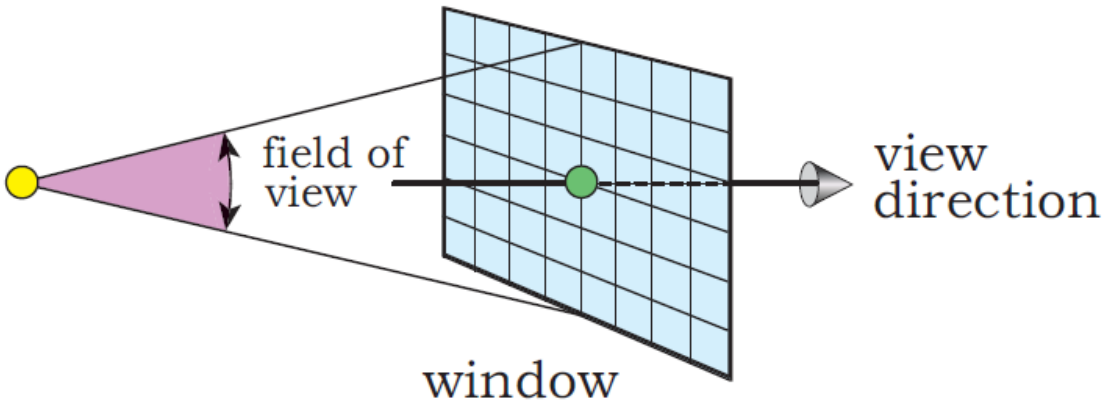
Distance/size of image plane can be chosen arbitrarily



# Field of view

Field of view is defined by:

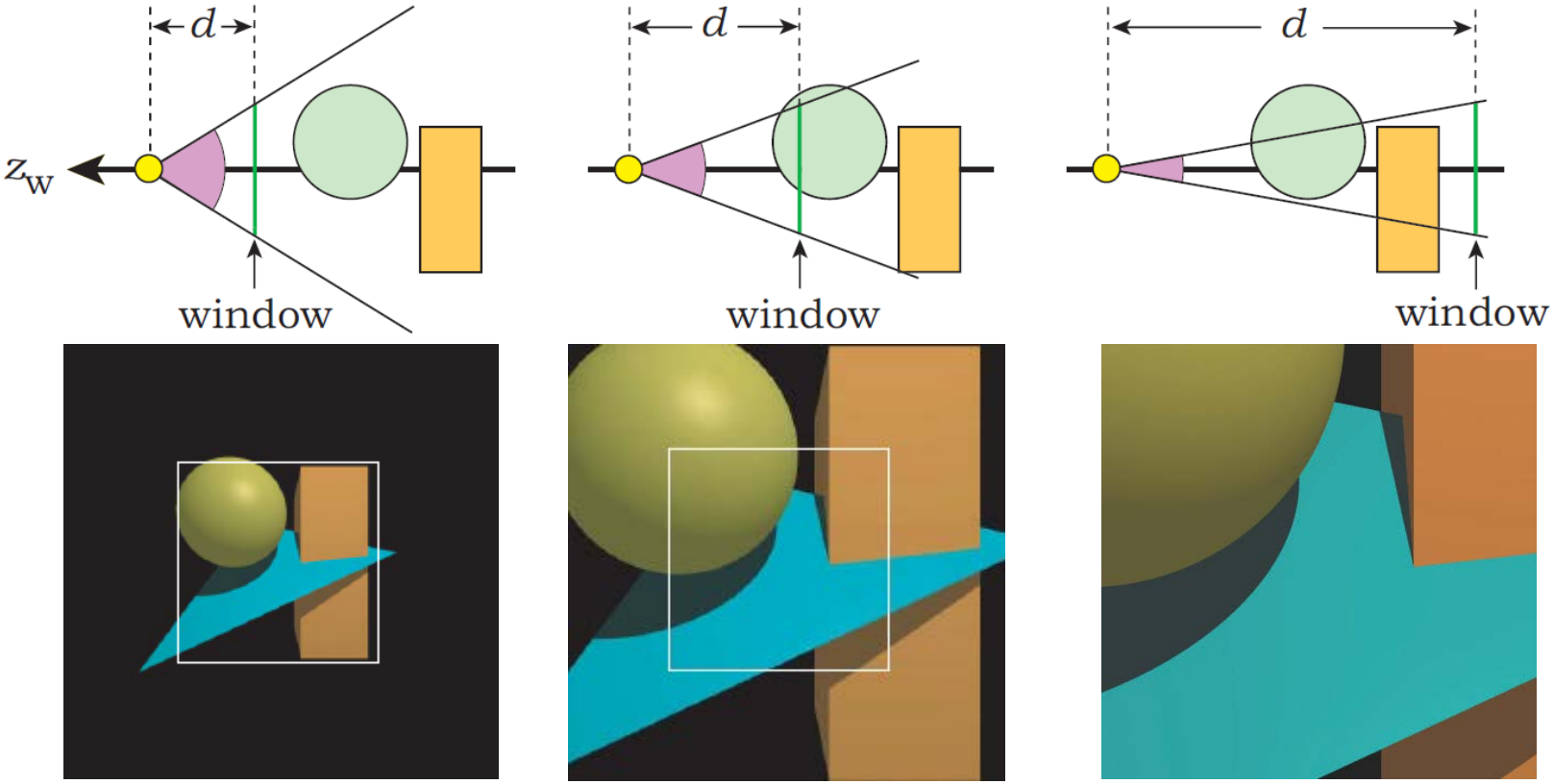
- Horizontal and vertical viewing angle
- … or distance to projection plane and width/height of image window




- Implicit assumptions: view plane perpendicular to view direction, field of view (window) symmetric around view direction

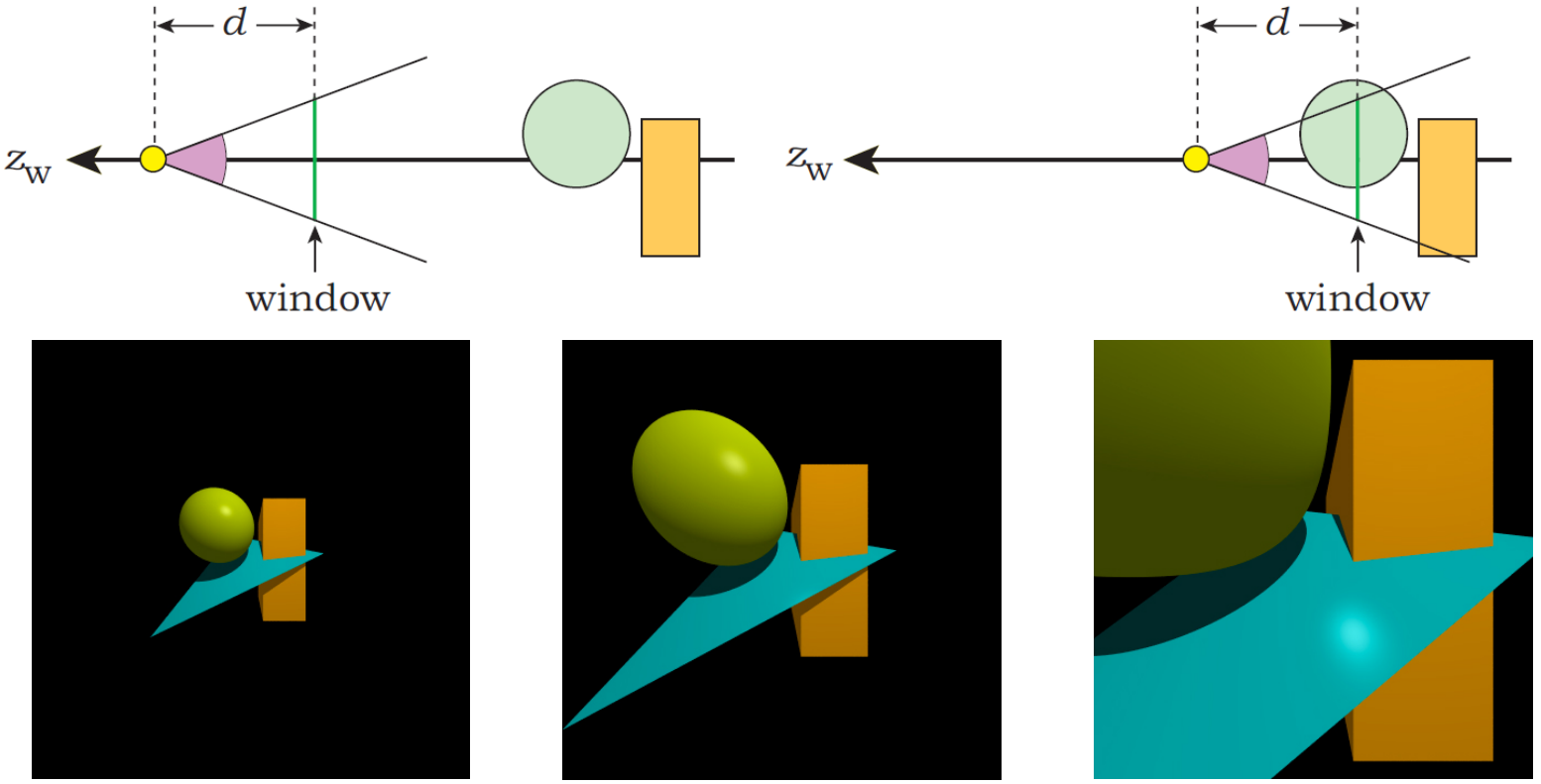
# Field of view

Narrowing the field of view, same camera position

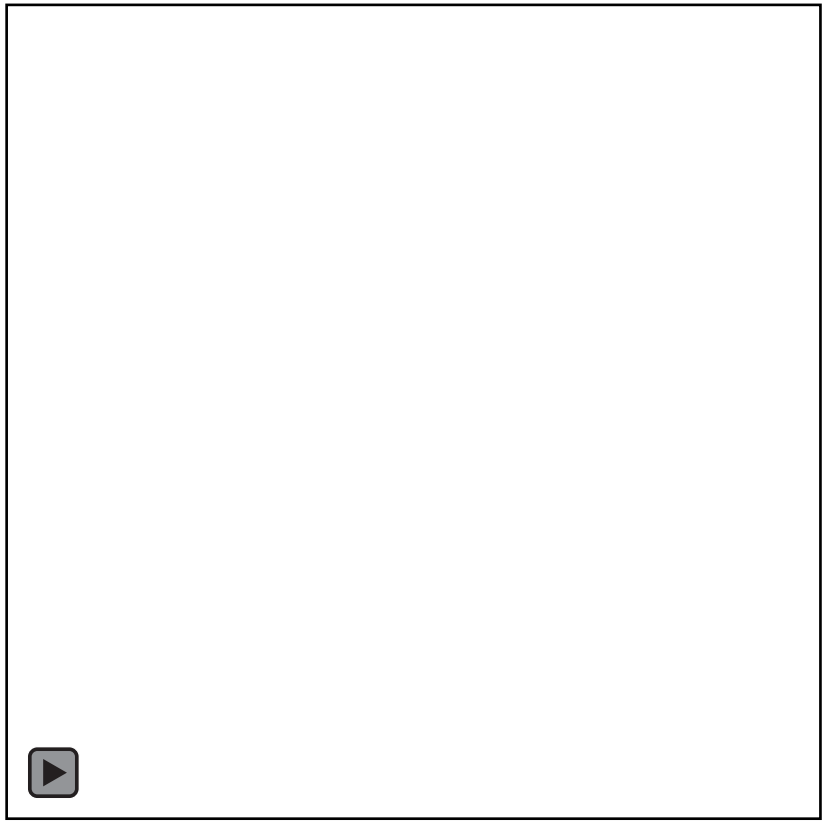


# Field of view

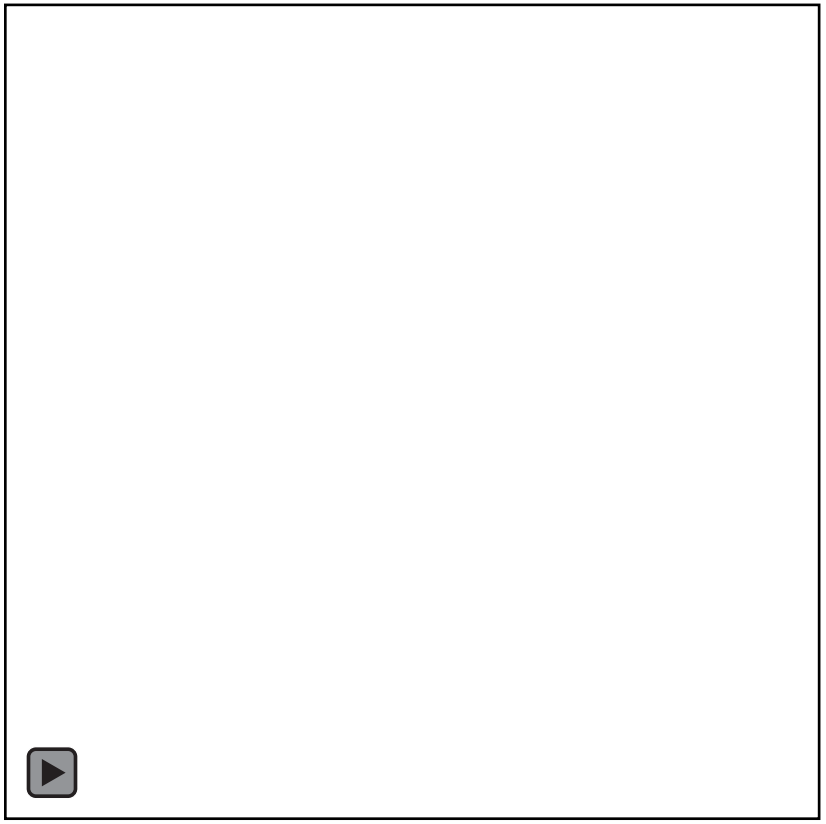
Moving the camera, same field of view



# Field of view

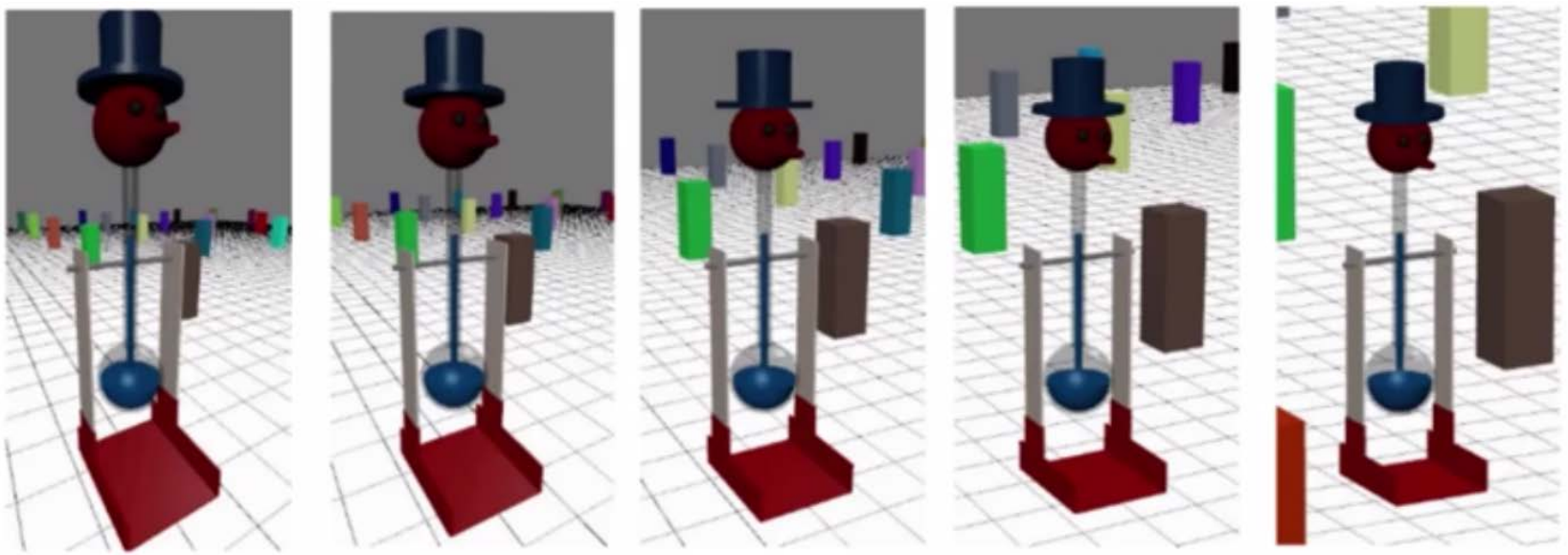


Camera static, narrowing field of view



Camera moves closer, same field of view

# Field of view

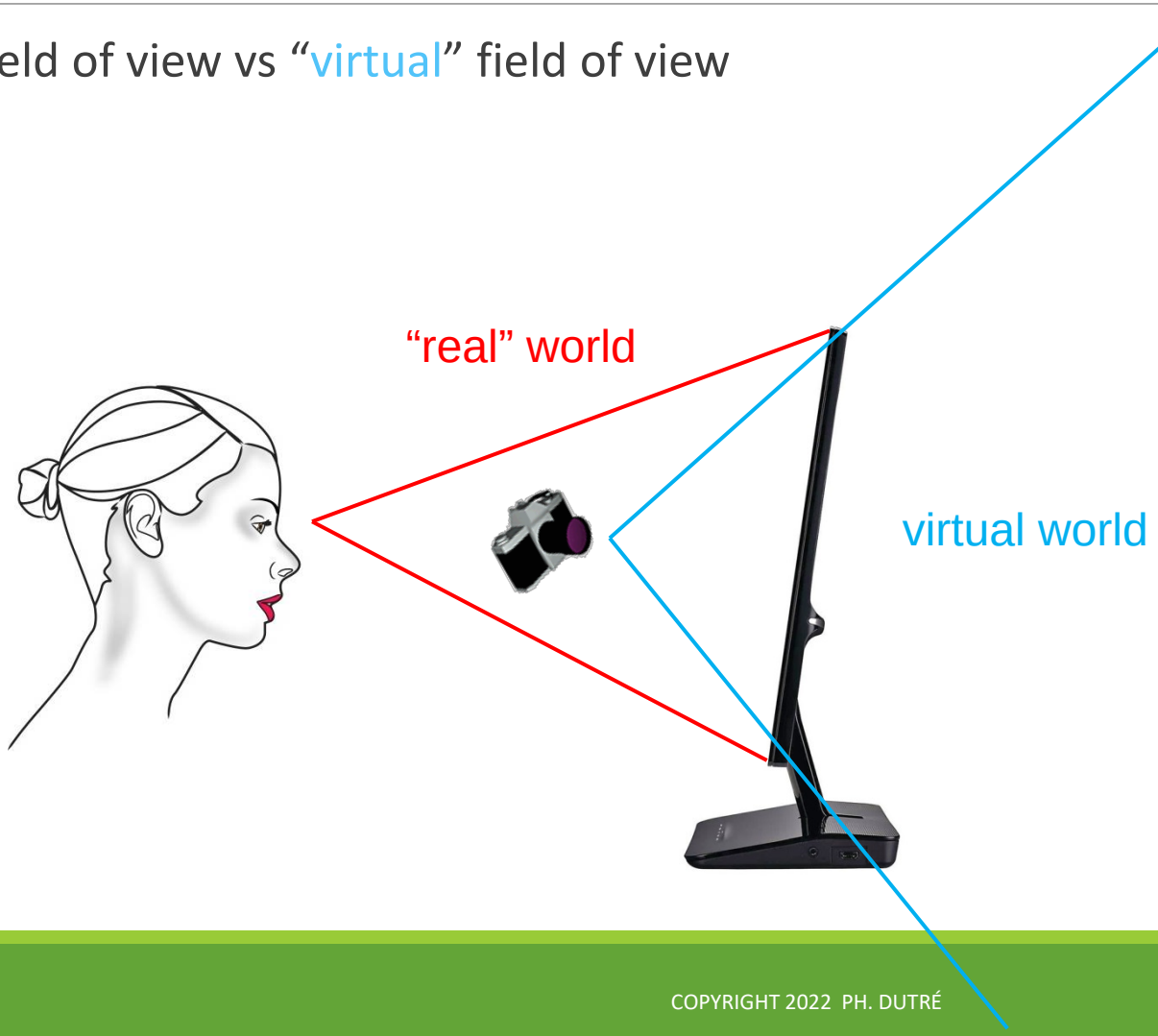


(Eric Haines)

move camera backwards, ***and*** narrowing field-of-view

# Field of view

"True" field of view vs "virtual" field of view



# How to compute perspective in computer graphics?

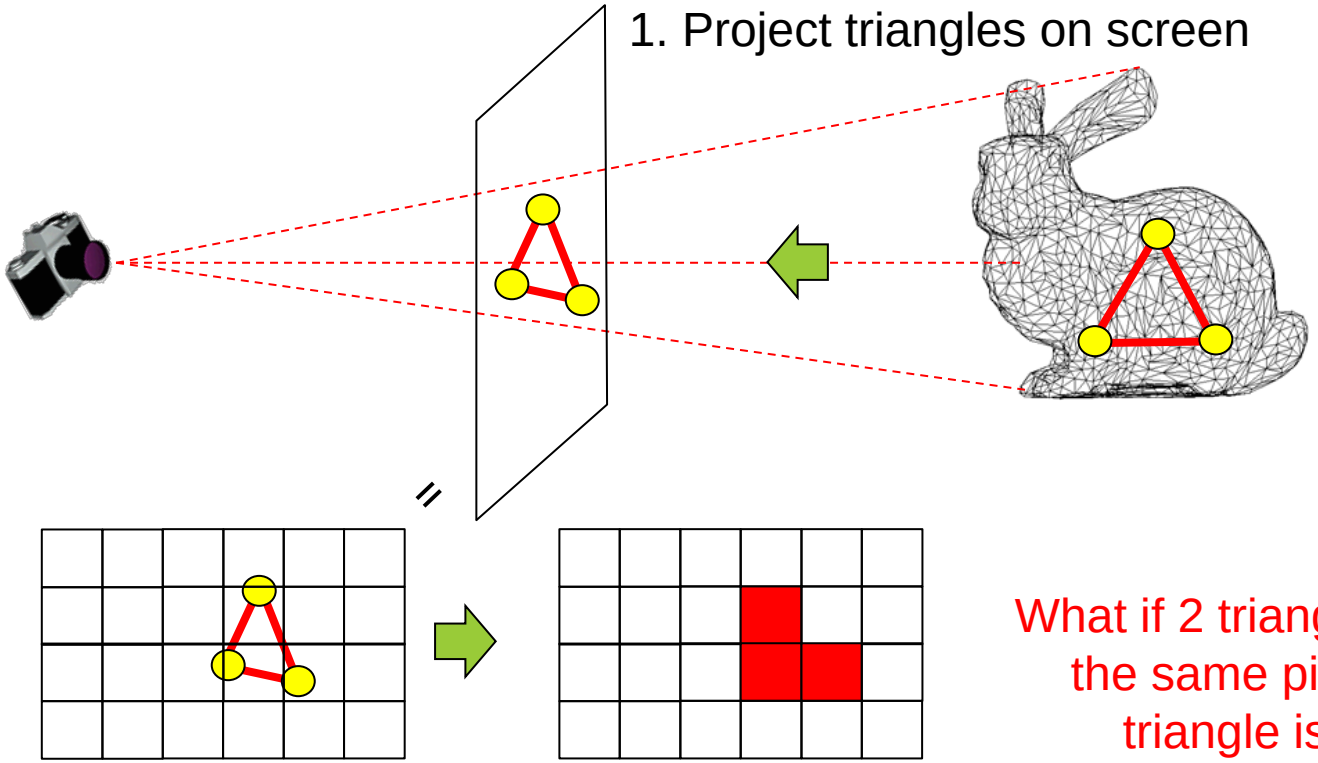
2 approaches:

- Project the objects on the screen
  - Object-driven
  - Projection needs to be transformed (discretized) into pixels

- Ray Tracing
  - Pixel-driven
  - “look through” pixel, figure out what is visible through pixel

# How to compute perspective in computer graphics?

(a) Project objects on screen



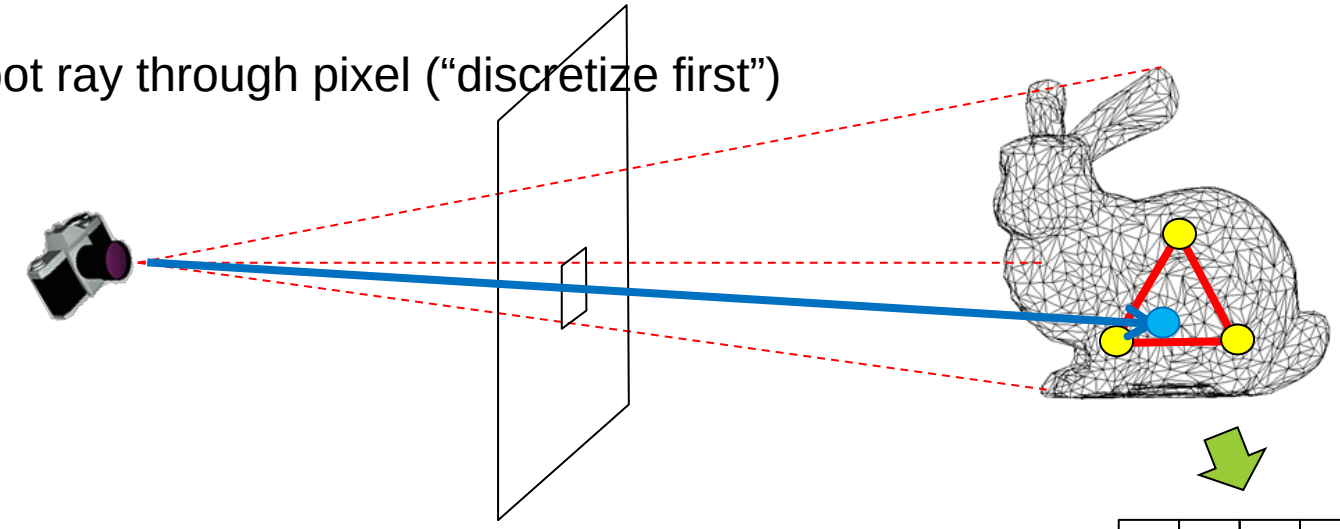

What if 2 triangles project to the same pixel? Which triangle is visible?

2. Discretize projected triangles into pixels (“discretize last”)

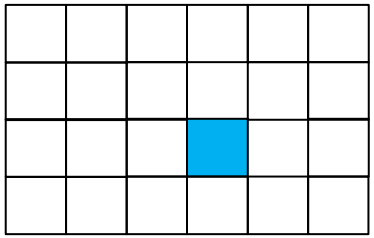
# How to compute perspective in computer graphics?

(b) Shoot ray through pixel

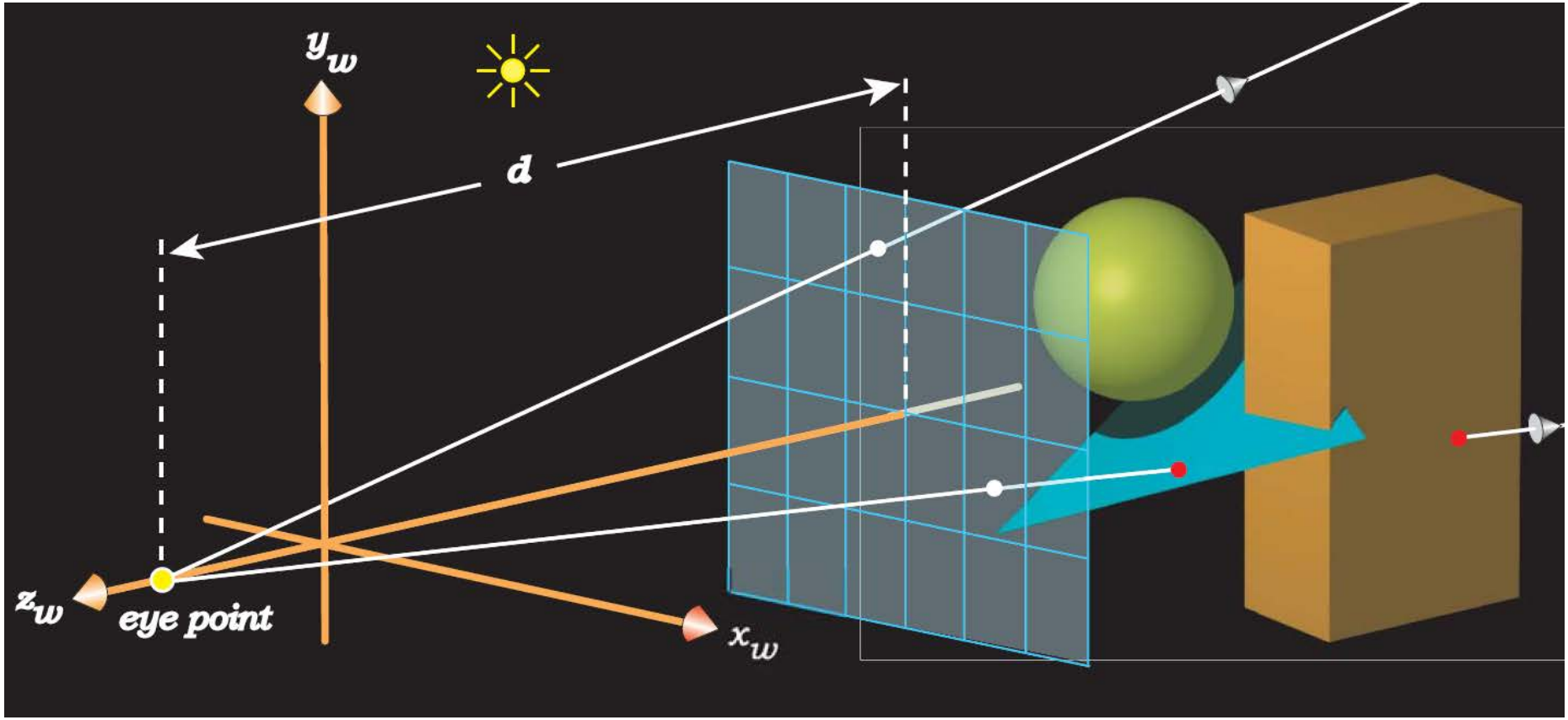
1. Shoot ray through pixel (“discretize first”)



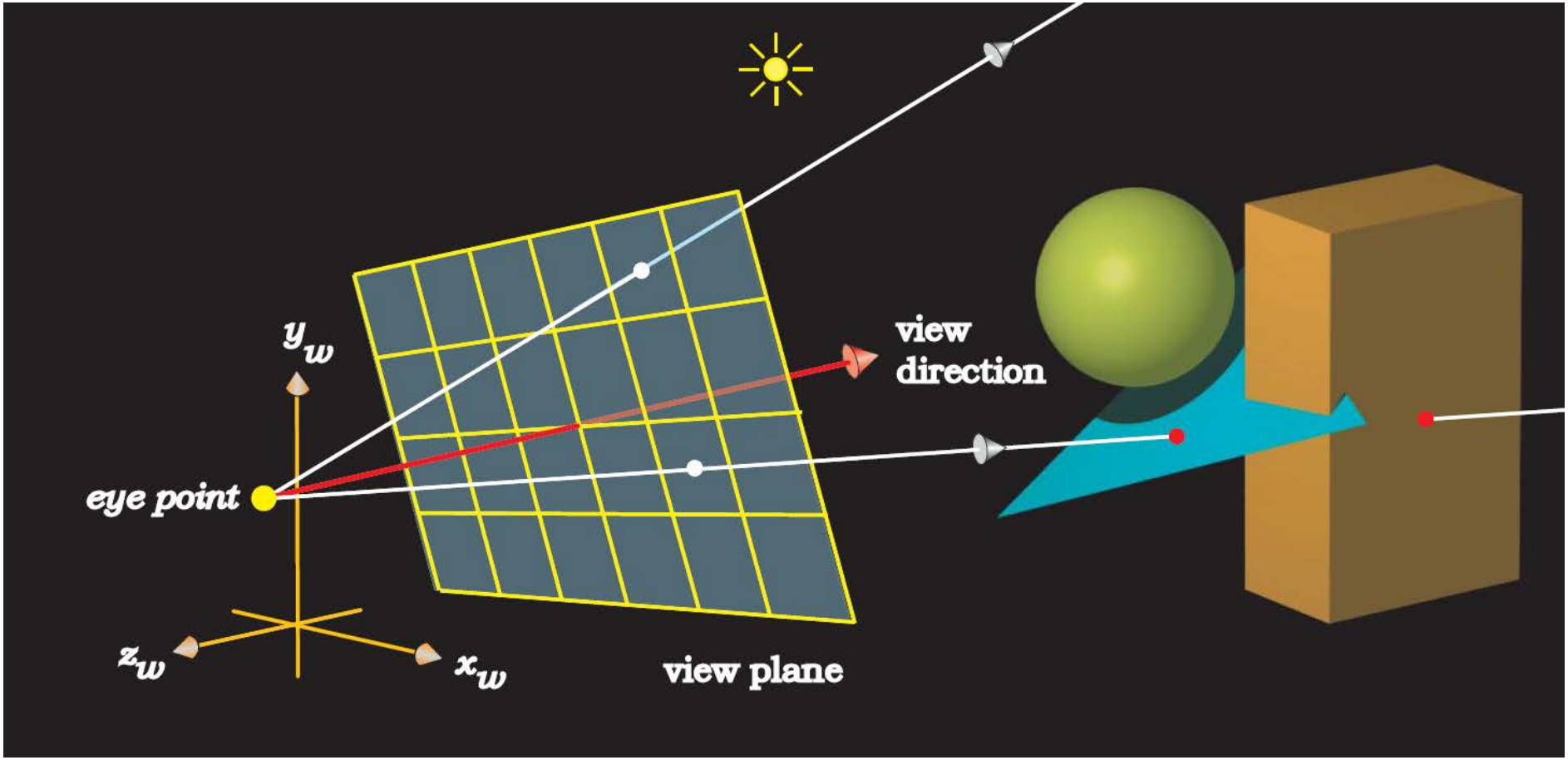
2. Find closest object
(visibility encoded in ray intersection query)



# Simple camera setup

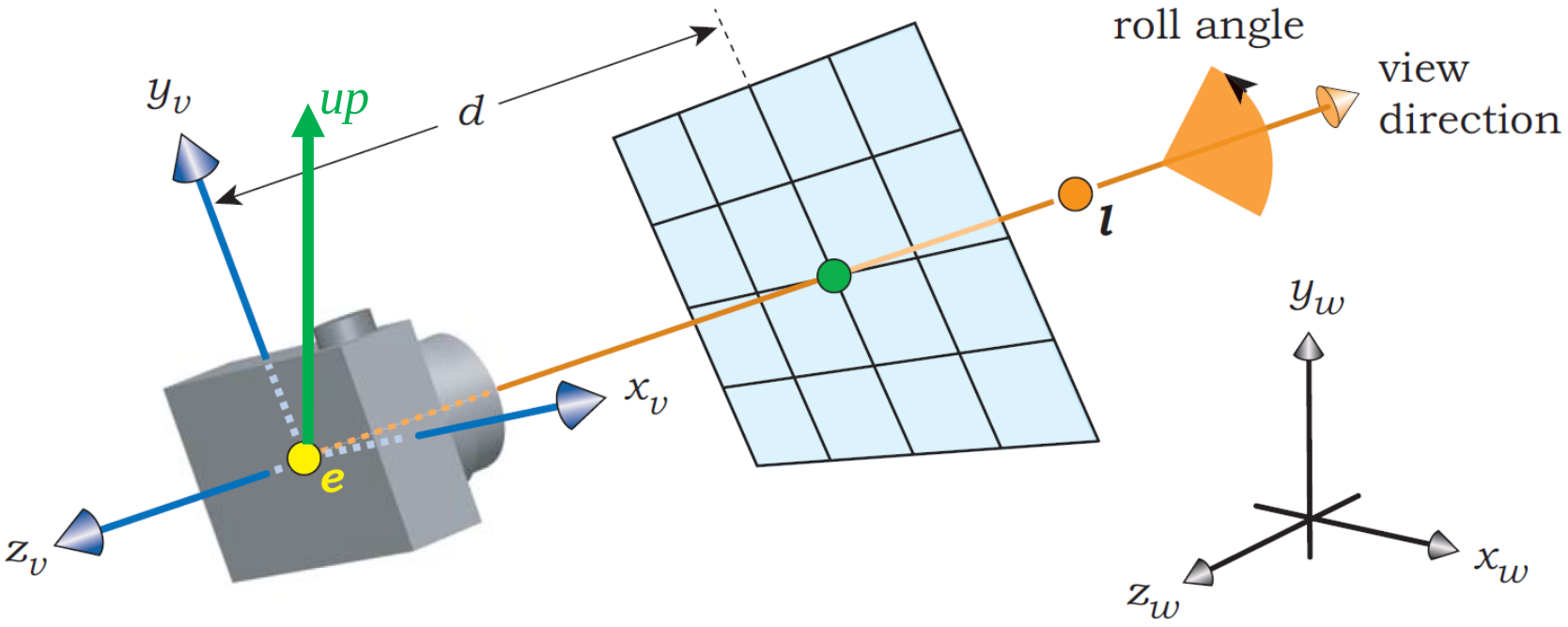


# General Camera Model

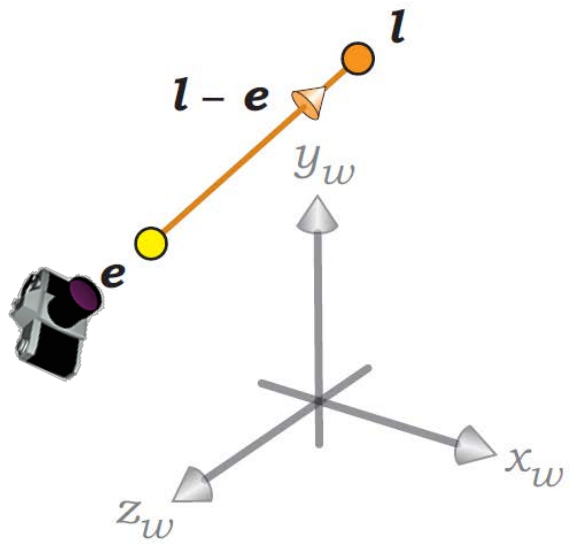


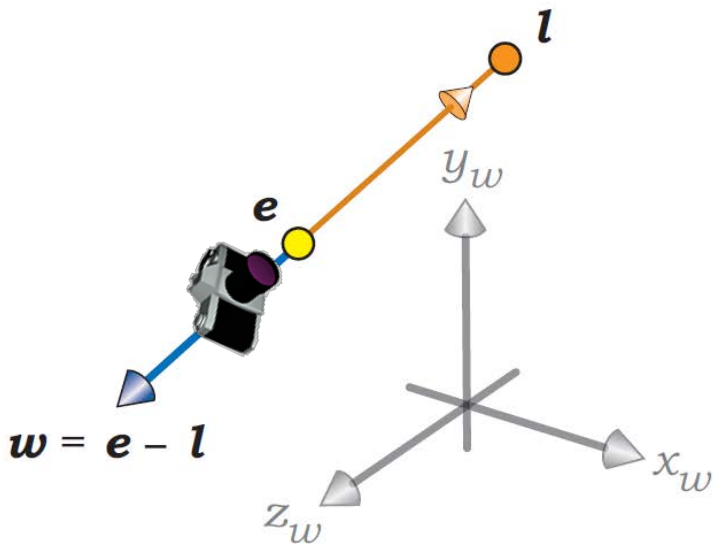
# General Camera Model

projection point e; look-at-point l; up vector; view plane distance d



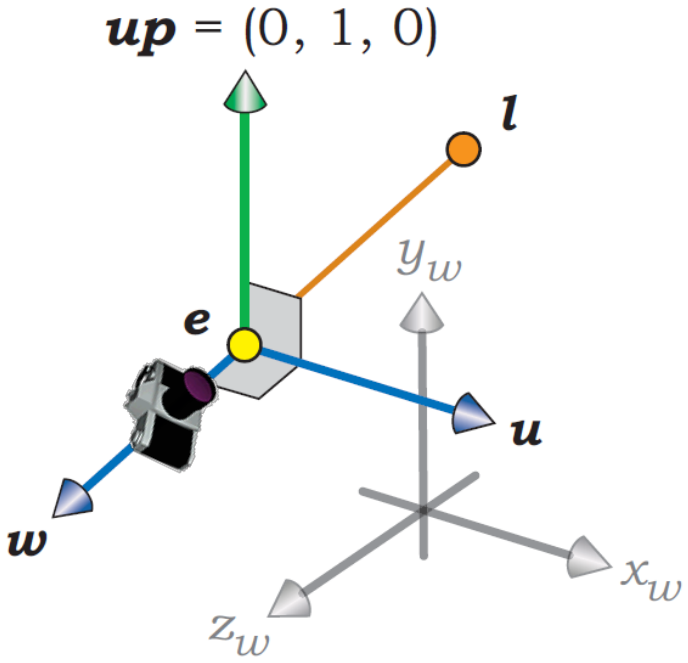
# Viewing coordinates

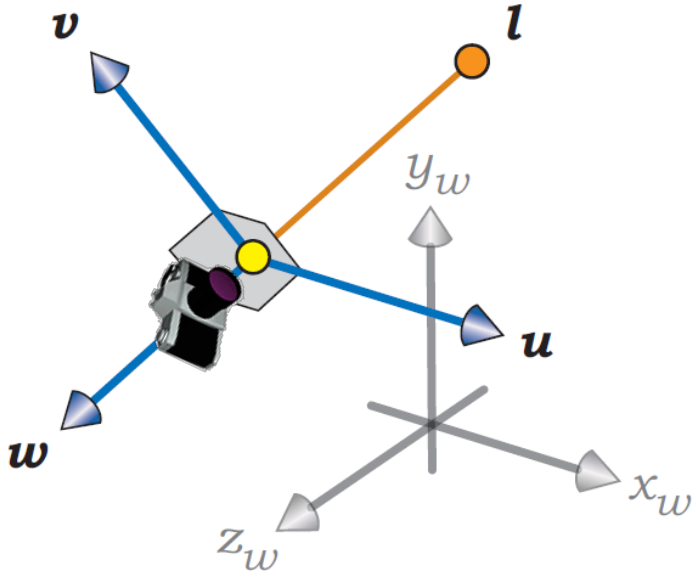



$$w = \frac{(e - l)}{\|e - l\|}$$

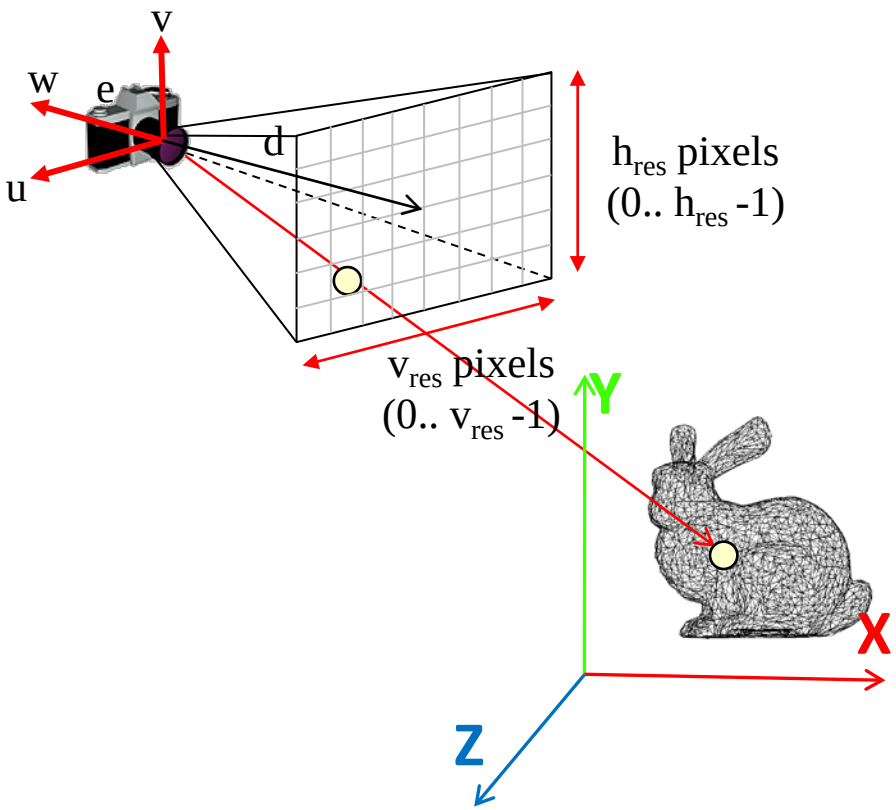
# Viewing coordinates




$$w = \frac{(e - l)}{\|e - l\|} \qquad u = \frac{up \times w}{\|up \times w\|} \qquad v = w \times u$$

# Viewing Rays



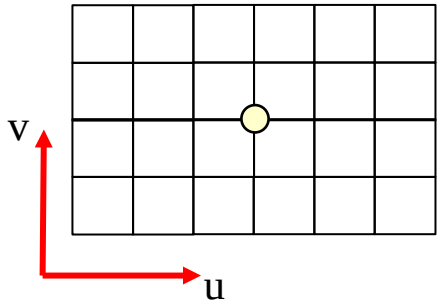

scalars

$$origin = e$$

$$direction = x \cdot u + y \cdot v - d \cdot w$$

$$x = s(c - h_{res}/2 + 0.5)$$

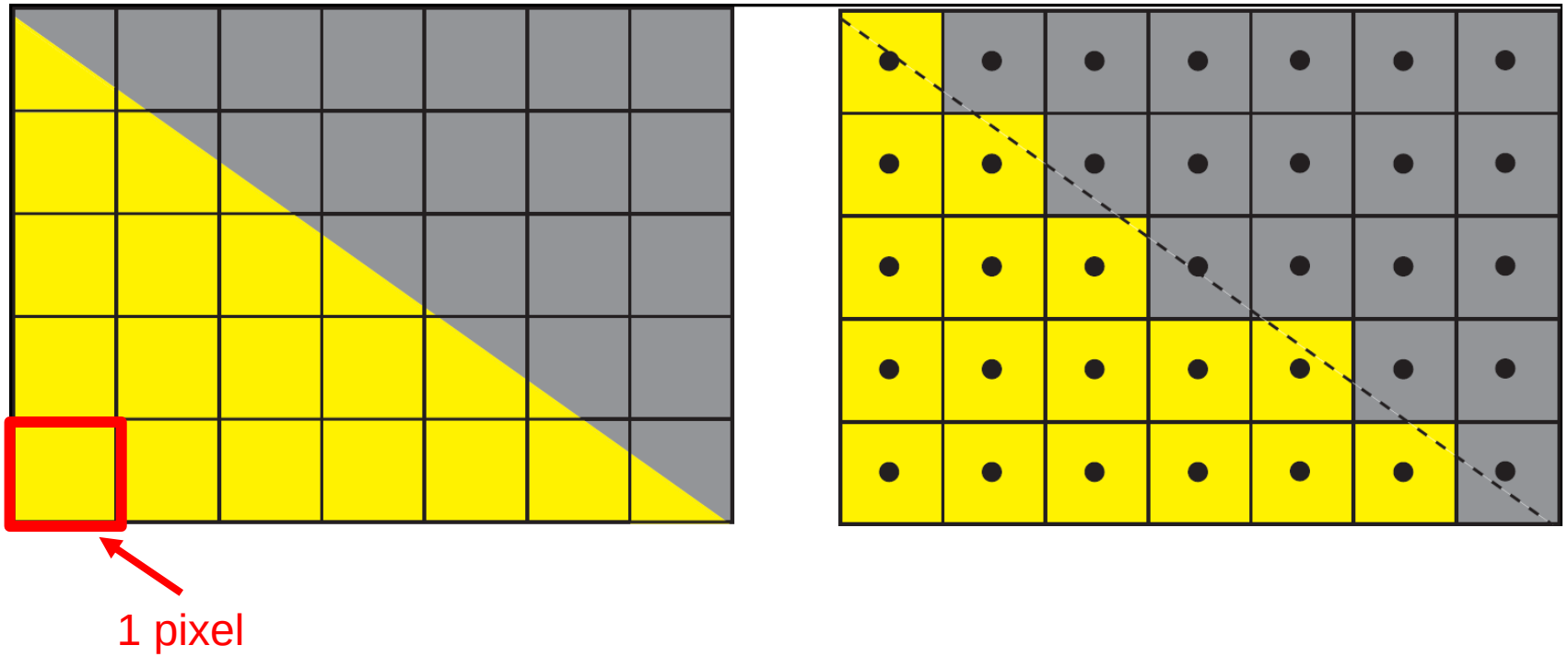
$$y = s(r - v_{res}/2 + 0.5)$$



# How to improve image quality?

So far:single ray through centre of pixel

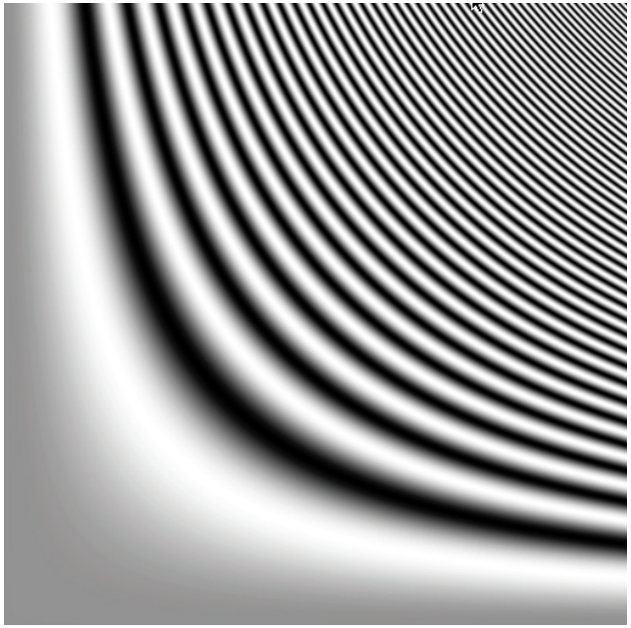
Problem: pixels have “mixed content”



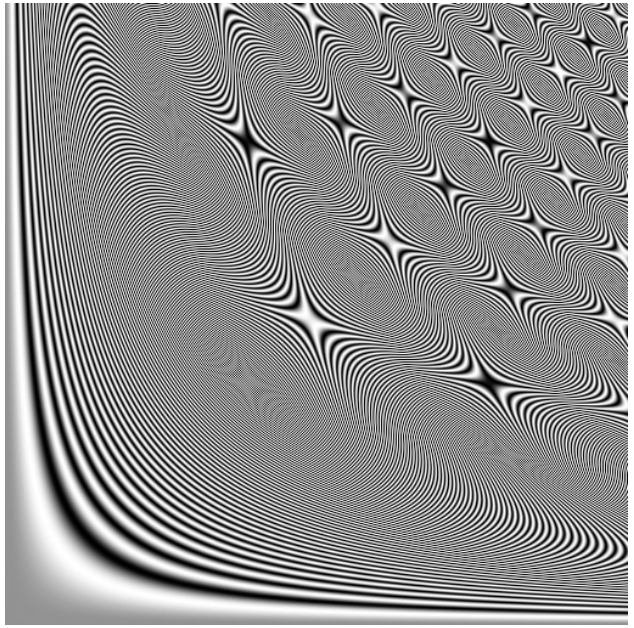
# Aliasing artefacts

$$f(x,y) = \frac{1}{2}(1 + \sin(x^2 y^2))$$

(images at 512 x 512 pixels, grey-value evaluated at centre of pixel)



$(x,y) \in [0, 3.79]^2$

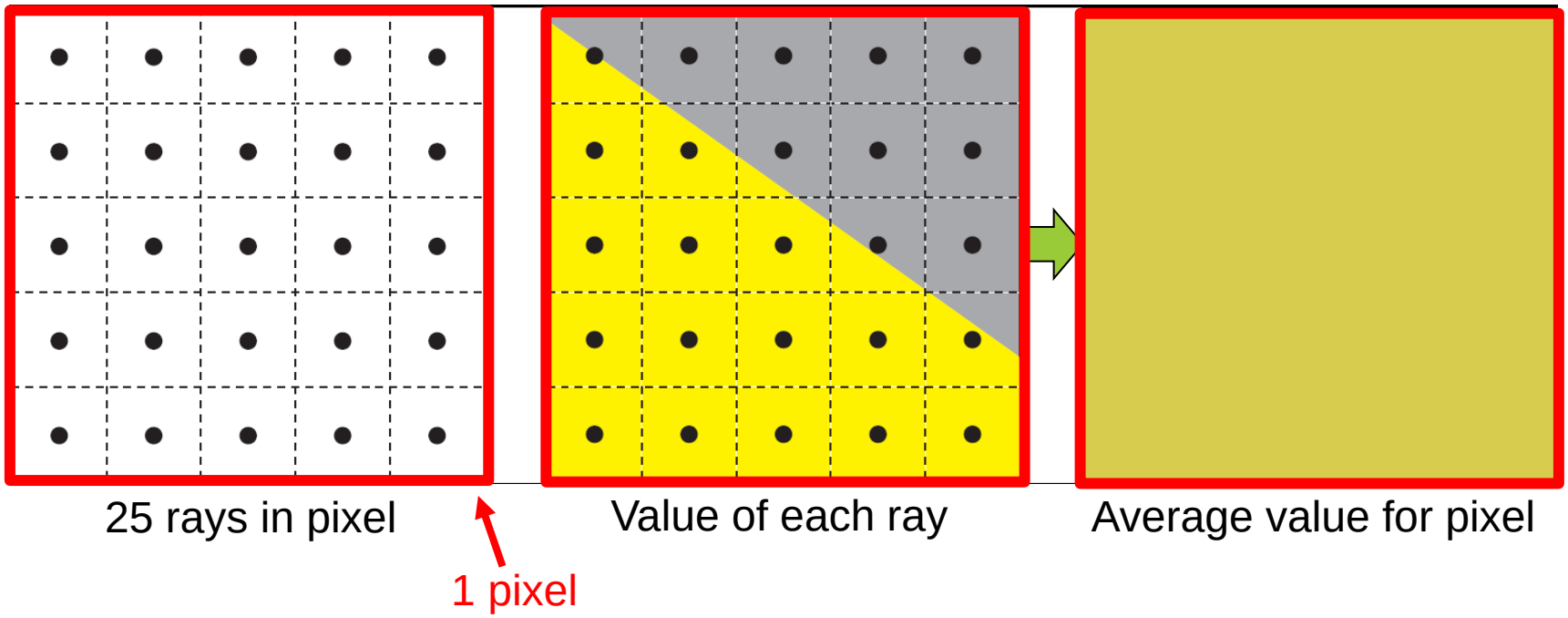


$(x,y) \in [0, 10.83]^2$

# Solutions for “aliasing”: “antialiasing”

Multiple viewing rays (“samples”) per pixel

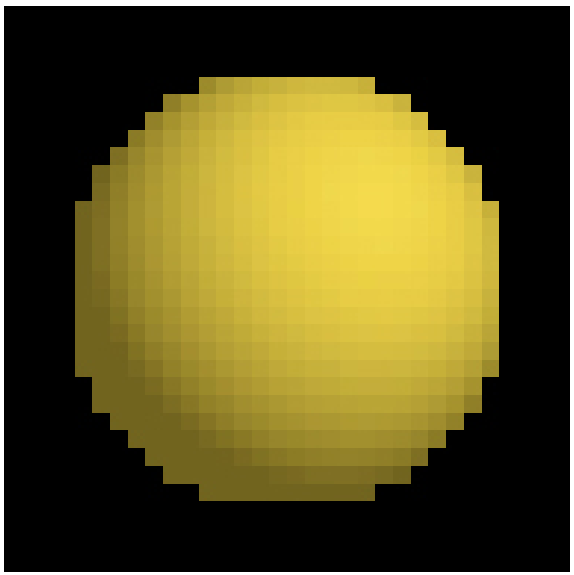
- E.g. regular grid of rays within pixel



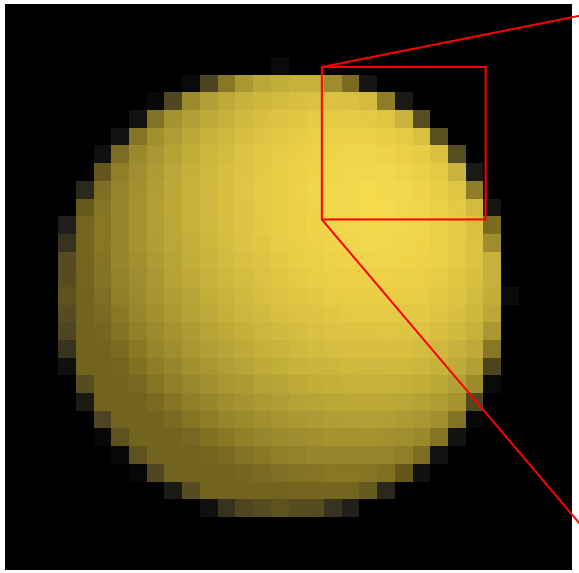
# Solutions for “aliasing”: “antialiasing”

Multiple viewing rays (“samples”) per pixel

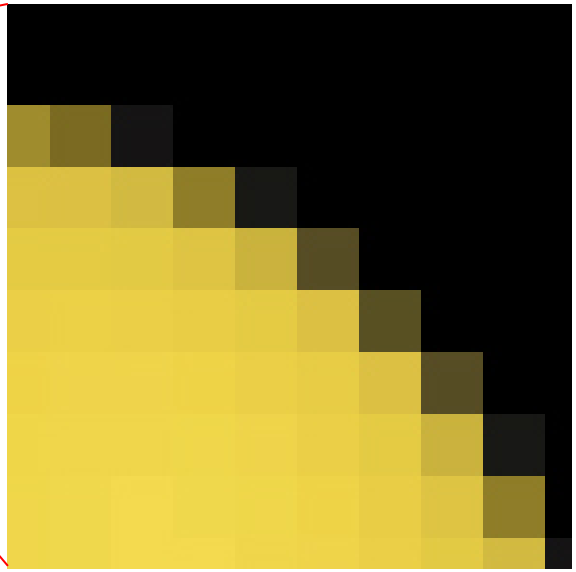
- E.g. regular grid of rays within pixel



1 ray per pixel



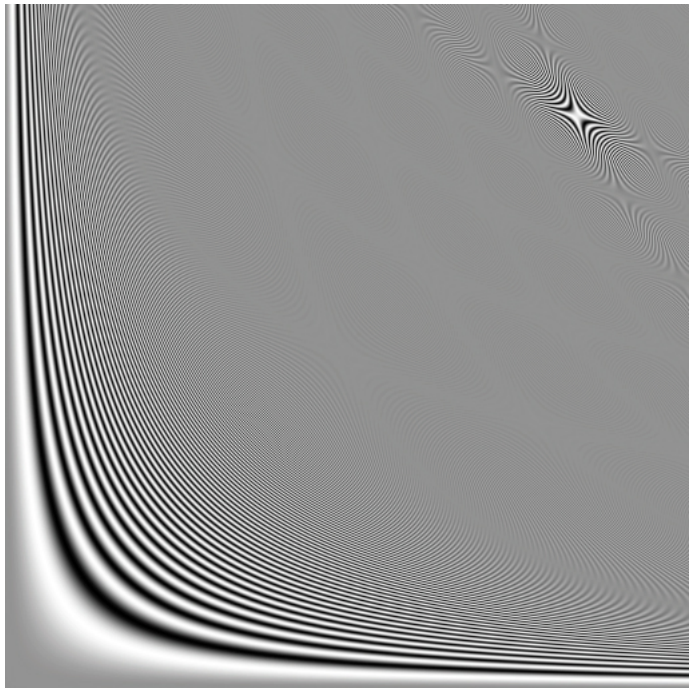
16 rays per pixel



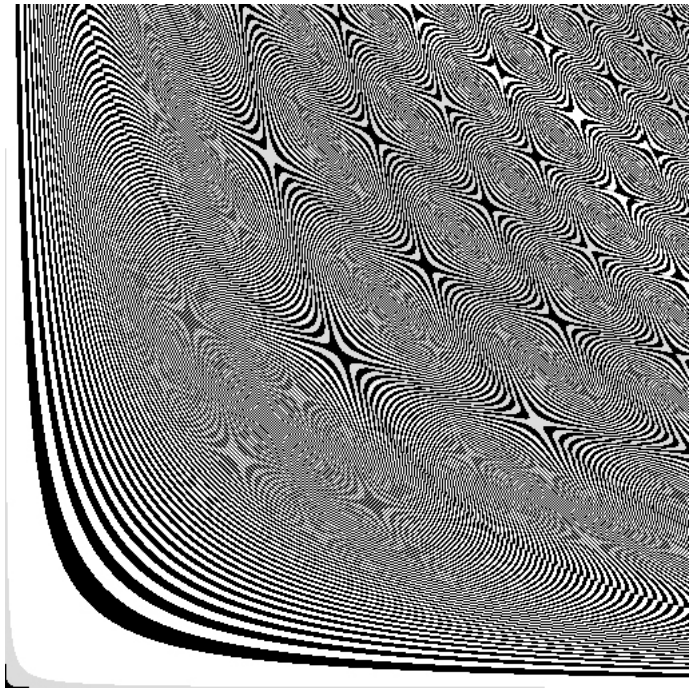
Close-up

# Solutions for “aliasing”: “antialiasing”

Regular grid: some frequencies / artefacts are still visible



(25 sample points per pixel)

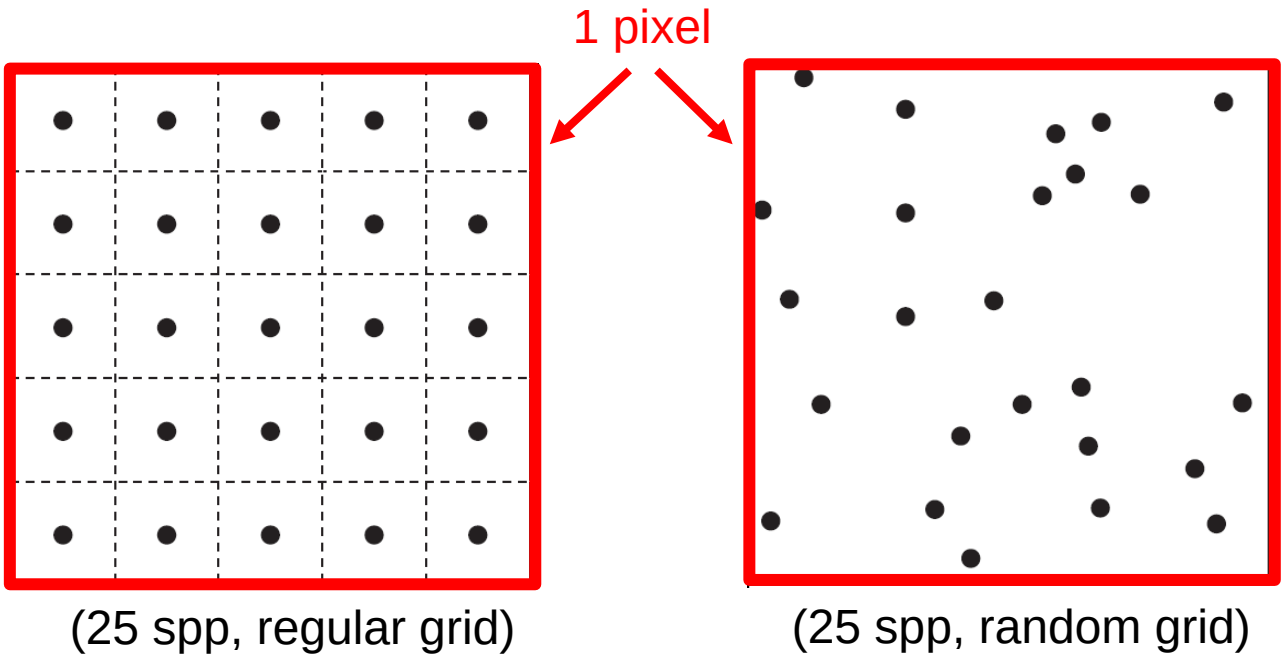


(25 sample points per pixel, contrast enhanced)

# Solutions for “aliasing”: “antialiasing”

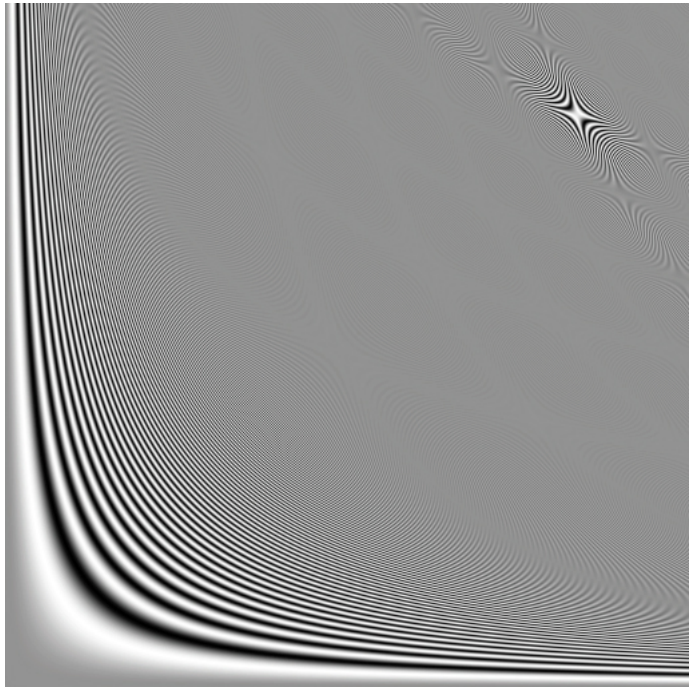
Replace regular grid by random grid

- (see also appendix on Monte Carlo integration)

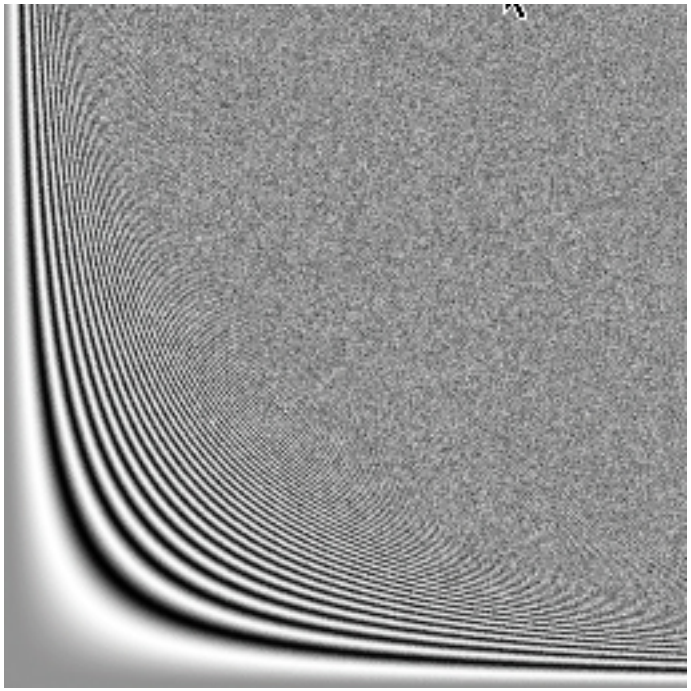


# Solutions for “aliasing”: “antialiasing”

Replace regular grid by random grid
... artefacts are traded for noise



(25 spp, regular grid)

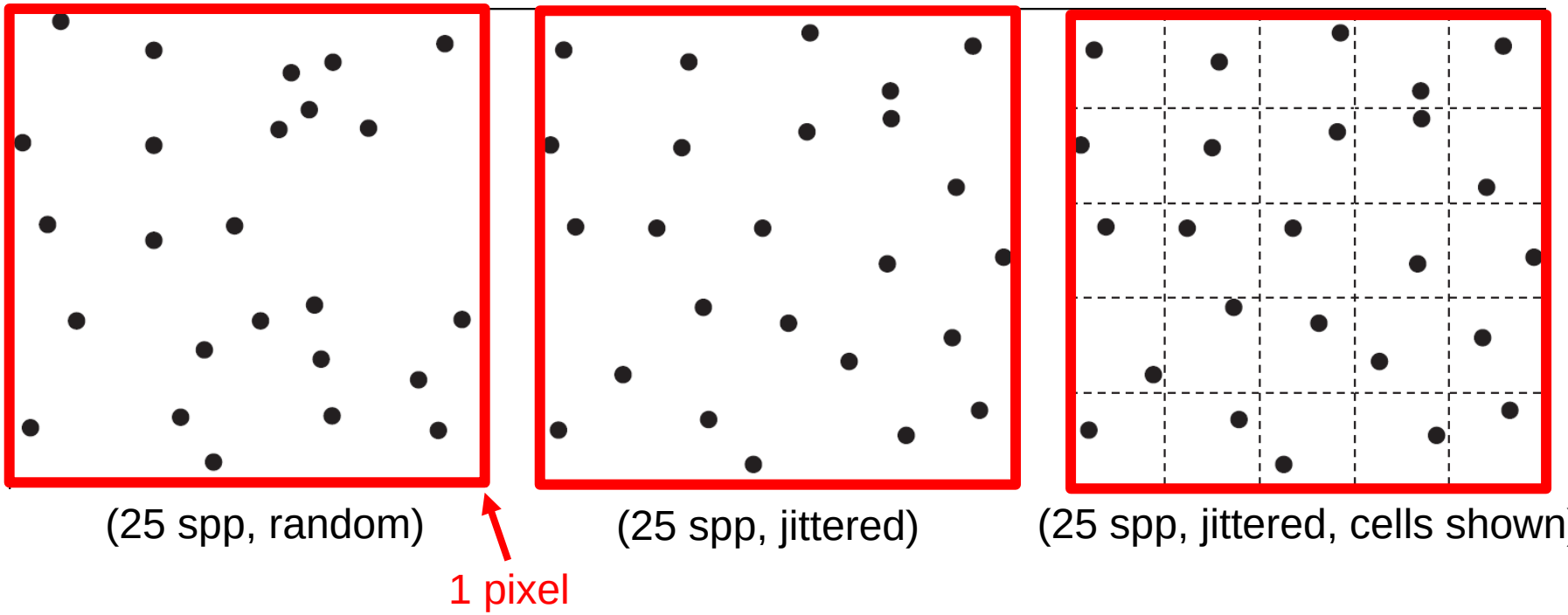


(25 spp, random grid)

# Solutions for “aliasing”: “antialiasing”

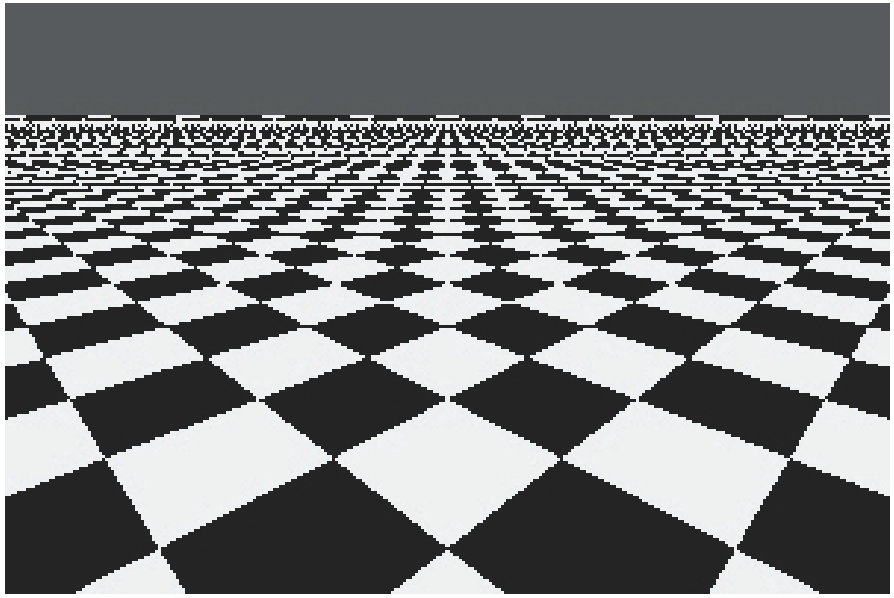
Replace regular grid by jittered grid

- “jiggle” regular points within their own cell

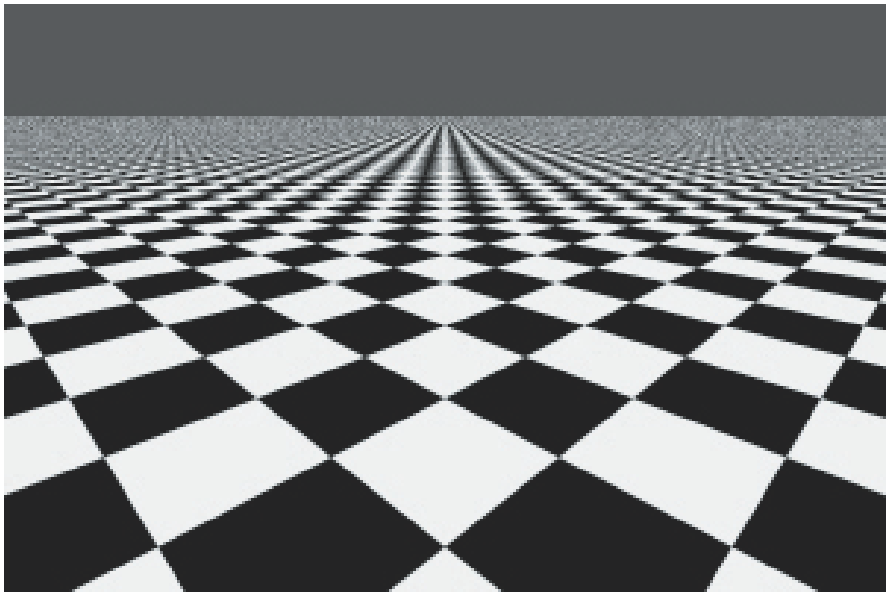


# Solutions for “aliasing”: “antialiasing”

E.g. checkerboard

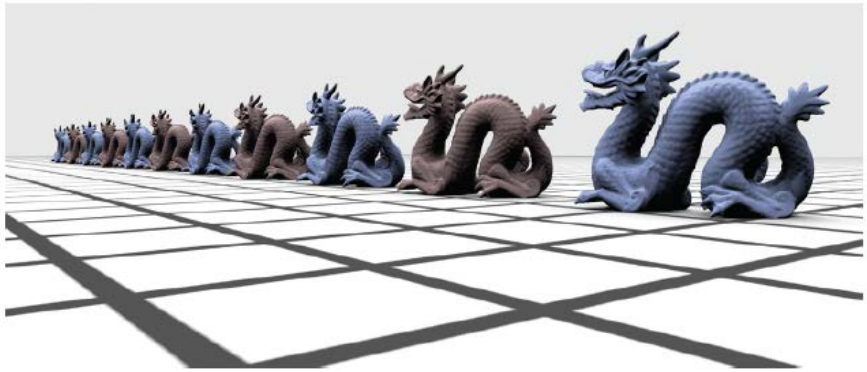


1 spp

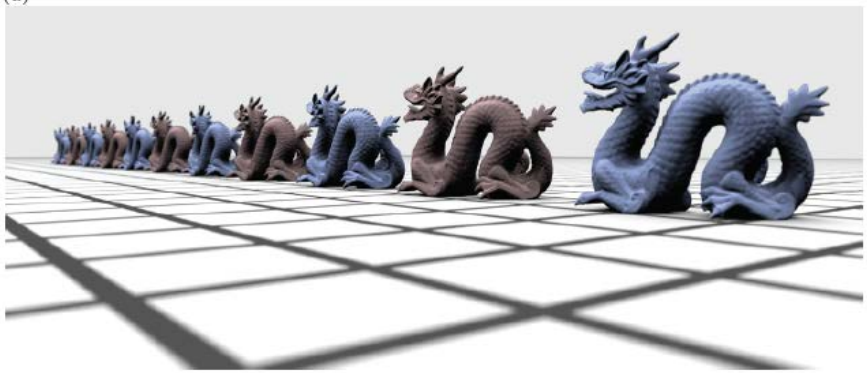


64 spp, jittered

# Depth of Field



(a)



(b)

**Figure 6.7:** (a) Scene rendered with no depth of field and (b) depth of field due to a relatively small lens aperture, which gives only a small amount of blurriness in the out-of-focus regions.

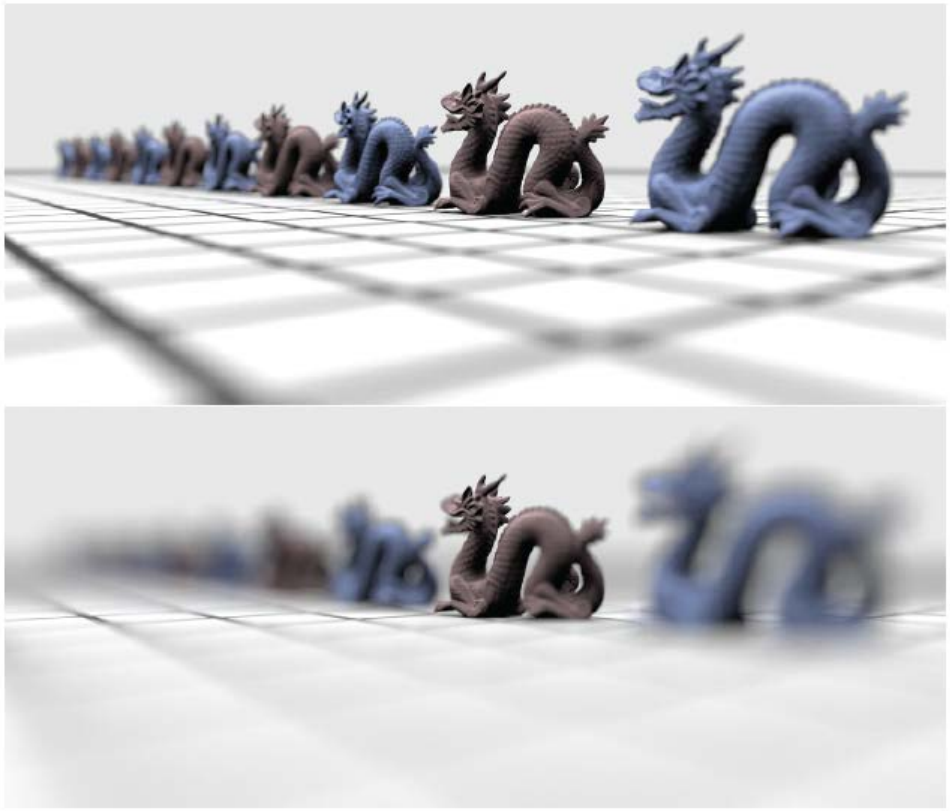
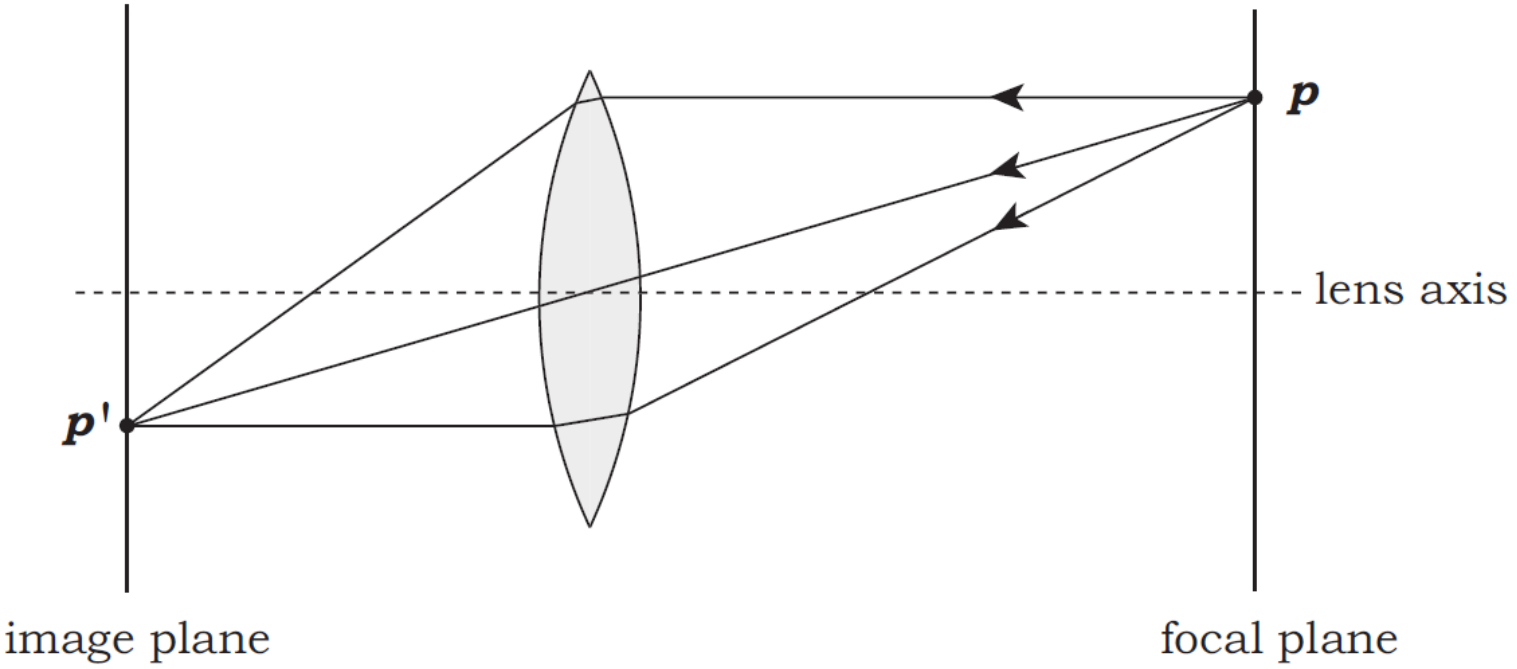


**Figure 6.8:** As the size of the lens aperture increases, the size of the circle of confusion in the out-of-focus areas increases, giving a greater amount of blur on the image plane.

# Depth of Field

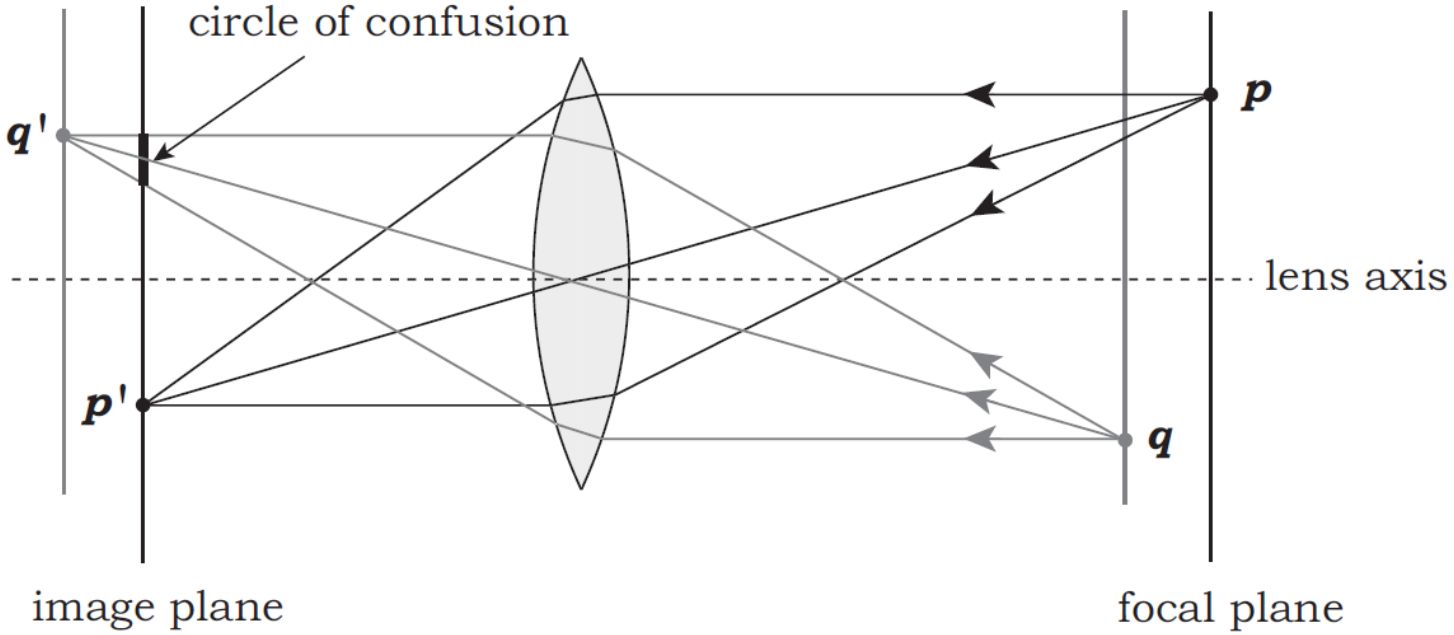
Thin-lens theory



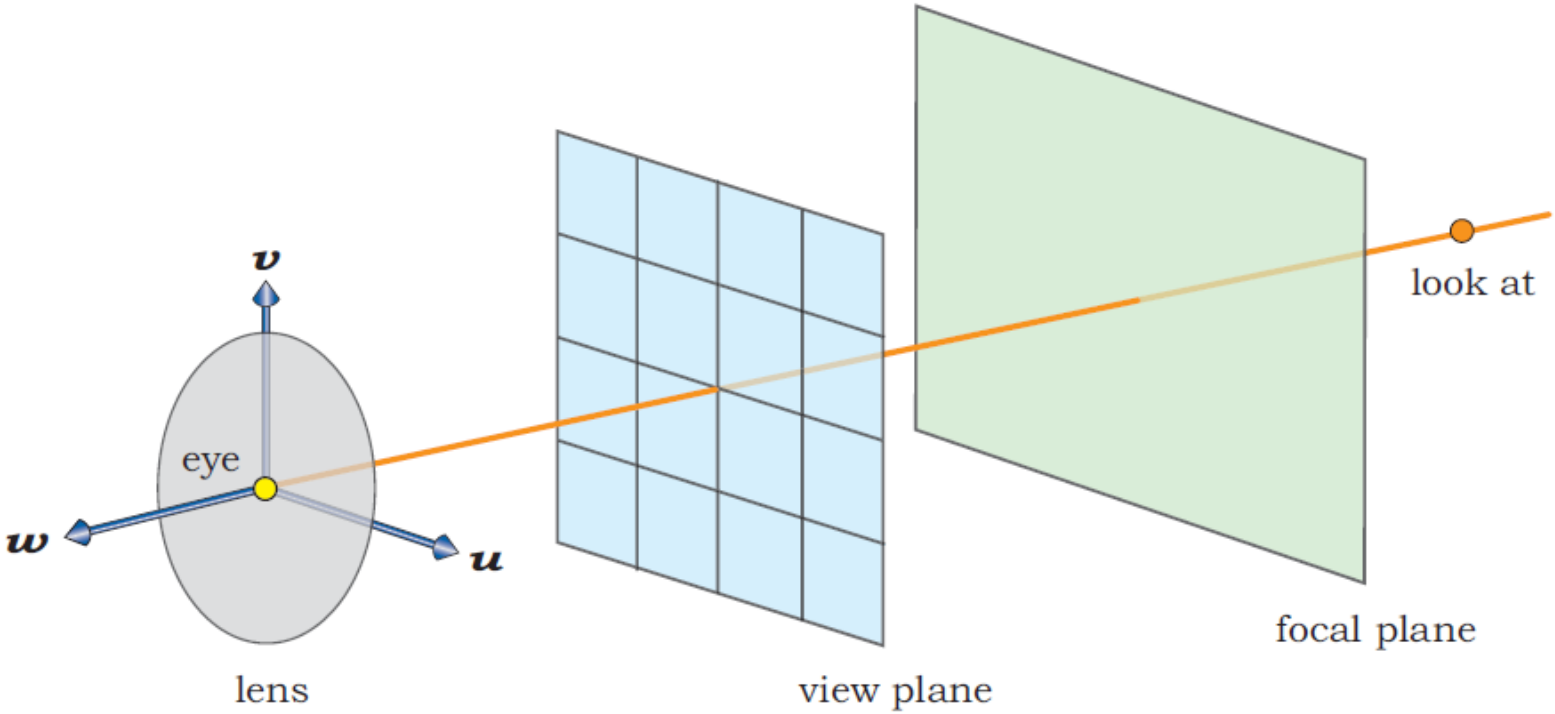
# Depth of Field

Depth of field = range of distances in which scene is in focus

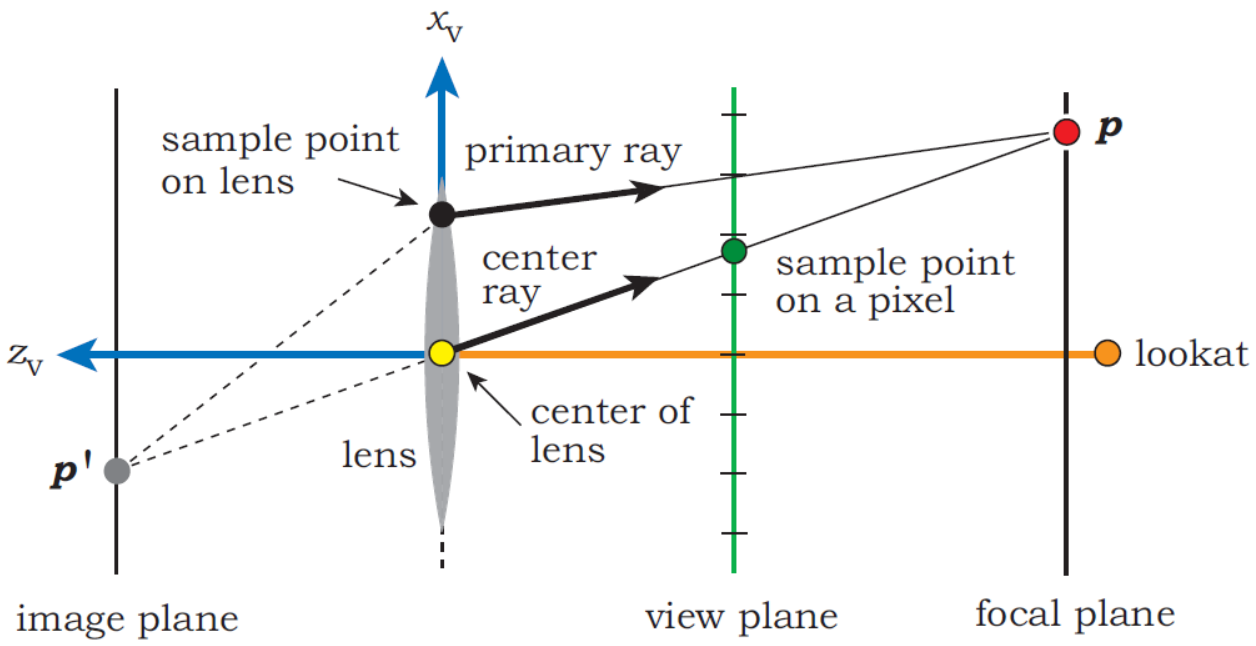
- Circle of confusion < area pixel



# Depth of Field in graphics



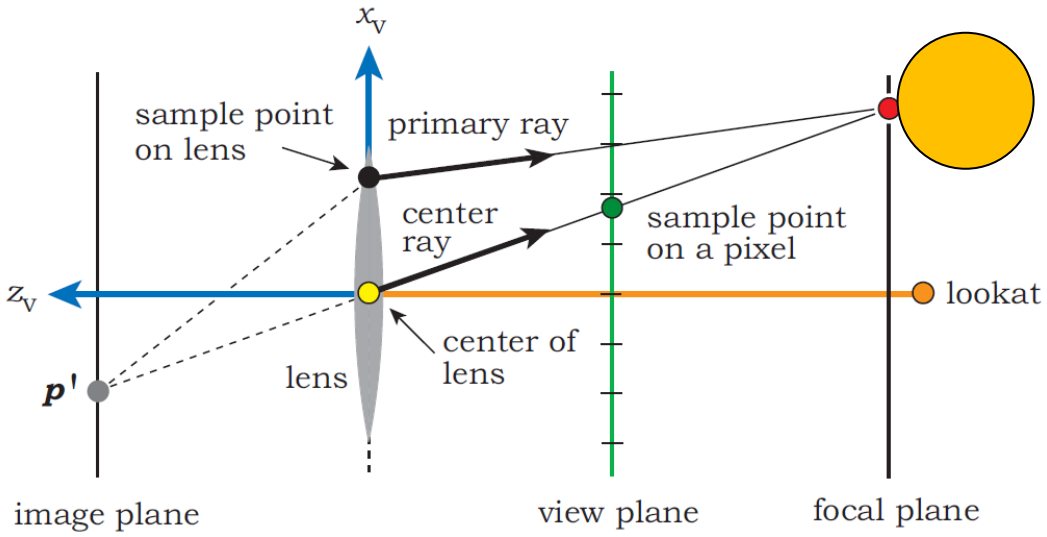
# Depth of Field in graphics




- Centre ray is used only to compute p
- Refraction through lens simulated
- No or limited Anti-Aliasing

# Depth of Field in graphics

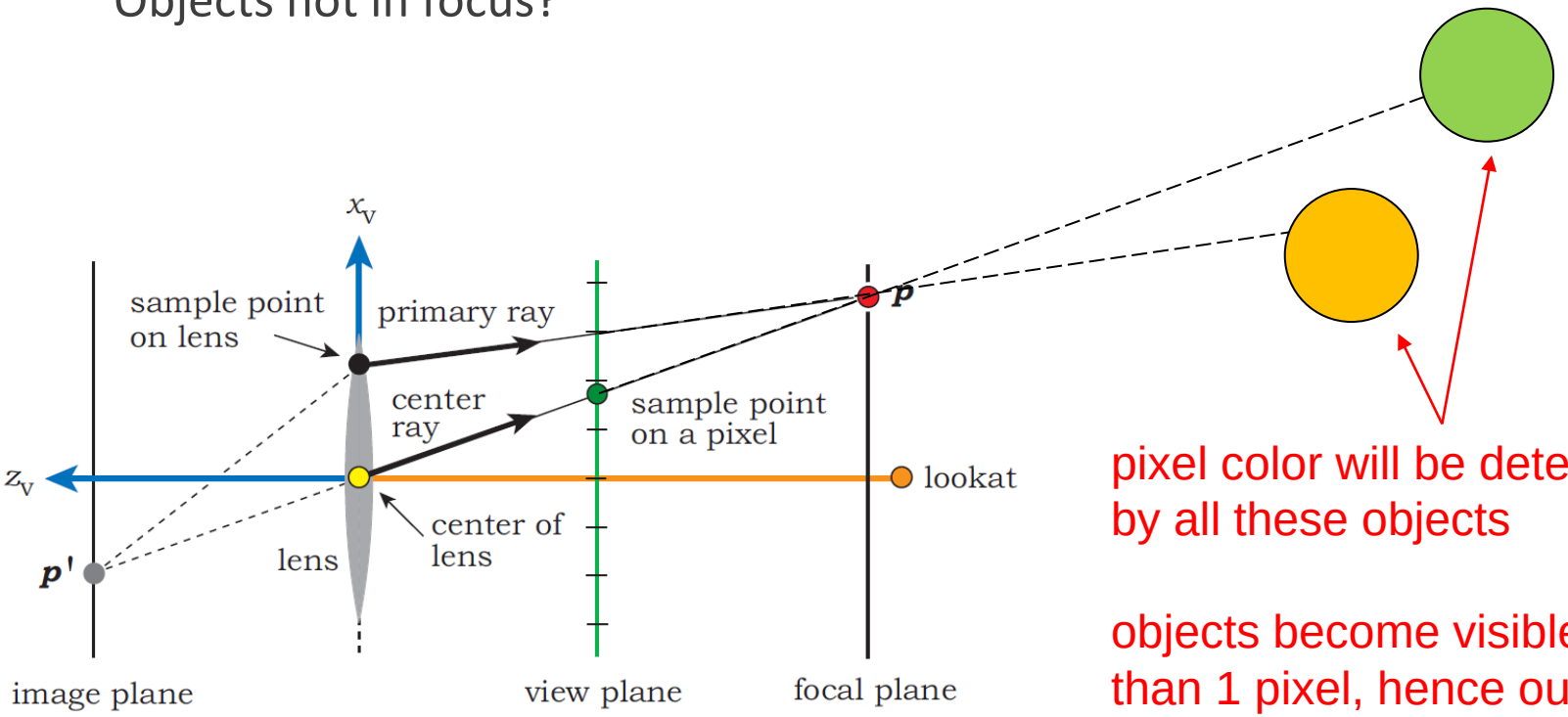
Object in focus?




pixel color determined by same object, irrespective of position on lens

# Depth of Field in graphics

Objects not in focus?

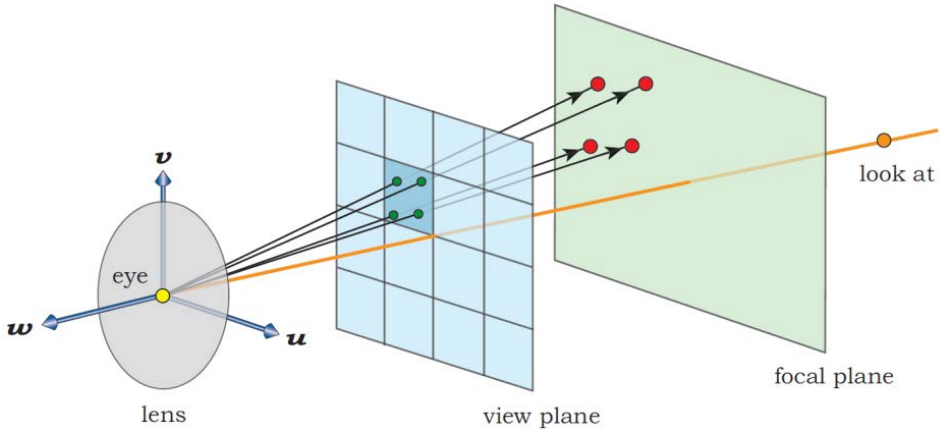



pixel color will be determined by all these objects

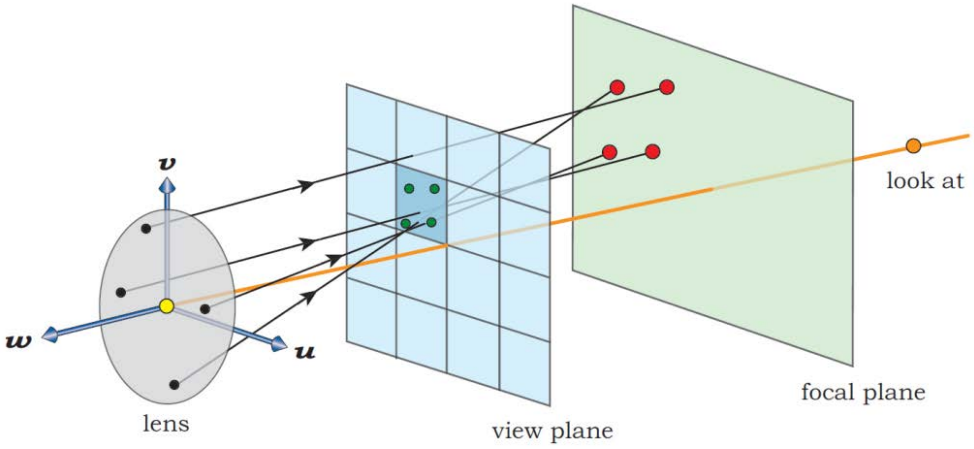
objects become visible in more than 1 pixel, hence out-of-focus

# Depth of Field in graphics

Anti-Aliasing: use different centre ray for each primary ray

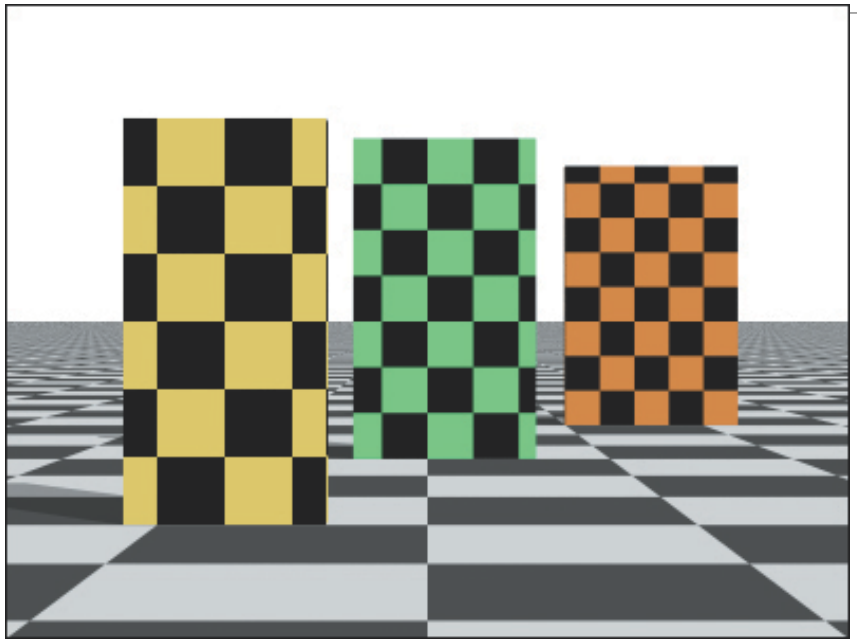



Center rays, used to compute points on focal plane

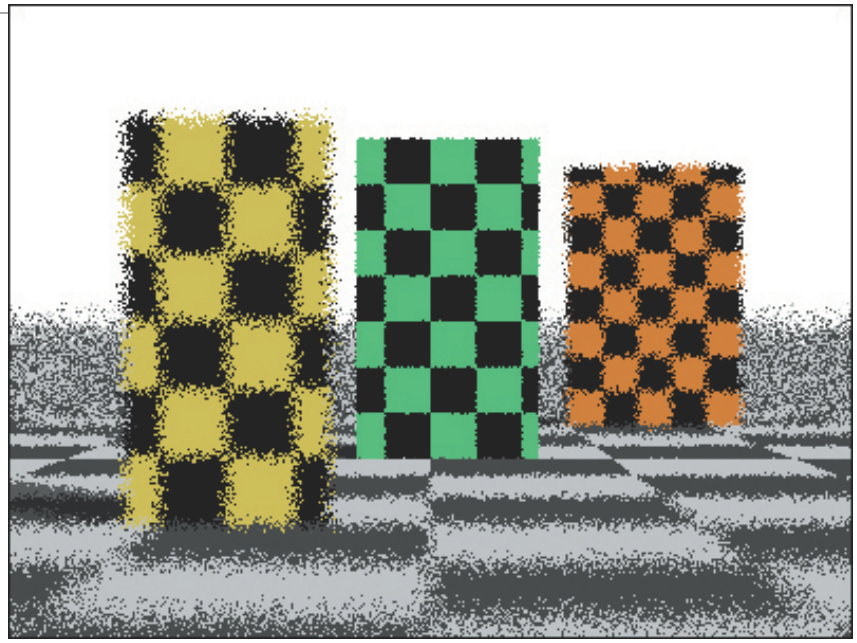



Primary rays, starting from random location on lens, direction determined by points on focal plane

# Depth of Field: results

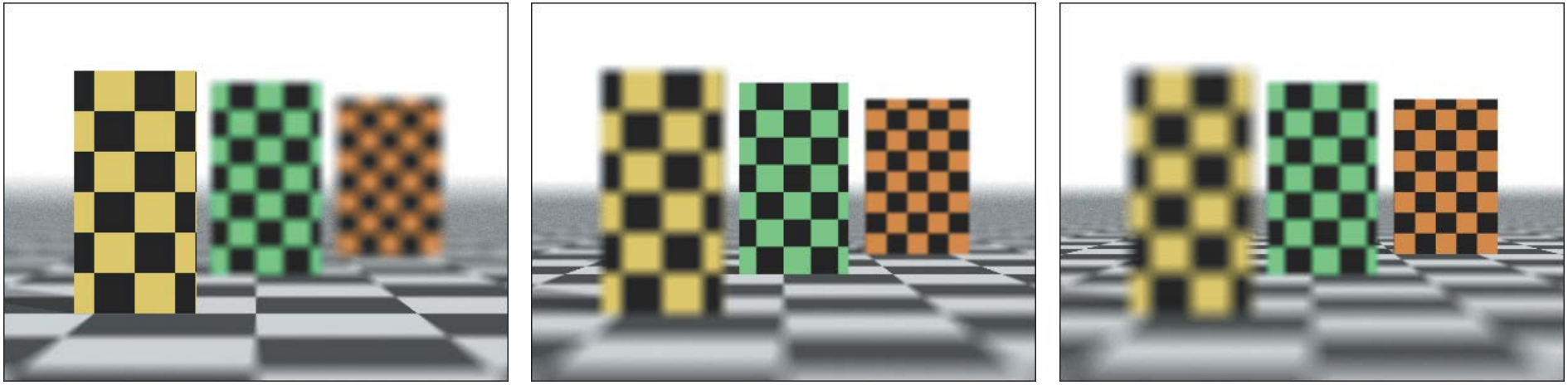


(pinhole camera)



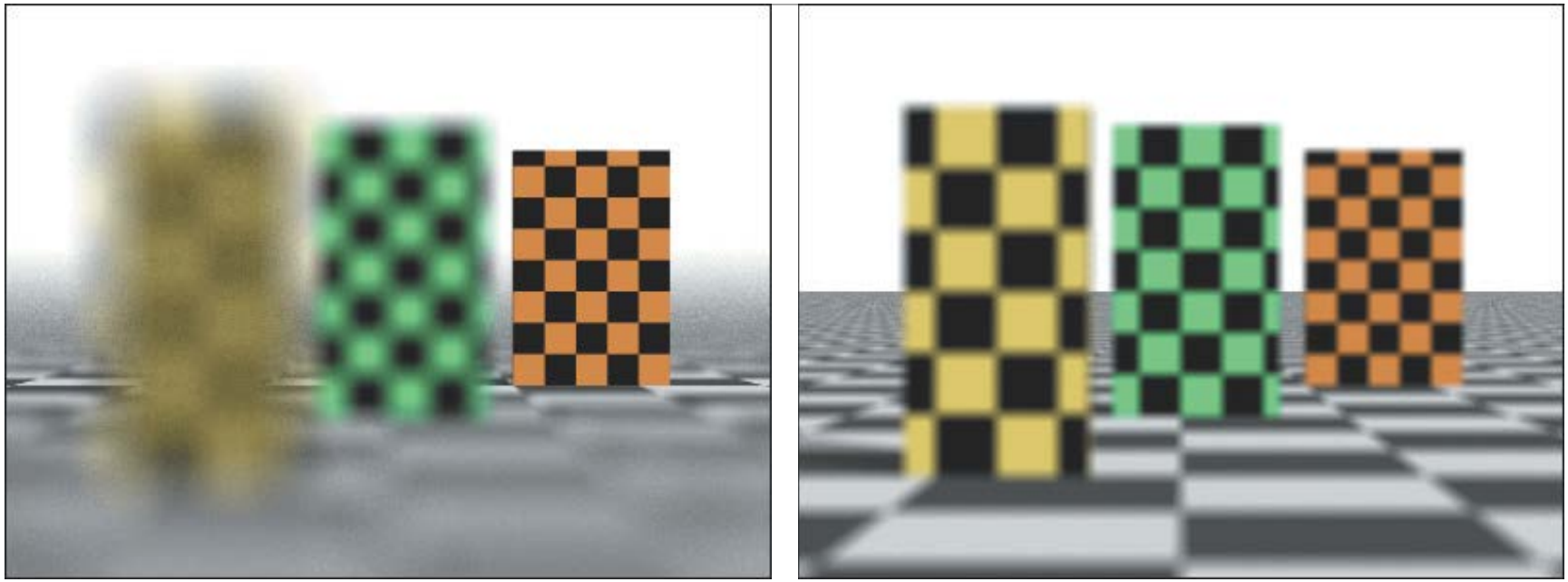
(1spp, depth of field)

# Depth of Field: results



(different focal distances)

# Depth of Field: results



(different lens radius)