TIEA311
Tietokonegrafiikan perusteet
kevät 2018

(“Principles of Computer Graphics” – Spring 2018)

TIEA311 Tietokonegrafiikan perusteet – kevät 2018 (“Principles of Computer Graphics” – Spring 2018)

Adapted from: *Wojciech Matusik*, and *Frédo Durand*: 6.837 Computer Graphics. Fall 2012. Massachusetts Institute of Technology: MIT OpenCourseWare, https://ocw.mit.edu/.

Frontpage of the local course version, held during Spring 2018 at the Faculty of Information technology, University of Jyväskylä:
`http://users.jyu.fi/~nieminen/tgp18/`

TIEA311 - Today in Jyväskylä (in Finnish)

The “steps of Jarno” (Ajattelumallia tehtävien ratkaisuun):

1. Luentomateriaali
2. Tehtävänanto (muista mitä aiemmissa tehtävissä on tehty/annettu)
3. Hae lähdekoodi ja testaa sen toiminta
4. Yhdistä teoria tehtävään ja lähdekoodiin, ymmärrä kokonaisuus
5. Hahmottele kevyt ”speksi” esim. paperille UML, prosessikaavio, ...

   ---

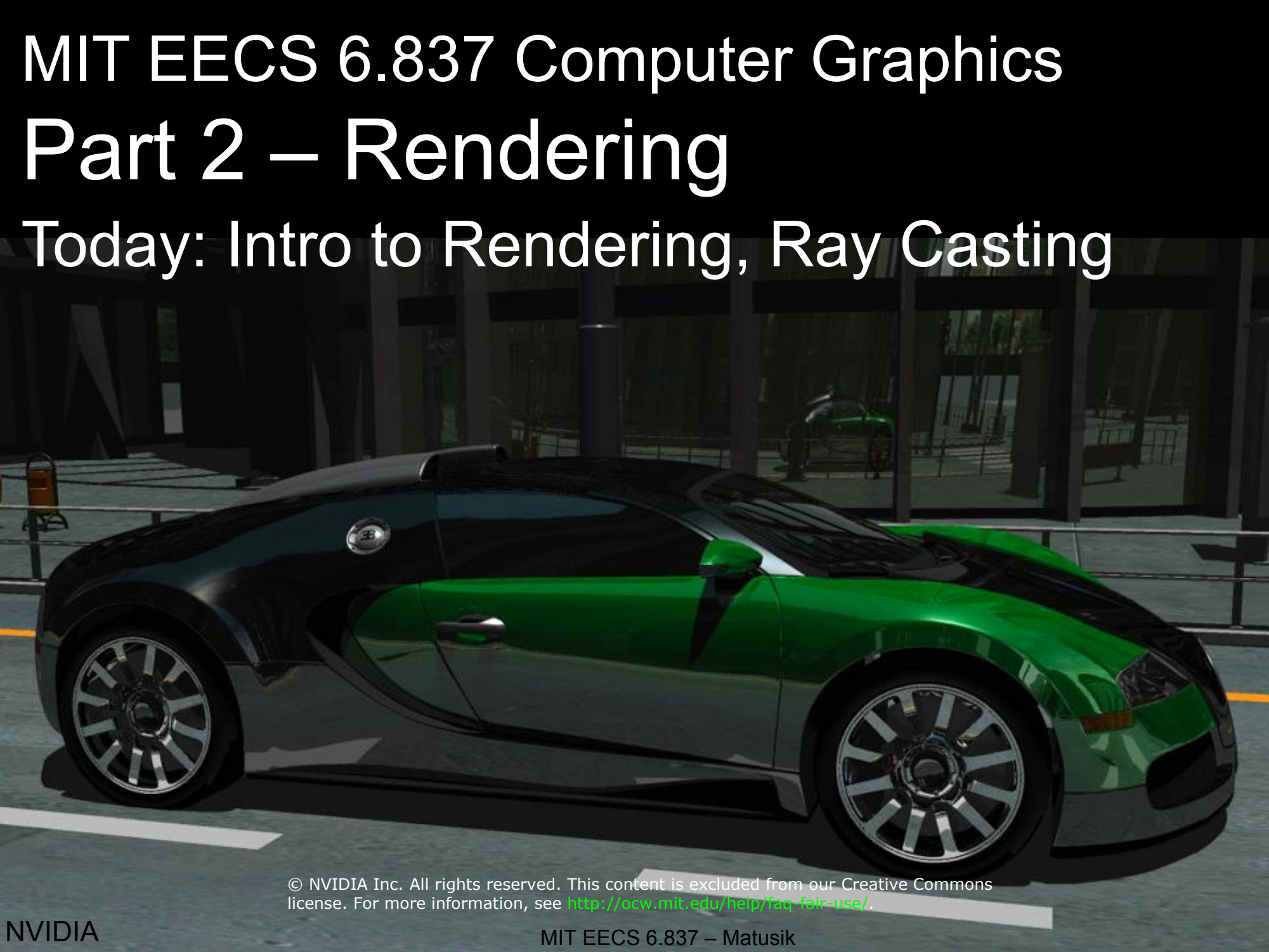
6. Tee osatehtävä 1
7. Päivitä ”speksi”
8. Tee osatehtävä 2
9. Päivitä ”speksi”

   ...

MIT EECS 6.837 Computer Graphics

Part 2 – Rendering

Today: Intro to Rendering, Ray Casting



NVIDIA

The Story So Far

- Modeling
  - splines, hierarchies, transformations, meshes, etc.
- Animation
  - skinning, ODEs, masses and springs
- **Now we'll to see how to generate an image given a scene description!**

TIEA311 - Rest of the Course in Jyväskylä

The next slide is verbatim from the MIT OCW course. Here is the current plan for the remaining part of our adapted short version:

- Ray Casting: cover fully. Ray Tracing: cover the basic idea, skip details $\rightarrow$ possible to continue as a “hobby project”; NOTE: teachers of “TIEA306 Ohjelmointityö” may be contacted regarding the possibility of receiving credit for hobby projects.
- Shading, texture mapping: Cover the principles up to Phong model and texture coordinates.
- Rasterization, z-buffering: cover basic ideas
- Global Illumination, Monte Carlo techniques: skip this $\rightarrow$ could be suitable for bachelor thesis (introductory topics) or even master thesis (current state-of-the-art techniques)
- Image-based rendering: skip $\rightarrow$ defer to advanced courses (e.g. “TIES411 Konenäkö ja kuva-analyysi”)
- Sampling and antialiasing: mostly skip $\rightarrow$ defer theory to “TIES324 Signaalinkäsittely” and techniques (possibly) to “TIES471 Reaaliaikainen renderöinti”.
- Shadow techniques, Graphics Hardware $\rightarrow$ defer to TIES471.

The Remainder of the Term

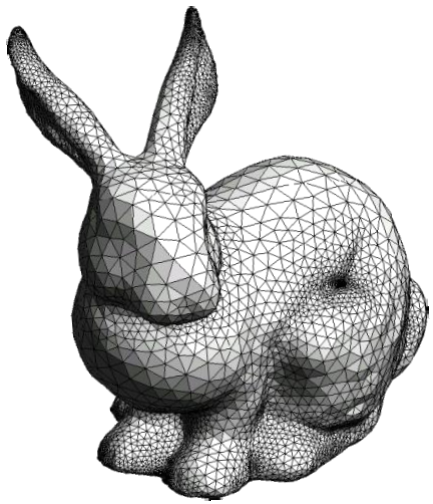
- Ray Casting and Ray Tracing
- Intro to Global Illumination
  - Monte Carlo techniques, photon mapping, etc.
- Shading, texture mapping
  - What makes materials look like they do?
- Image-based Rendering
- Sampling and antialiasing
- Rasterization, z-buffering
- Shadow techniques
- Graphics Hardware

[Lehtinen et al. 2008]

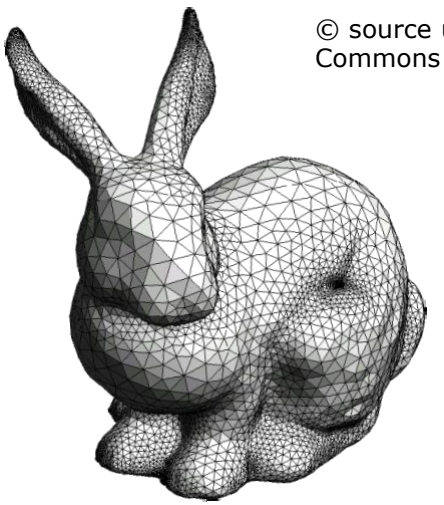
Today

- What does *rendering* mean?

- Basics of ray casting



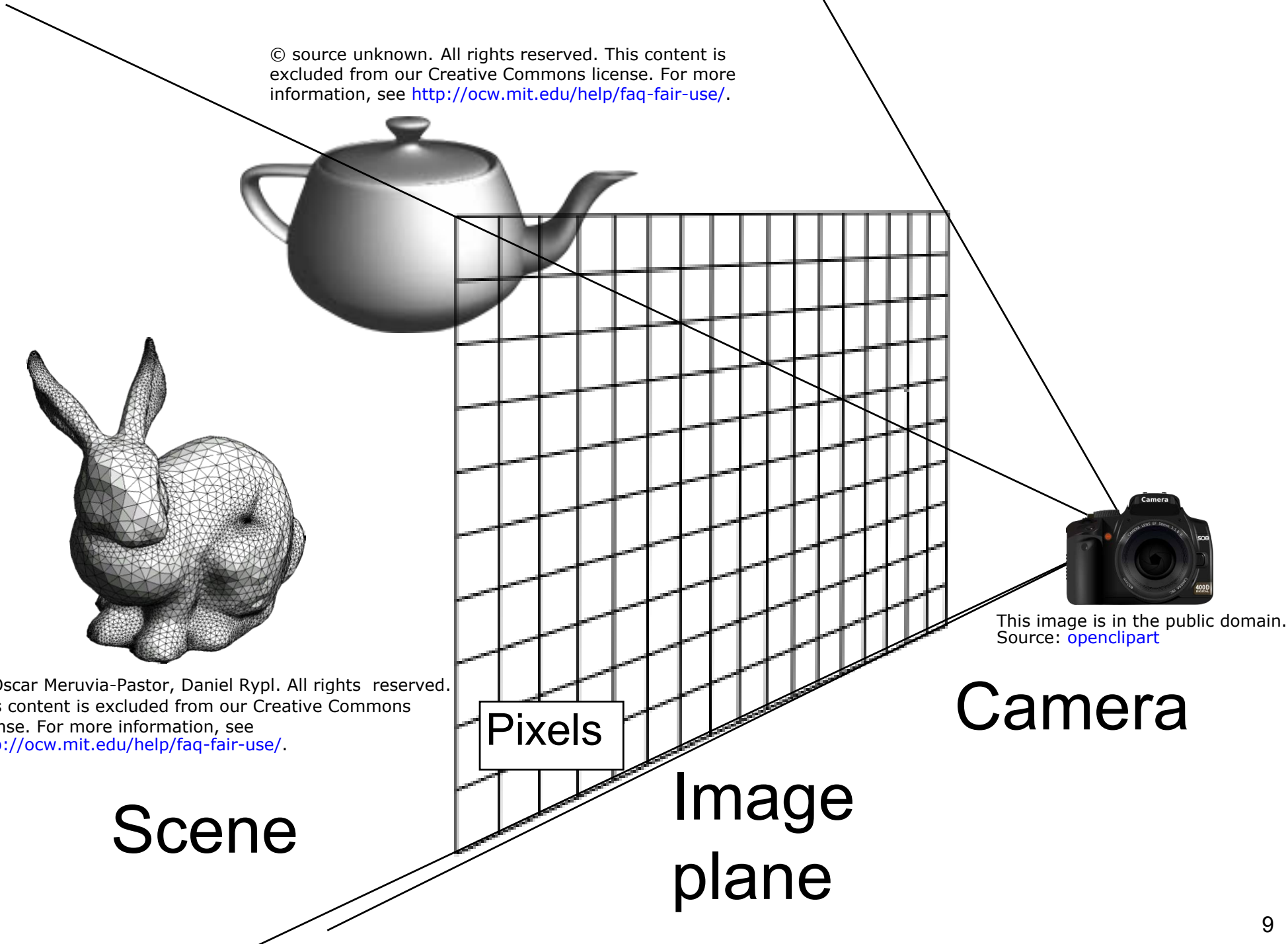
Scene



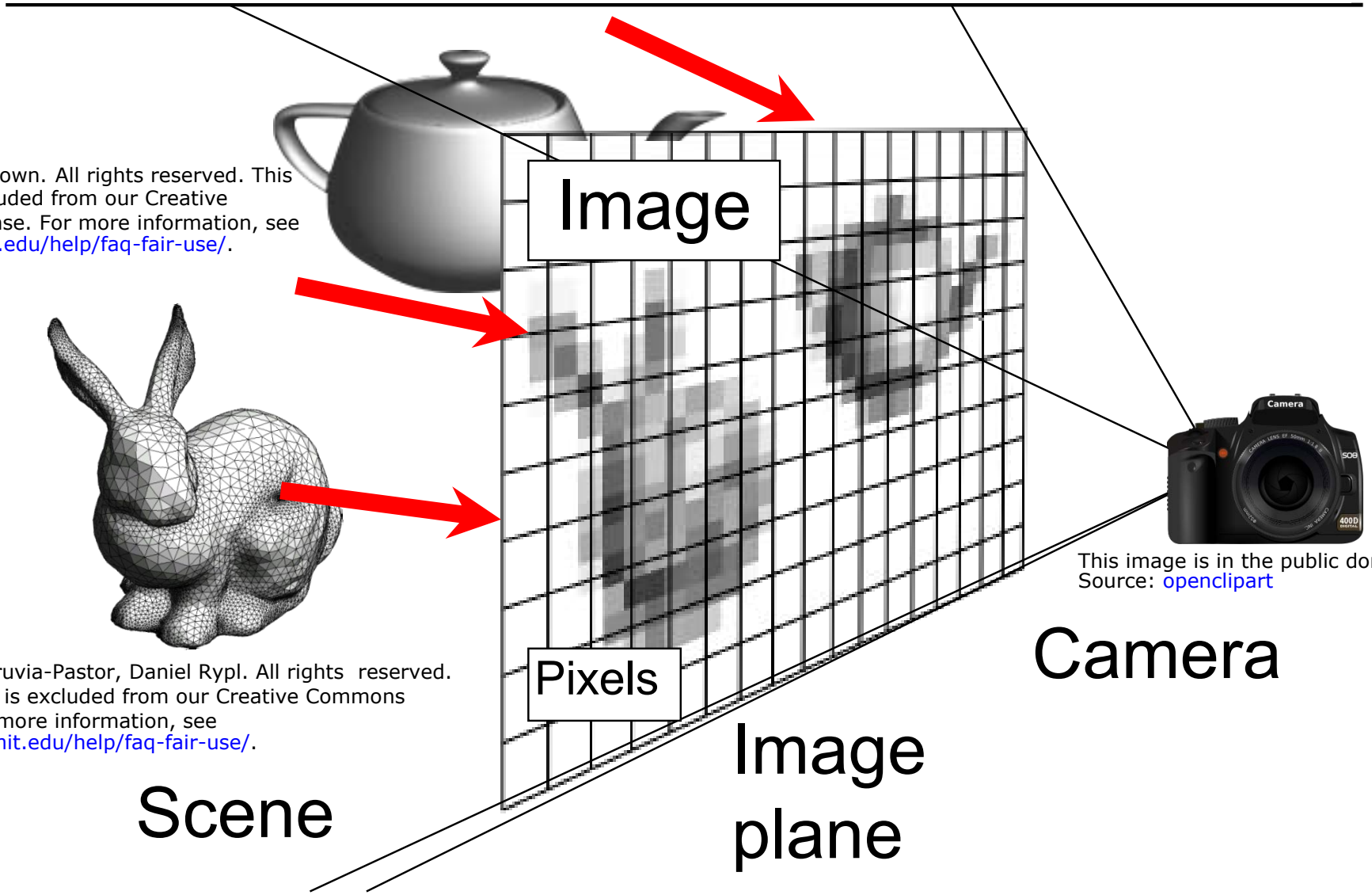
Scene

This image is in the public domain.
Source: openclipart

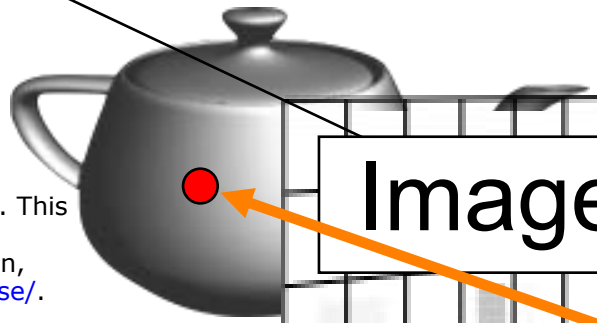
Camera


© source unknown. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
Camera
This image is in the public domain.
Source: openclipart
© Oscar Meruvia-Pastor, Daniel Rypl. All rights reserved. This content is excluded from our Creative Commons license. For more information, see http://ocw.mit.edu/help/faq-fair-use/.
Pixels
Camera
Scene
Image plane

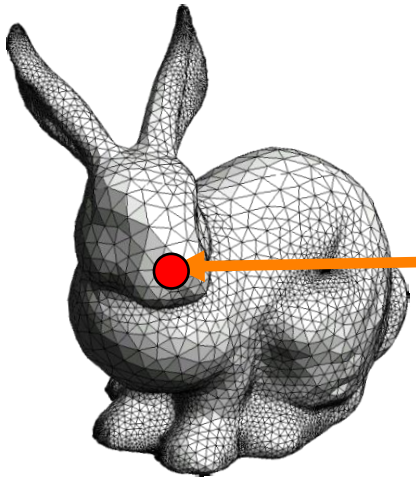
Rendering = Scene to Image



Rendering – Pinhole Camera



Image



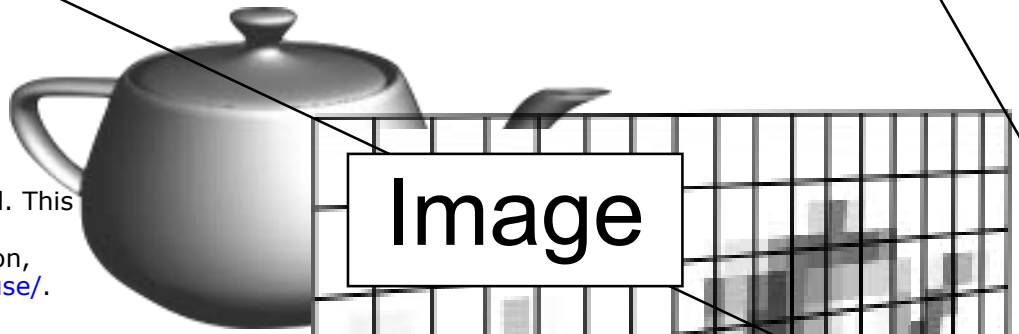
This image is in the public domain.
Source: openclipart

Pixels

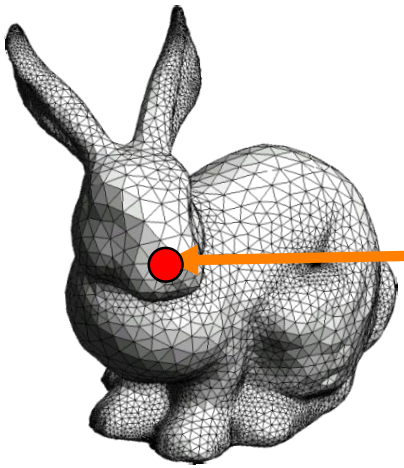
Scene

Each pixel corresponds to one ray. We need to figure out which scene point each one hits.

Rendering



Image



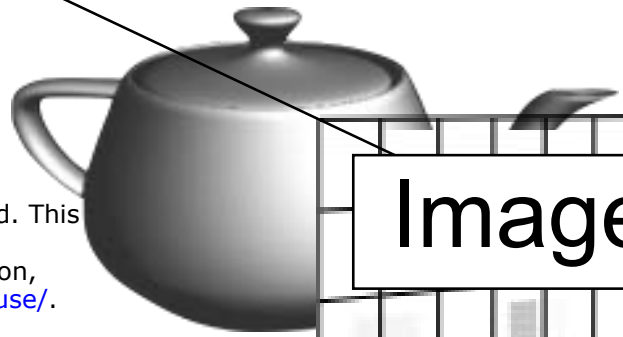
This image is in the public domain.
Source: openclipart

Pixels

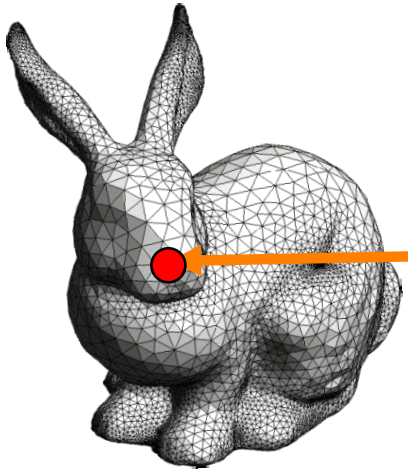
Scene

What's the color you put in each pixel?

Rendering



Image



This image is in the public domain. Source: openclipart

Pixels

Pixel Color Determined by *Lighting/Shading*

Scene

Rendering

- "Rendering" refers to the entire process that produces color values for pixels, given a 3D representation of the scene
- Pixels correspond to rays; need to figure out the **visible** scene point along each ray
  - Called "hidden surface problem" in older texts
  - "Visibility" is a more modern term
  - Also, we assume (for now) a single ray per pixel

Rendering

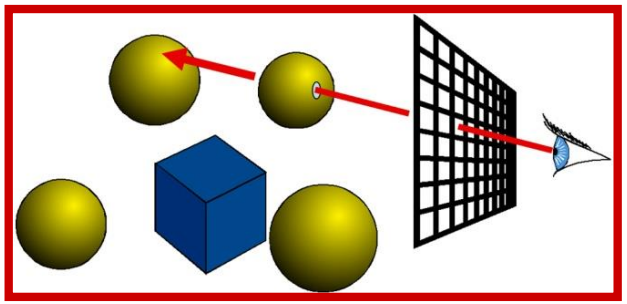
- "Rendering" refers to the entire process that produces color values for pixels
- Pixels correspond to rays; need to figure out the **visible** scene point along each ray
  - Called "hidden surface problem" in older texts
  - "Visibility" is a more modern term
  - Also, we assume (for now) a single ray per pixel
- Major algorithms: **Ray casting and rasterization**

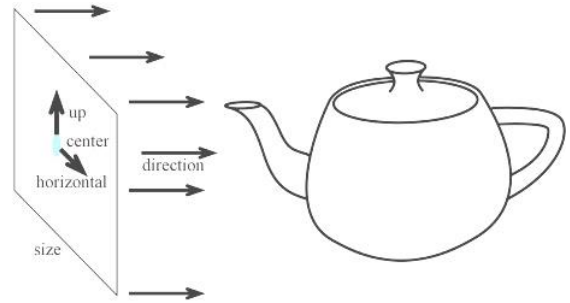
- Note: We are assuming a pinhole camera (for now)

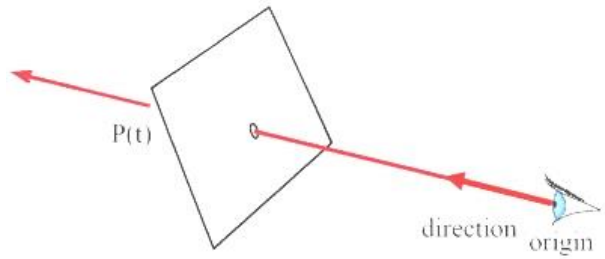
Questions?

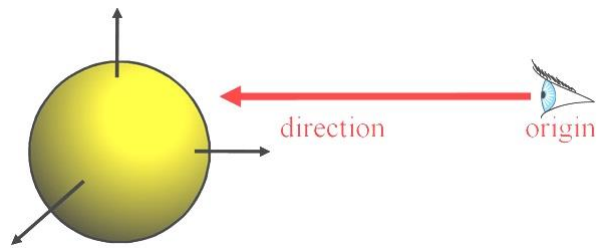
Ray Casting

- Ray Casting Basics
- Camera and Ray Generation
- Ray-Plane Intersection
- Ray-Sphere Intersection



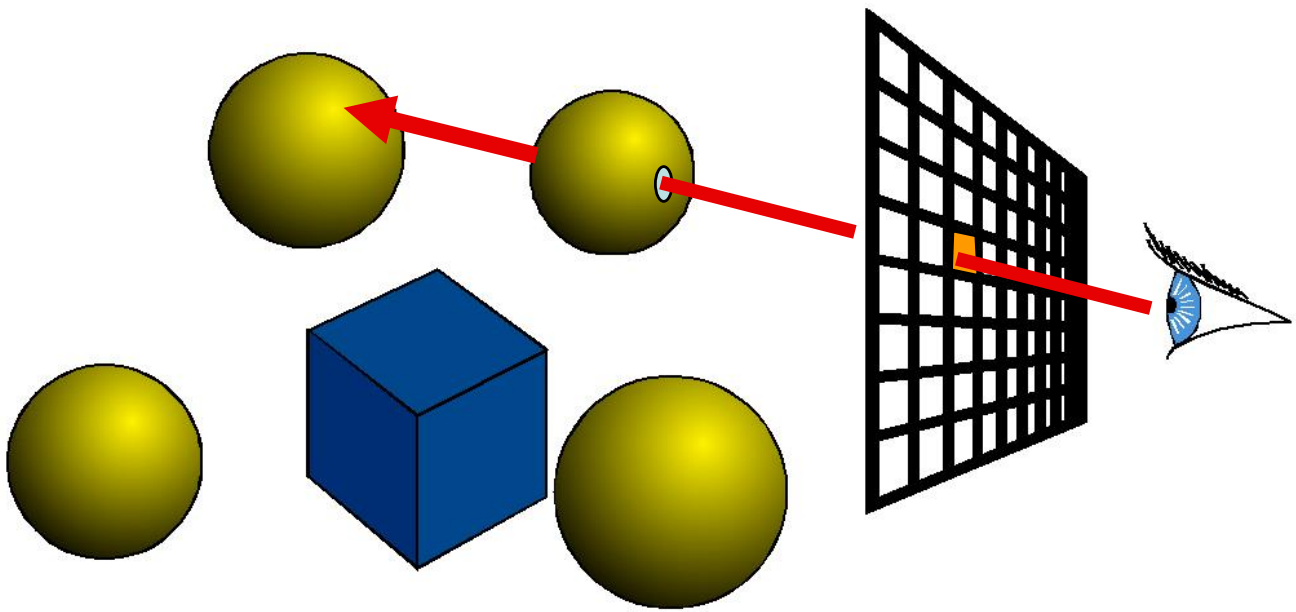






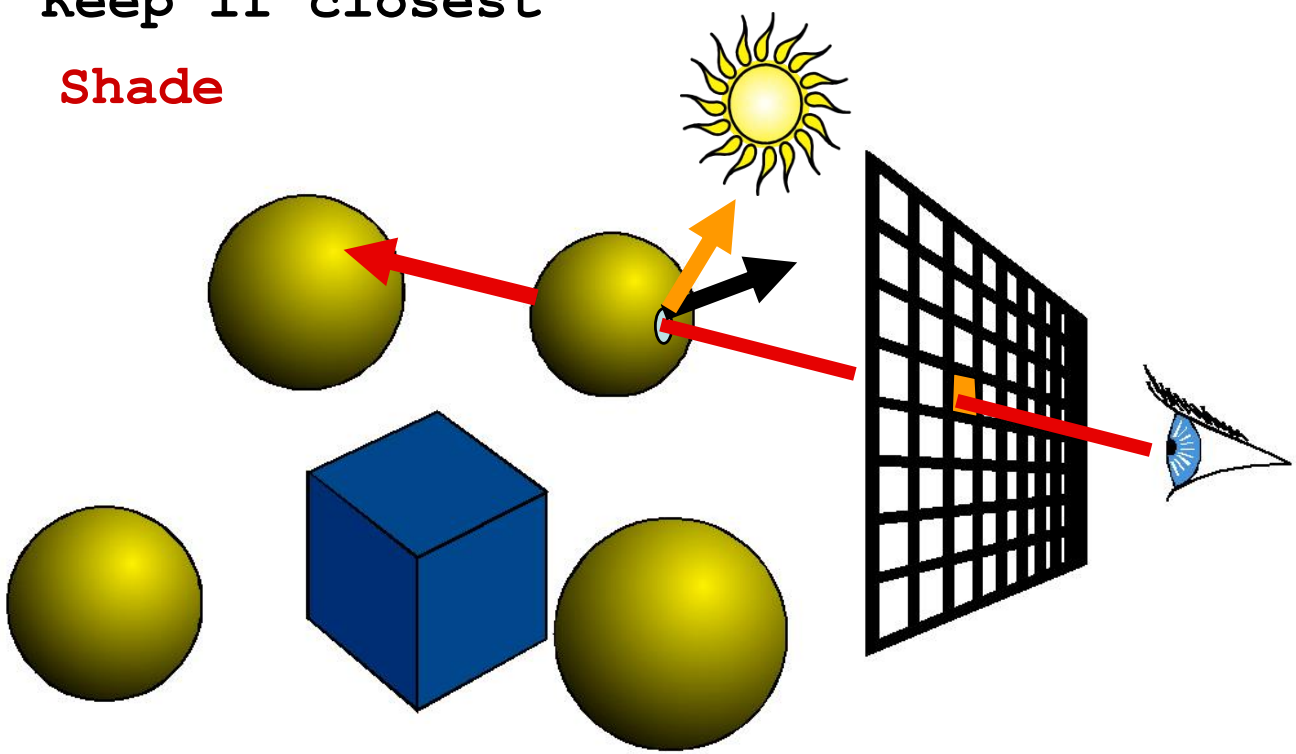
Ray Casting

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
```



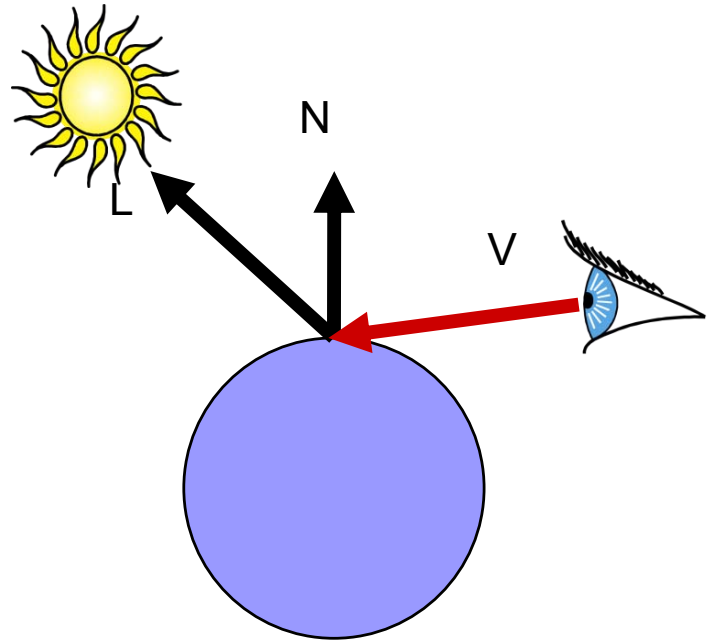
Shading

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
        Shade
```

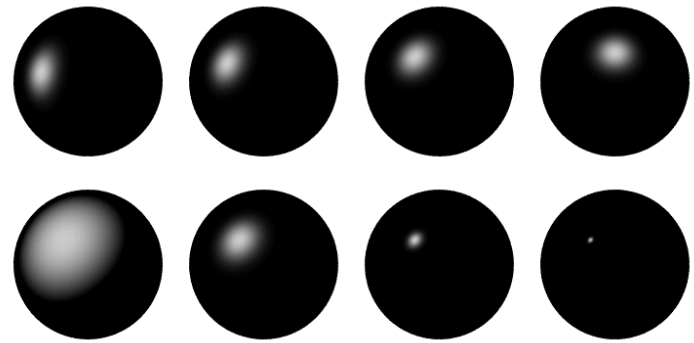


Shading = What Surfaces Look Like

- Surface/Scene Properties
  - surface normal
  - direction to light
  - viewpoint
- Material Properties
  - Diffuse (matte)
  - Specular (shiny)
  - …
- Light properties
  - Position
  - Intensity, ...
- Much more!



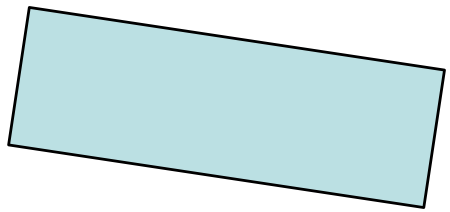
Diffuse sphere

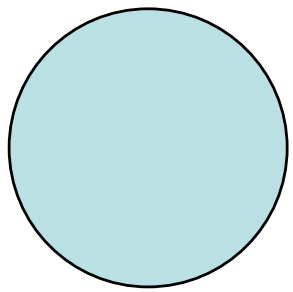


Specular spheres

Ray Casting vs. Ray Tracing

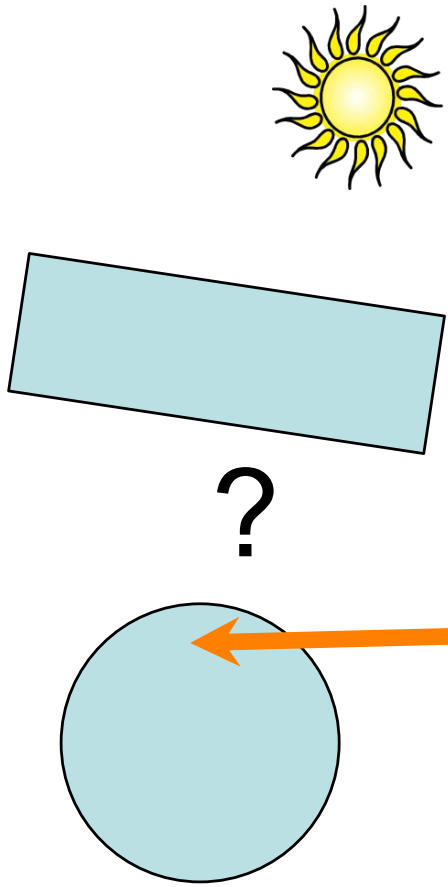
- Let's think about shadows...





This image is in the public domain.
Source: openclipart

Ray Casting vs. Ray Tracing

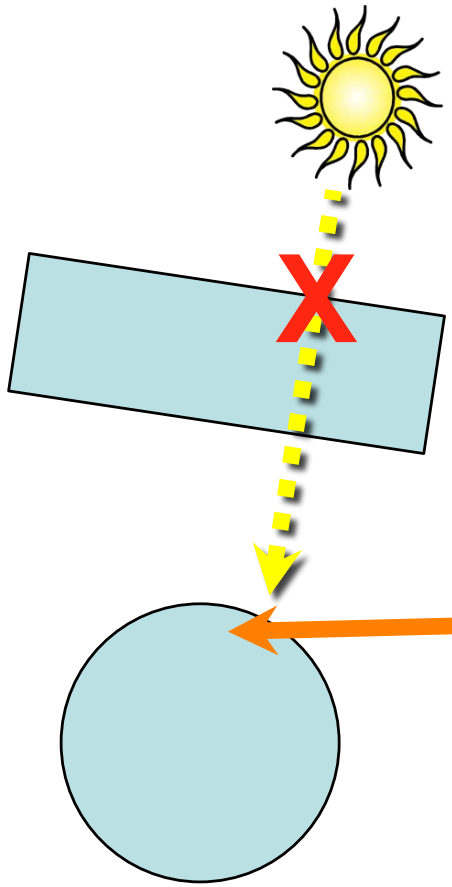


?

This image is in the public domain.
Source: openclipart

"camera rays" are rays from the camera to the scene

Ray Casting vs. Ray Tracing

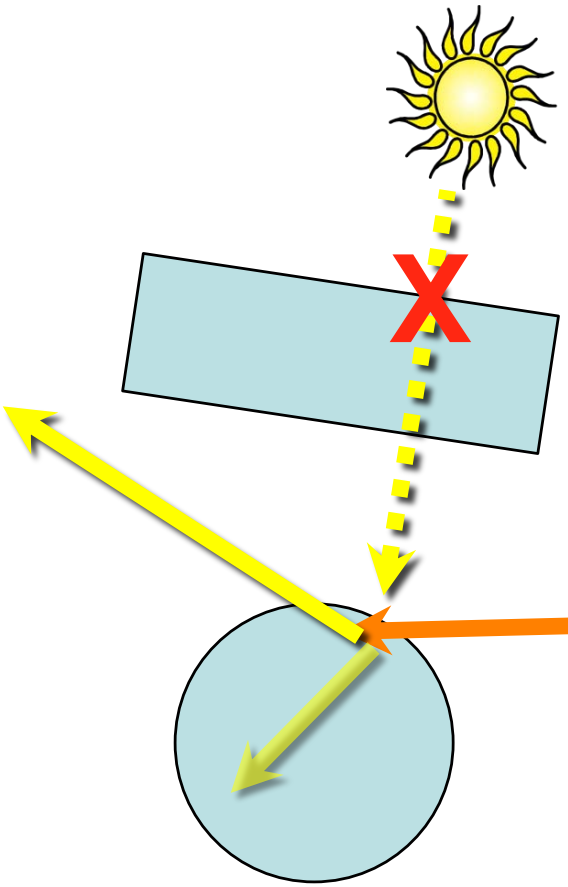


ray from light to hit point is blocked, i.e., **point is in shadow**

This image is in the public domain.
Source: openclipart

Ray Casting vs. Ray Tracing

- Ray casting = eye rays only, tracing = also secondary

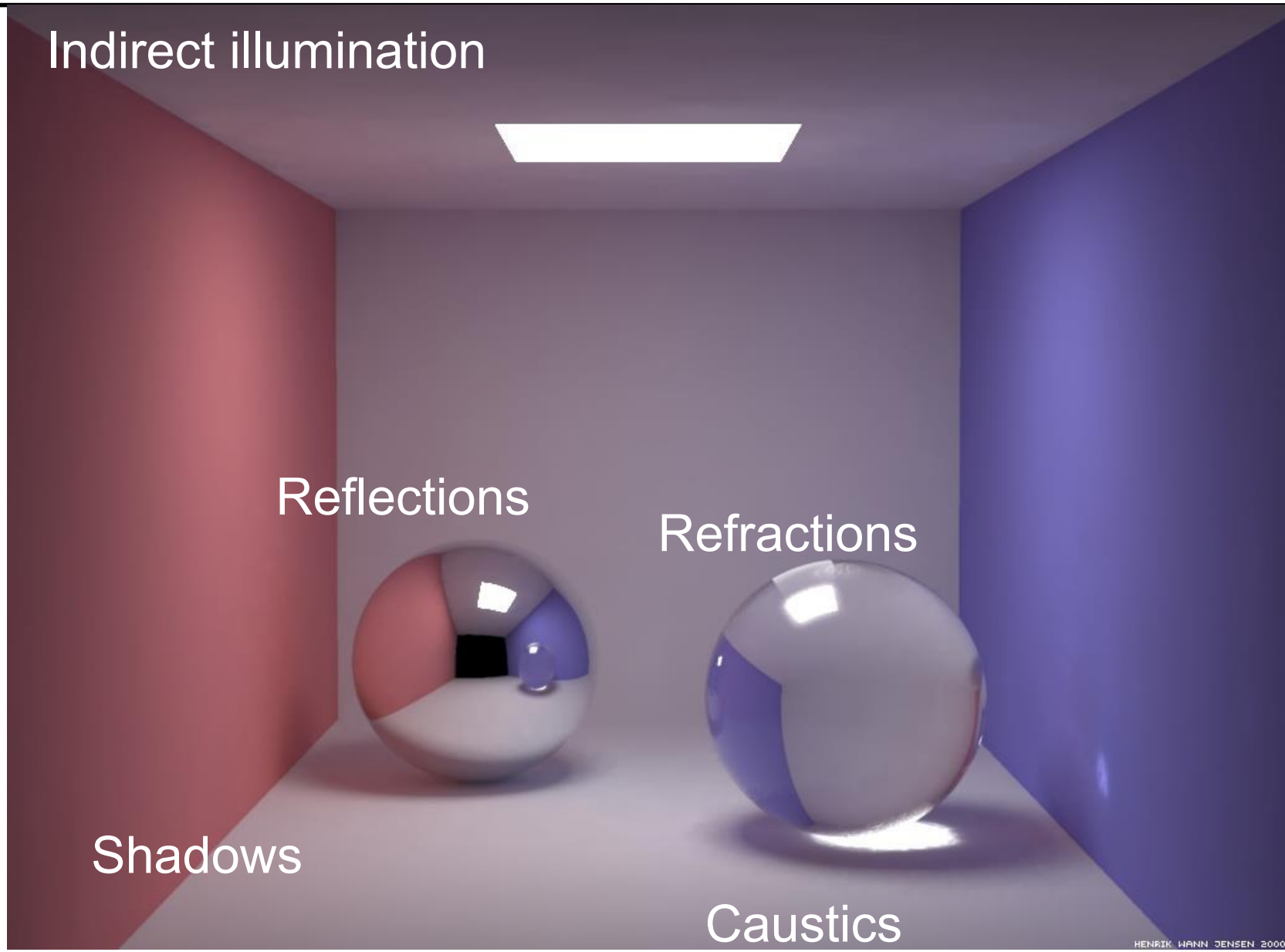


Secondary rays are used for testing shadows, doing reflections, etc.

This image is in the public domain.
Source: openclipart

We'll do all this a little later!

Secondary Rays

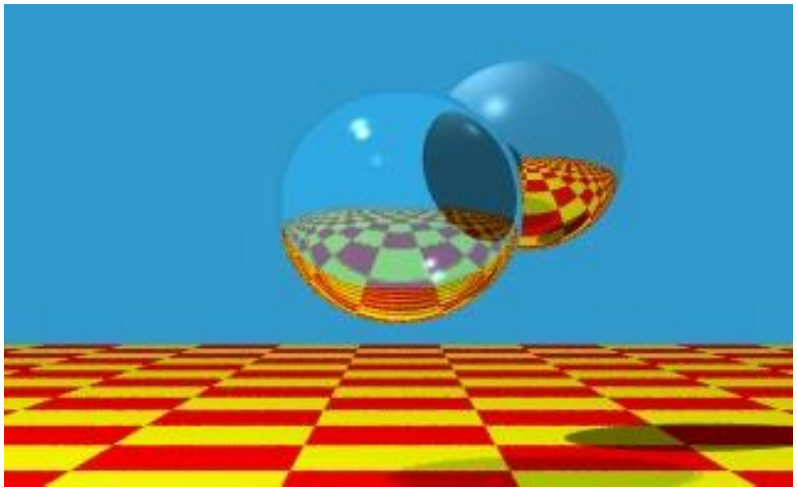


Courtesy of Henrik Wann Jensen. Used with permission.

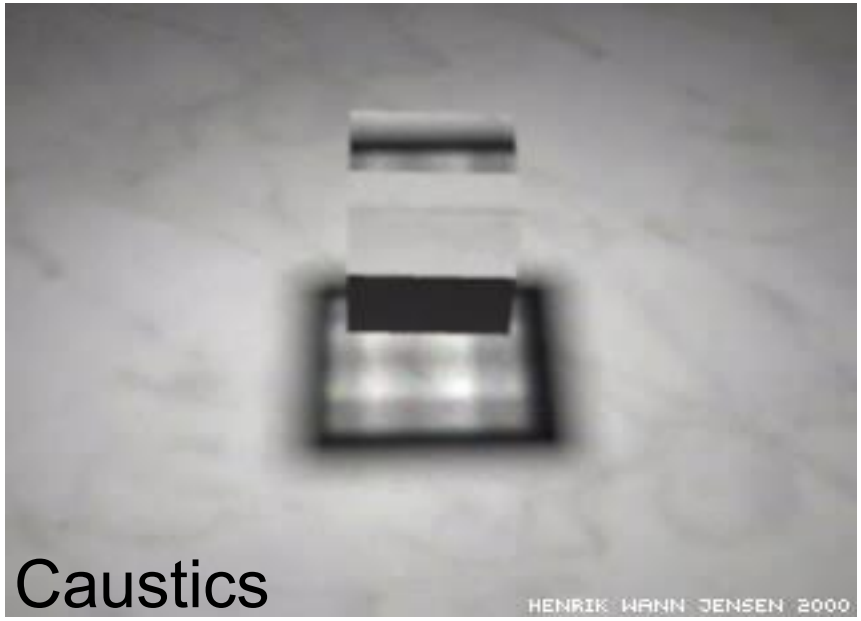
Henrik Wann Jensen

Ray Tracing

Courtesy of Henrik Wann Jensen. Used with permission.



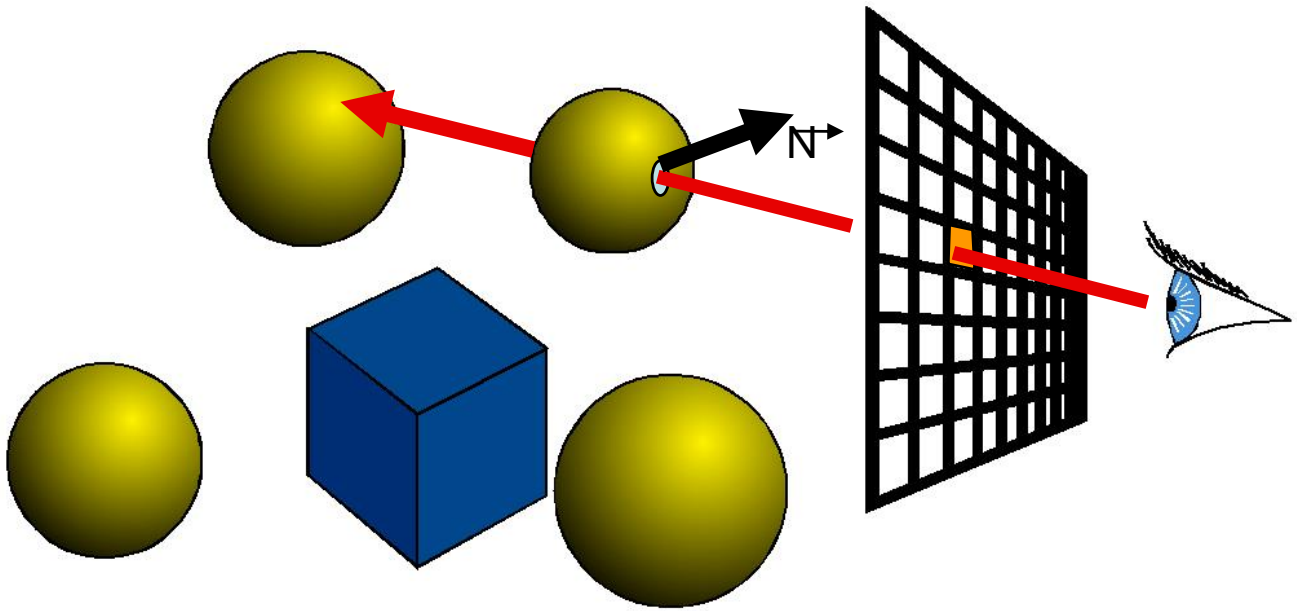
Reflections, refractions



Questions?

Ray Casting

```
For every pixel
   Construct a ray from the eye
   For every object in the scene
       Find intersection with the ray
       Keep if closest
    Shade depending on light and normal vector
```

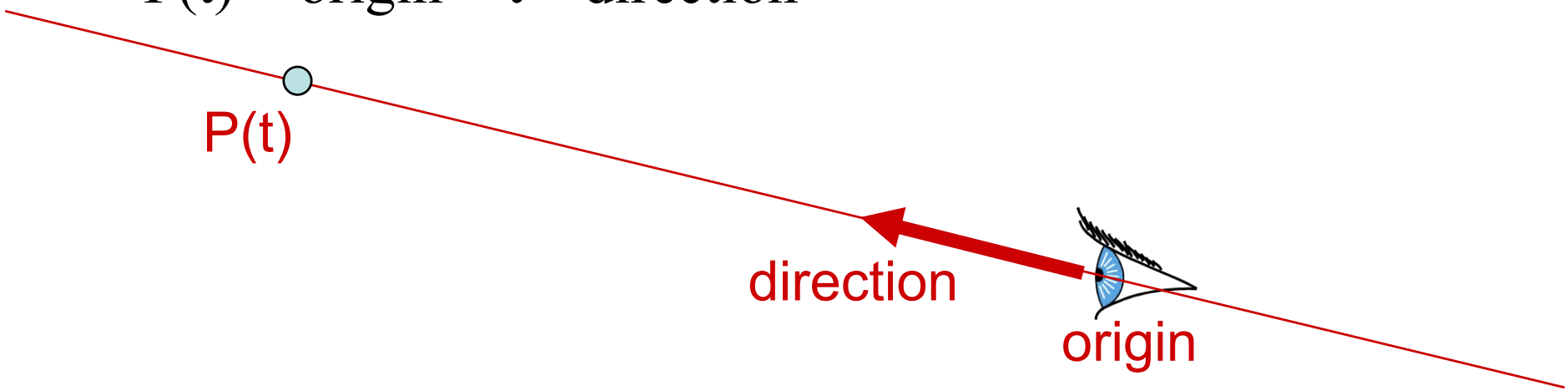


Finding the intersection point and normal is the central part of ray casting

Ray Representation

- Origin – Point
- Direction – Vector
  - normalized is better
- Parametric line
  - P(t) = origin + t * direction

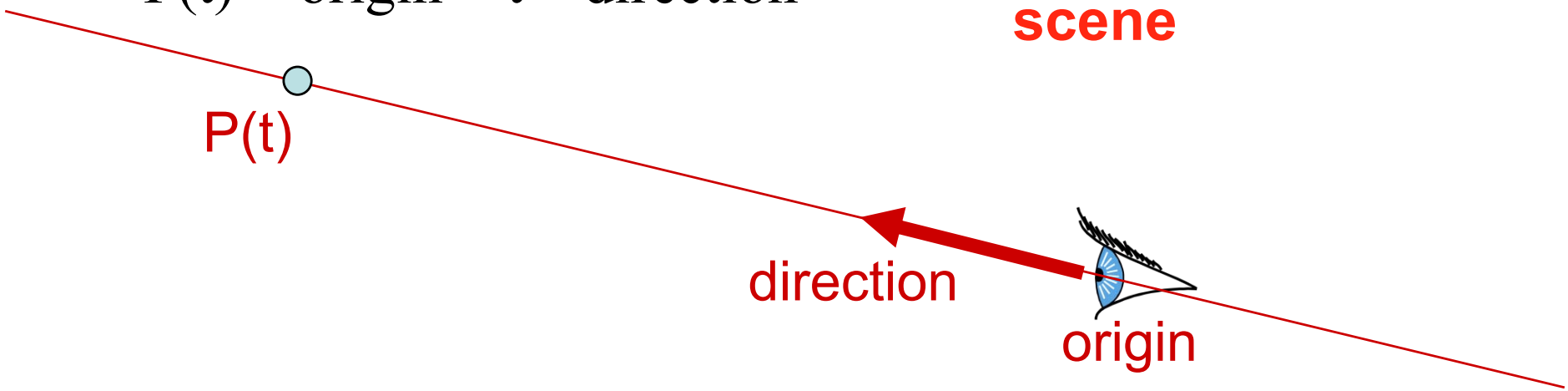
How would you represent a ray?



Ray Representation

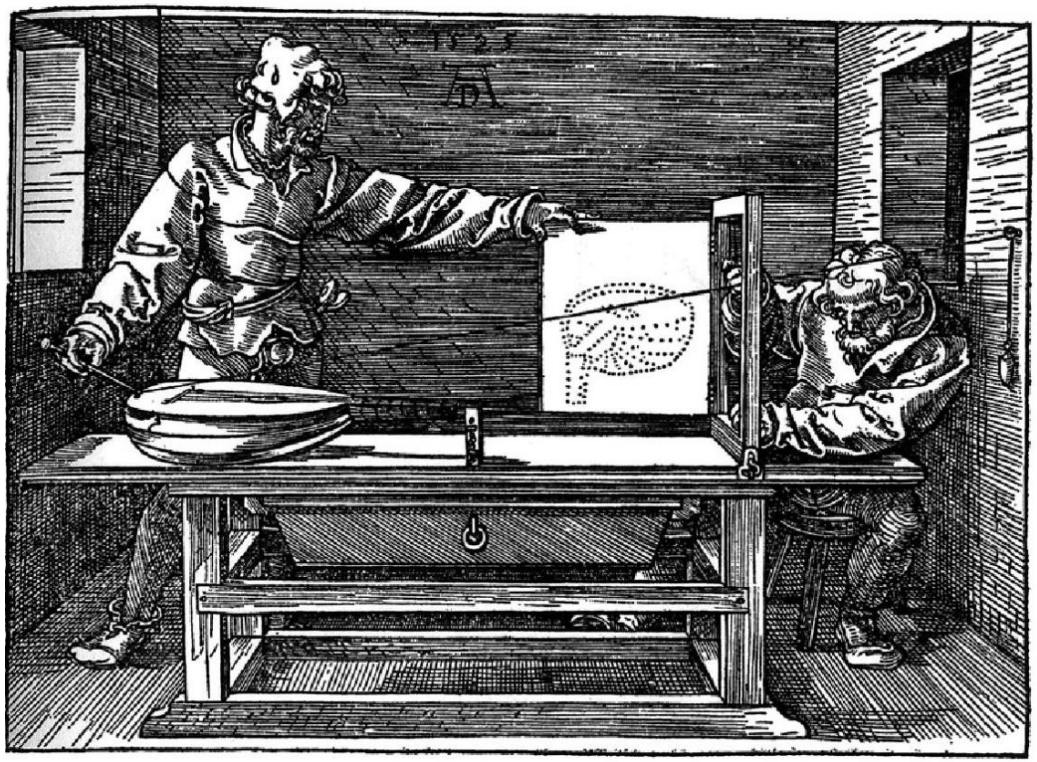
- Origin – Point
- Direction – Vector
  - normalized is better
- Parametric line
  - P(t) = origin + t * direction

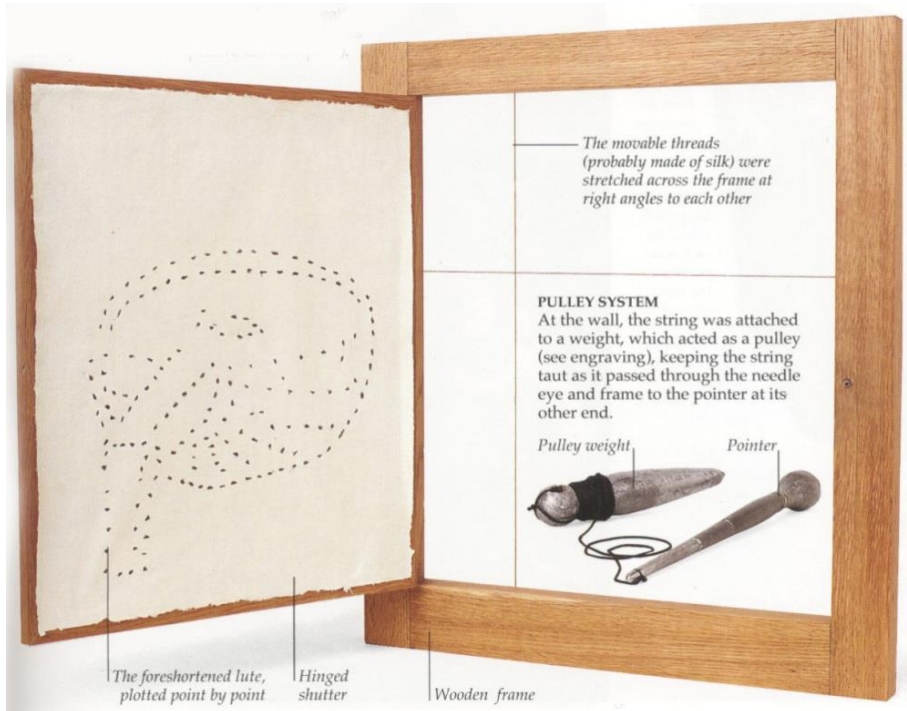
Another way to put the ray casting problem statement:
Find smallest t > 0 such that P(t) lies on a surface in the scene



Dürer's Ray Casting Machine

- Albrecht Dürer, 16th century



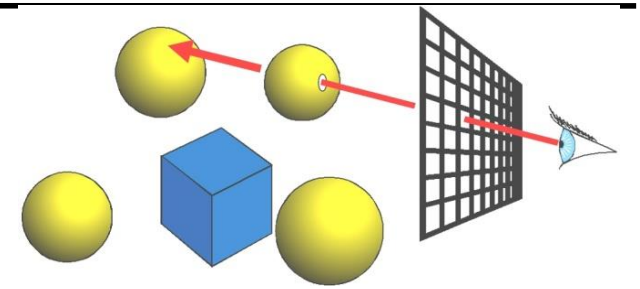


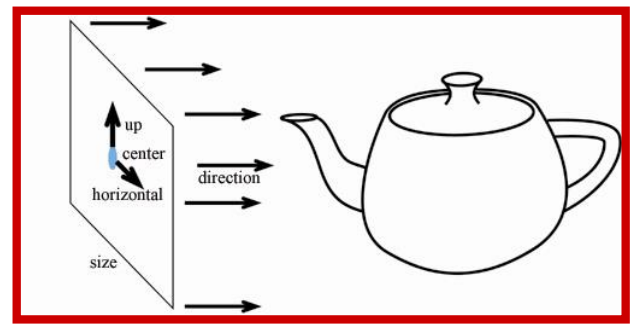
Dürer's Ray Casting Machine

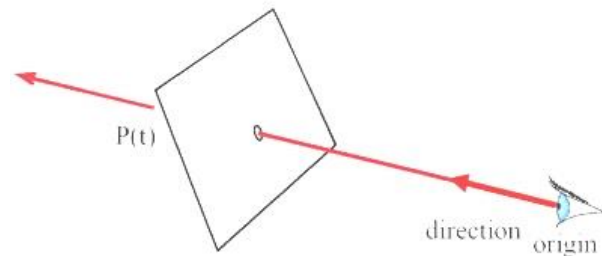
- Albrecht Dürer, 16th century

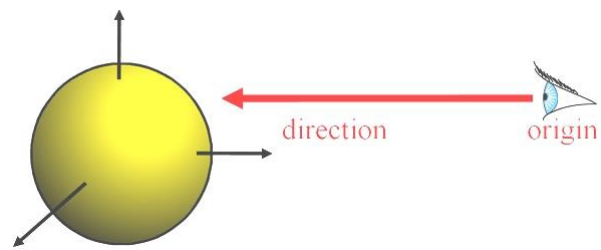
Ray Casting

- Ray Casting Basics
- Camera and Ray Generation
- Ray-Plane Intersection
- Ray-Sphere Intersection



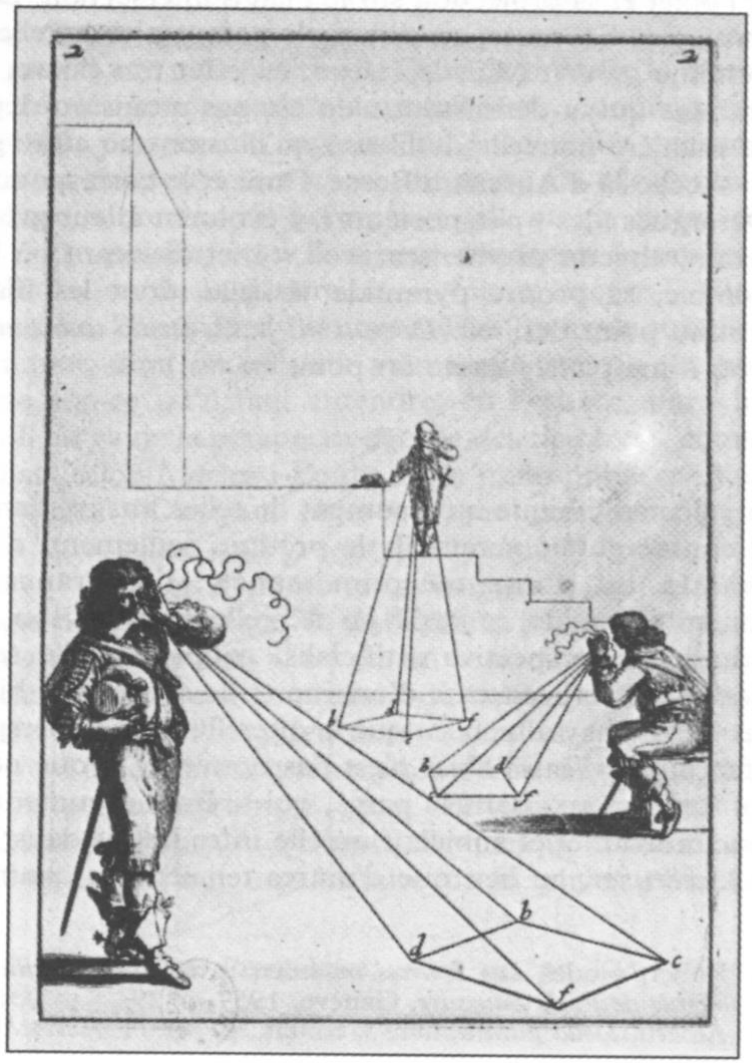






Cameras

```
For every pixel
  Construct a ray from the eye
  For every object in the scene
    Find intersection with ray
      Keep if closest
```

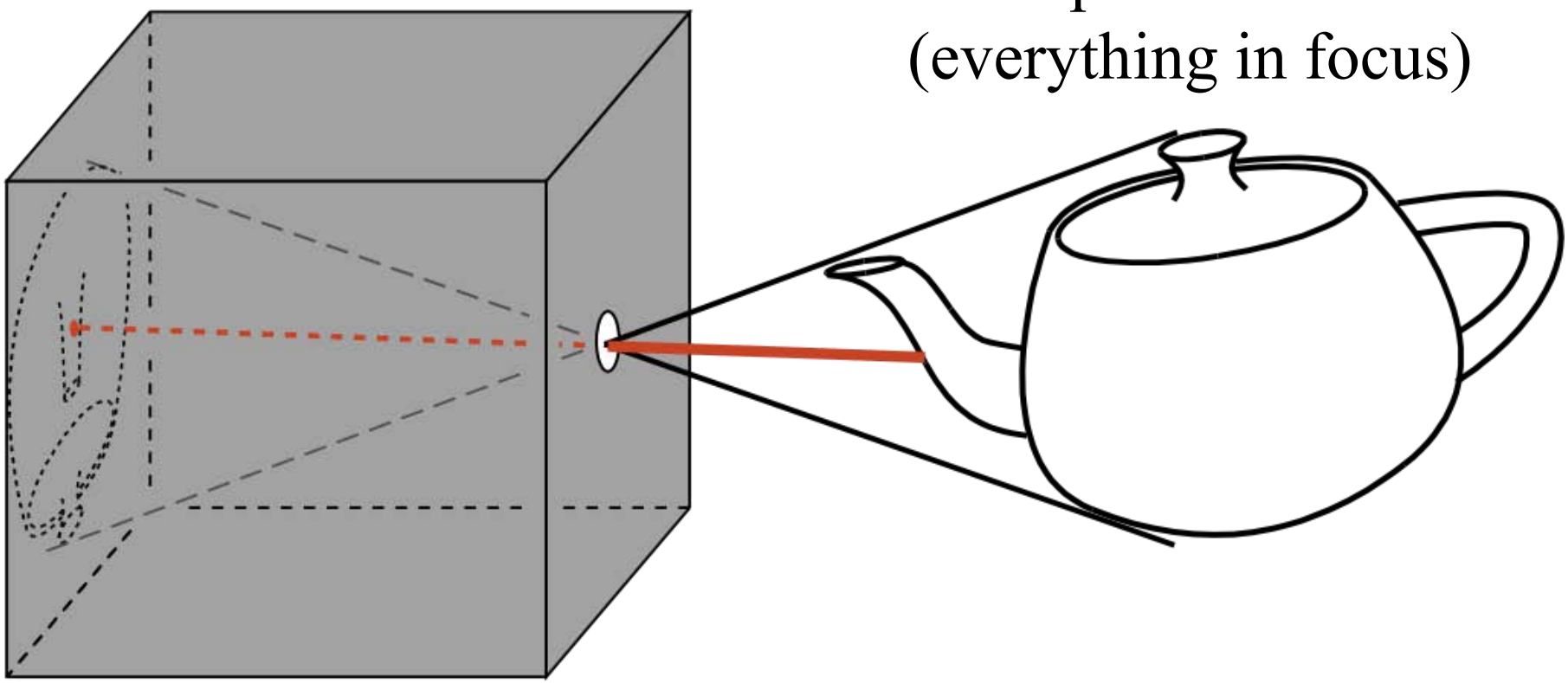


Abraham Bosse, *Les Perspecteurs*. Gravure extraite de la *Manière*

Pinhole Camera

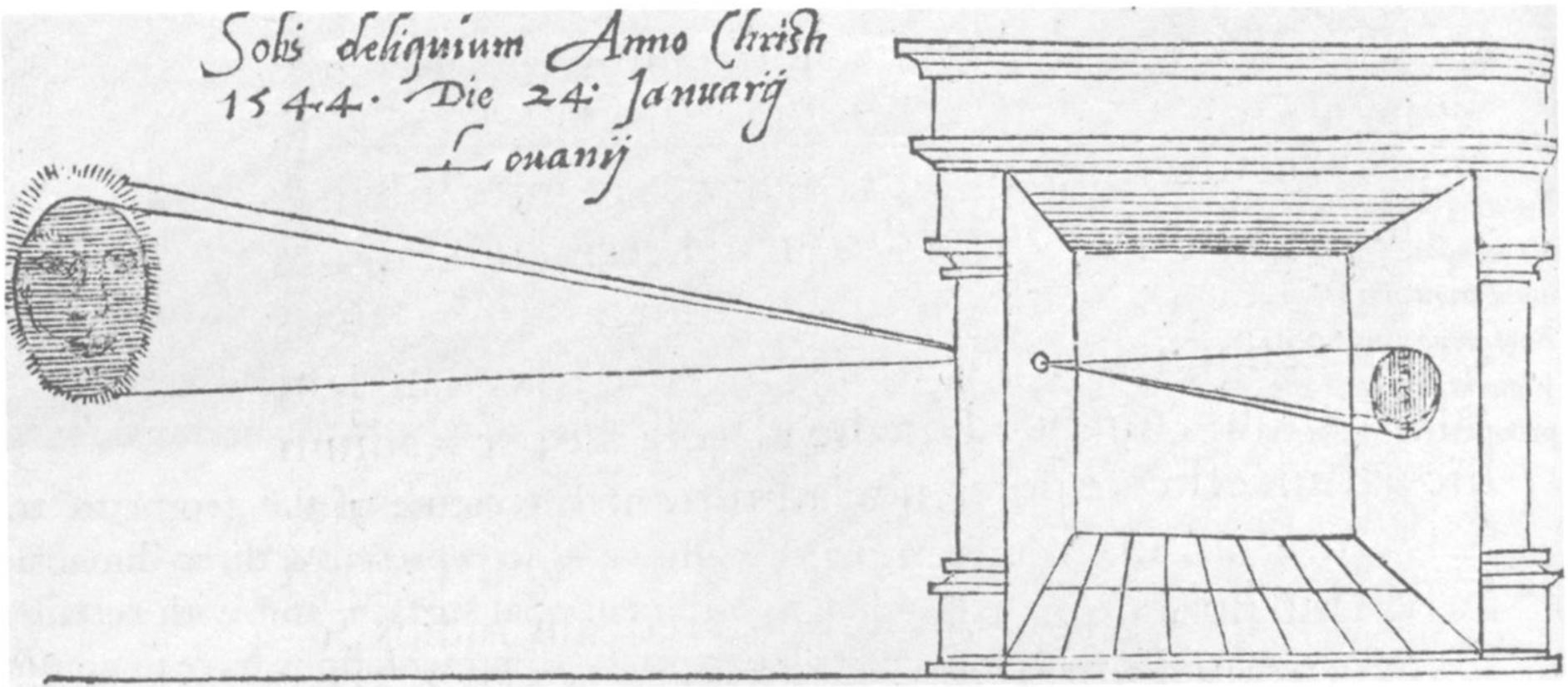
- Box with a tiny hole
- Inverted image
- Similar triangles

- Perfect image if hole infinitely small
- Pure geometric optics
- No depth of field issue (everything in focus)

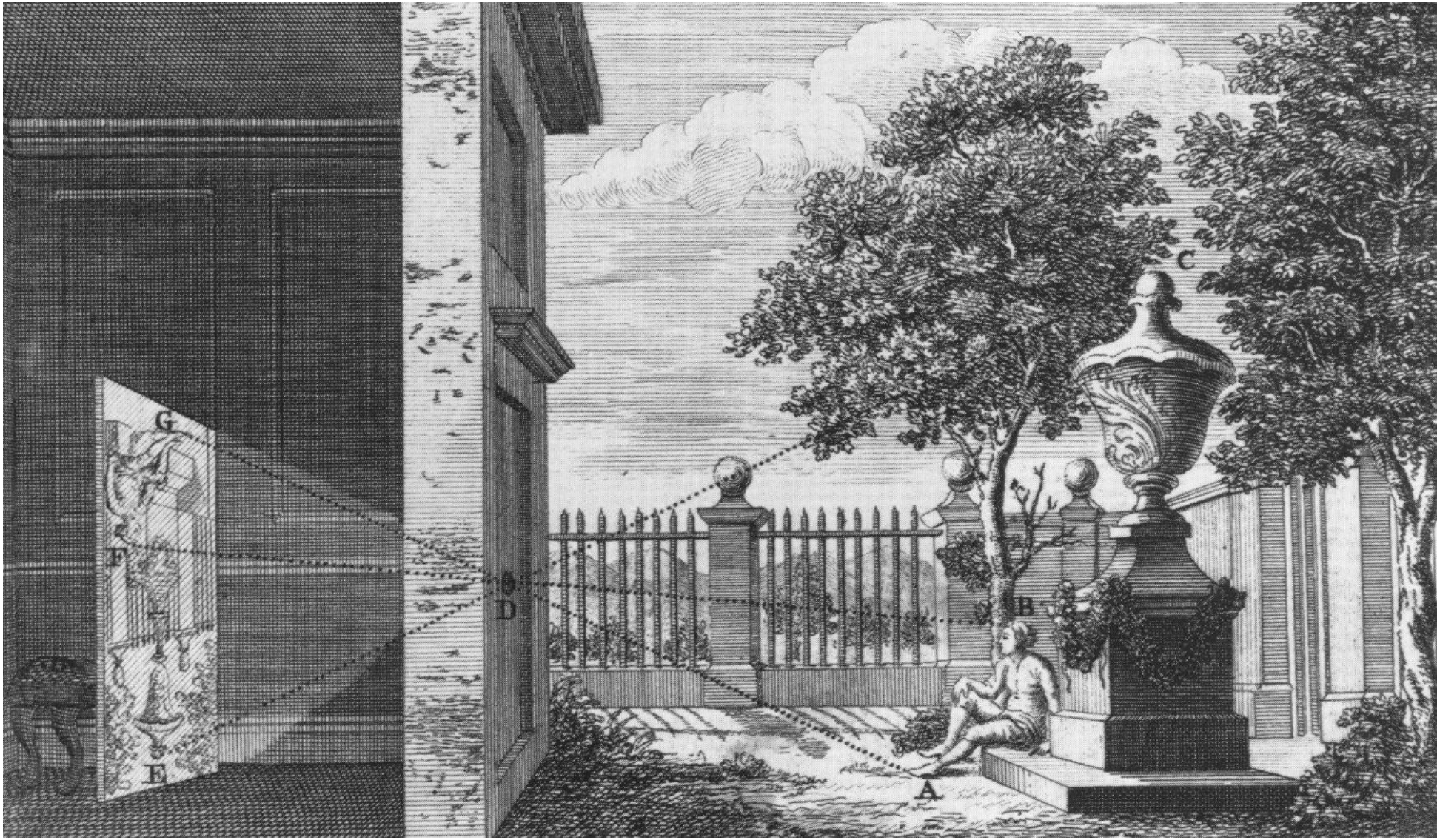


Oldest Illustration

- From Gemma Frisius, 1545



Also Called “Camera Obscura”



Camera Obscura Today

Images removed due to copyright restrictions -- please see
http://www.abelardomorell.net/photography/cameraobsc_01/cameraobsc_17.html
http://www.abelardomorell.net/posts/camera-obscura/
http://www.abelardomorell.net/photography/cameraobsc_49/cameraobsc_63.html
for further details.

Abelardo Morell

www.abelardomorell.net

Camera Obscura Today

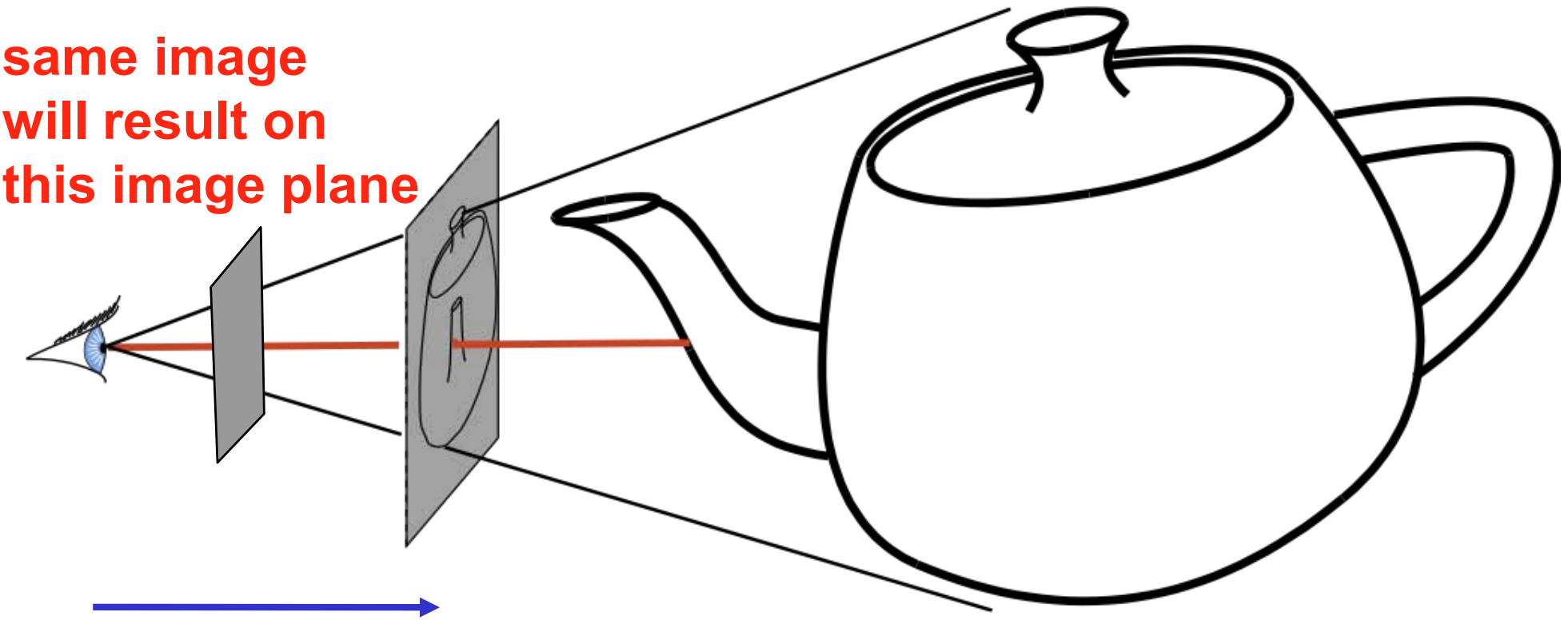
Images removed due to copyright restrictions -- please see
http://www.abelardomorell.net/photography/cameraobsc_01/cameraobsc_17.html
http://www.abelardomorell.net/posts/camera-obscura/
http://www.abelardomorell.net/photography/cameraobsc_49/cameraobsc_63.html
for further details.

Abelardo Morell

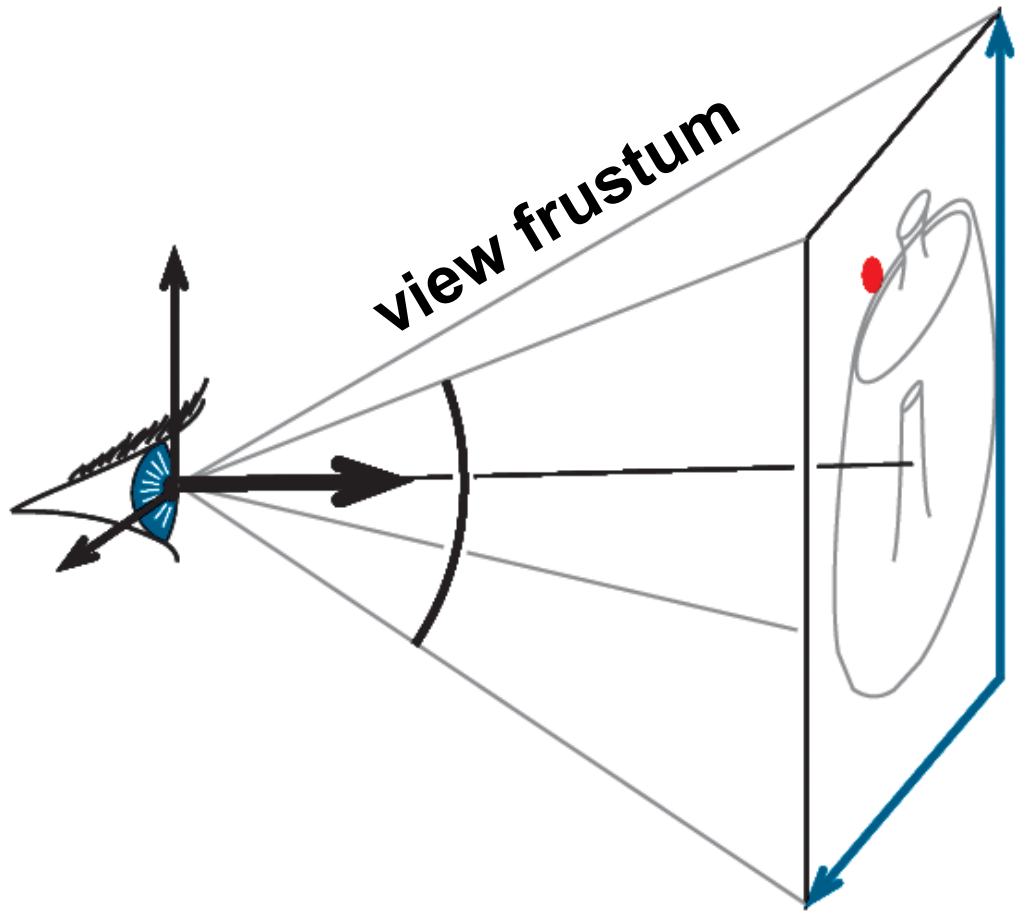
www.abelardomorell.net

Simplified Pinhole Camera

- Eye-image pyramid (view frustum)
- Note that the distance/size of image are arbitrary

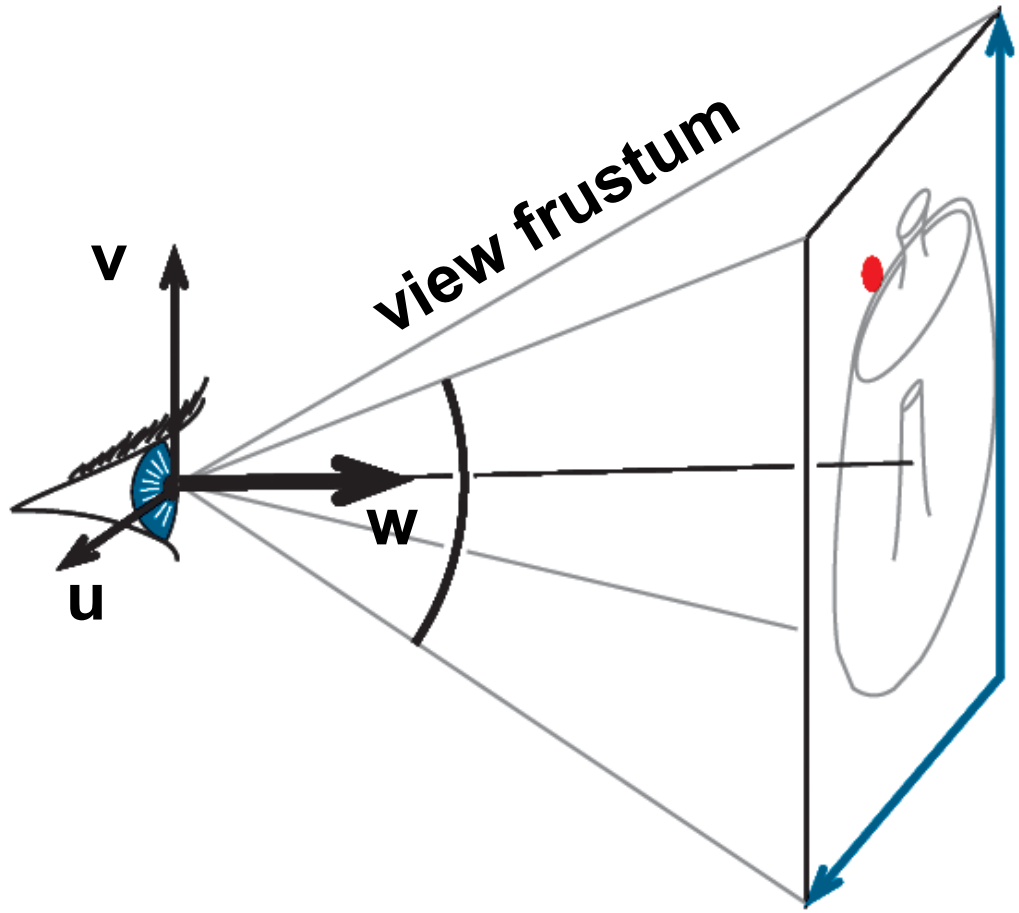


Camera Description?



Camera Description?

- Eye point ***e*** *(center)*
- Orthobasis ***u, v, w*** *(horizontal, up, direction)*



Object coordinates

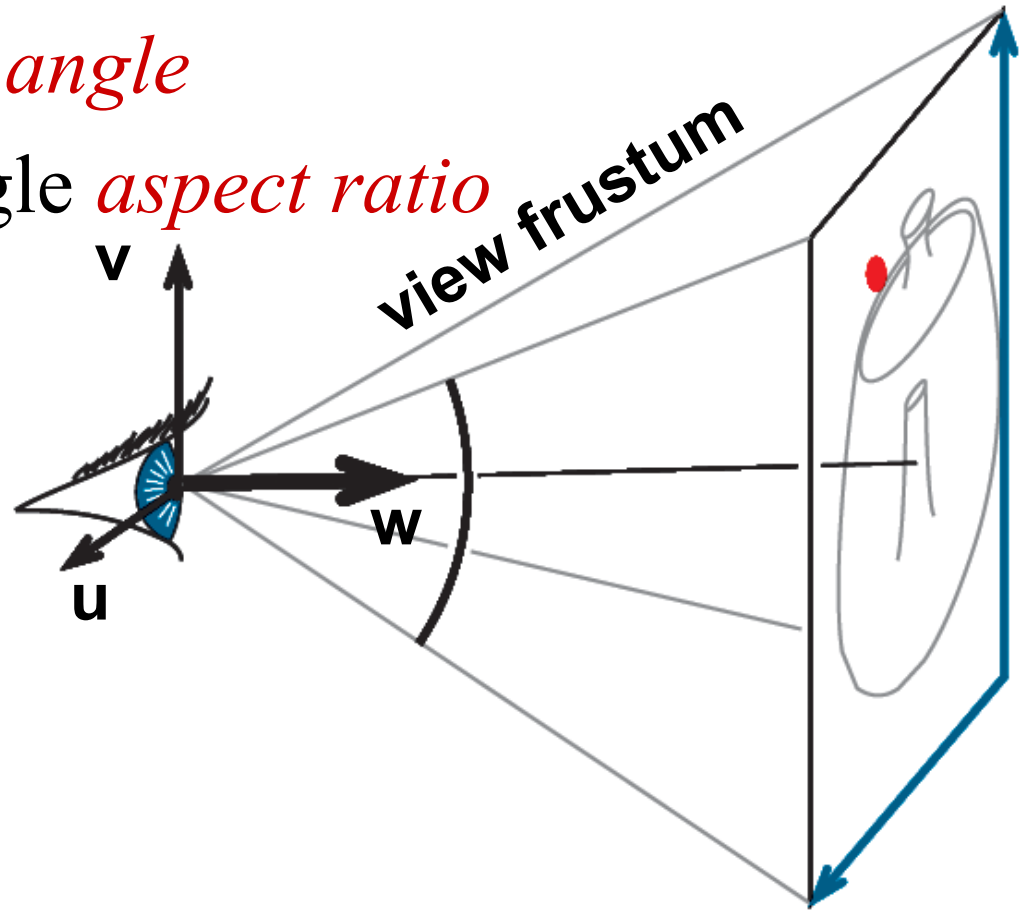
World coordinates

View coordinates

Image coordinates

Camera Description?

- Eye point ***e*** *(center)*
- Orthobasis ***u, v, w*** *(horizontal, up, direction)*
- Field of view *angle*
- Image rectangle *aspect ratio*



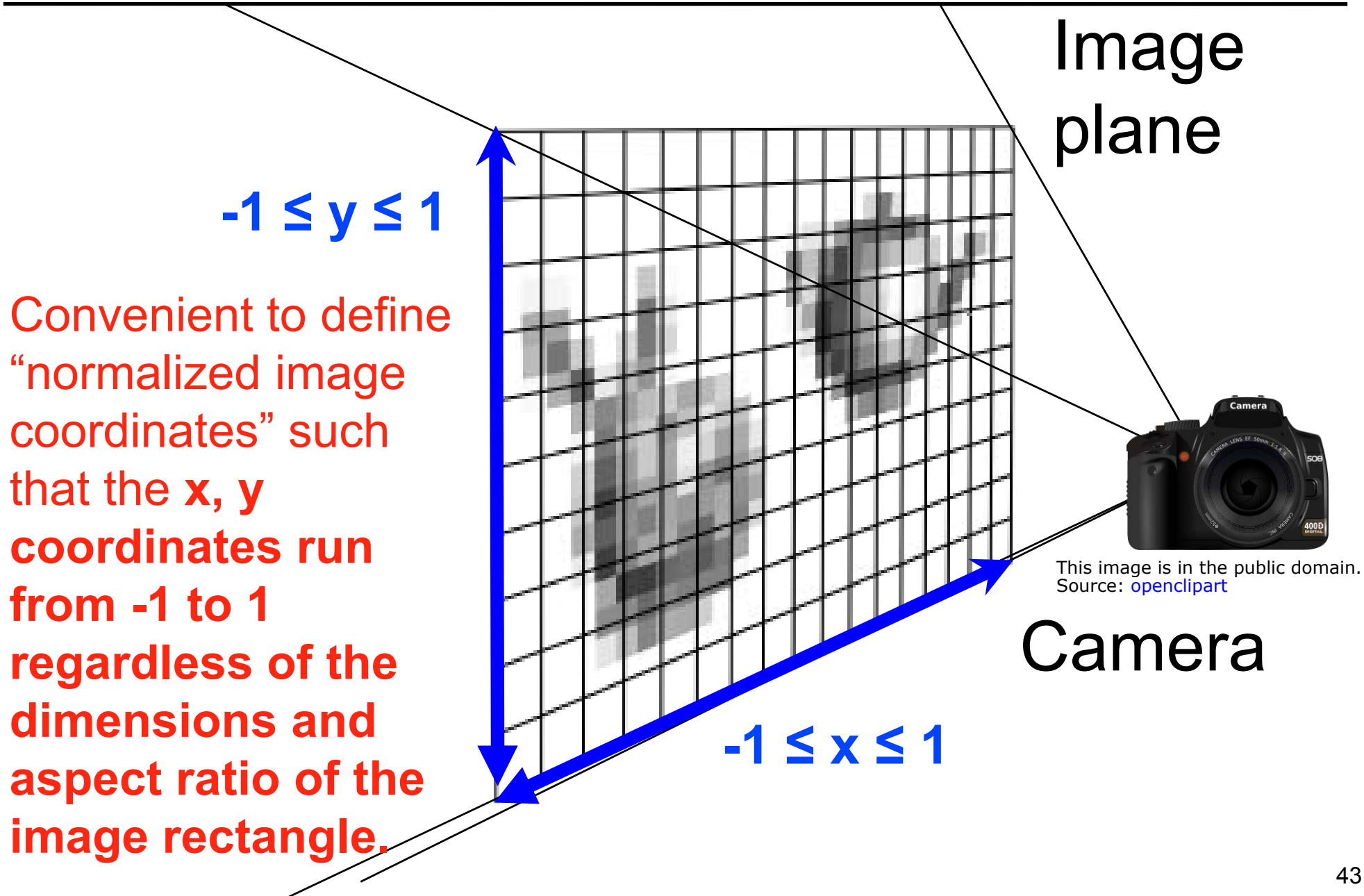
Object coordinates

World coordinates

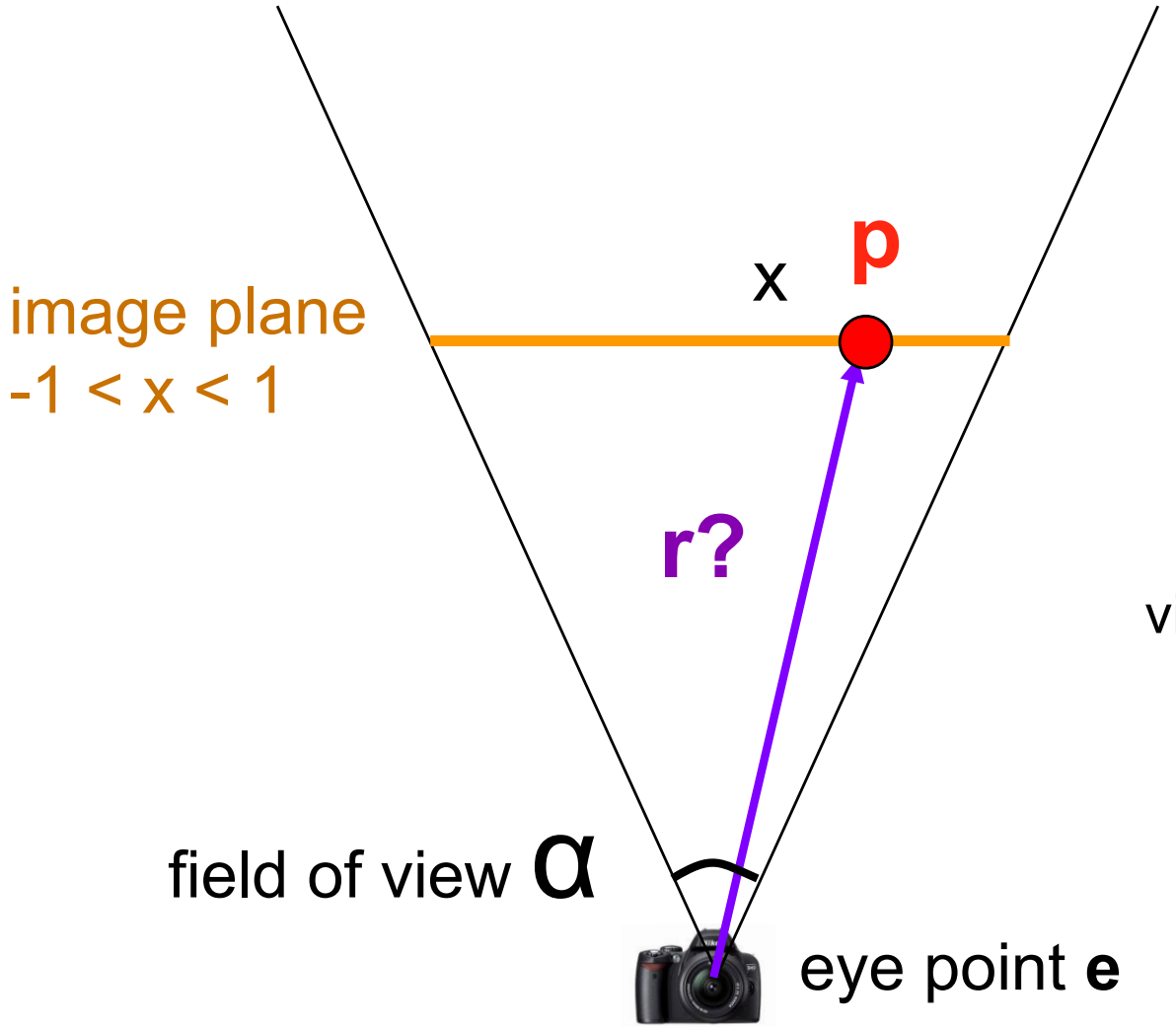
View coordinates

Image coordinates

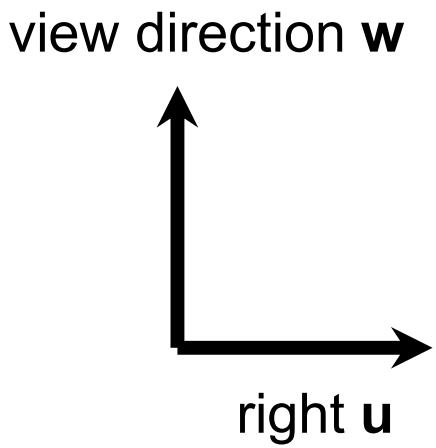
Image Coordinates



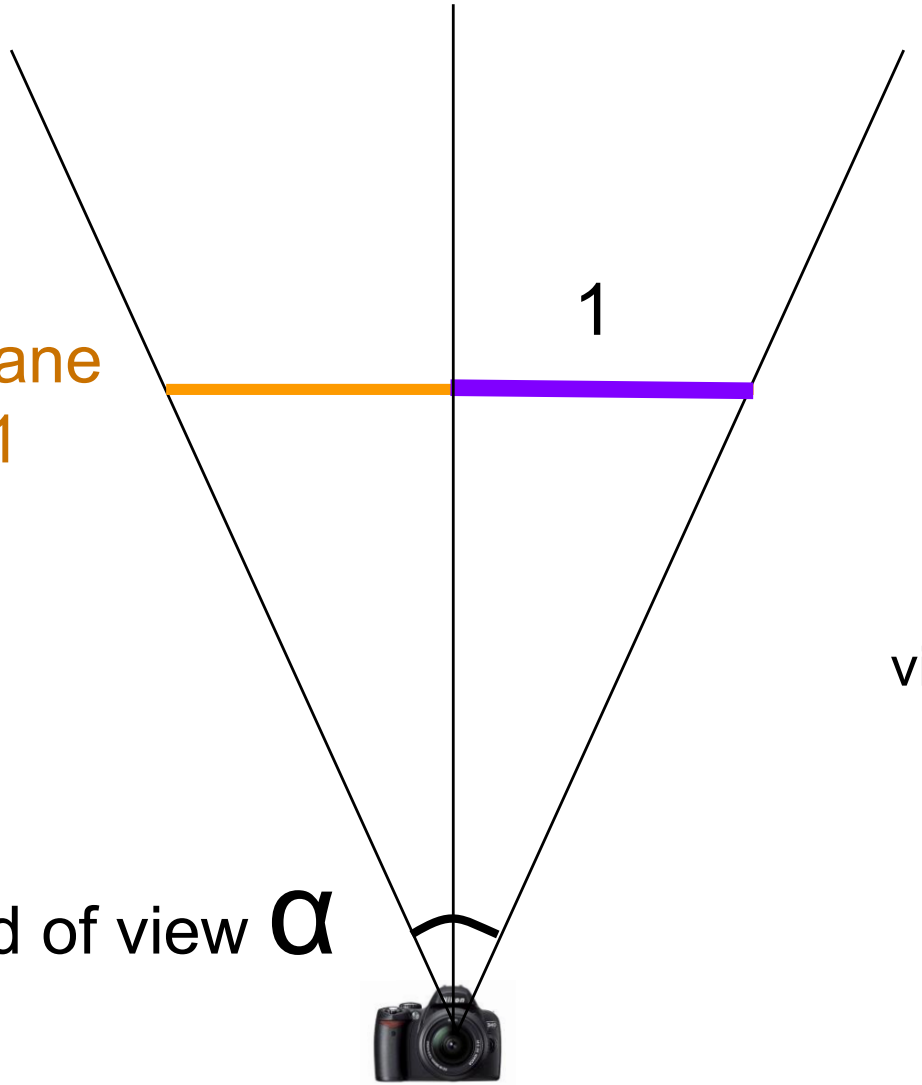
Ray Generation in 2D



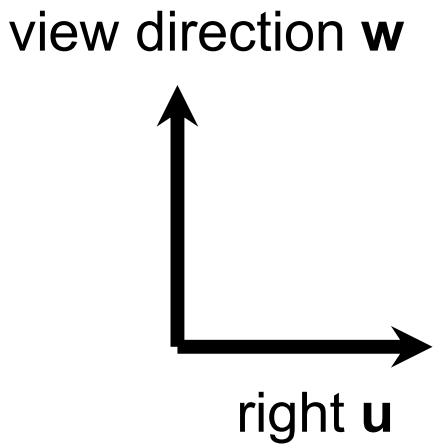
p is point on image plane at coordinate x, we want to know the direction of the ray **r**



Ray Generation in 2D



What is the distance D to the screen so that the normalized coordinates go to 1?



Corrections

CORRECTION: In the following few slides, the ideas are brilliantly visualized, but some of the equations are rubbish. These are OK:

$$tan(\alpha/2) = 1/D \rightarrow D = 1/\,tan(\alpha/2)$$

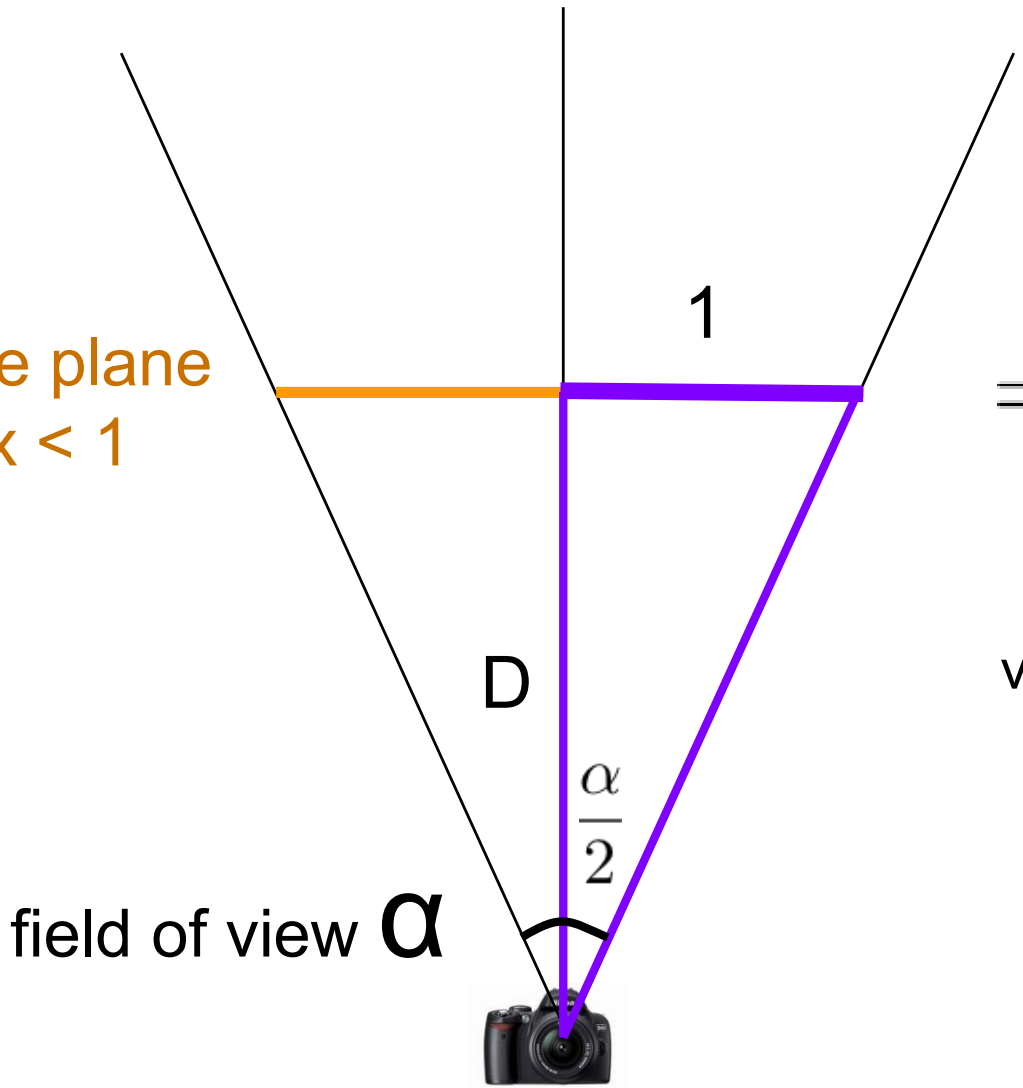
The others should read as:

$$\mathbf{r} = \mathbf{p} - \mathbf{e} = x\mathbf{u} + D\mathbf{w}$$

And (for the 3D case):

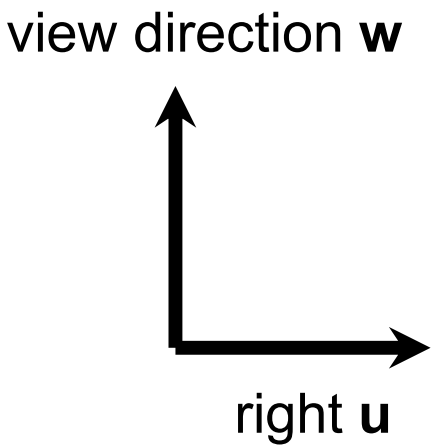
$$\mathbf{r} = x\mathbf{u} + \text{aspect} \cdot y\mathbf{v} + D\mathbf{w}$$

Ray Generation in 2D

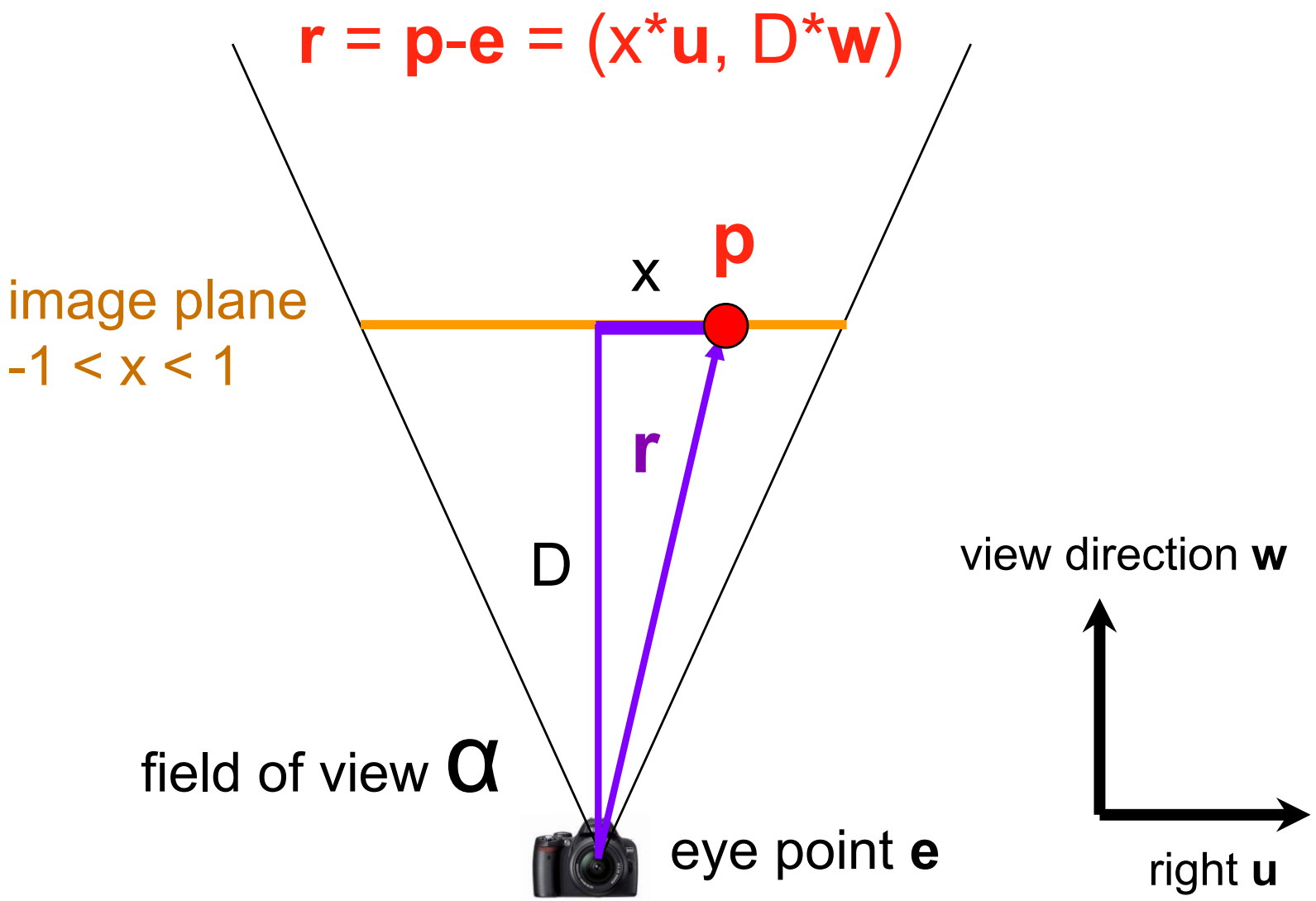


$$\tan\frac{\alpha}{2} = \frac{1}{D}$$

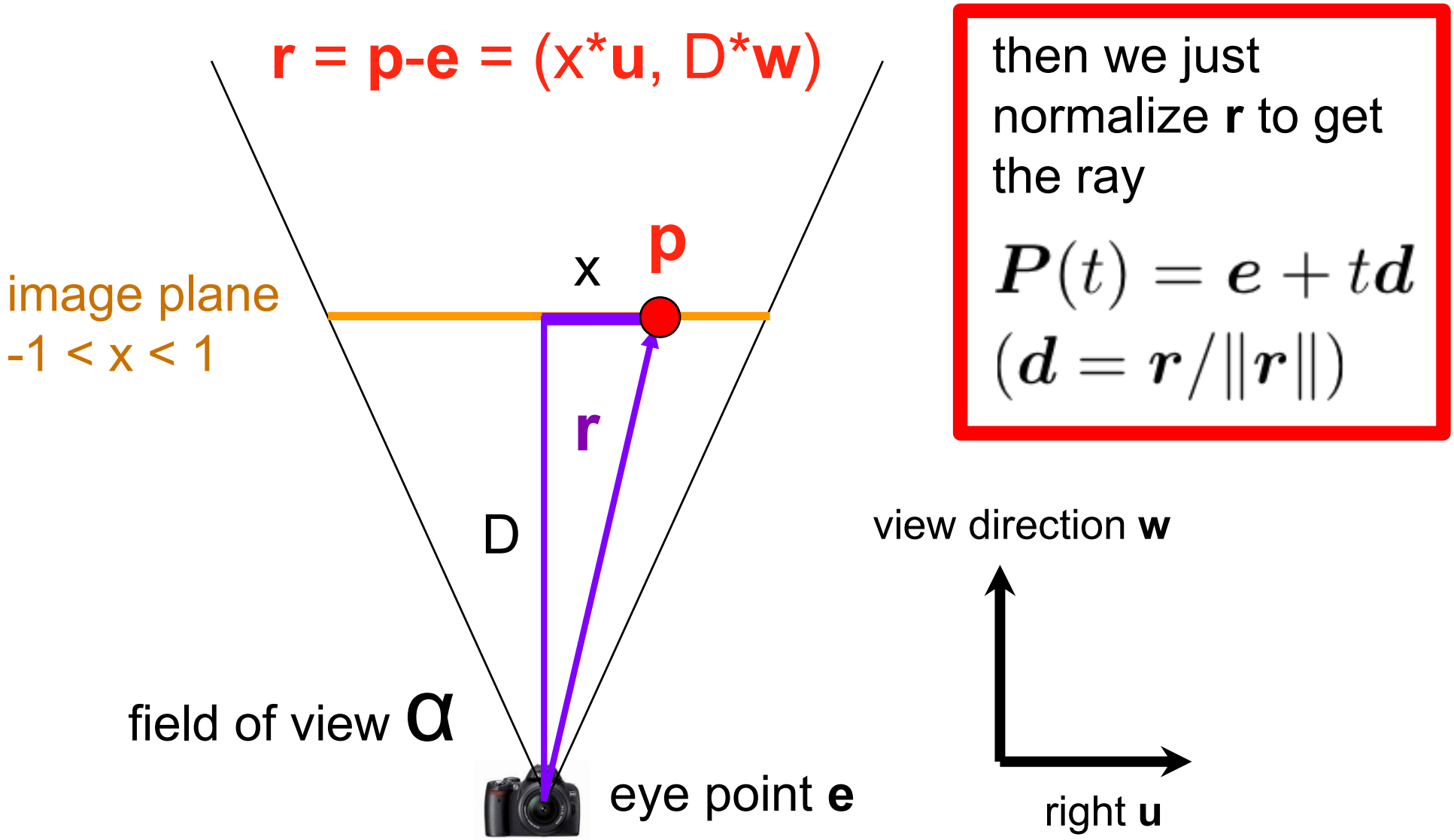
$$\Rightarrow D = \frac{1}{\tan(\alpha/2)}$$



Ray Generation in 2D



Ray Generation in 2D



then we just normalize **r** to get the ray

$$\boldsymbol{P}(t) = \boldsymbol{e} + t\boldsymbol{d}$$

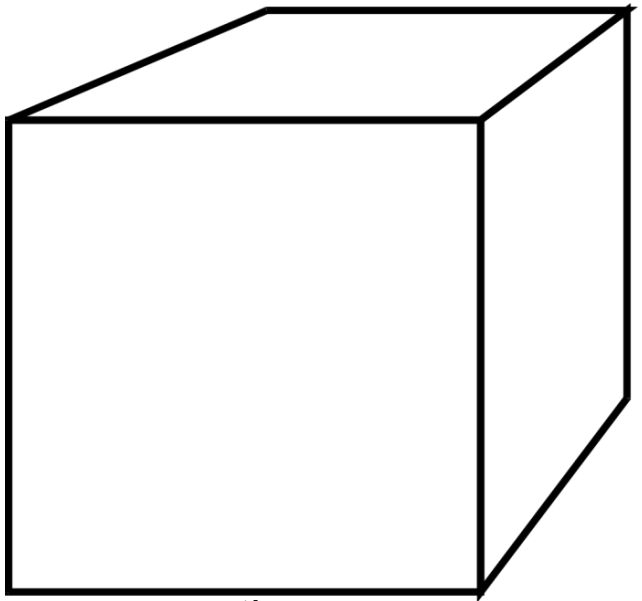
$$(\boldsymbol{d} = \boldsymbol{r}/\|\boldsymbol{r}\|)$$

That was 2D, 3D is just as simple

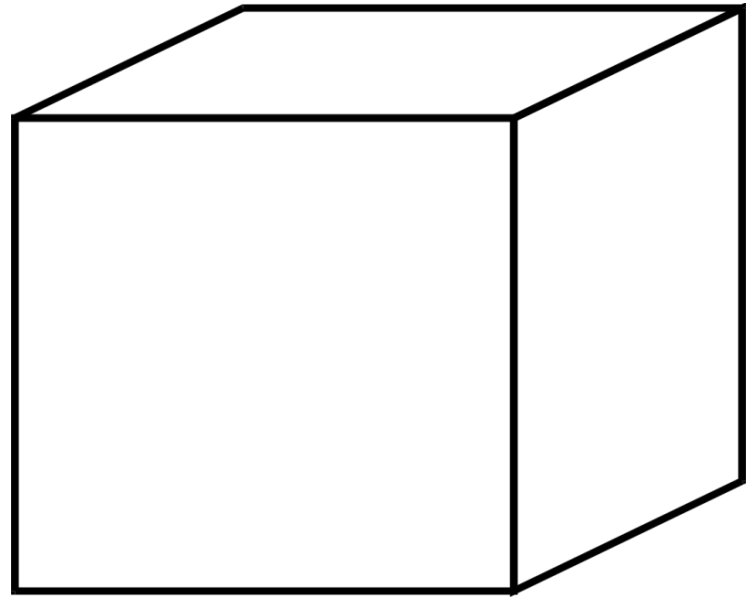
- *y* coordinate is treated just like *x*, except accounting for aspect ratio
  - **r** = (x***u**, aspect*y***v**, D***w**)
  - Again, **u**, **v**, **w** are the basis vectors of the view coordinate system
  - Aspect ratio handles non-square viewports
    - Think of your 16:9 widescreen TV

- The point of the exercise with computing D was to allow us to use the [-1,1] image coordinate system regardless of field of view.

Perspective vs. Orthographic



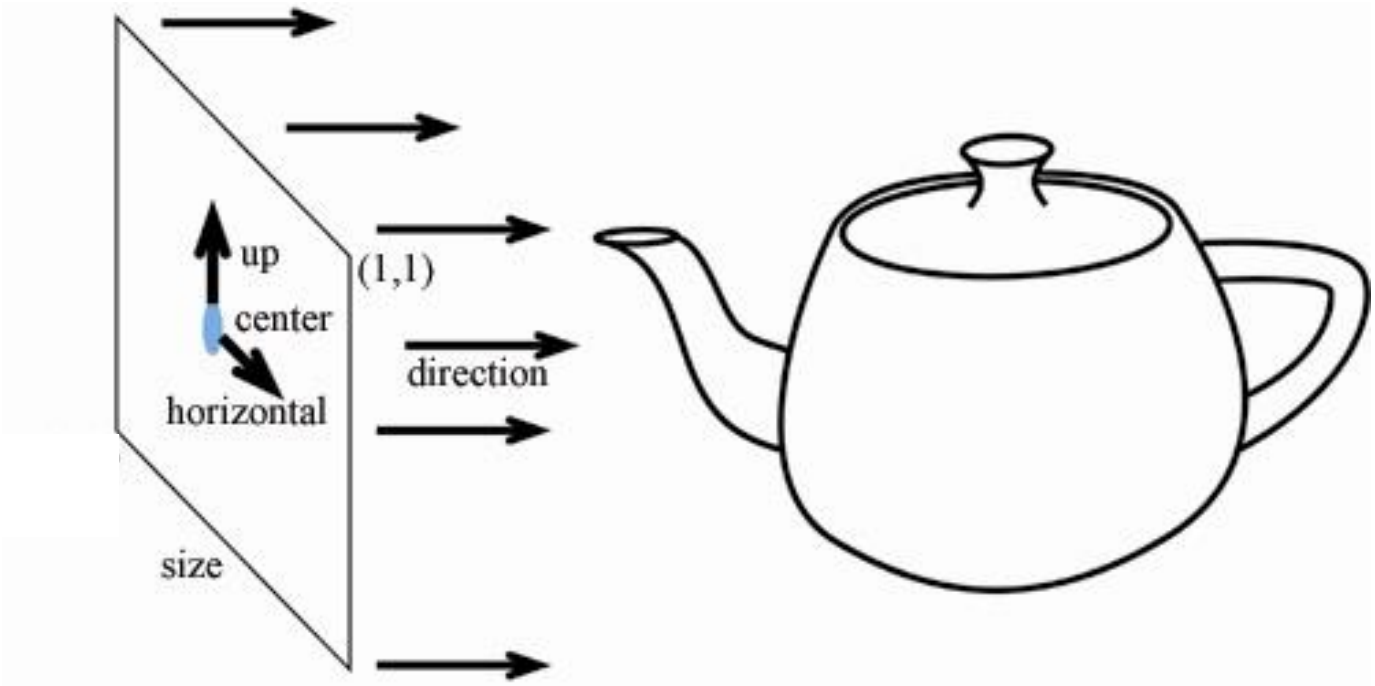
perspective



orthographic

- Parallel projection
- No foreshortening
- No vanishing point

Orthographic Camera



- Ray Generation?
  - Origin = **e** + x*size***u** + y*size***v**
  - Direction is constant: **w**

Other Weird Cameras

- E.g. fish eye, omnimax, parabolic

CAVE Columbia University

Questions?

Courtesy of Paul Rademacher. Used with permission.

Even Funkier Multiperspective Imaging