COMPUTER GRAPHICS

rendering virtual worlds

Course Overview

Mathematical toolbox

Ray caster

Camera & ray generation

Ray-plane intersection

Depth images

Starter code & next week

BASED ON MIT 6.837

slides adapted & project started code translated to Swift by Dion Larson
original course materials available for free here

COURSE DETAILS

Wednesday's at 3pm

90 min sessions, will occasionally run long

Roughly 2.5 hours of homework a week

Join #computer-graphics right after class

Message @dion on Slack for help

REQUIREMENTS

Show up to class on time

Attempt implementations on your own

No laptops out unless specified otherwise

GOALS

Write a ray caster

Extend it be a ray tracer

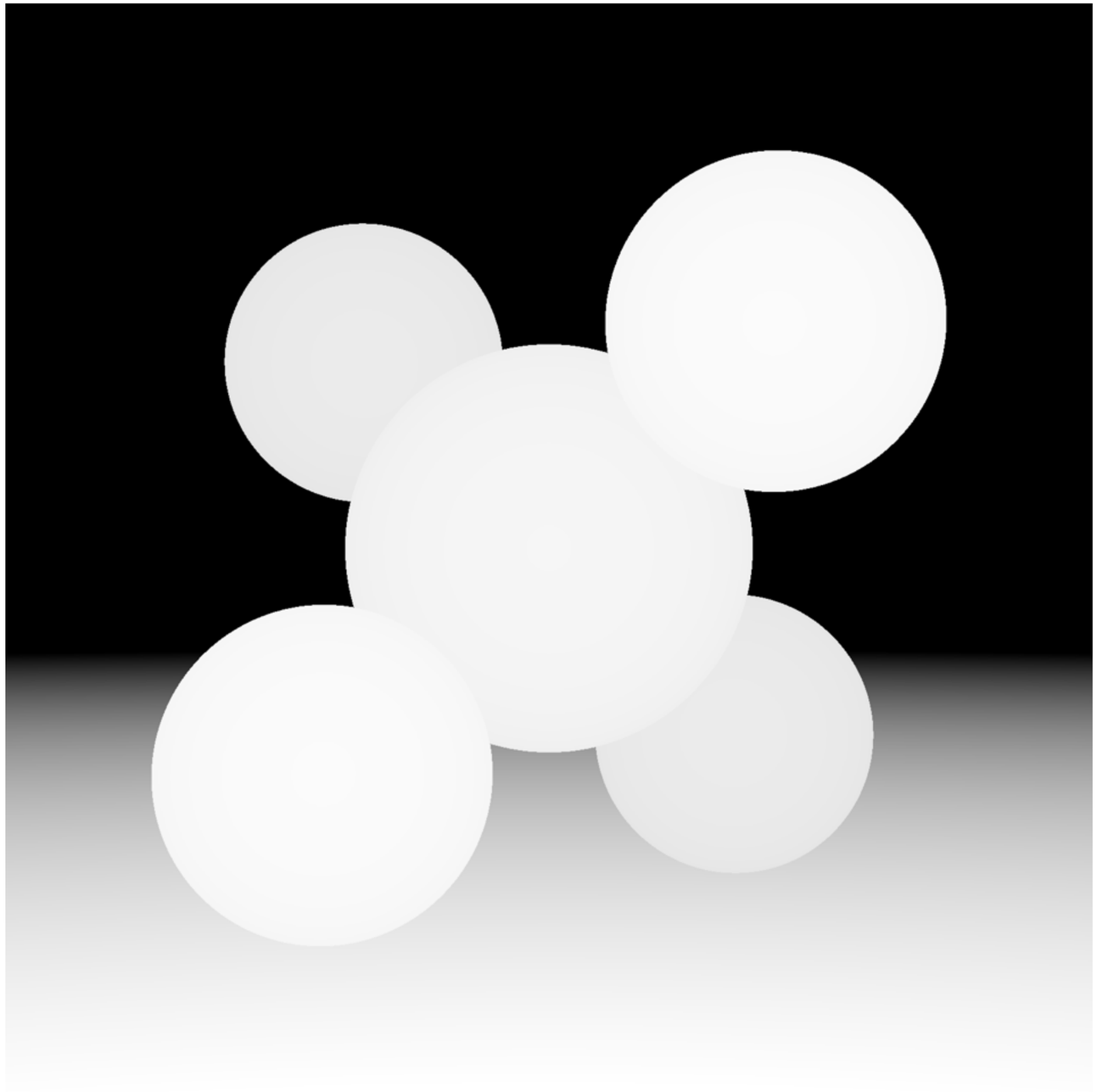
We will not use OpenGL, WebGL, CUDA, OpenCL, Metal this semester

RENDER DISTANCE TO GEOMETRY

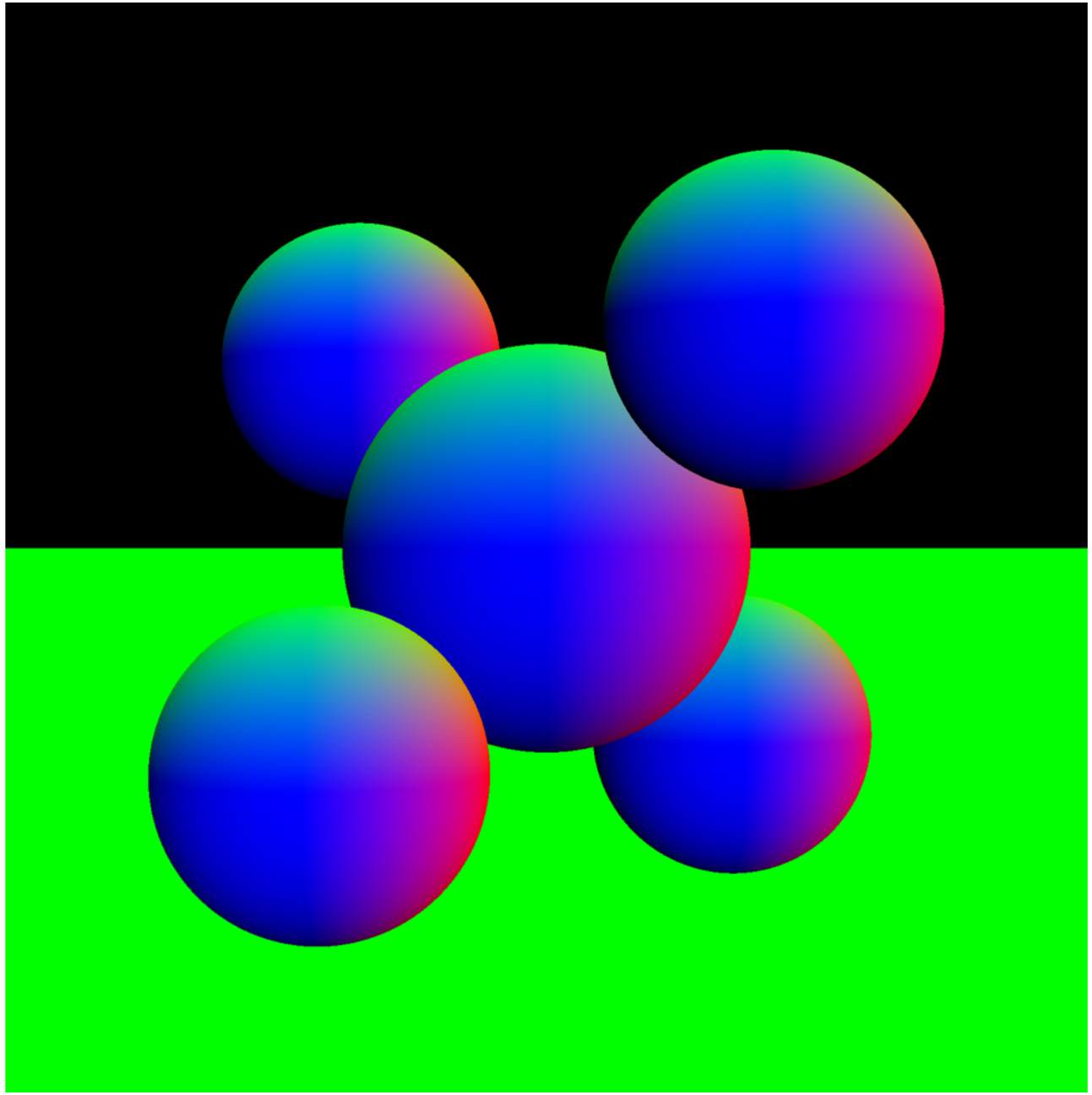
Camera representation

Plane intersection

Sphere intersection



RENDER NORMALS

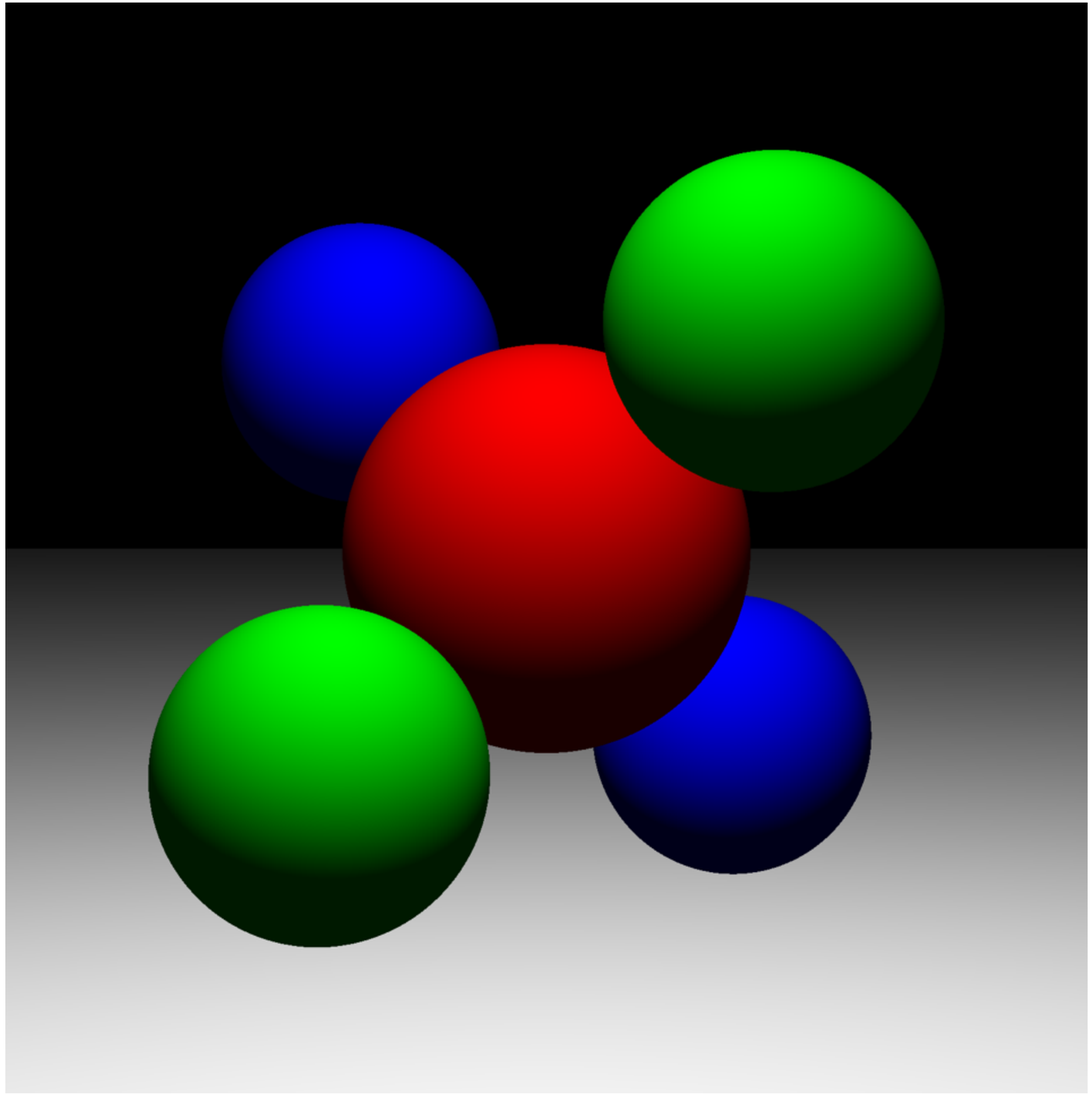


MATERIALS & SHADING

Lights

Diffuse shading

Phong shading

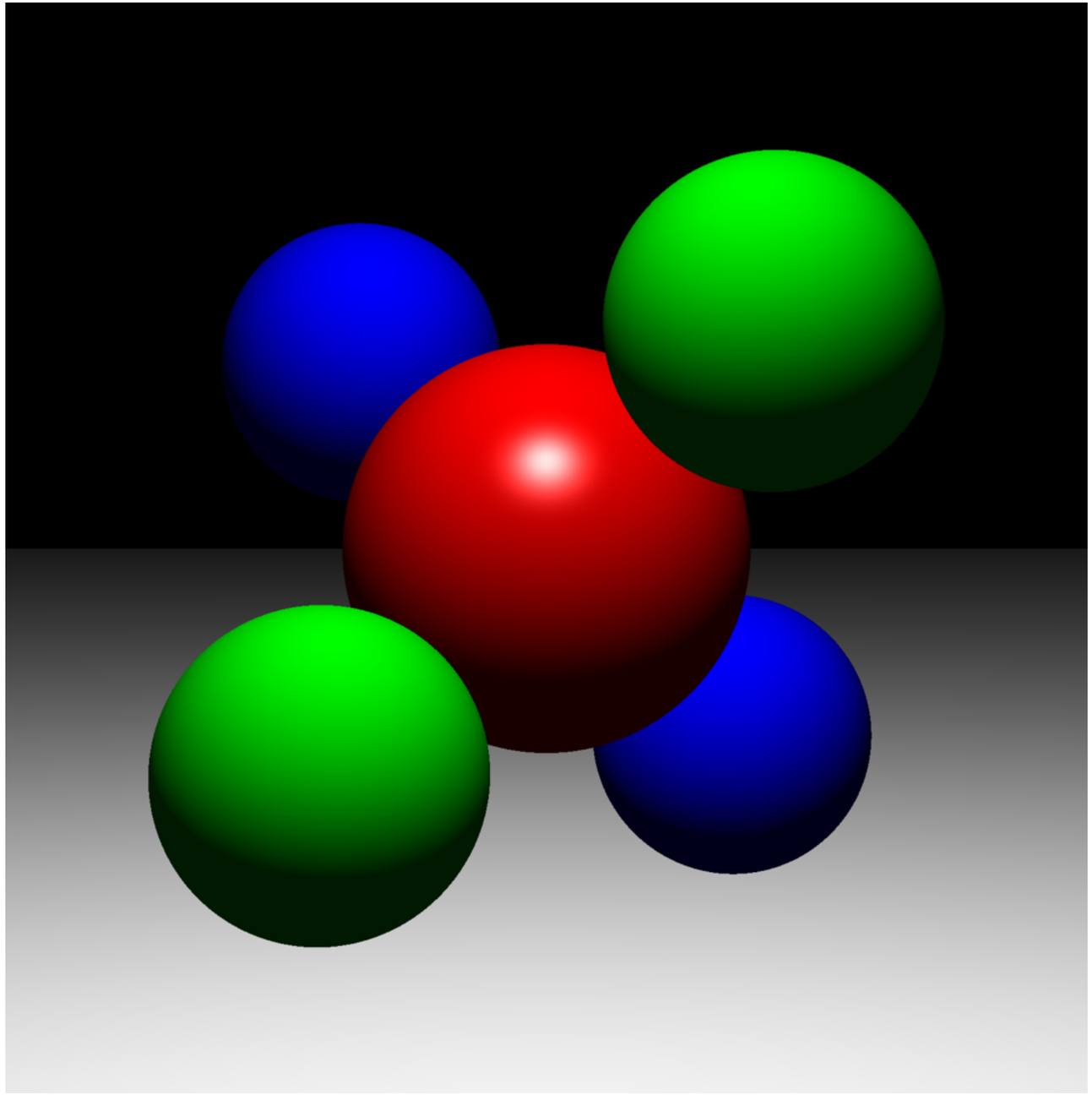


MATERIALS & SHADING

Lights

Diffuse shading

Phong shading

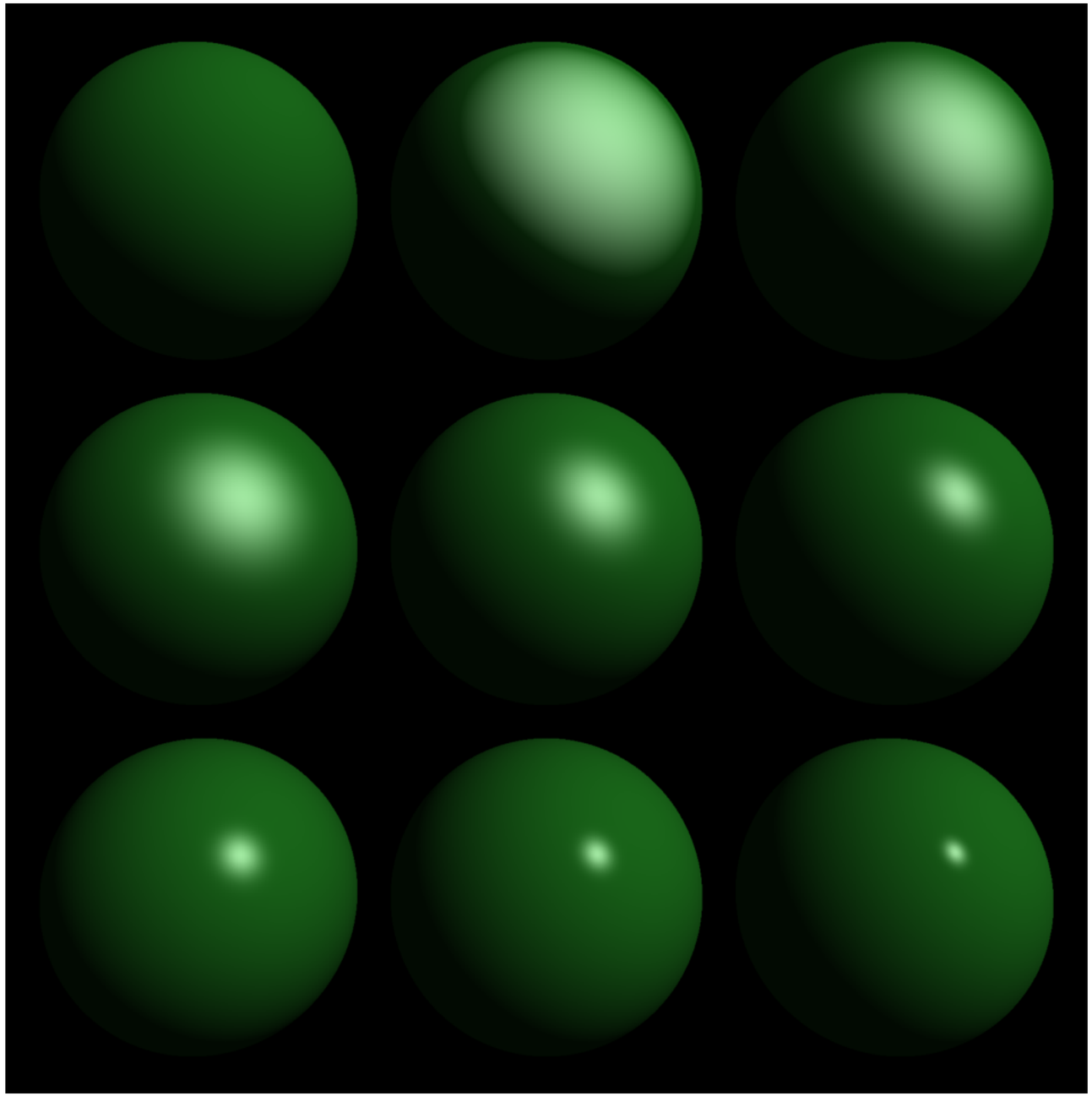


MATERIALS & SHADING

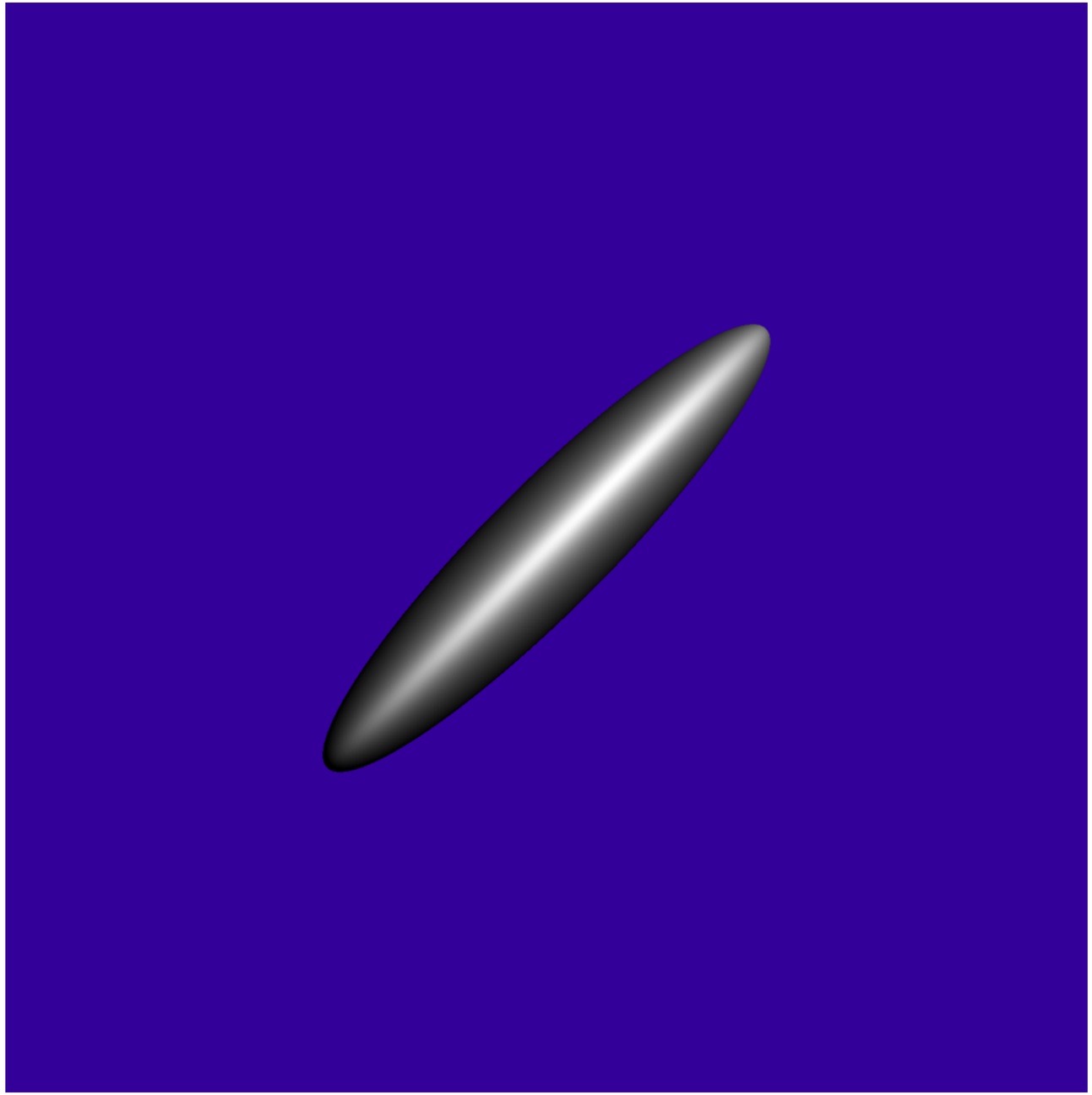
Lights

Diffuse shading

Phong shading



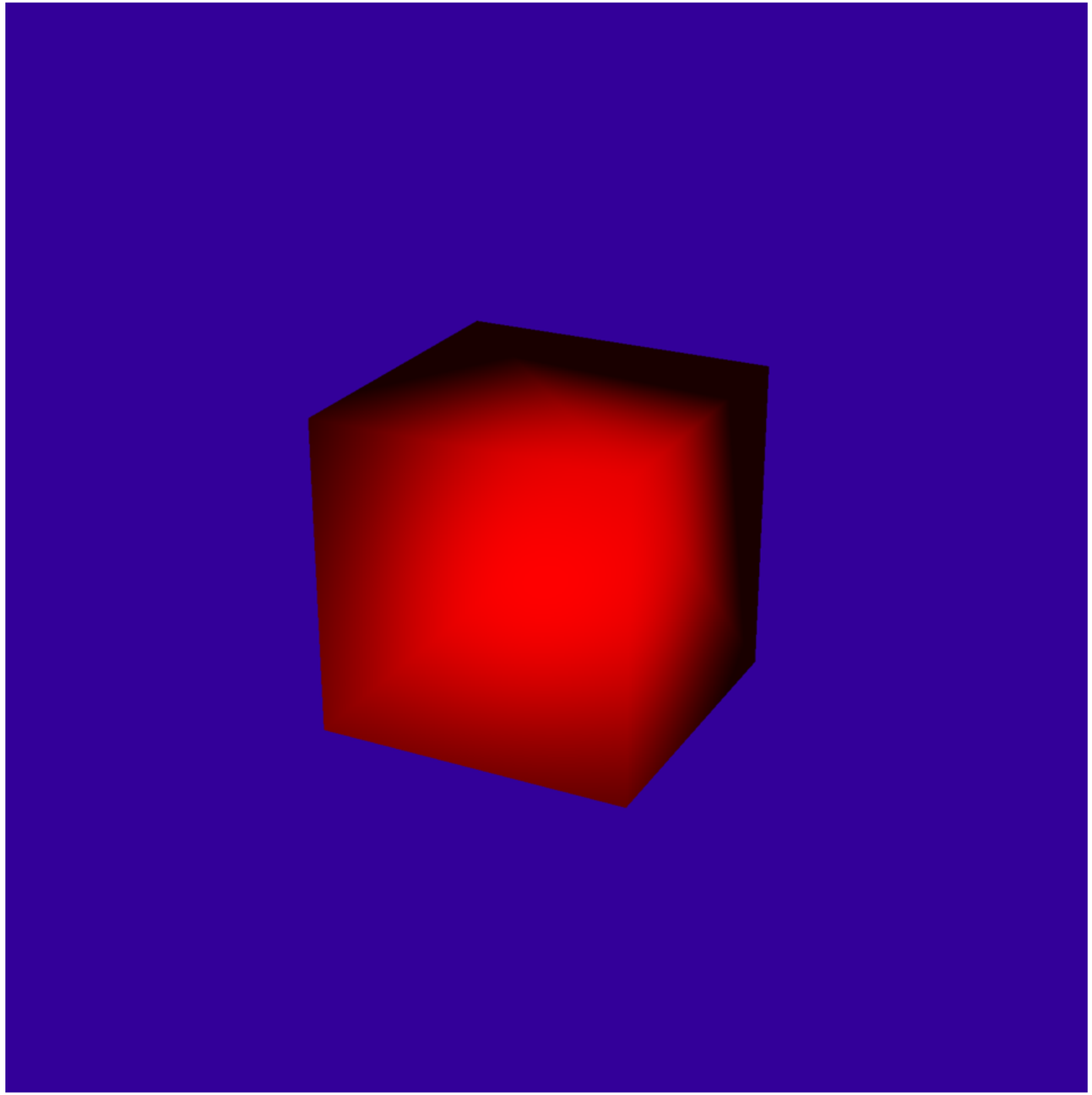
TRANSFORMS



TRIANGLES

Triangle mesh intersection

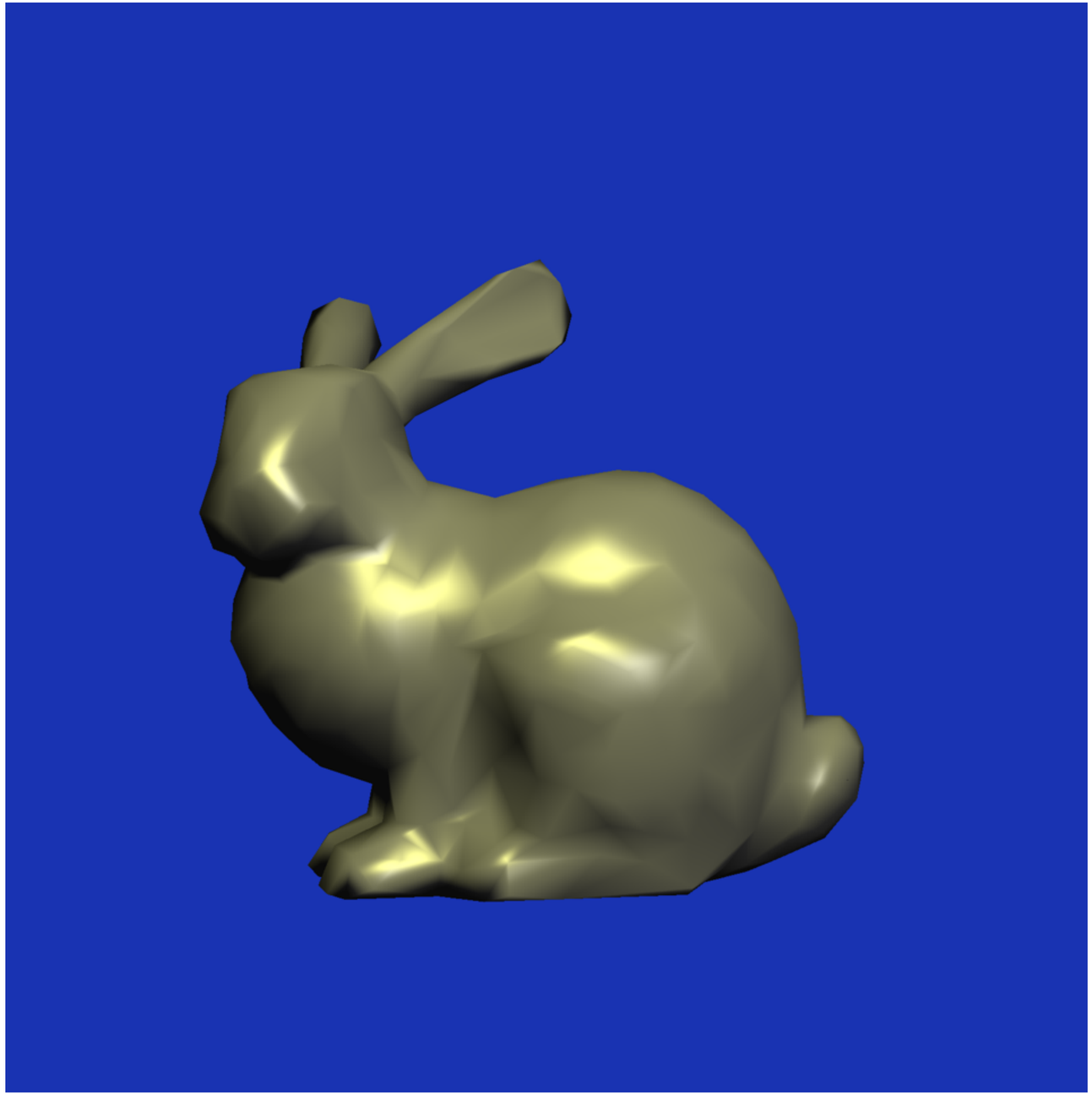
Optimization



TRIANGLES

Triangle mesh intersection

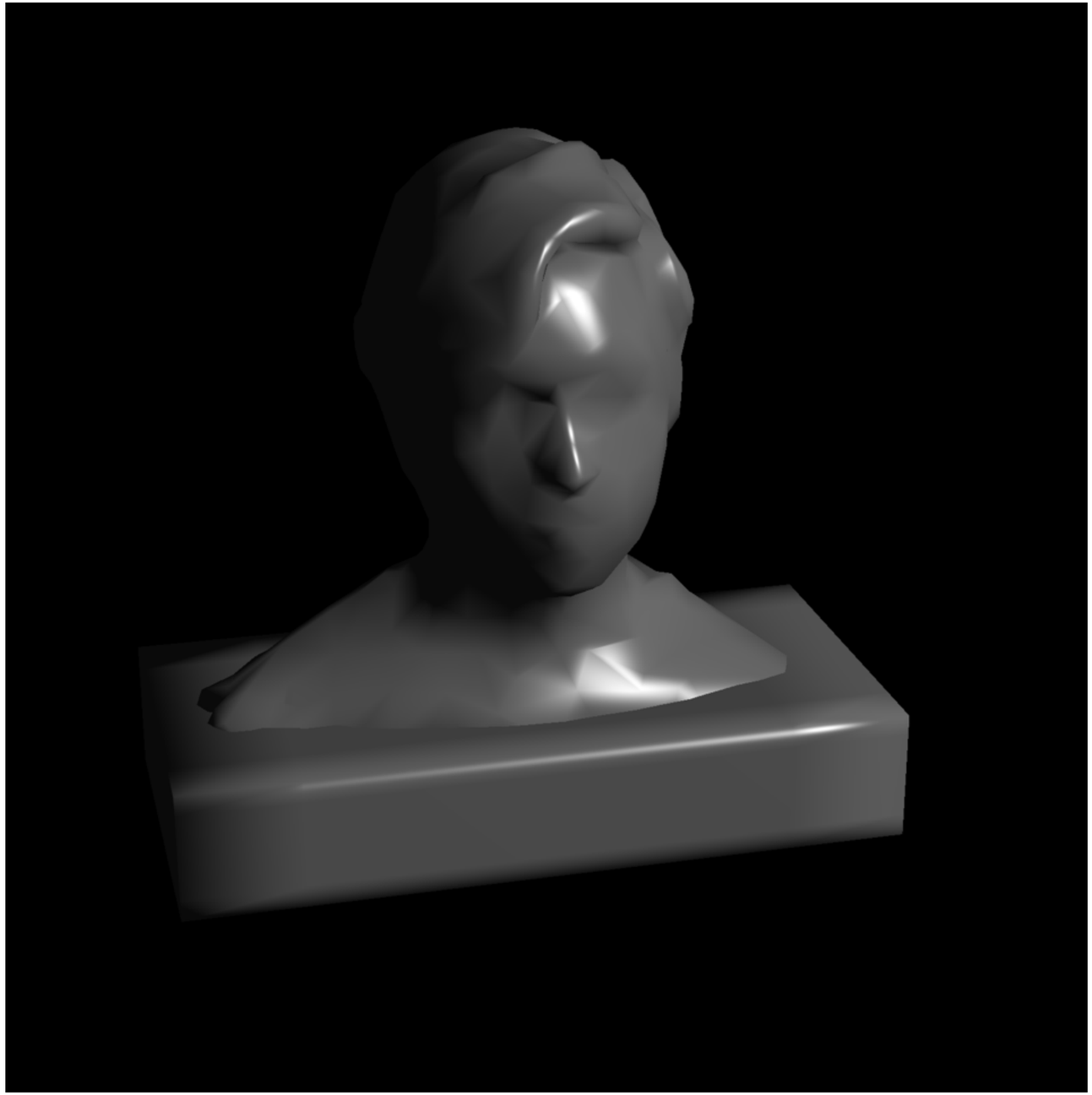
Optimization



TRIANGLES

Triangle mesh intersection

Optimization

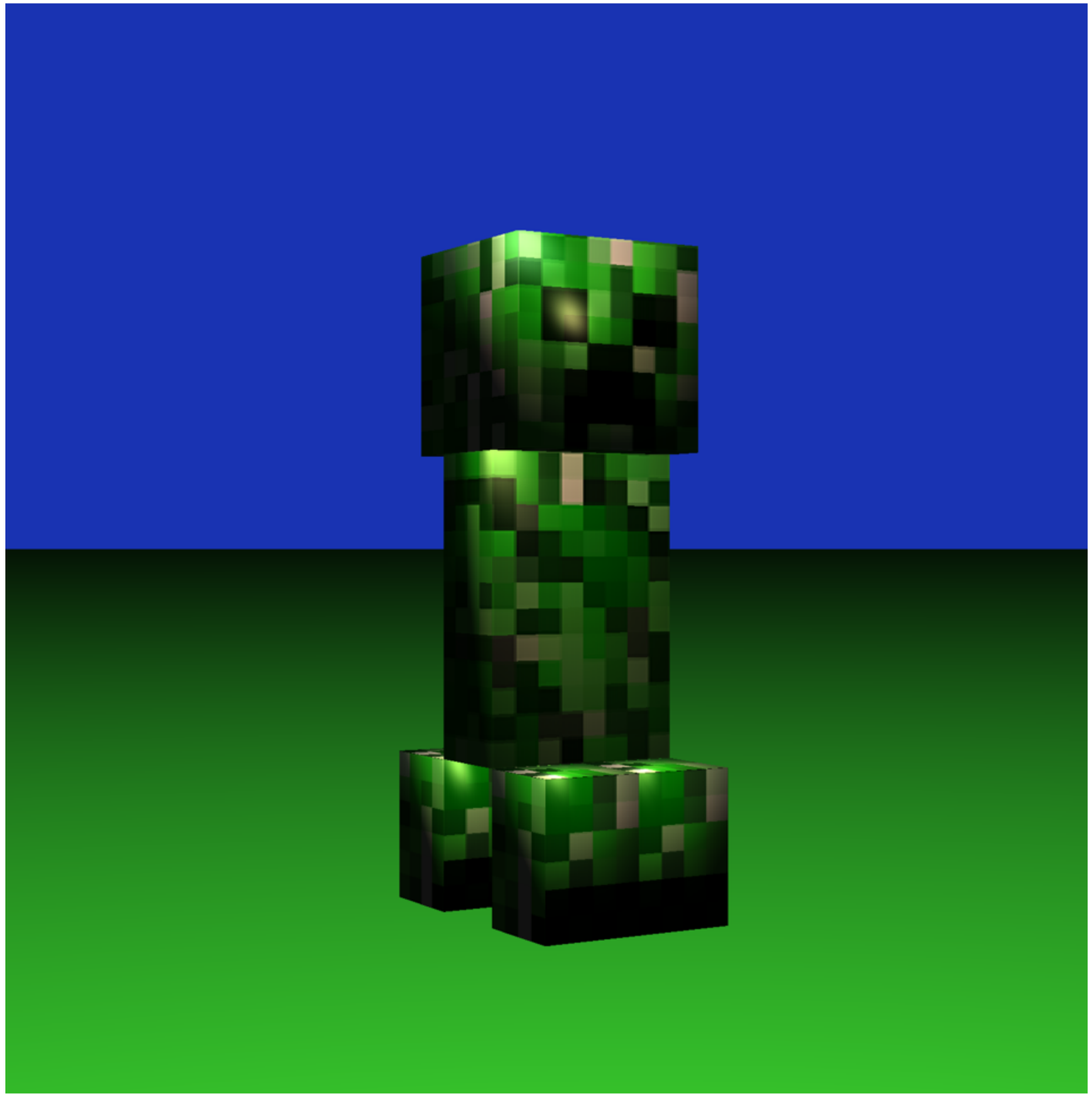


TRIANGLES

Triangle mesh intersection

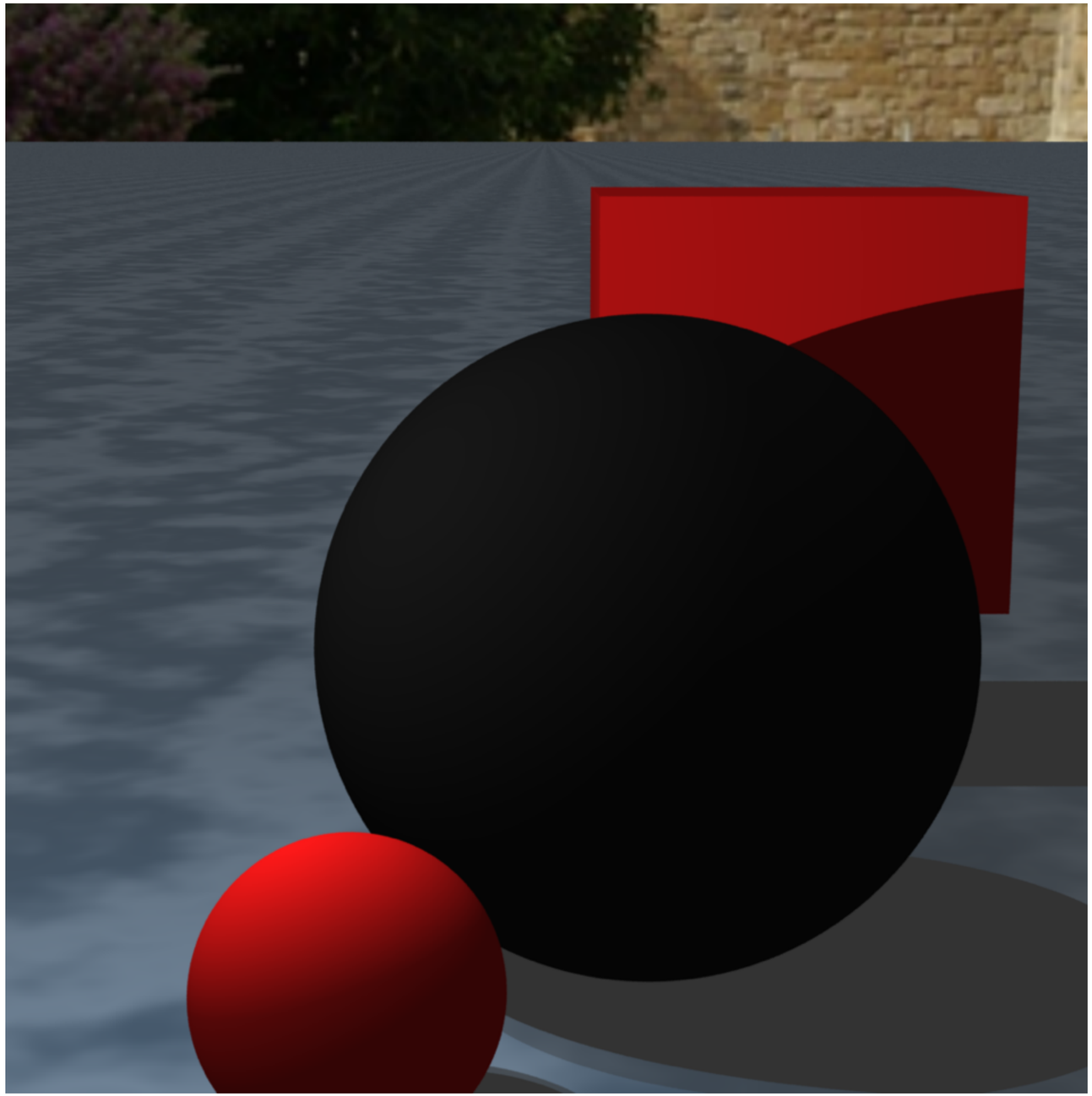
Optimization

TEXTURES

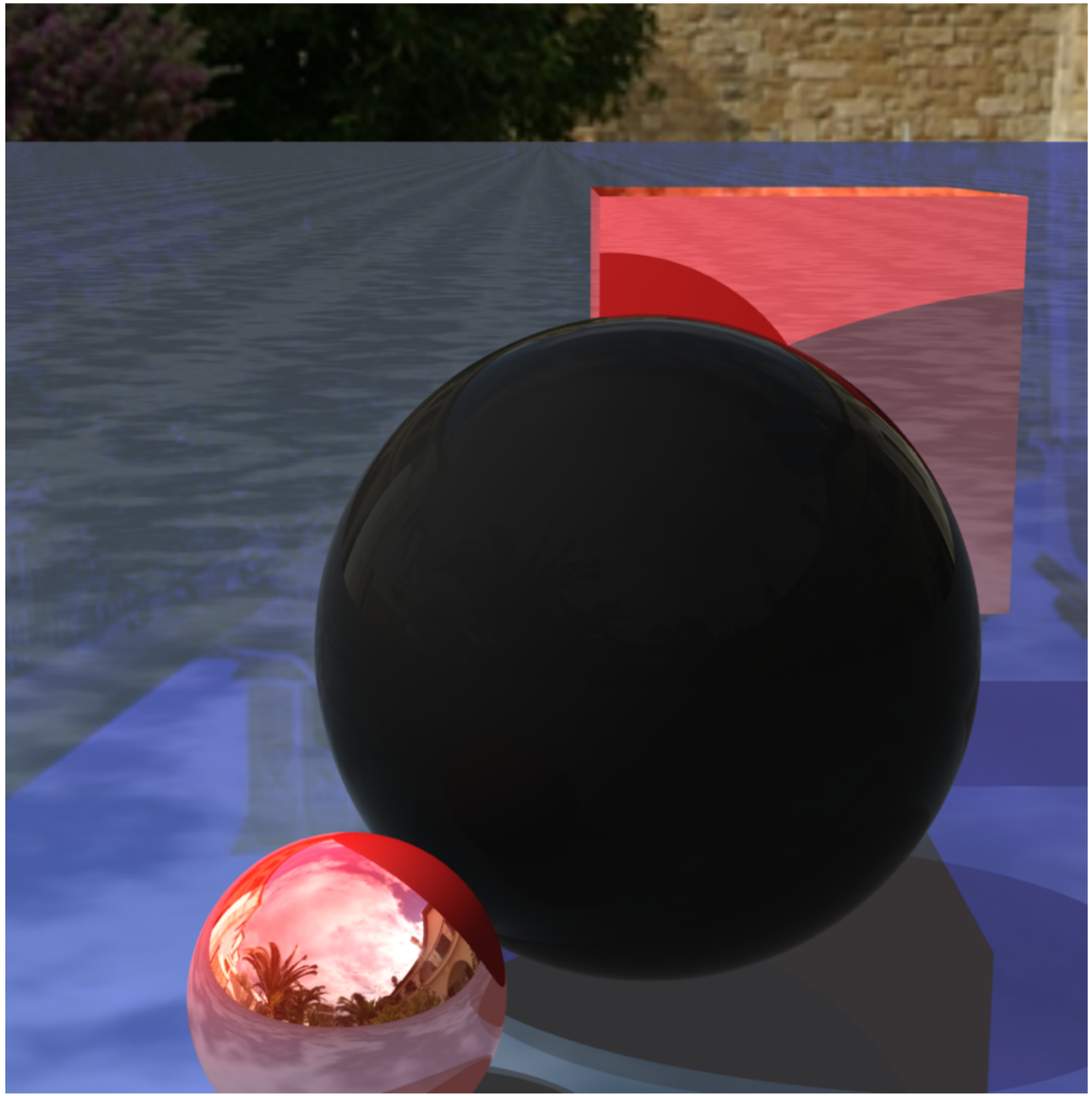


TEXTURES

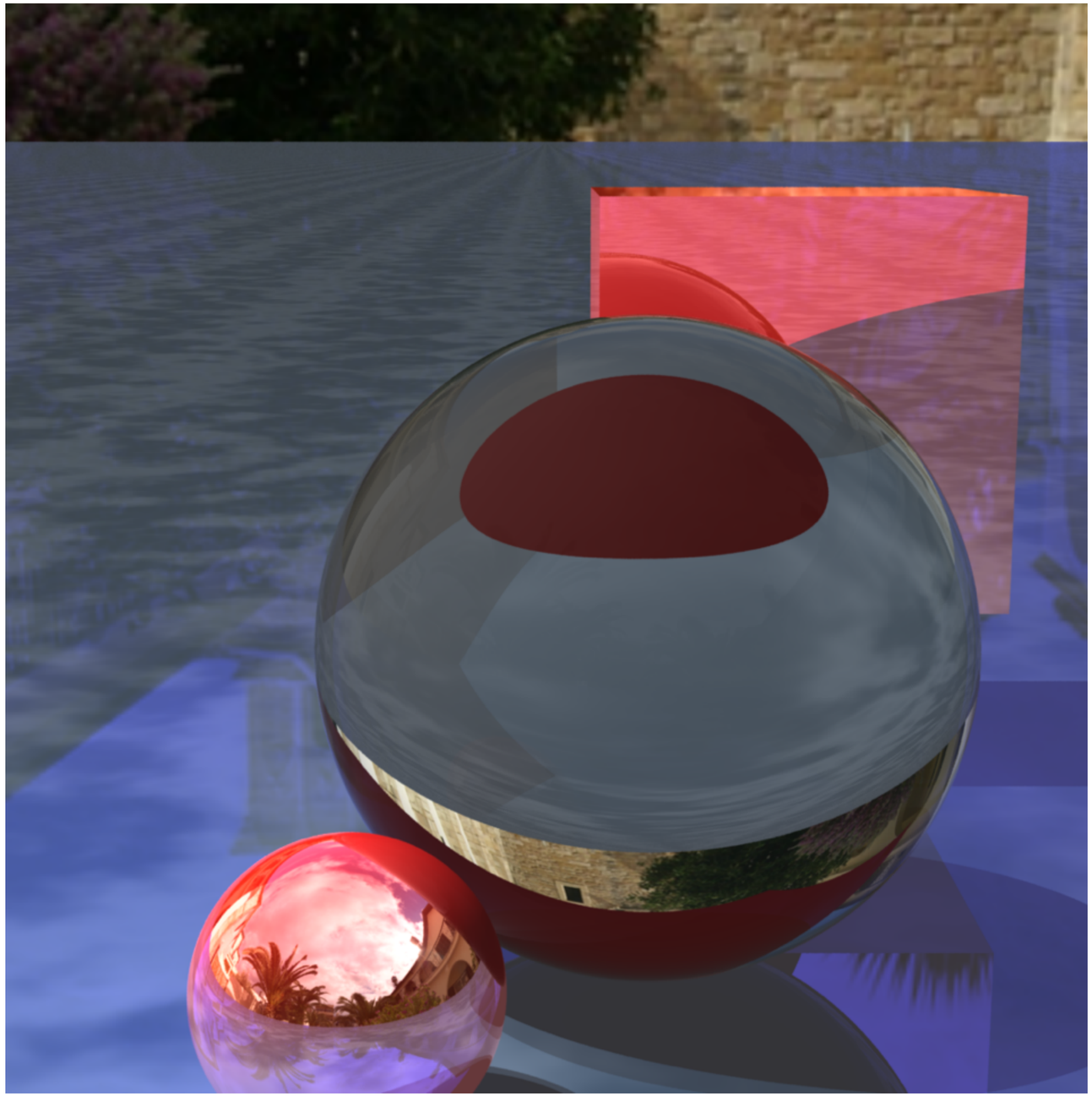
SHADOWS



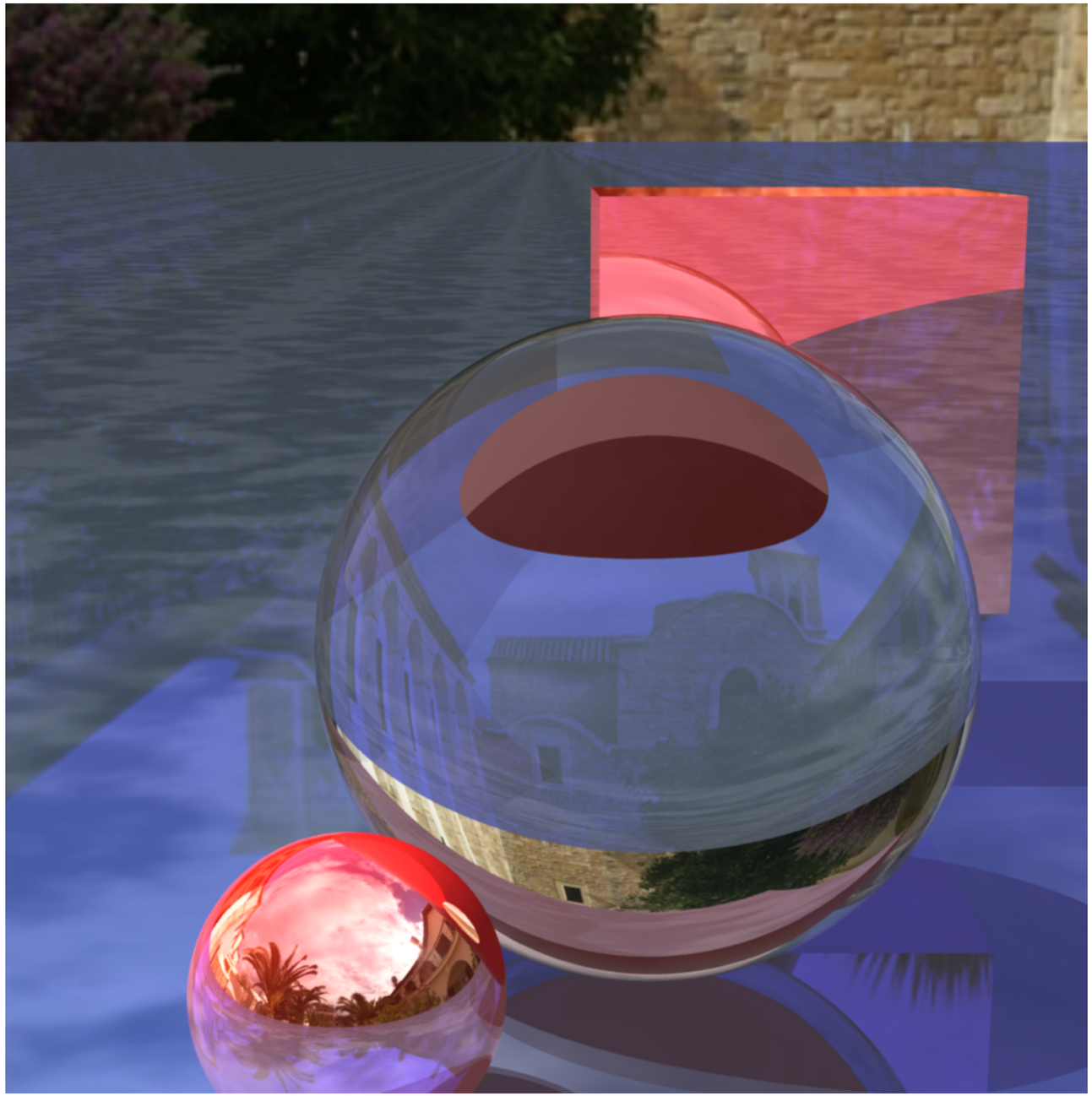
REFLECTION & REFRACTION



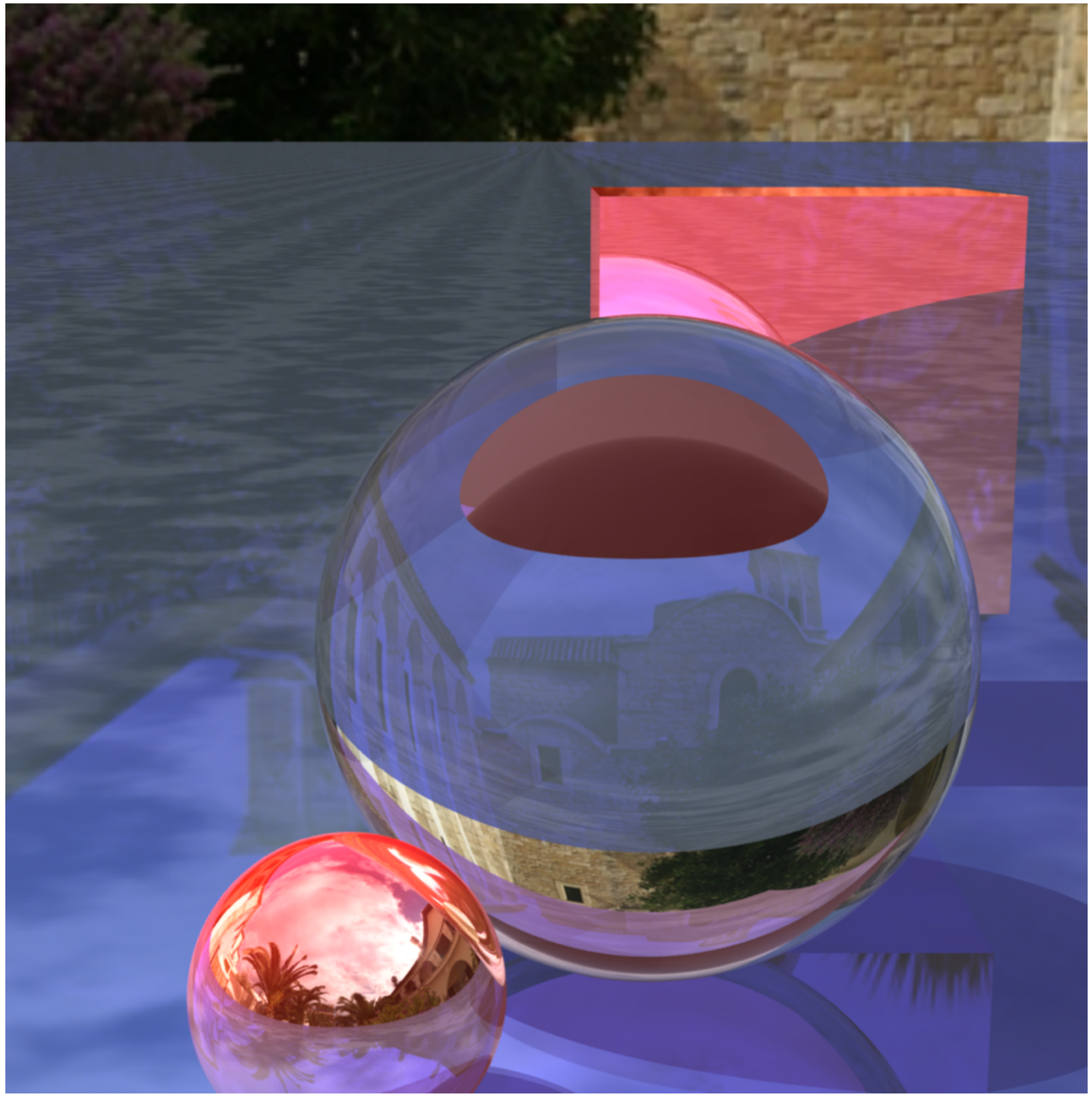
REFLECTION & REFRACTION



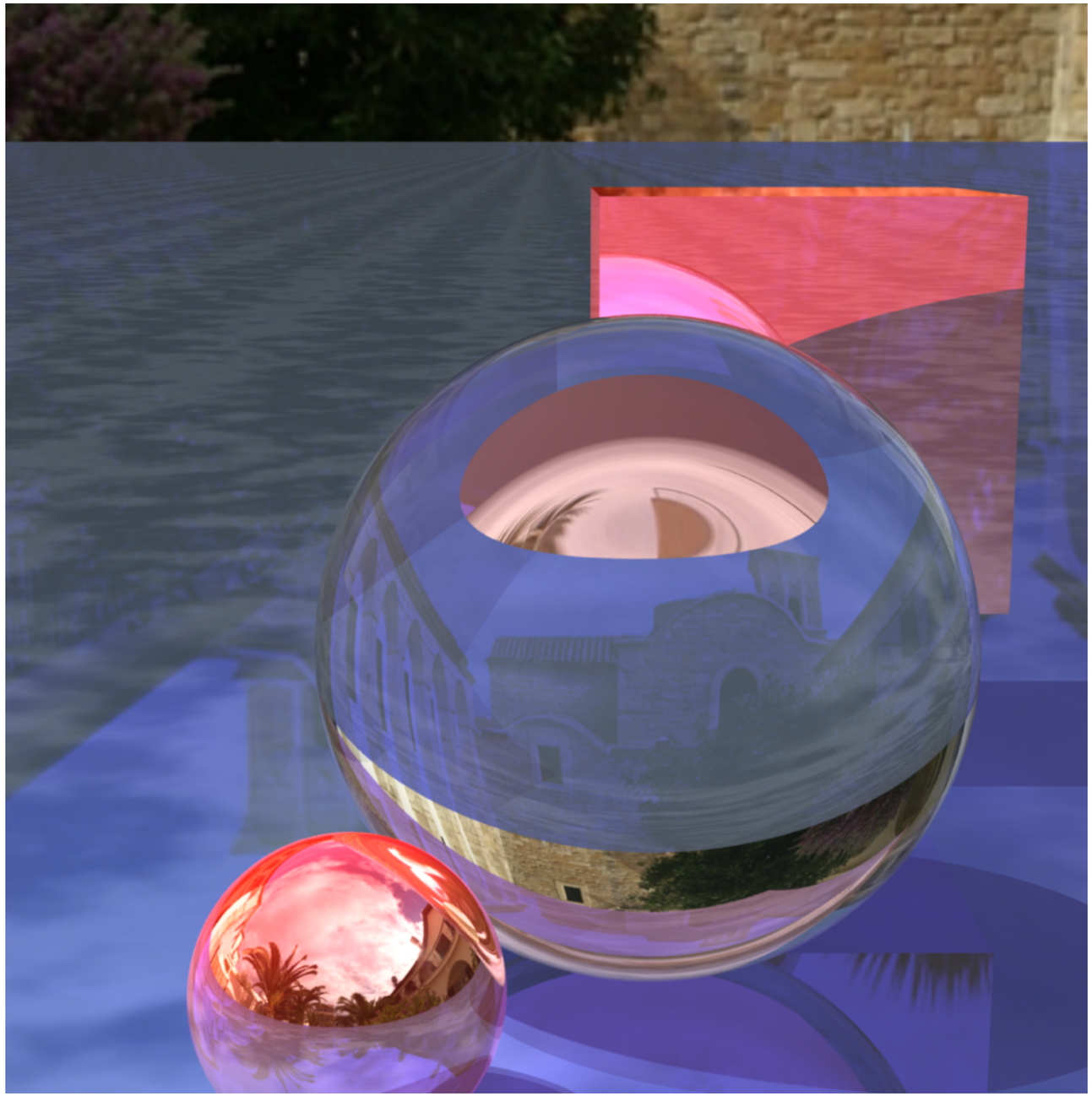
REFLECTION & REFRACTION



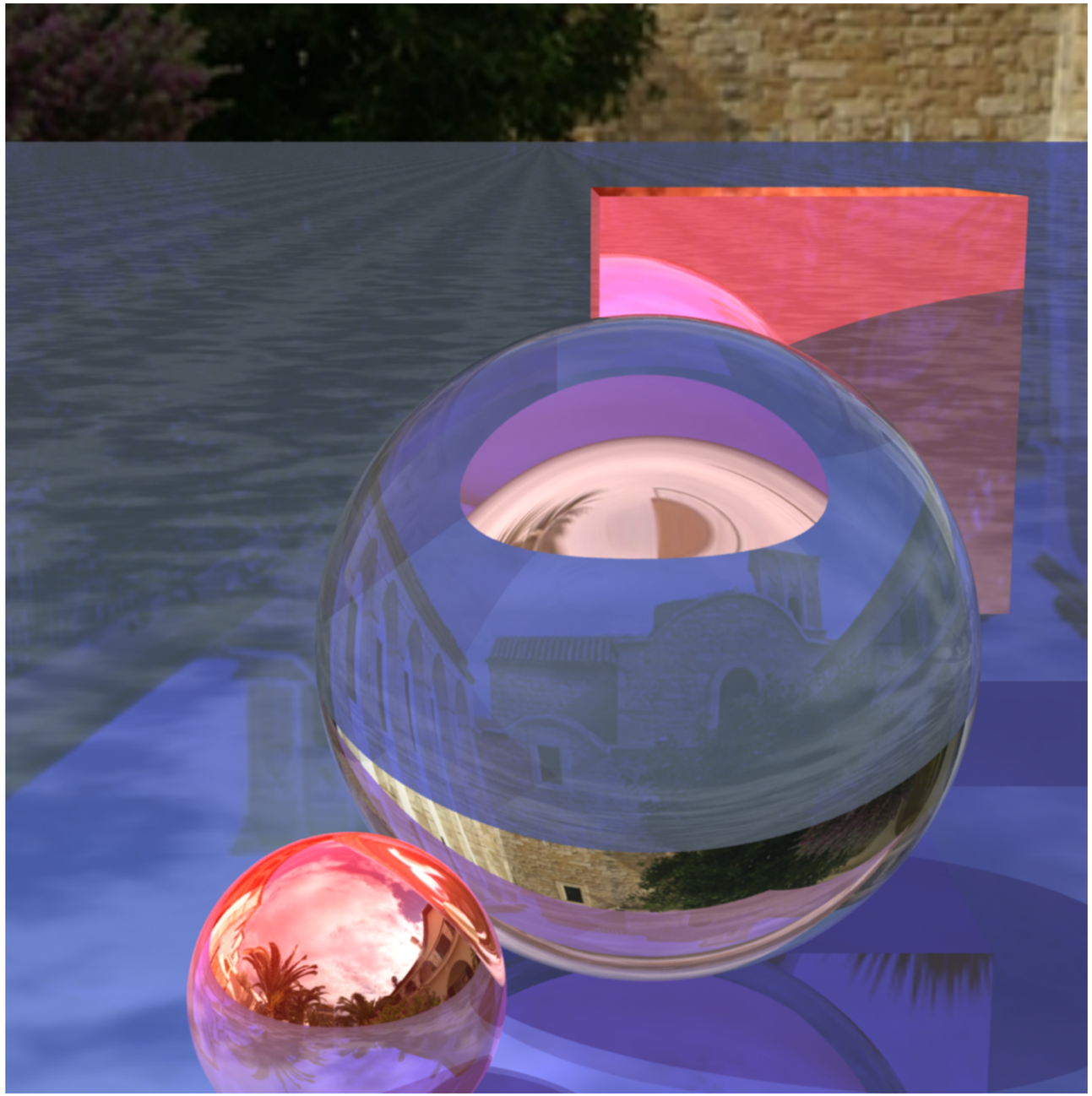
REFLECTION & REFRACTION



REFLECTION & REFRACTION



REFLECTION & REFRACTION



PROCEDURAL TEXTURES

FINAL RESULT

Course Overview

Mathematical toolbox

Ray caster

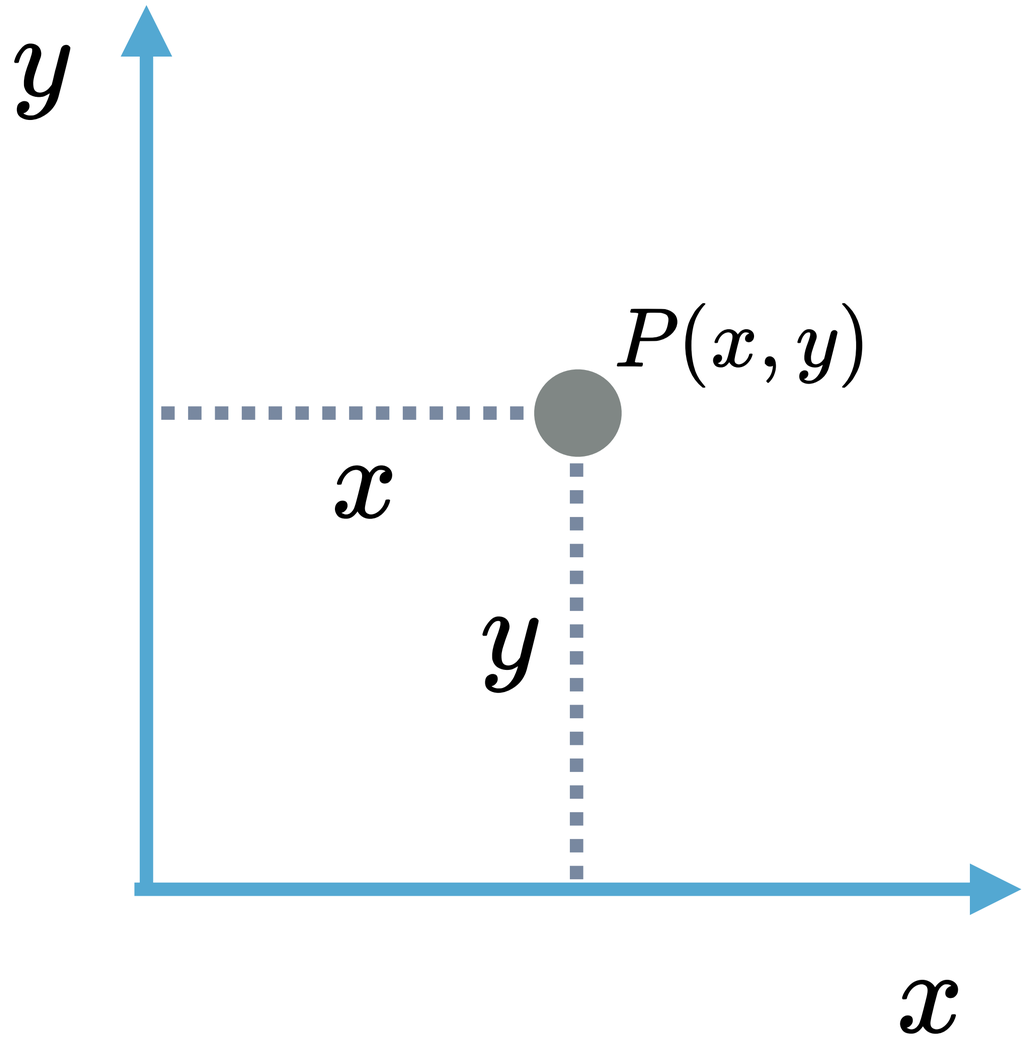
Camera & ray generation

Ray-plane intersection

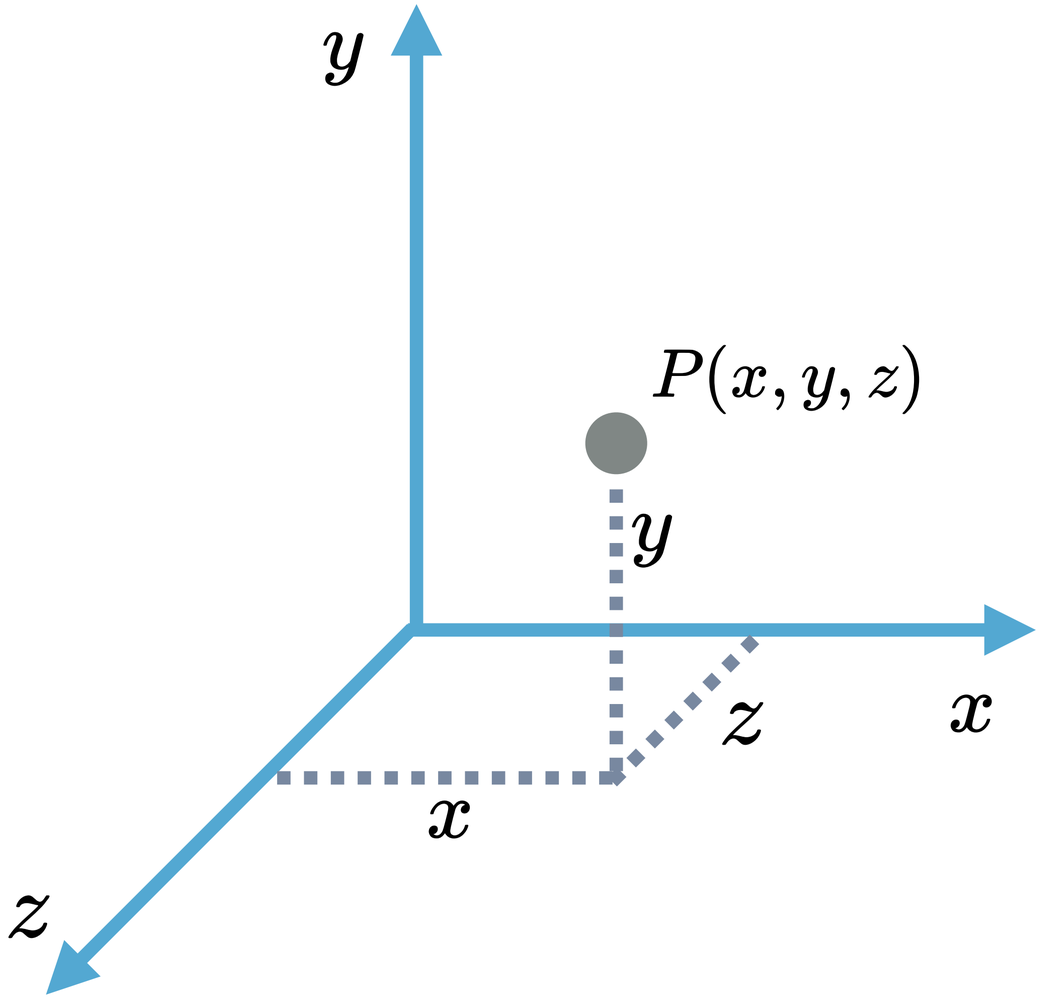
Depth images

Starter code & next week

POINTS IN 2D SPACE



POINTS IN 3D SPACE



VECTORS

Vectors have magnitude and direction

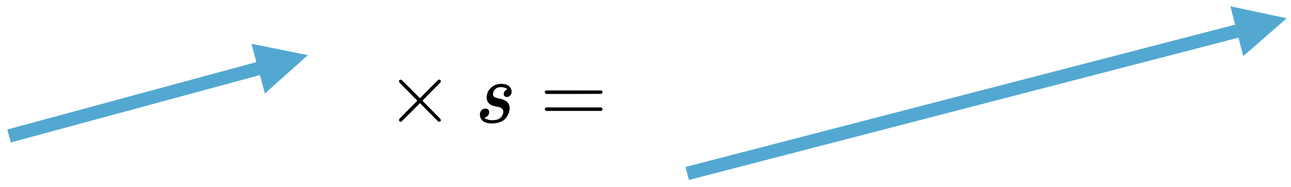
Used throughout computer graphics

$$\vec{A} = \ < A_x, A_y, A_z >$$

SCALING VECTORS

Multiply each element by the scalar *s*

$$\boldsymbol{As} = \ < \boldsymbol{A_x s}, \boldsymbol{A_y s}, \boldsymbol{A_z s} >$$



NORMALIZING VECTORS

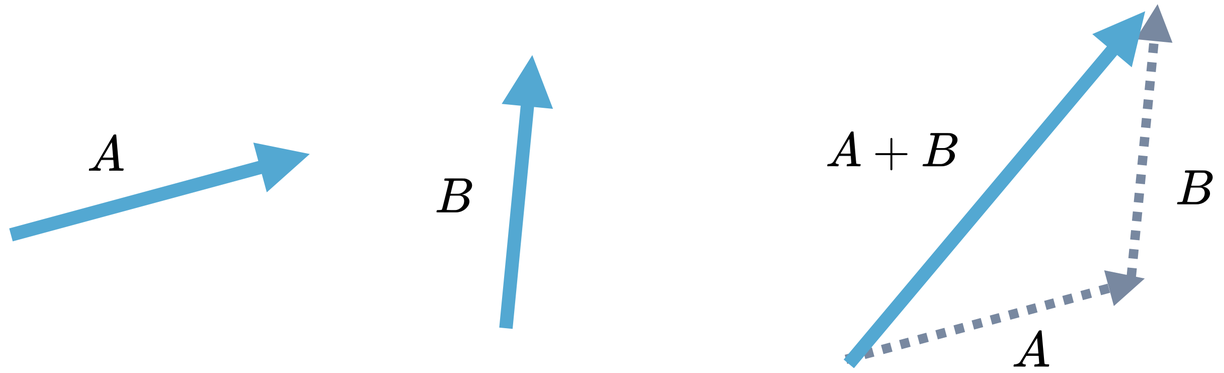
Create a vector with same direction but length of one

$$normalize(A) = \frac{A}{||A||}$$

VECTOR ADDITION

Add each element to the corresponding element of other vector

$$\mathbf{A} + \mathbf{B} = \ < A_x + B_x,\ A_y + B_y,\ A_z + B_z >$$



DOT PRODUCT

Extremely useful for light calculations in shading step

$$\mathbf{A} \cdot \mathbf{B} = \sum_{i=1}^{3} A_i B_i = A_x B_x + A_y B_y + A_z B_z$$

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\| \cos\theta$$

When perpendicular

$$\mathbf{A} \cdot \mathbf{B} = 0$$

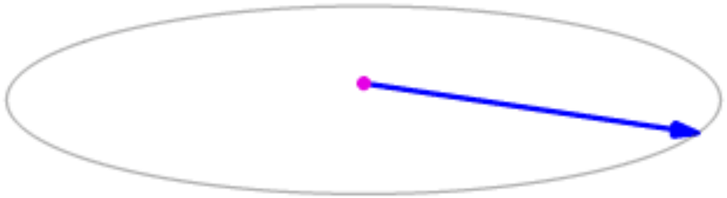
When parallel

$$\mathbf{A} \cdot \mathbf{B} = \|\mathbf{A}\| \, \|\mathbf{B}\|$$

CROSS PRODUCT

Follows the right hand rule

Used to find normals



$$A \times B = \|A\|\|B\| \sin\theta \, n$$

Course Overview

Mathematical toolbox

Ray caster

Camera & ray generation

Ray-plane intersection

Depth images

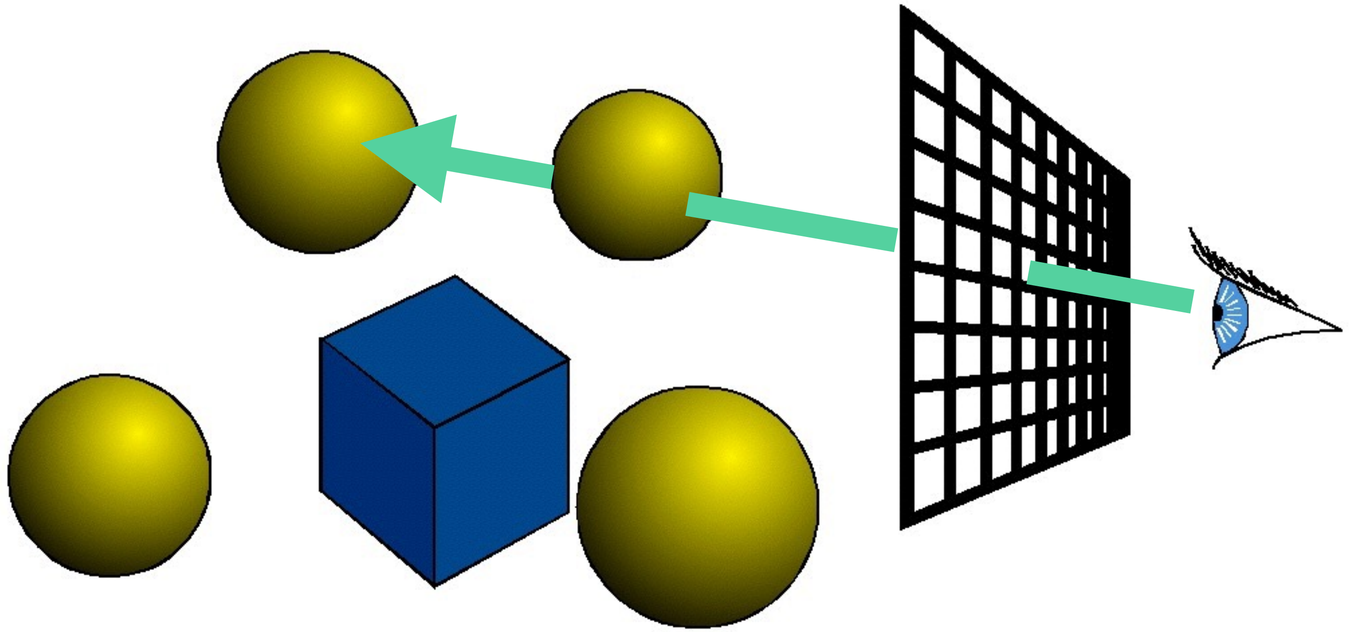
Starter code & next week

WHAT IS A RAY CASTER?

Renders 2D images from 3D geometry

Simulation of a camera

Different from rasterization



```
for every pixel
  construct ray from eye to pixel
    for every object in the scene
      find intersection t with ray
      save closest t
  shade closest using lights, normal, material
```

Course Overview

Mathematical toolbox

Ray caster

Camera & ray generation

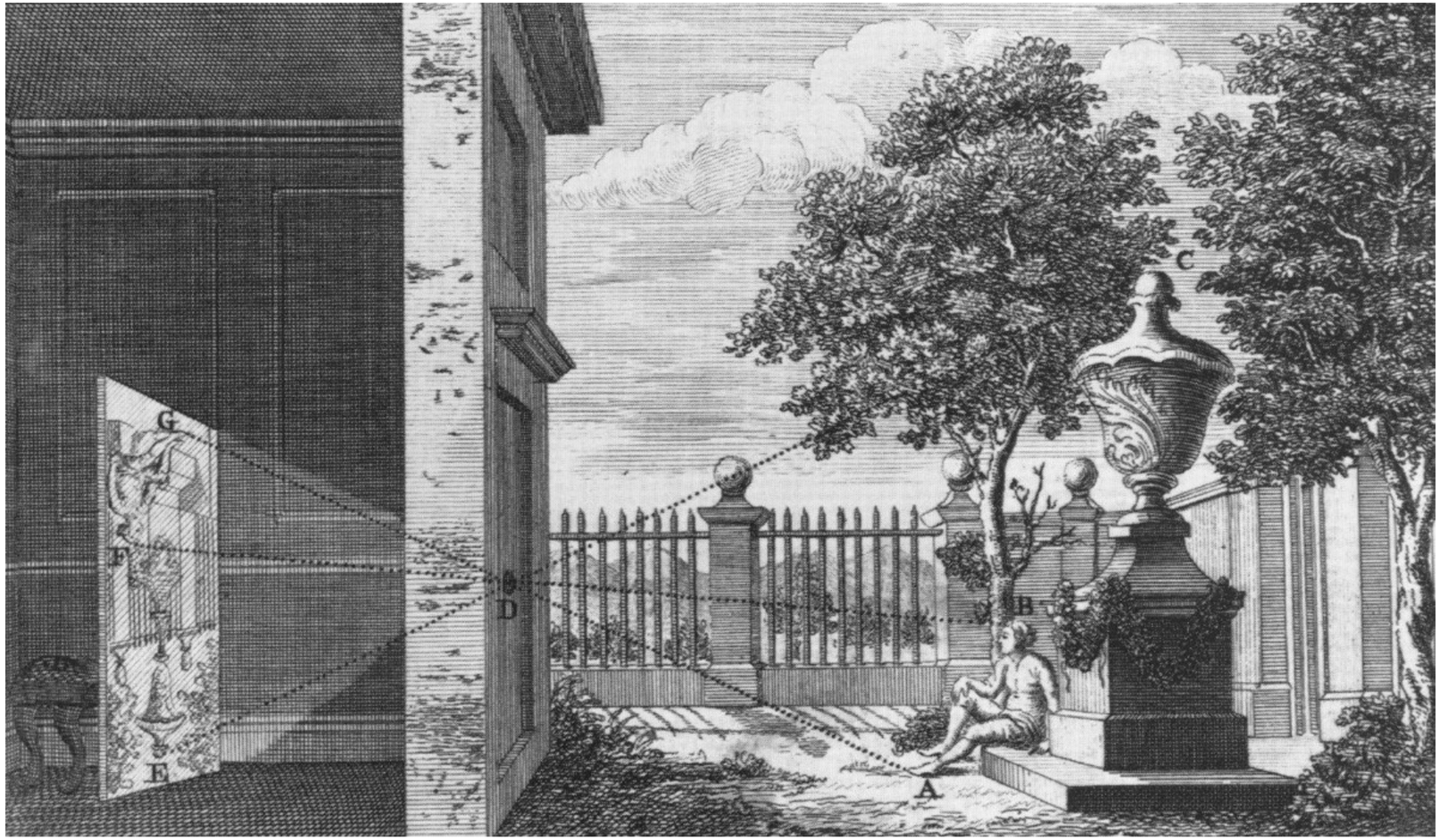
Ray-plane intersection

Depth images

Starter code & next week

```
for every pixel
  construct ray from eye to pixel
    for every object in the scene
      find intersection t with ray
      save closest t
  shade closest using lights, normal, material
```

CAMERA OBSCURA



PINHOLE CAMERA

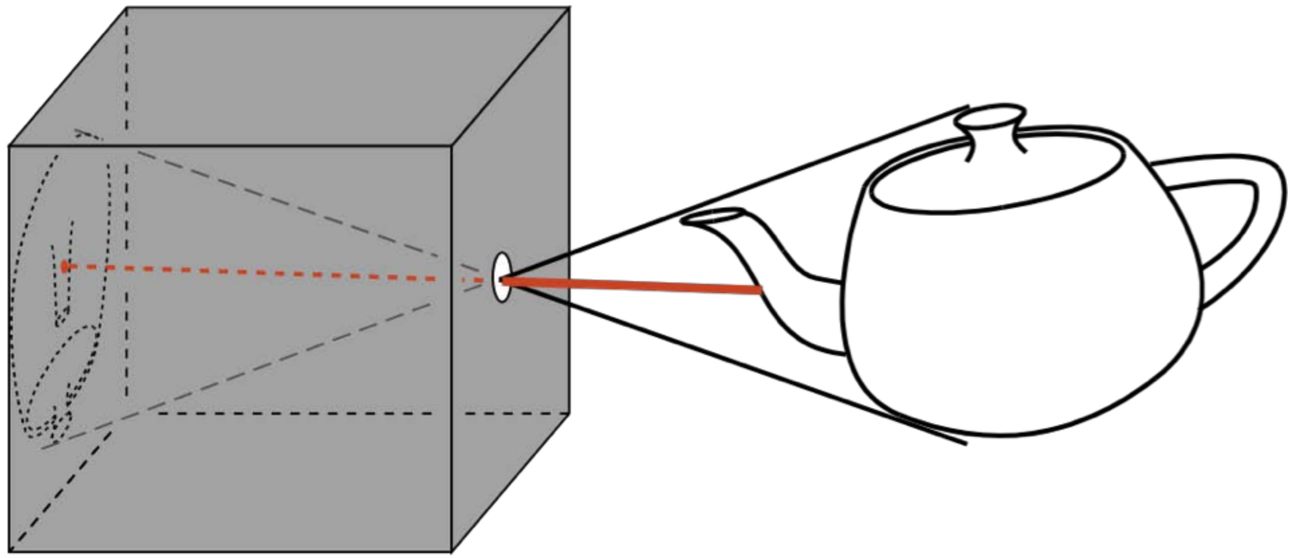
box with tiny hole

inverted image

everything in focus due to "pinhole"

pure geometric optics

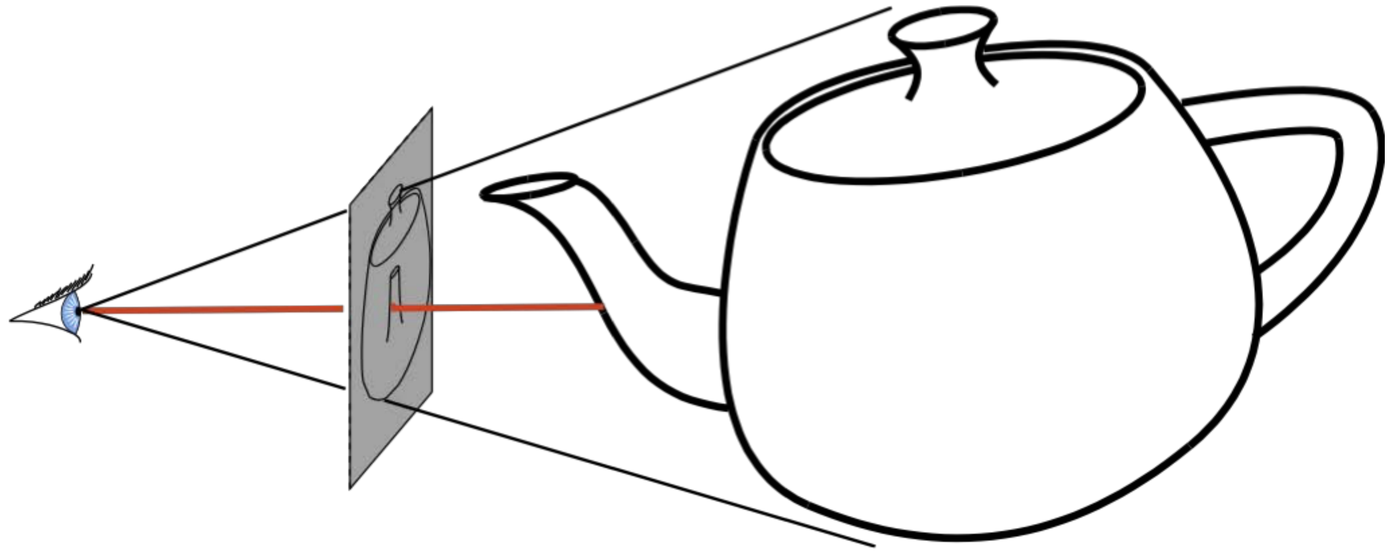
similar to what we'll be doing



SIMPLIFIED PINHOLE CAMERA

Eye-image pyramid (view frustum)

Distance/size of image is arbitrary



PERSPECTIVE CAMERA

Eye point ***e***
(center)

Orthobasis ***u, v, w***
(horizontal, up, direction)

Field of view ***angle***

Image rectangle ***aspect ratio***

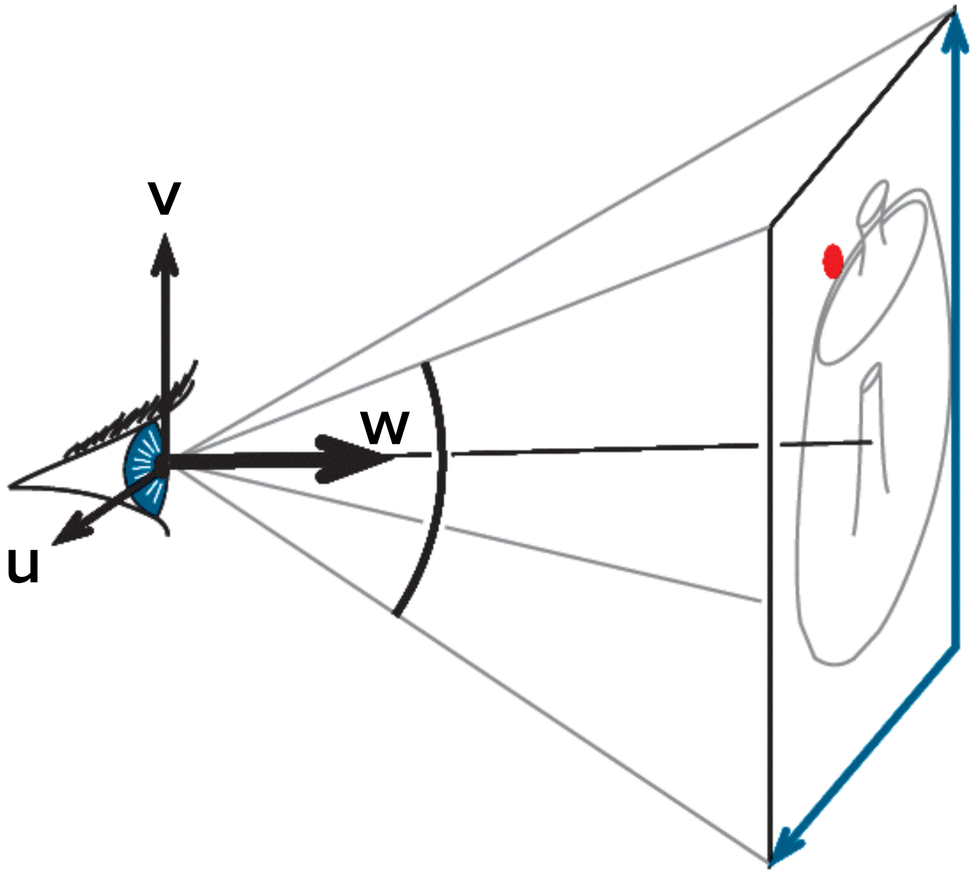
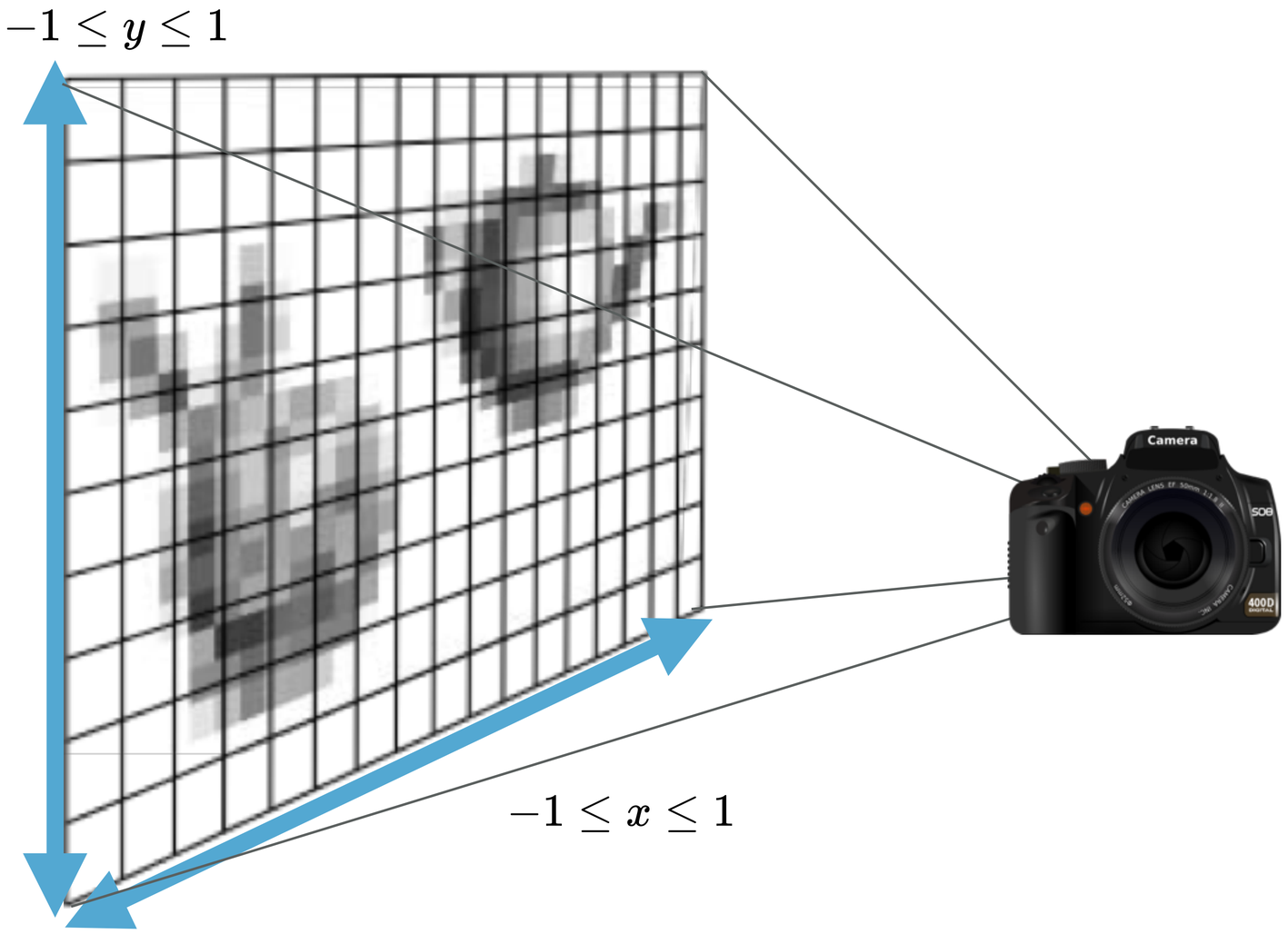


IMAGE COORDINATES

we want to use "normalized coordinates" to make our life easier

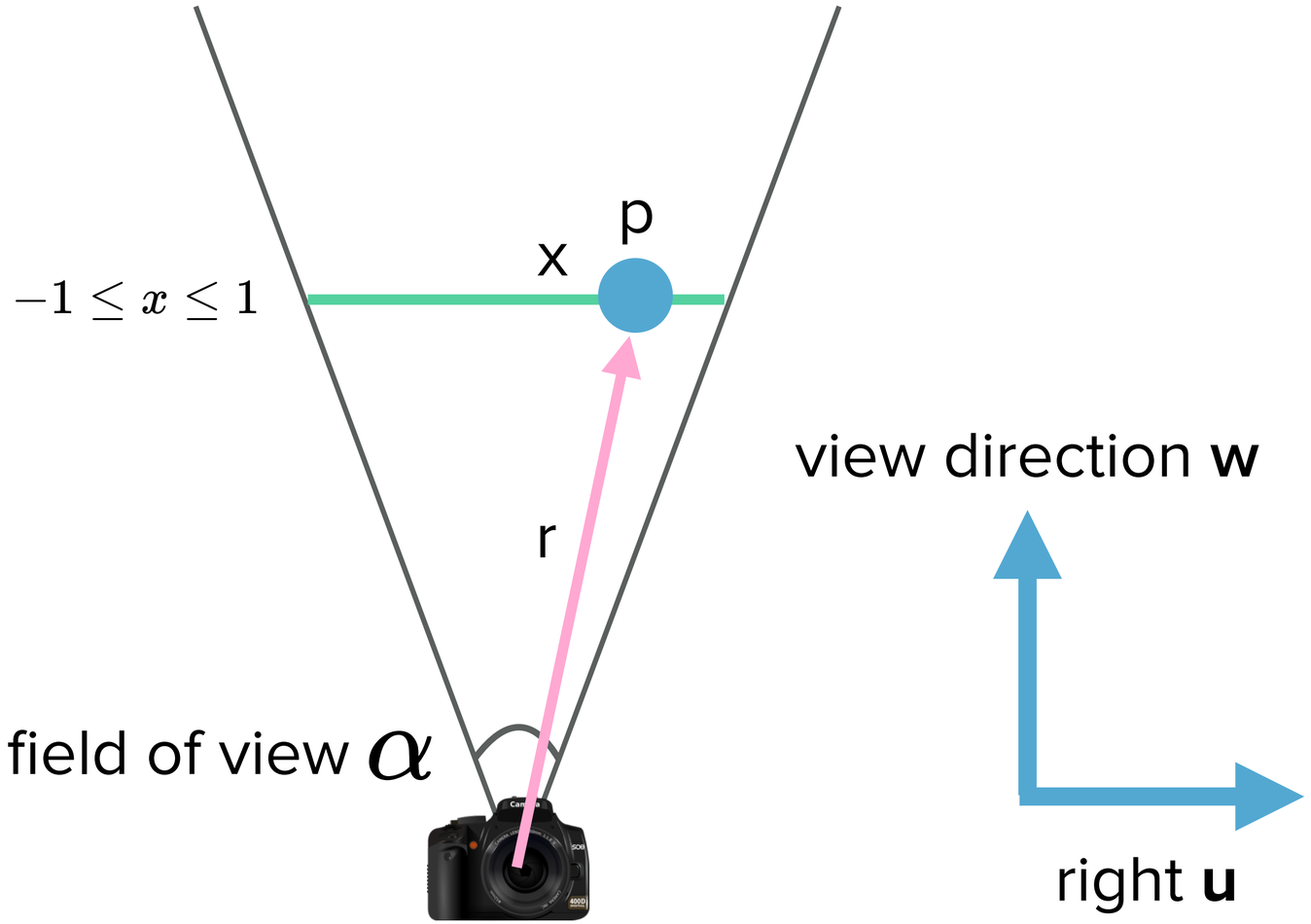
x, y should go from -1 to 1 regardless of aspect ratio



RAY GENERATION IN 2D

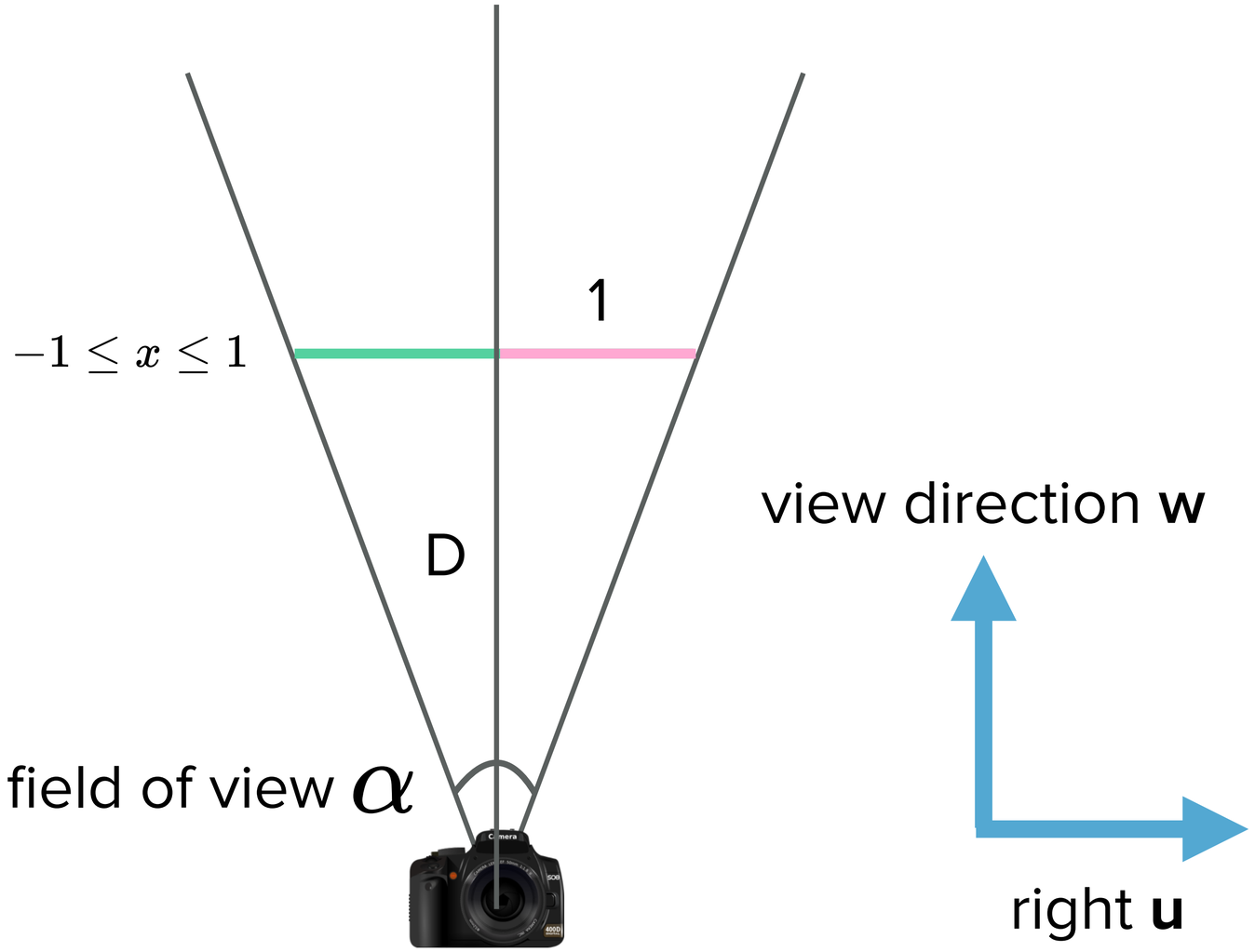
p *is a point on the image plane at coordinate* ***x***

What is the direction of the ray **r** *from the camera to the point?*



RAY GENERATION IN 2D

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

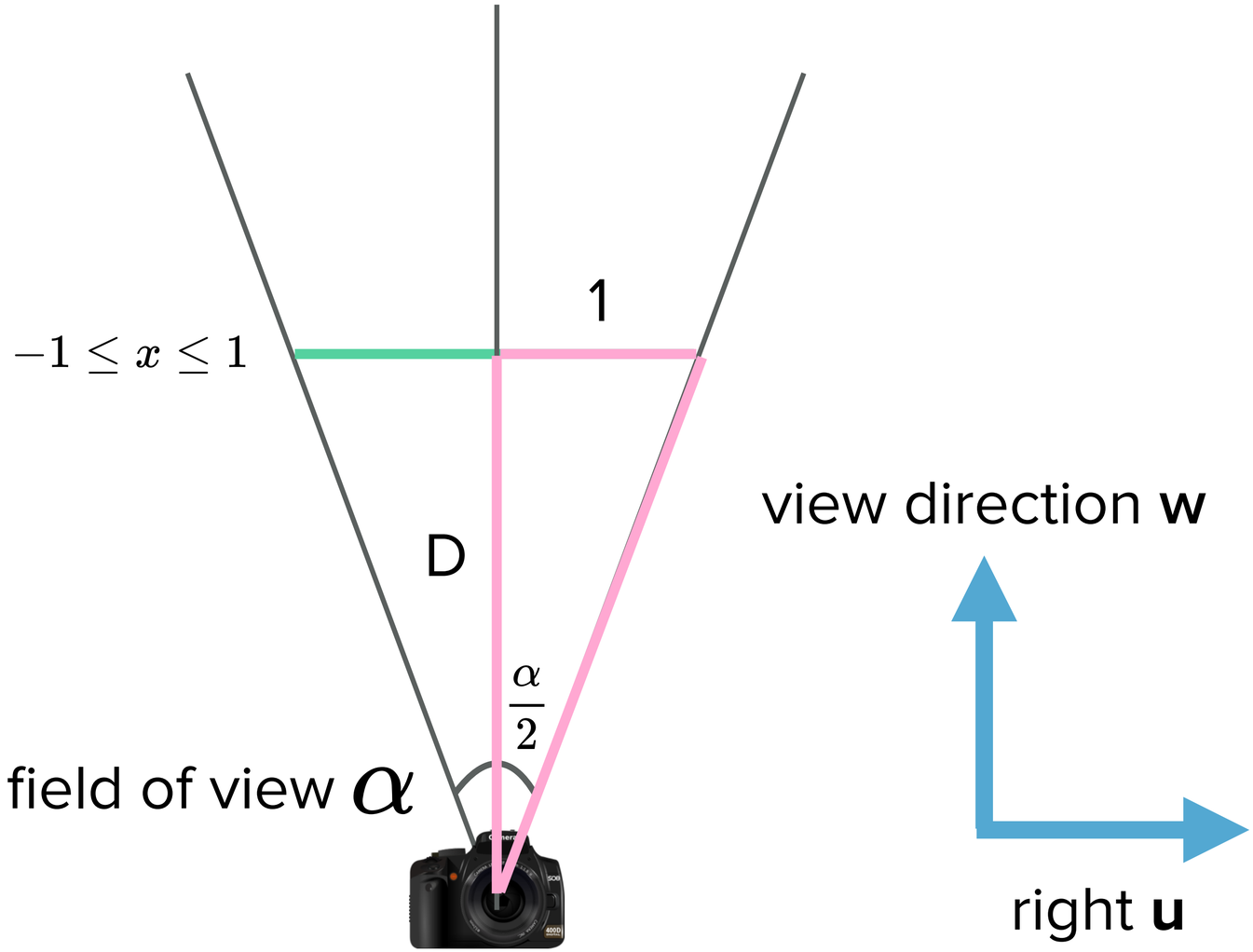


RAY GENERATION IN 2D

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

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

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



RAY GENERATION IN 2D

Calculate the ray

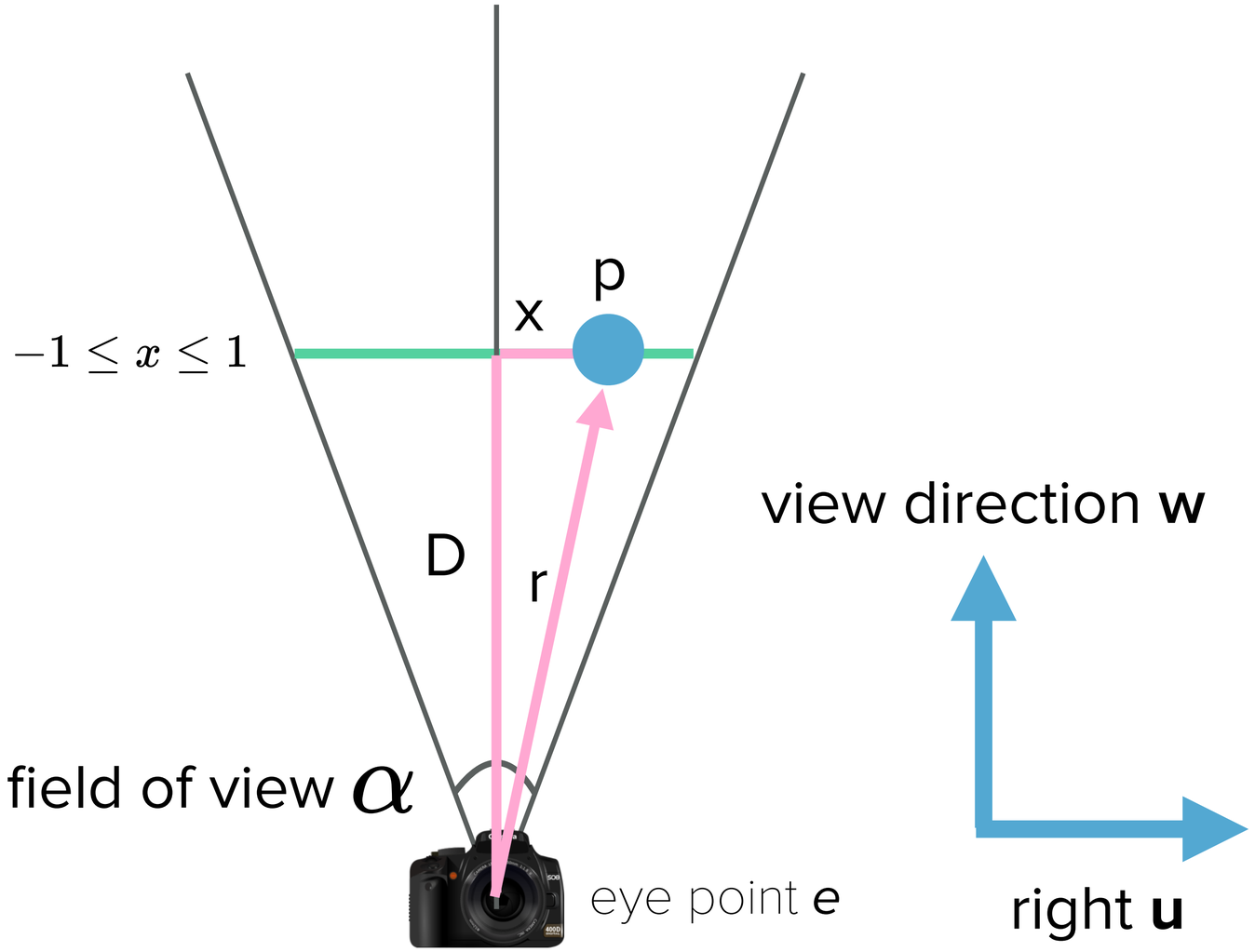
$$r = p - e = (xu, Dw)$$

Normalize it to get direction

$$d = \frac{r}{||r||}$$

Any point on ray can be expressed by

$$P(t) = e + td$$



3D WORKS JUST THE SAME

y is same as **x** but accounts for aspect ratio

$$r = x * u + aspect * y * v + D * w$$

Aspect ratio is for non-square views (16:9, etc)

Allows us to use [-1, 1] for image coordinates

ORTHOGRAPHIC CAMERA

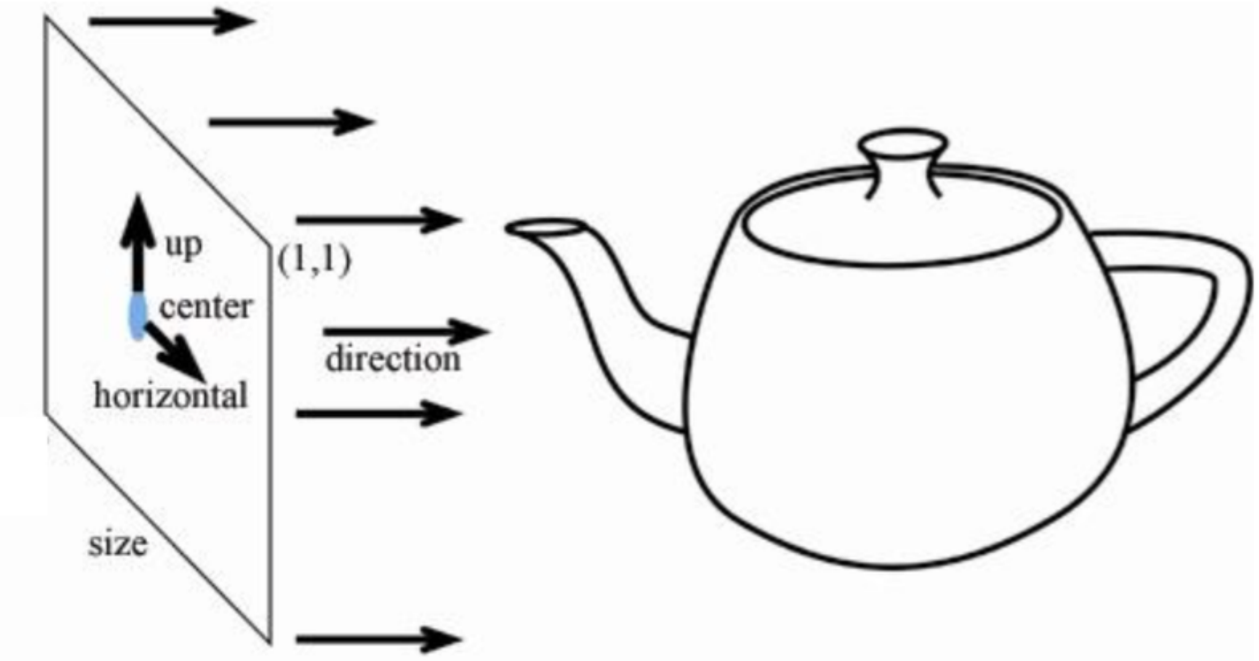
move the origin with each pixel

$$origin = e + x * size * u + y * size * v$$

direction is always **w**

creates a fisheye style camera

different cameras allow for different effects



Course Overview

Mathematical toolbox

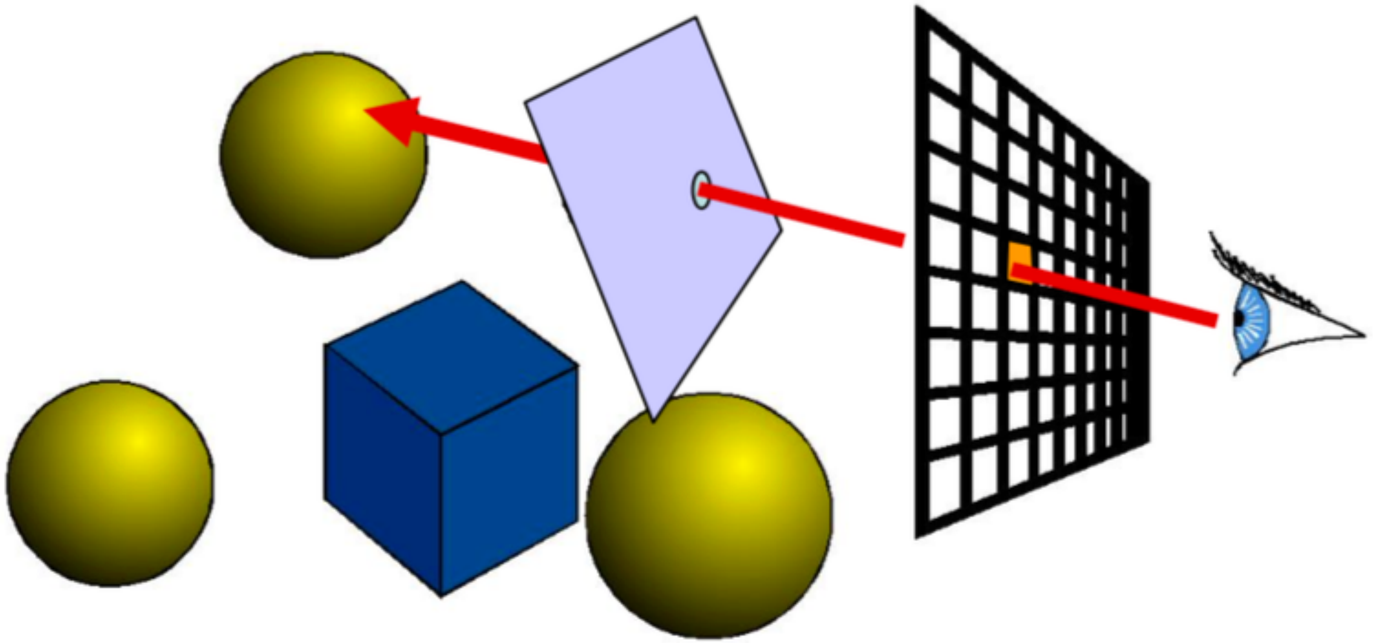
Ray caster

Camera & ray generation

Ray-plane intersection

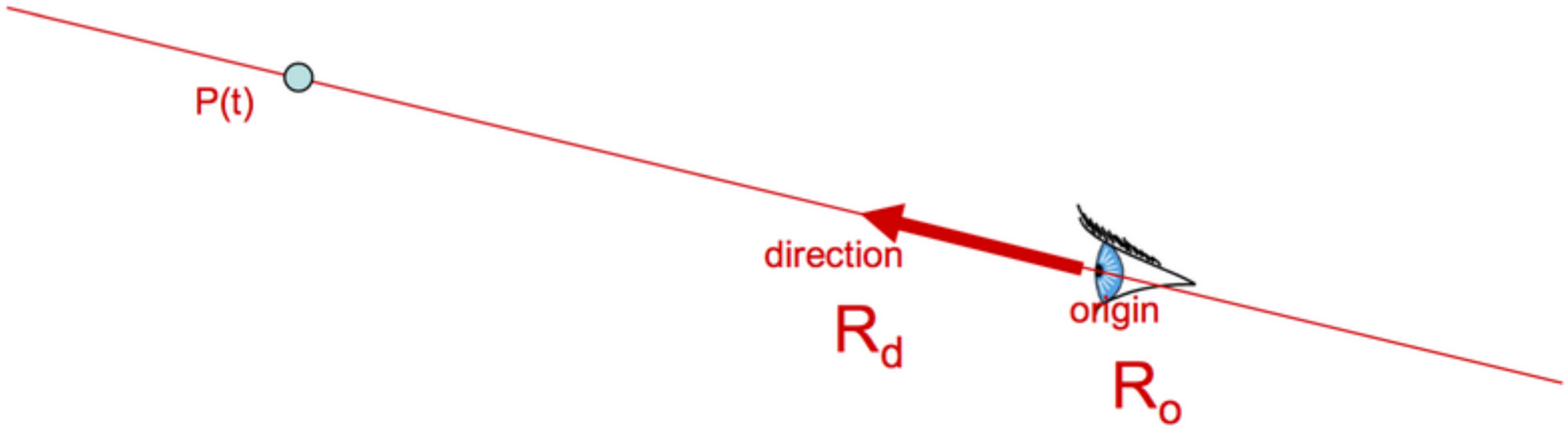
Depth images

Starter code & next week



```
for every pixel
  construct ray from eye to pixel
    for every object in the scene
      find intersection t with ray
      save closest t
  shade closest using lights, normal, material
```

PARAMETRIC LINE



$$P(t) = R_o + t * R_d$$

PLANE EQUATION

Infinite plane is defined by

$$P_o = (x_o, y_o, z_o)$$

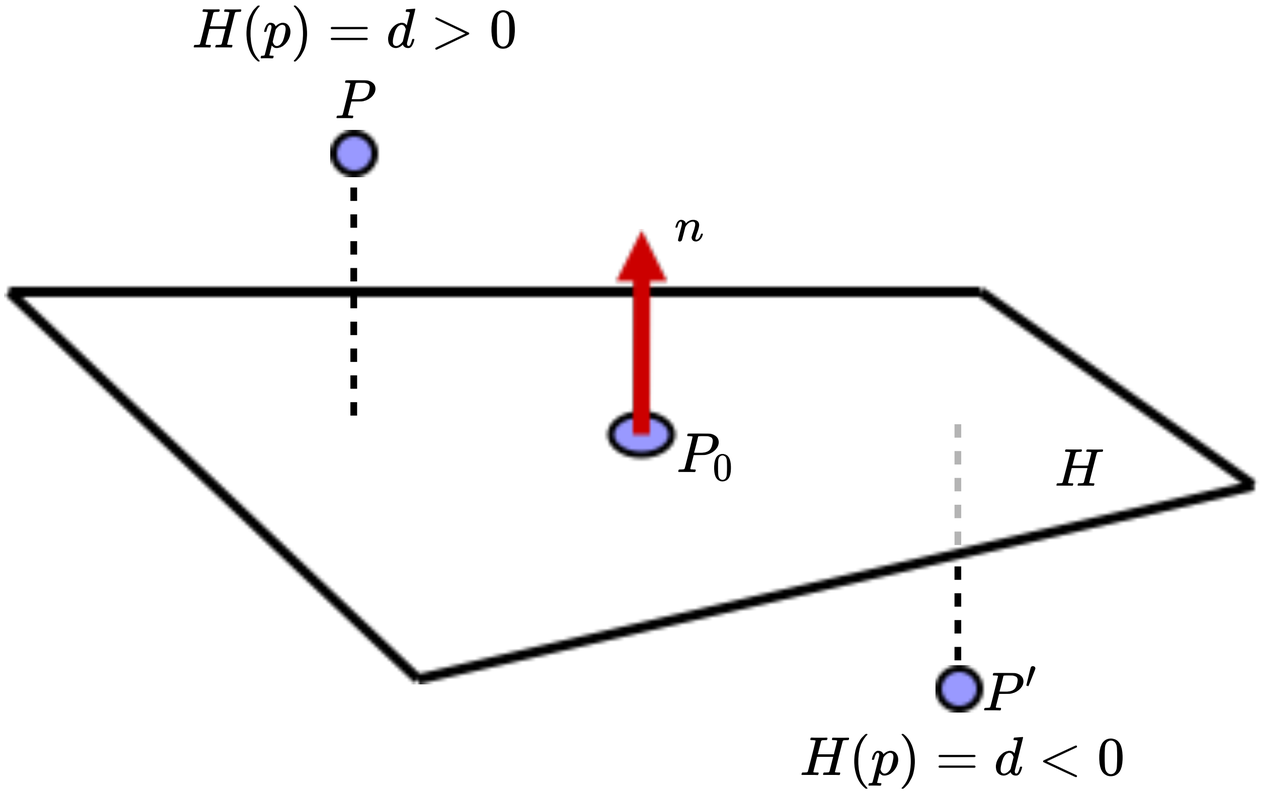
$$n = \ < A, B, C >$$

Implicit plane equation

$$H(P) = Ax_o + By_o + Cz_o + D = 0$$

$$n \cdot P_o + D = 0$$

*If **n** is normalized, signed distance to plane is **H(P)***



EXPLICIT VS IMPLICIT

Ray equation is explicit $P(t) = R_o + tR_d$

Parametric, generates points

Hard to verify point is on ray

Plane equation is implicit $H(P) = n \cdot P + D = 0$

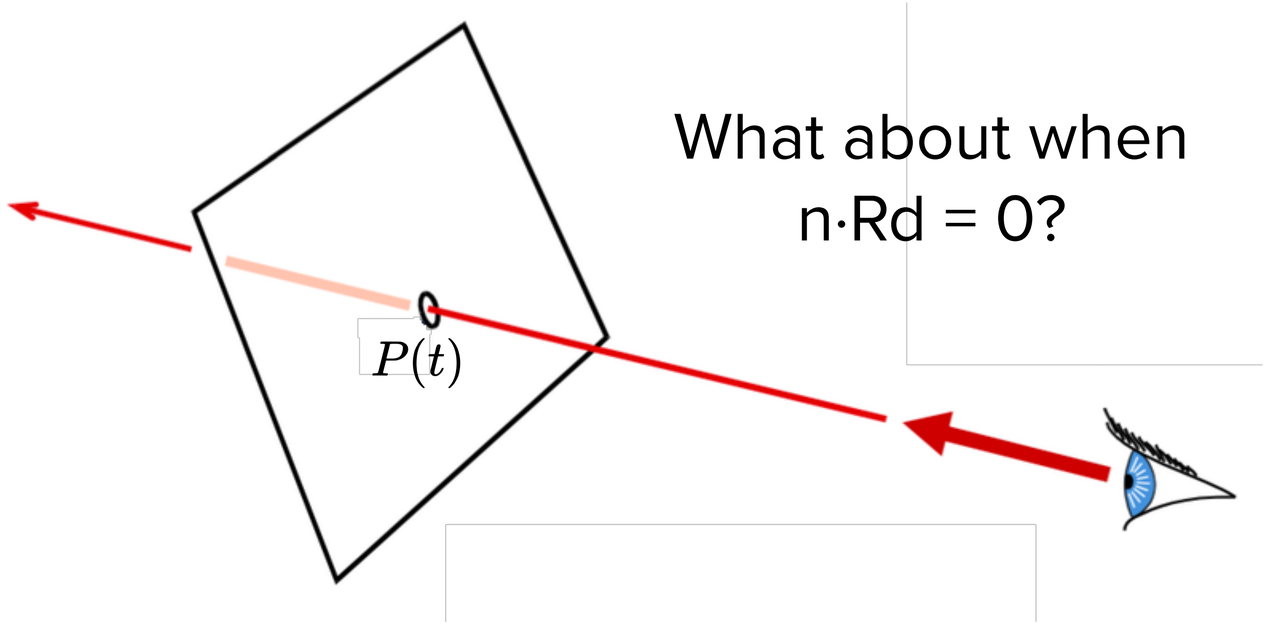
Solution of equation, does not generate points

Verifies point is on the plan

RAY-PLANE INTERSECTION

Intersection happens when both equations satisfied

Insert explicitly ray equation into implicit plane equation and solve for *t*



What about when n·Rd = 0?

$$P(t) = R_o + tR_d$$

$$H(P) = n \cdot P + D = 0$$

$$n \cdot (R_o + tR_d) + D = 0$$

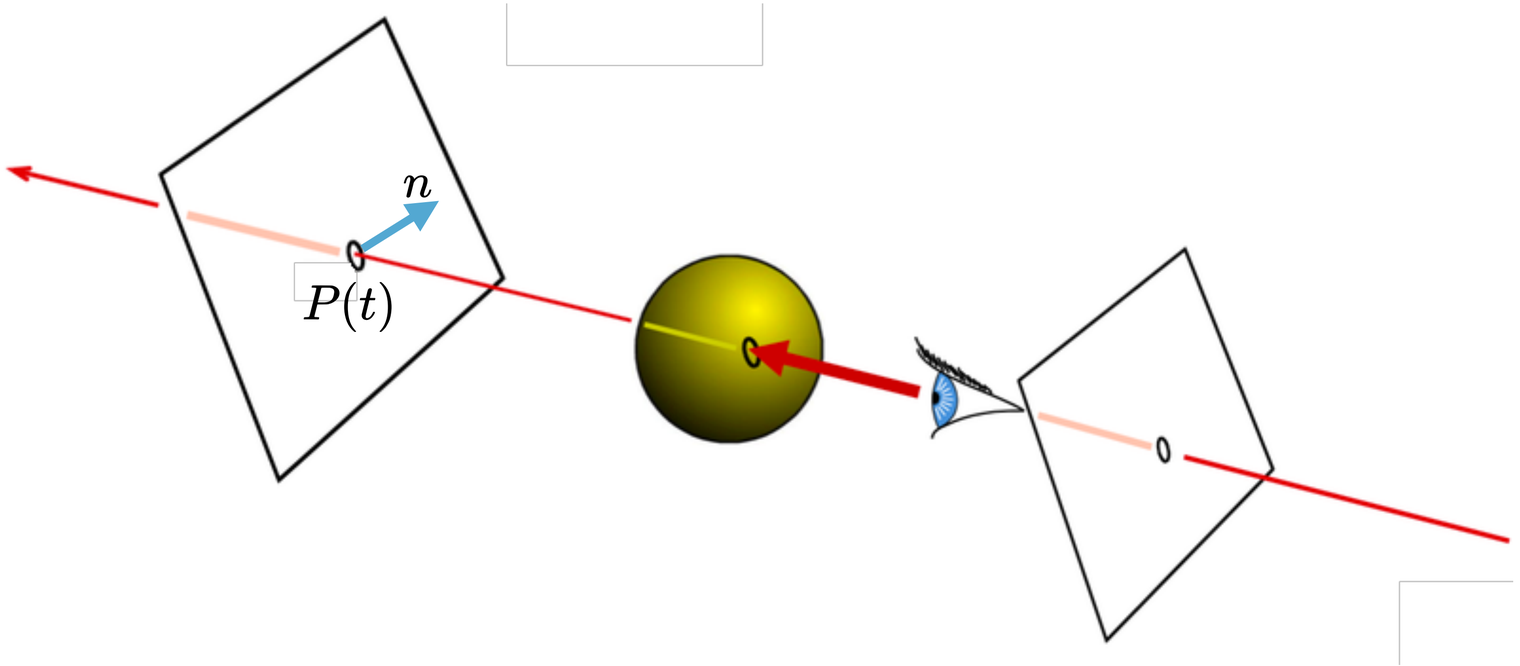
$$t = \frac{-(D + n \cdot R_o)}{n \cdot R_d}$$

FILLING IN THE REST

t must be closer than previous and in range

$$t < t_{current} \qquad t > t_{min}$$

Normal is constant throughout a plane



Course Overview

Mathematical toolbox

Ray caster

Camera & ray generation

Ray-plane intersection

Depth images

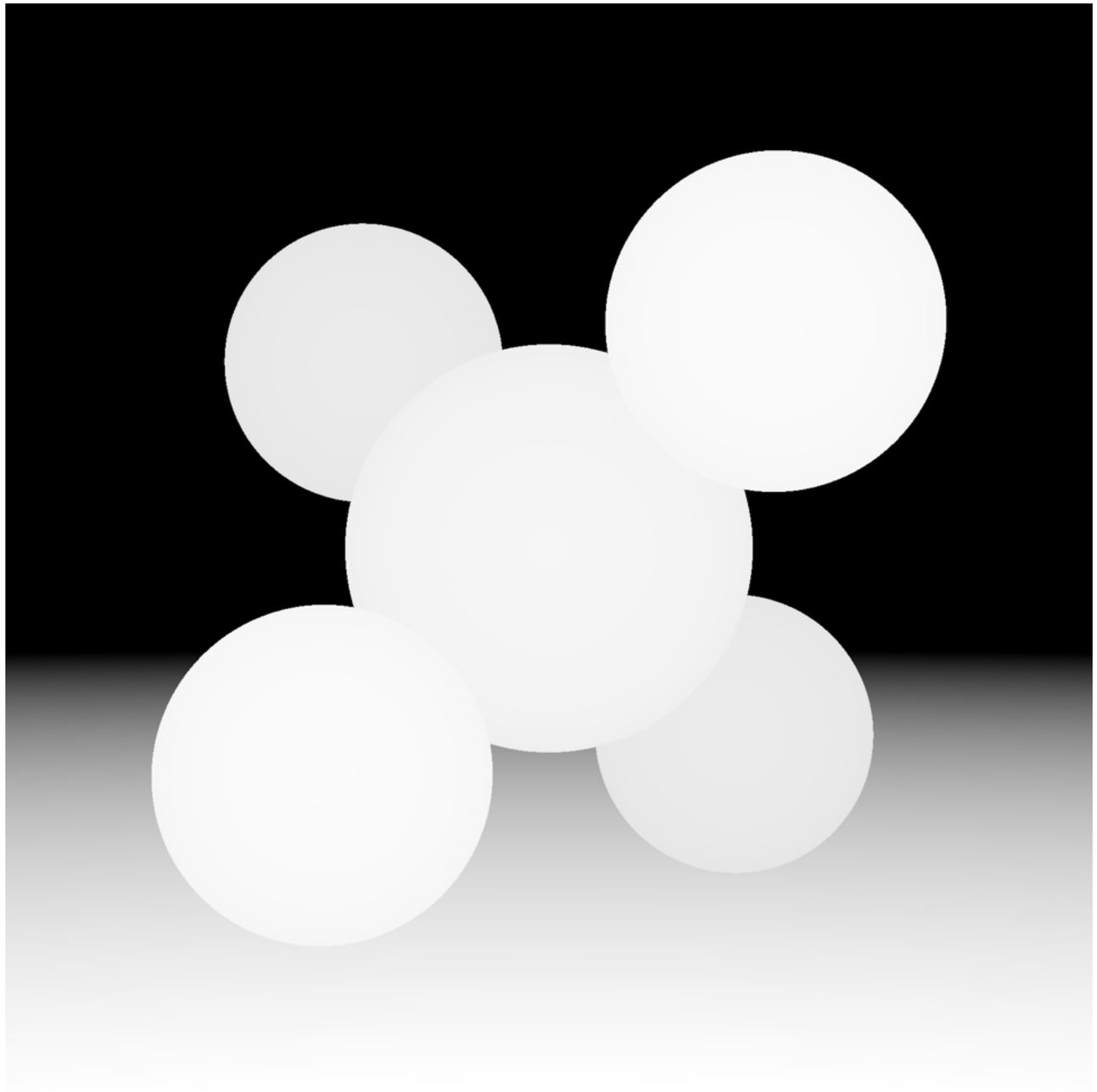
Starter code & next week

SAVE DEPTH IN GRAYSCALE

Lighter pixels represent close objects

Goes to black with infinity

Useful for debugging until we start shading



Course Overview

Mathematical toolbox

Ray caster

Camera & ray generation

Ray-plane intersection

Depth images

Starter code & next week

STARTER CODE

https://github.com/dionlarson/Ray-Tracer-Skeleton-Swift-OSX

DUE NEXT SESSION

camera, plane intersection, depth image