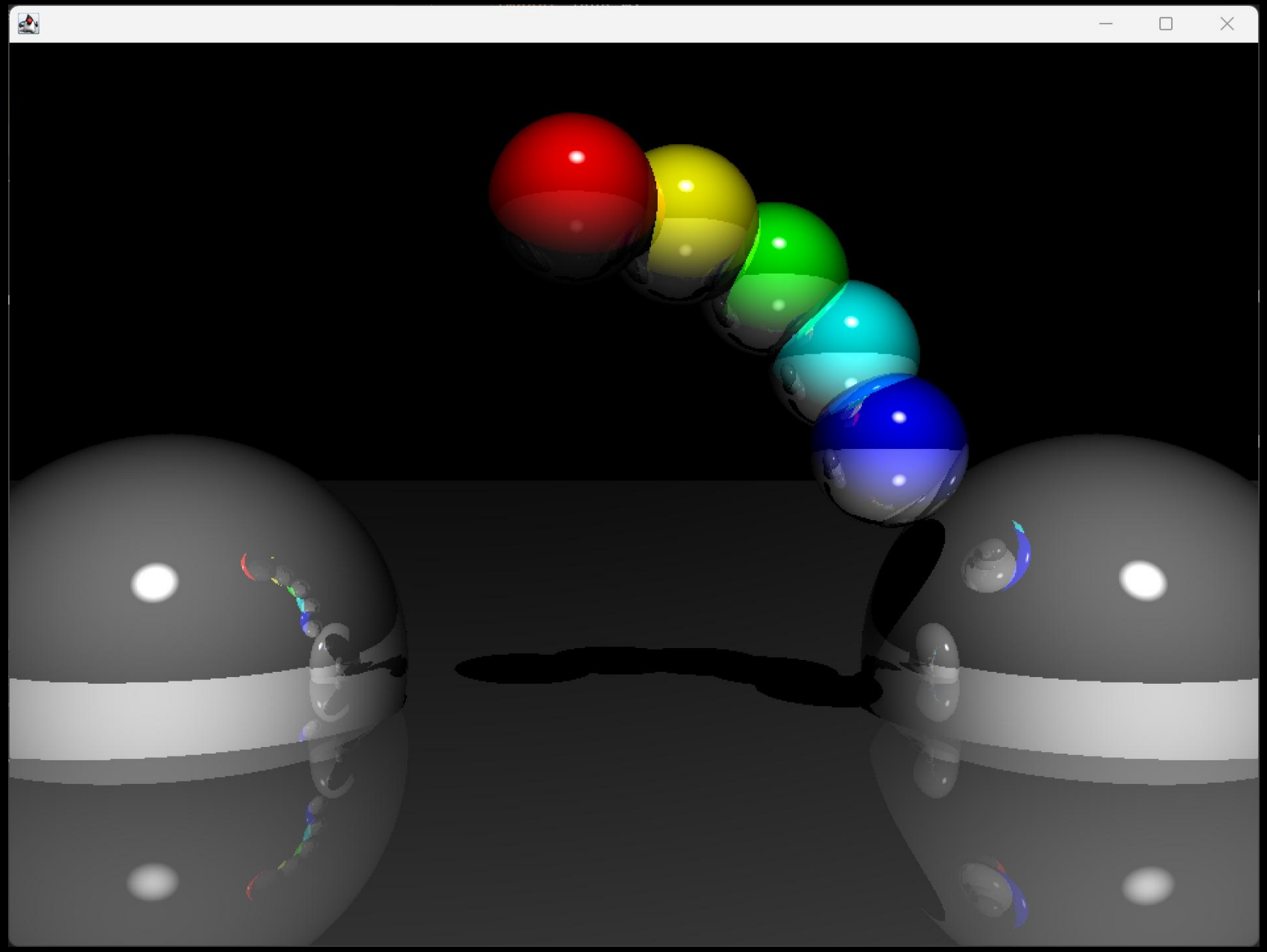
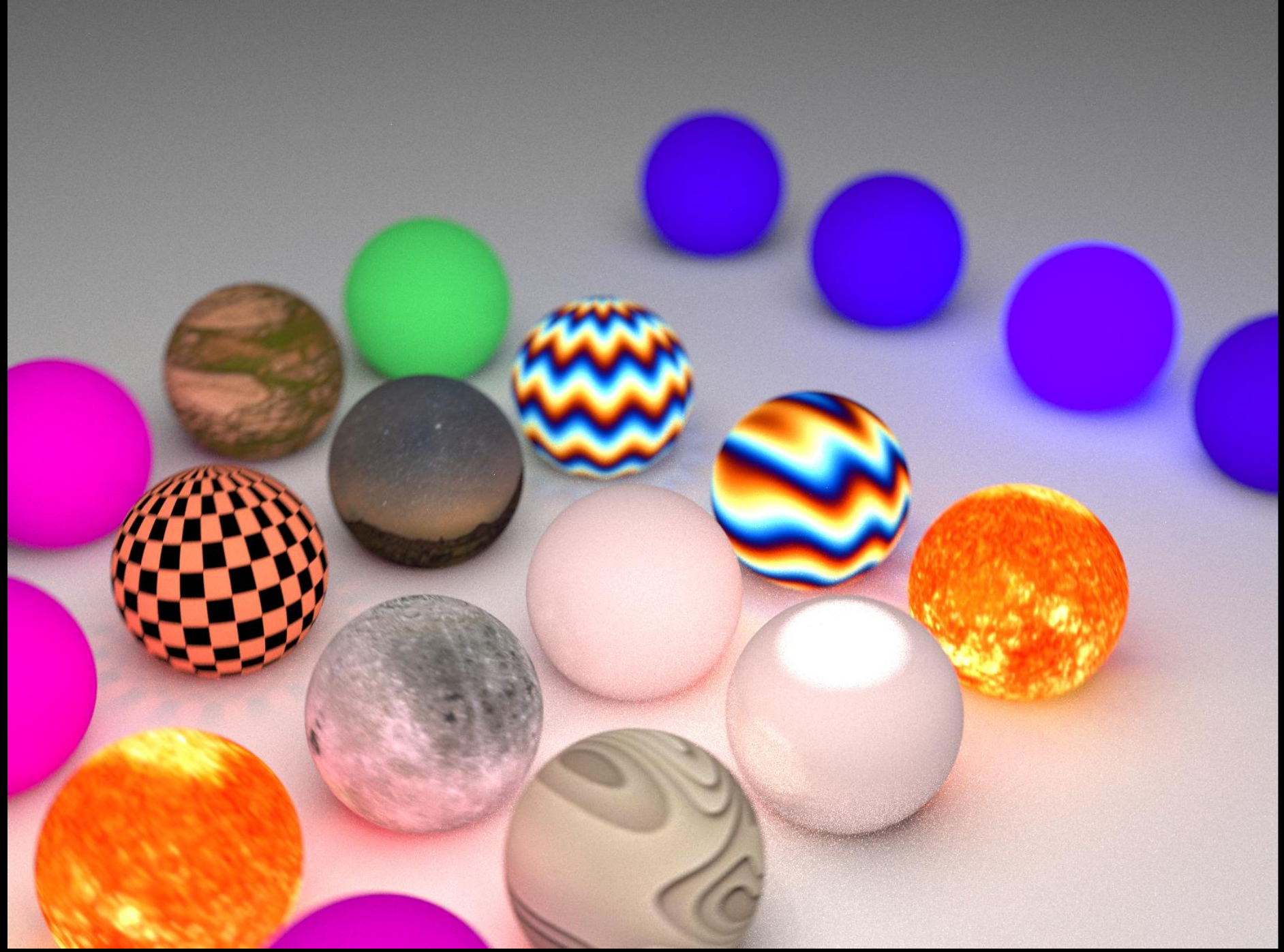
Path Tracing I

Computer Graphics
Fall Semester 2025

S. Felix

University of Applied Sciences and Arts Northwestern Switzerland
School of Computer Science





“Depth of Field” by M. Albrecht

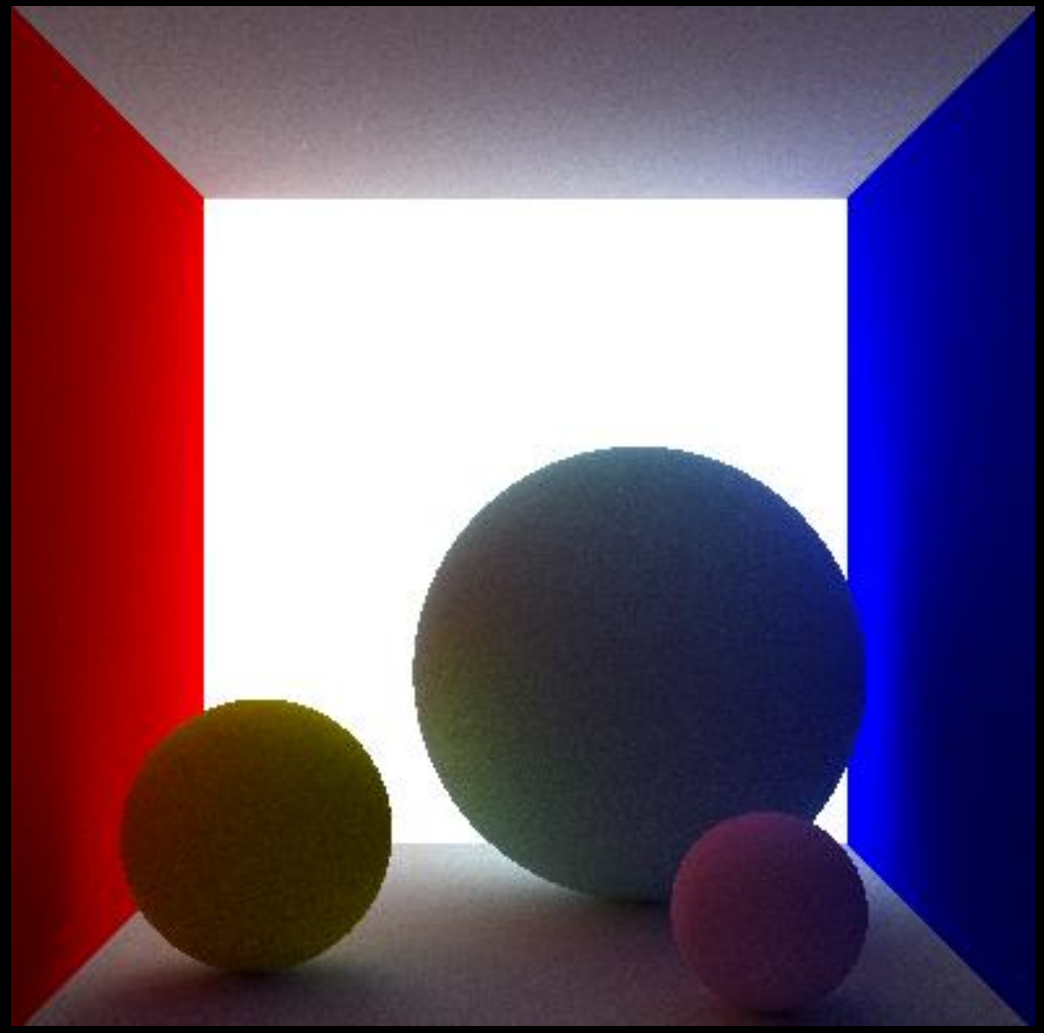
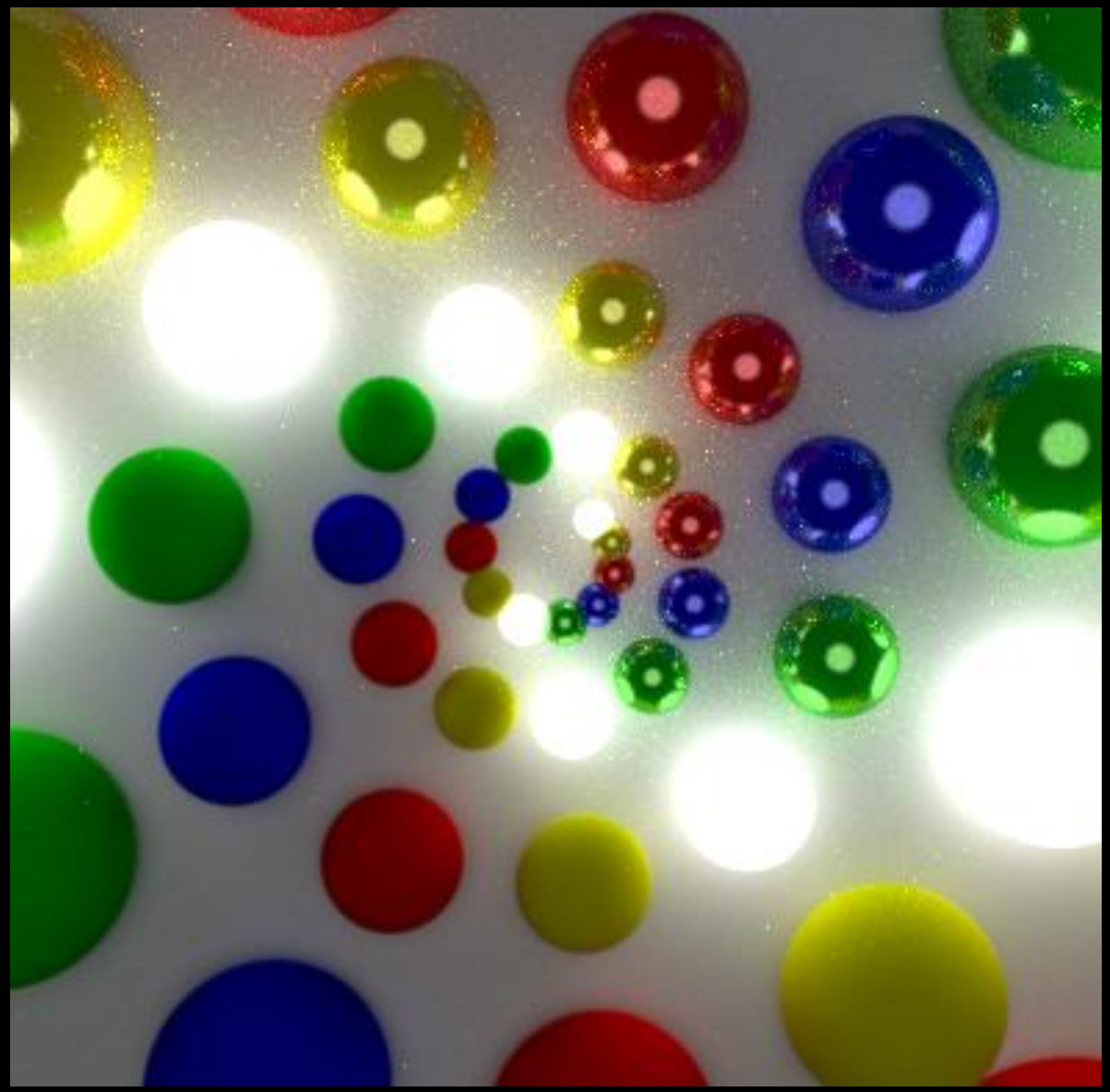
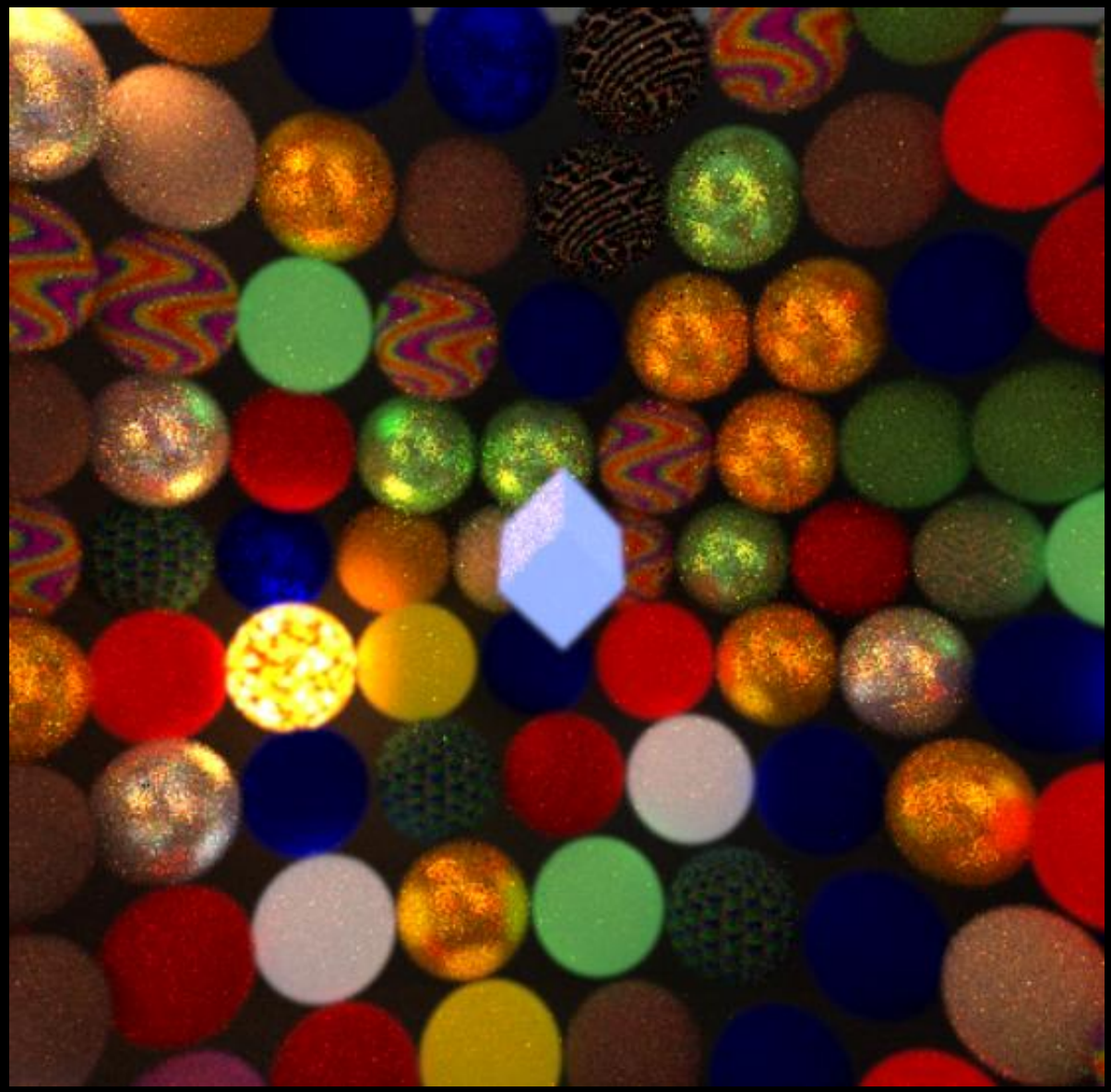


Image by S. Burkhardt

“Adventures in Copyright Infringement” by G. Ferrali





Spirals by S. Burkhardt, Cube by J. Wyss

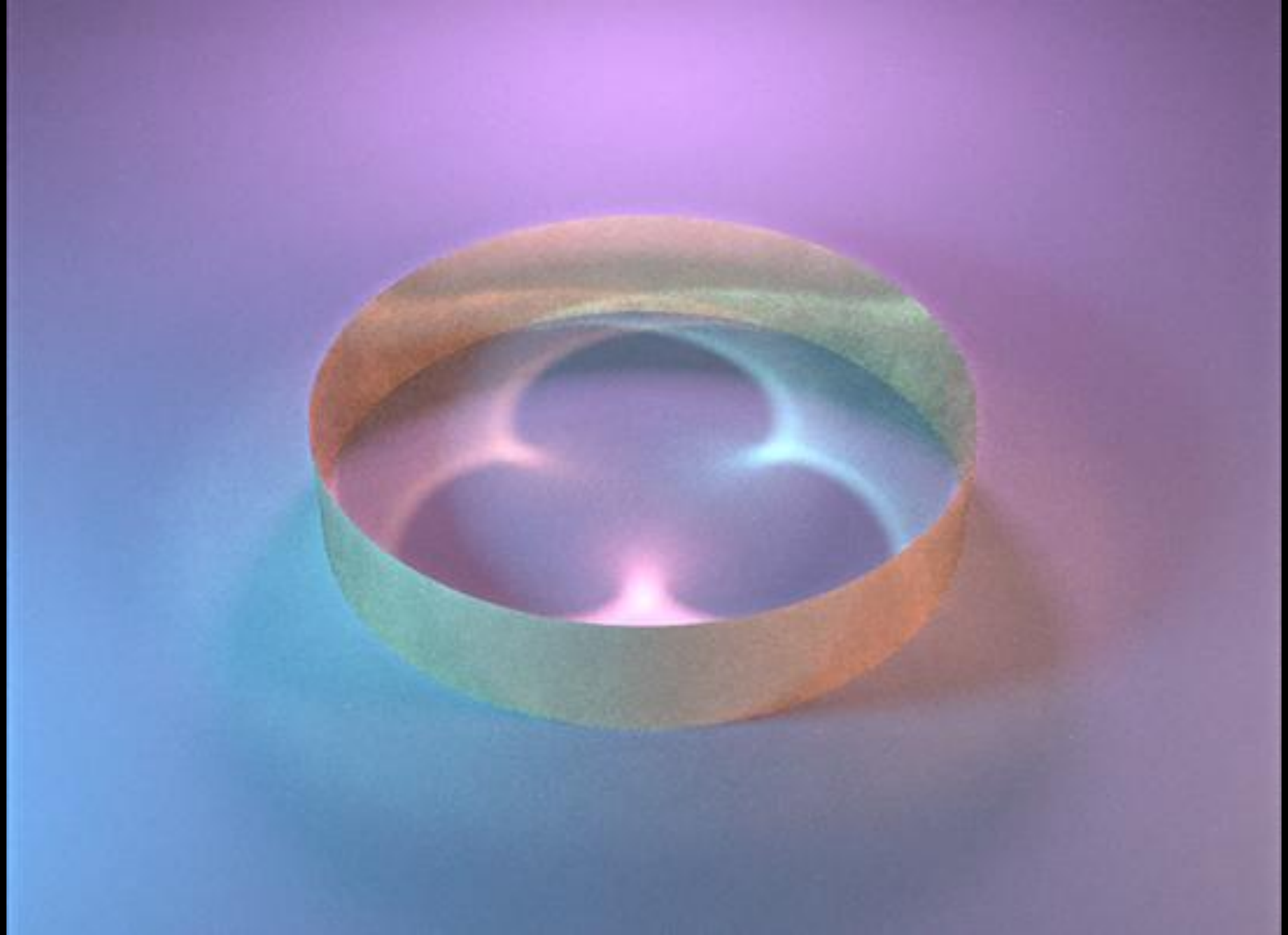


Image by N. Matter

Light, simplified

Light sources emit photons

Photons move in **straight lines**, until they hit something

Each photon is monochromatic

(of a single **color**/wavelength)

We see photons when they reach our eyes

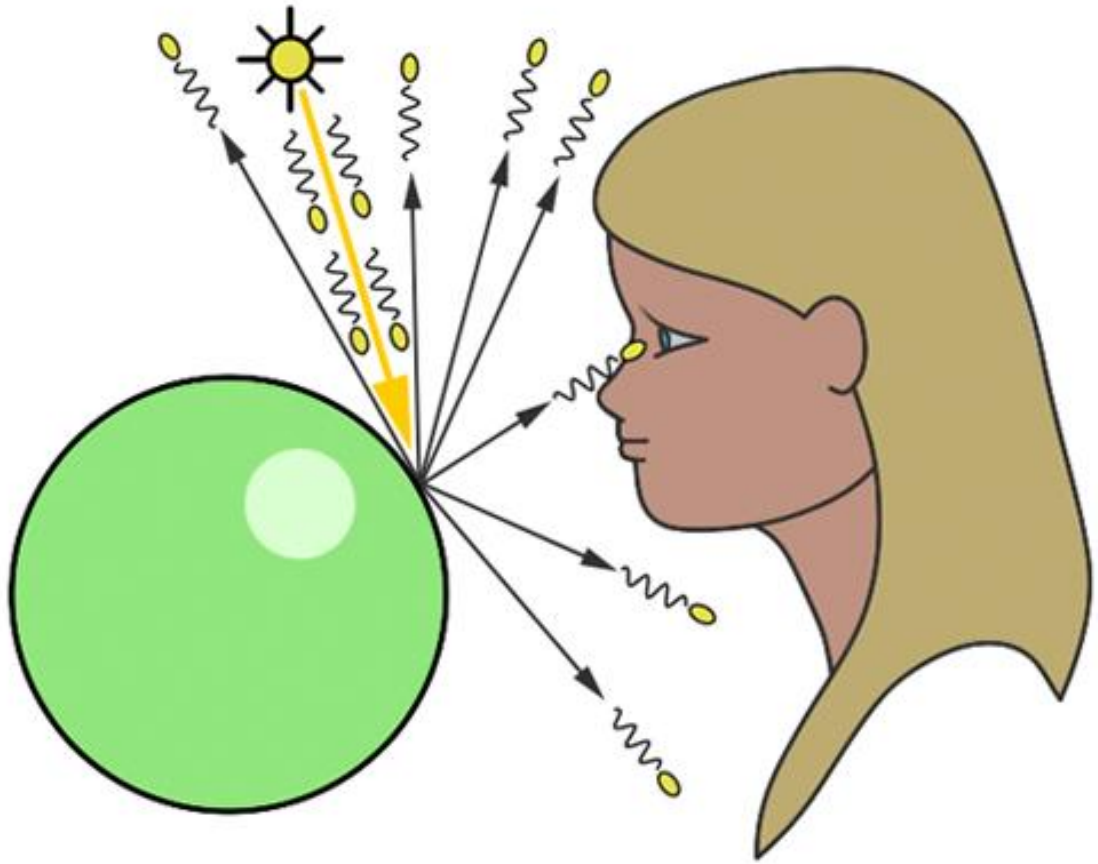
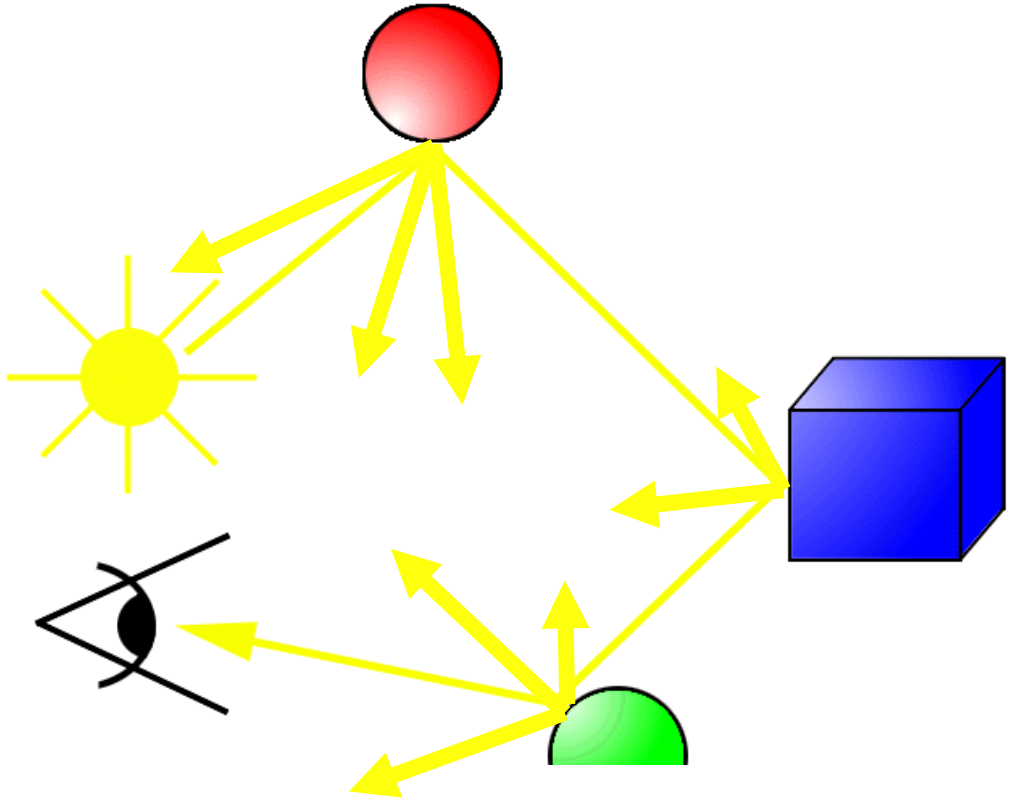


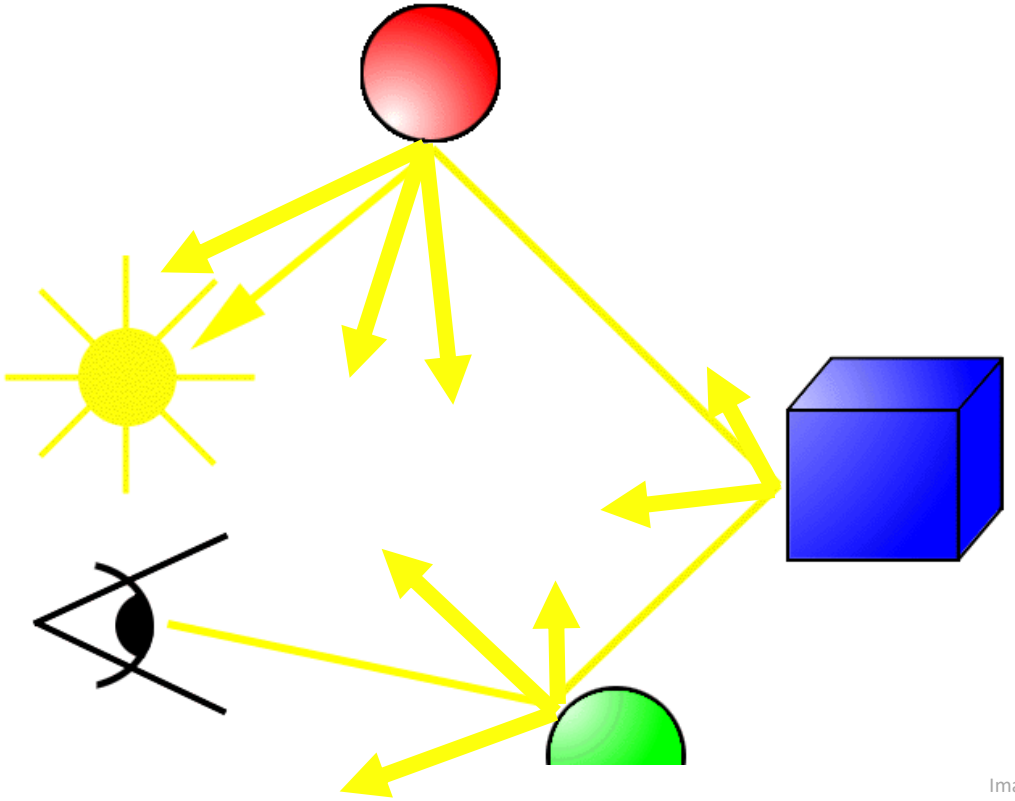
Image: Scratchapixel

Ray Tracing Approaches

Forward Ray Tracing



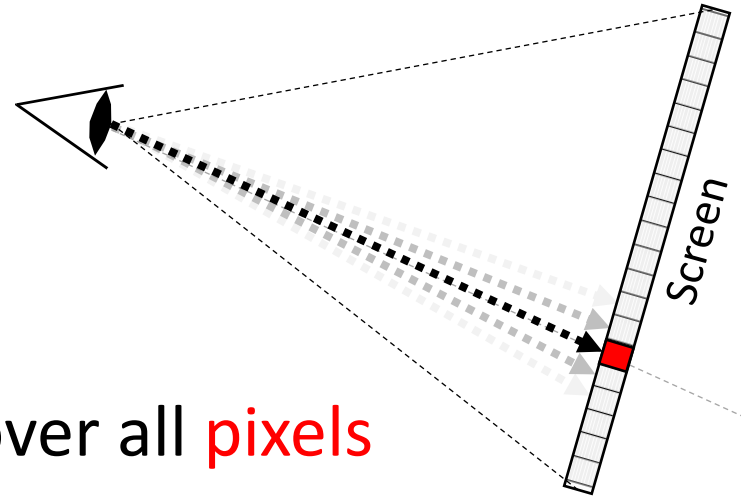
Backward Ray Tracing



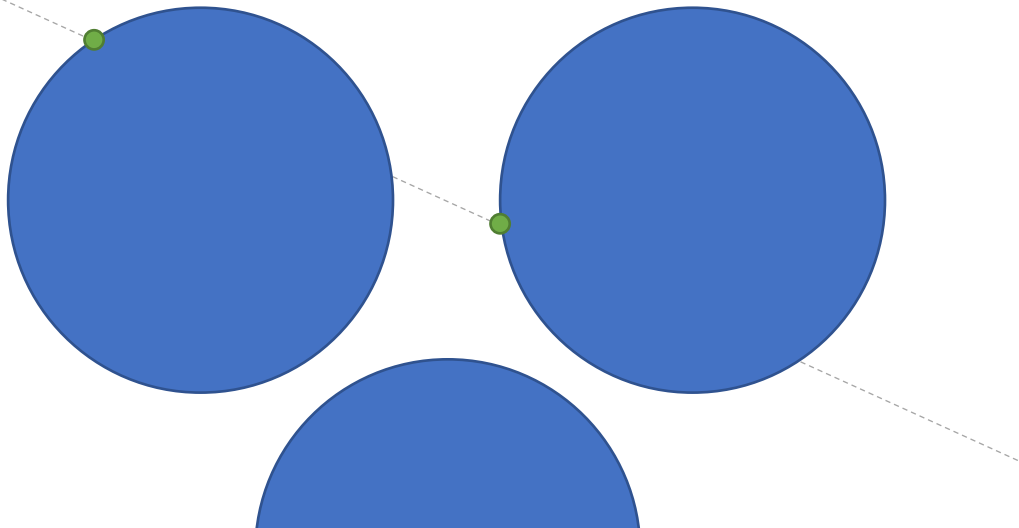
Images: Stanford

Whitted, Turner. "An Improved Illumination Model for Shaded Display." *Communications* (1980).

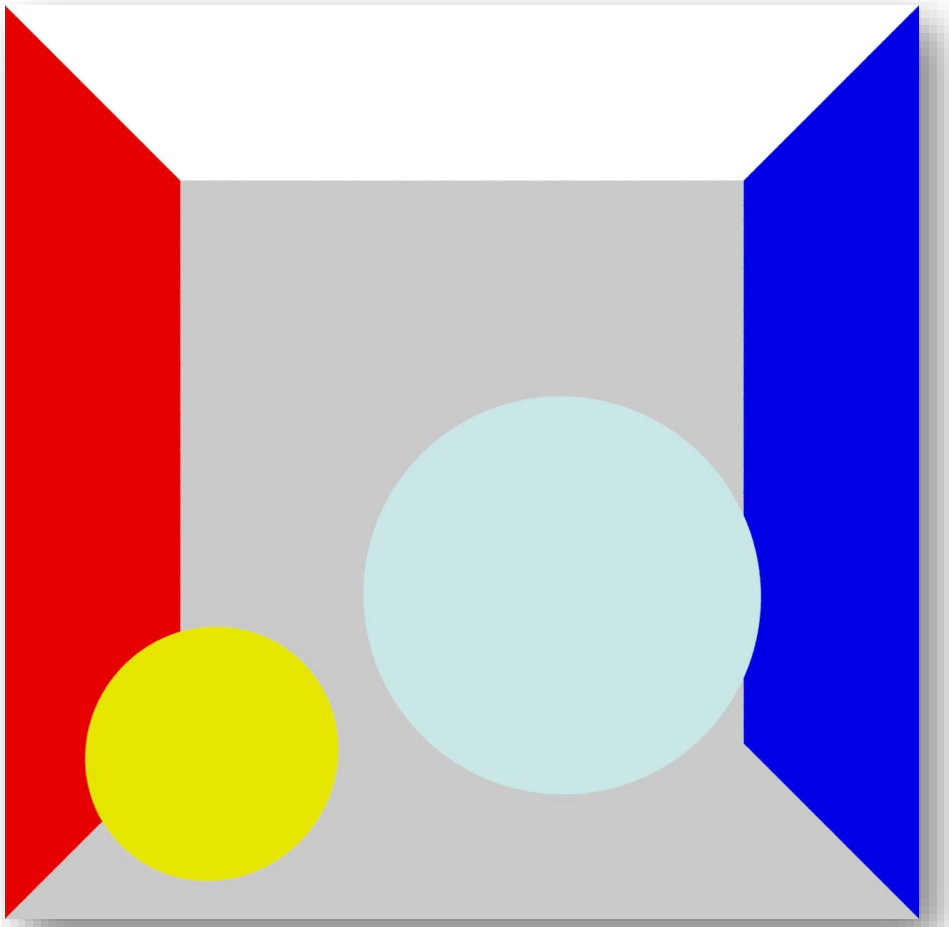
Whitted Ray Tracing



1. Iterate over all pixels
2. Generate an eye ray
   ⋯▸ for each pixel
3. Find **first** Hit-Point
4. Determine color
5. Set pixel to color



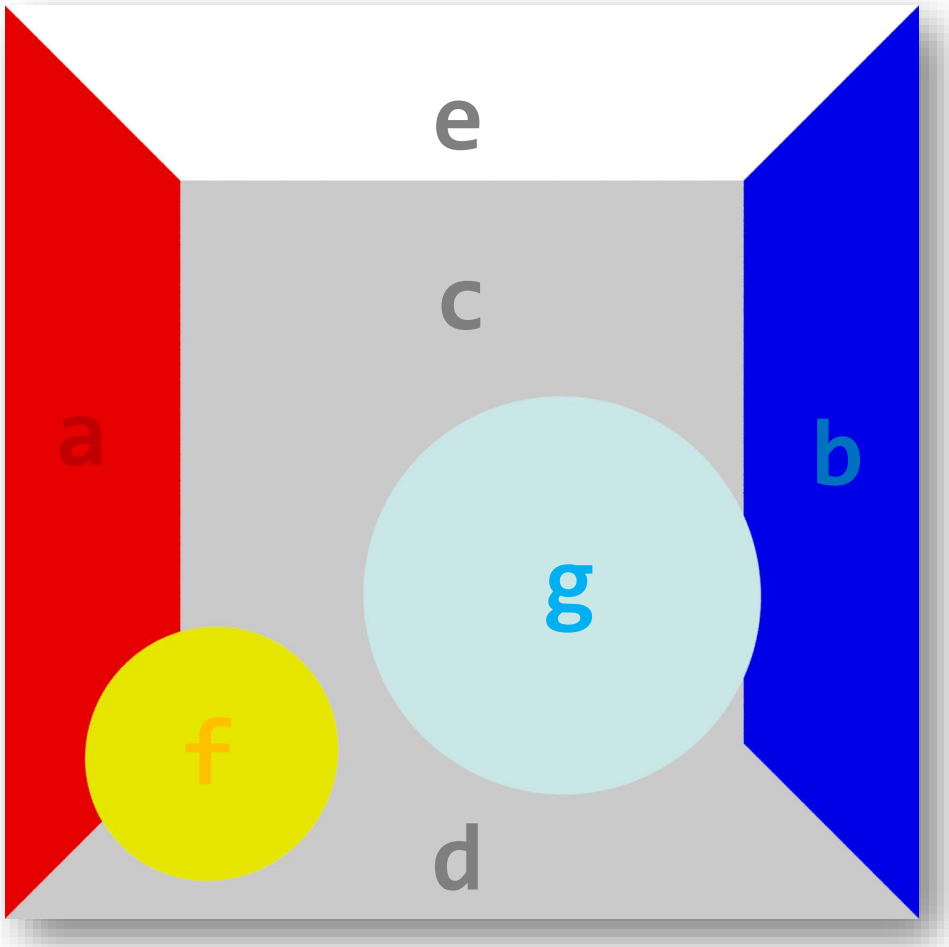
Cornell Box



Our first approximation

Cornell Box, built with paper (1980)

Cornell Box




```
Eye    = [0, 0, -4]
LookAt = [0, 0,  6]
FOV = 36°

Scene
{
  Sphere(Center=[-1001,     0,   0], r=1000, Color=       Red) a
  Sphere(Center=[ 1001,     0,   0], r=1000, Color=      Blue) b
  Sphere(Center=[    0,     0,1001], r=1000, Color=      Gray) c
  Sphere(Center=[    0, -1001,   0], r=1000, Color=      Gray) d
  Sphere(Center=[    0,  1001,   0], r=1000, Color=     White) e
  Sphere(Center=[ -0.6,  -0.7,-0.6], r= 0.3, Color=    Yellow) f
  Sphere(Center=[  0.3,  -0.4, 0.3], r= 0.6, Color=LightCyan) g
}
```

Avoid 100% intensity. Most materials reflect less than ~80%.

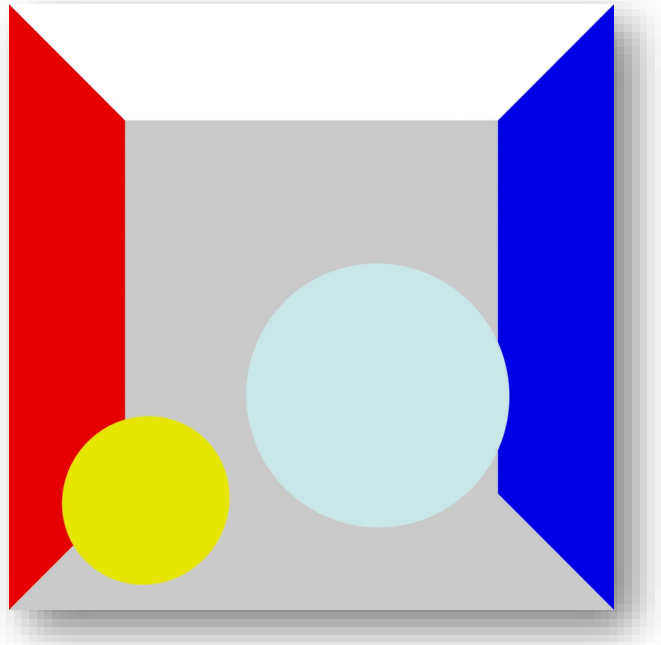
```
(Vec3 o, Vec3 d) CreateEyeRay(Vec3 Eye, Vec3 LookAt,
                              float FOV, Vec2 Pixel) { … }

HitPoint FindClosestHitPoint(Scene s, Vec3 o, Vec3 d) { … }

Color ComputeColor(Scene s, Vec3 o, Vec3 d) { … }
```

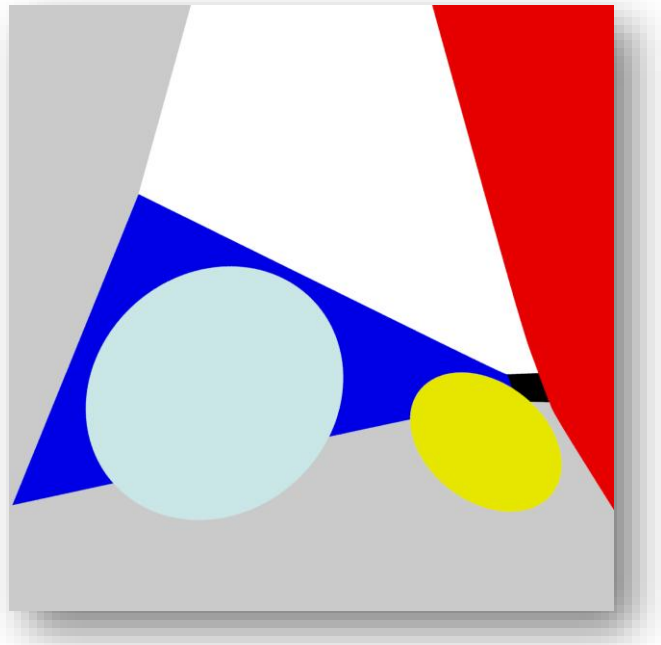
This Week's Lab

Exchange the gradient computation with the basis for a path tracer. It should support:

- Spheres
- Colors
- **Eye = [-0.9, -0.5, 0.9]**
  LookAt = [0, 0, 0]
  FOV = 110°

1. Keep the code from last week running
2. Take screenshots of the results





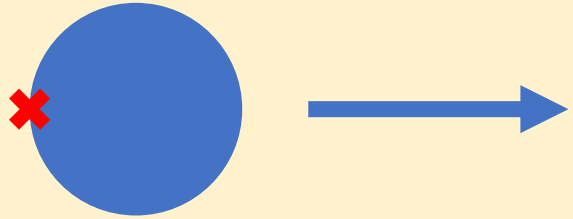
Common Problems

- Incorrect formulas
- Using normalized vectors instead of non-normalized ones and vice-versa
- Mixing units, radians and degrees, coordinate systems
- Truncating integer divisions

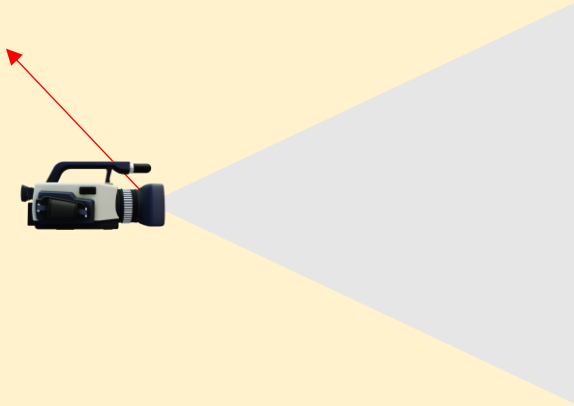
How to Debug

Write Unit Tests for…

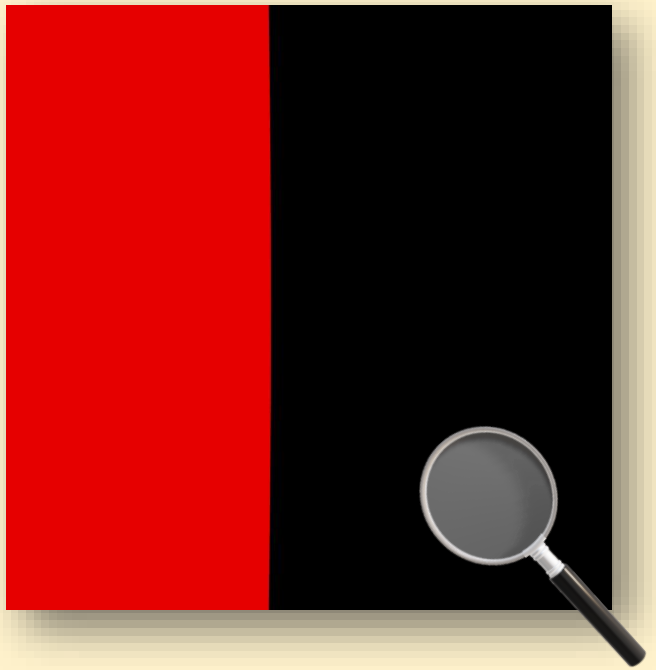
- …ray-sphere intersection from the front
- …ray-sphere intersection from the side
- …ray-sphere intersection from the back
- …ray pointing away from sphere



- …eye ray generation for the center pixel looking forward
- …eye ray generation for the center pixel looking right
- …eye ray generation for the left-most pixel looking forward
- …eye ray generation for the top-most pixel looking forward



How to Debug



1. Remove most spheres from the test scene
2. Identify a problematic pixel
3. Place a breakpoint for that pixel
4. Single-step through the code for that pixel
5. Are the numbers plausible?
   - Positive/negative
   - Small/big
   - Correct direction

```
141     for (int x = 0; x < bitmap.Width; x++)
142     {
143         for (int y = 0; y < bitmap.Height; y++)
144         {
145             var pixel = new Pixel(x, y);
146
147             if (x == 316 && y == 17)
148                 Console.WriteLine(value: "interesting");
149
150             (Vector3 Origin, Vector3 Direction) = CreateEyeR
                  halfWidth) / halfWidth, (pixel.y - halfHeight)
151
```