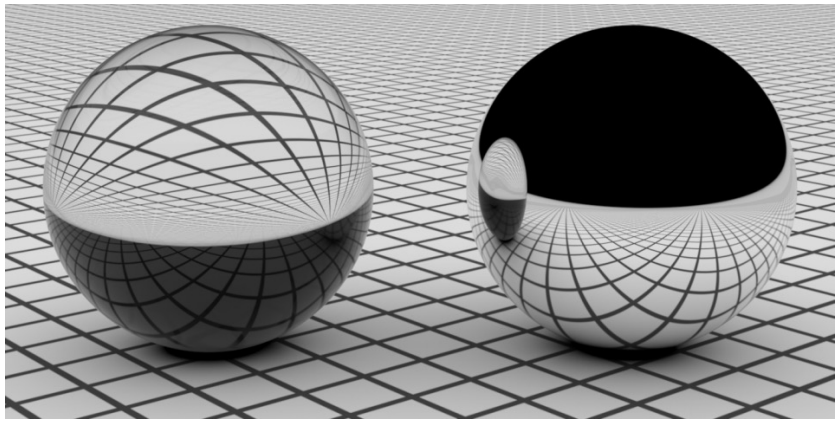
Ray Tracing I: Basics

Today



- `pbrt` **overview**
- **Basic algorithms**
- **Ray-surface intersection**
- **Efficiency**
- ***Floating-point error***

Next lecture

- **Accelerating ray tracing of large numbers of geometric primitives**

Light Rays

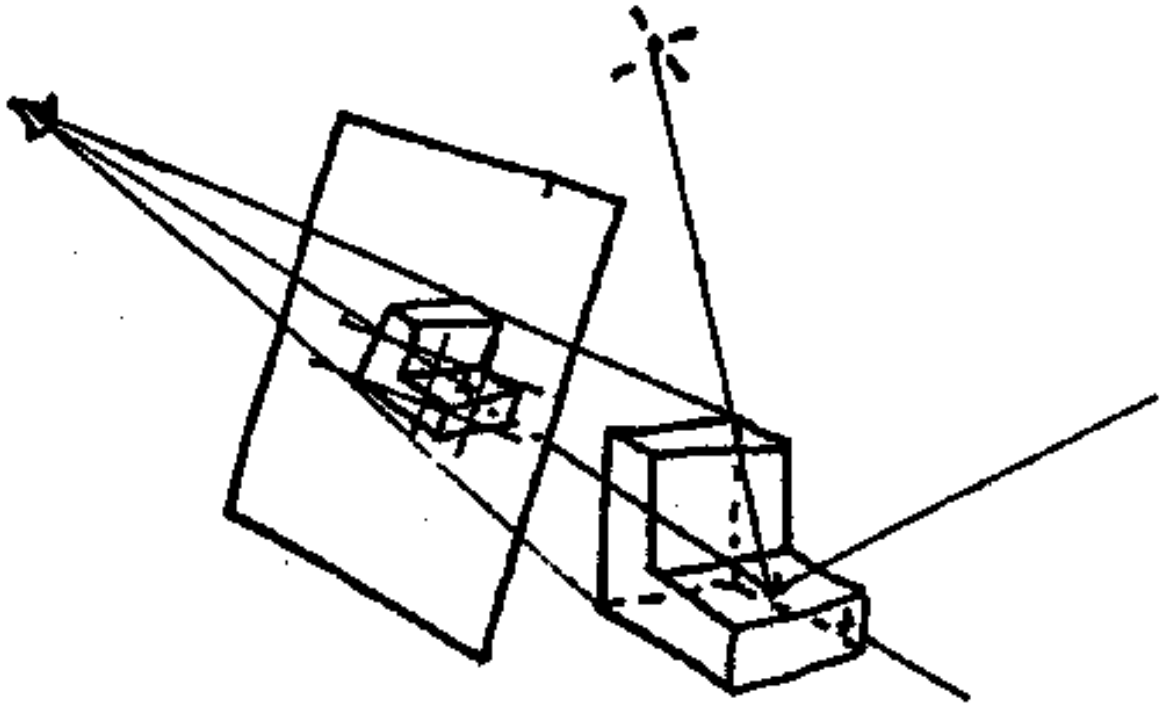
Three ideas about light rays

1. **Light travels in straight lines (mostly)**
2. **Light rays do not interfere with each other if they cross (light is invisible!)**
3. **Light rays travel from the light sources to the eye (but the physics is invariant under path reversal - reciprocity).**

Ray Tracing in Computer Graphics

Appel 1968 - Ray casting

1. Generate an image by casting one ray per pixel
2. Check for shadows by sending a ray to the light



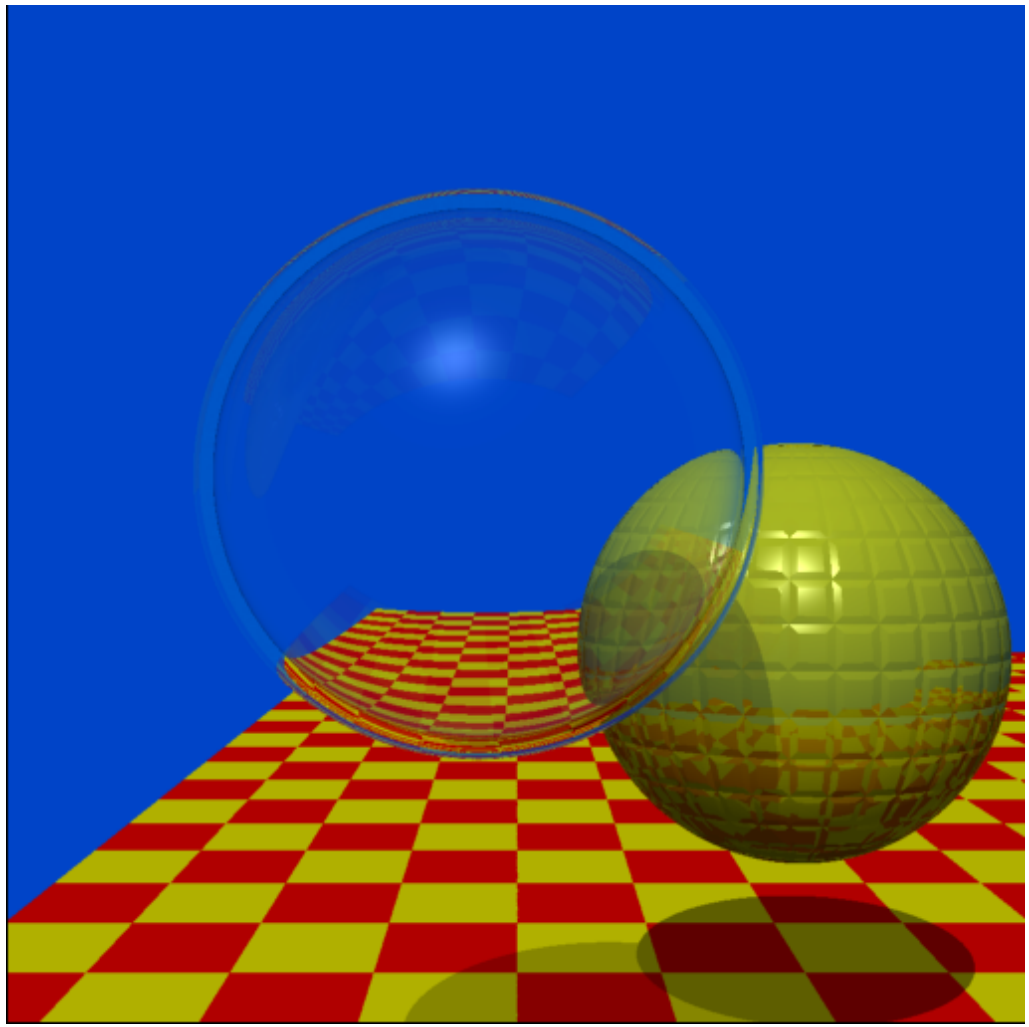
Ray Tracing in Computer Graphics

"An improved
Illumination model
for shaded display"
T. Whitted,
CACM 1980

1. Always send ray
to the light source (unless
glass or mirror)

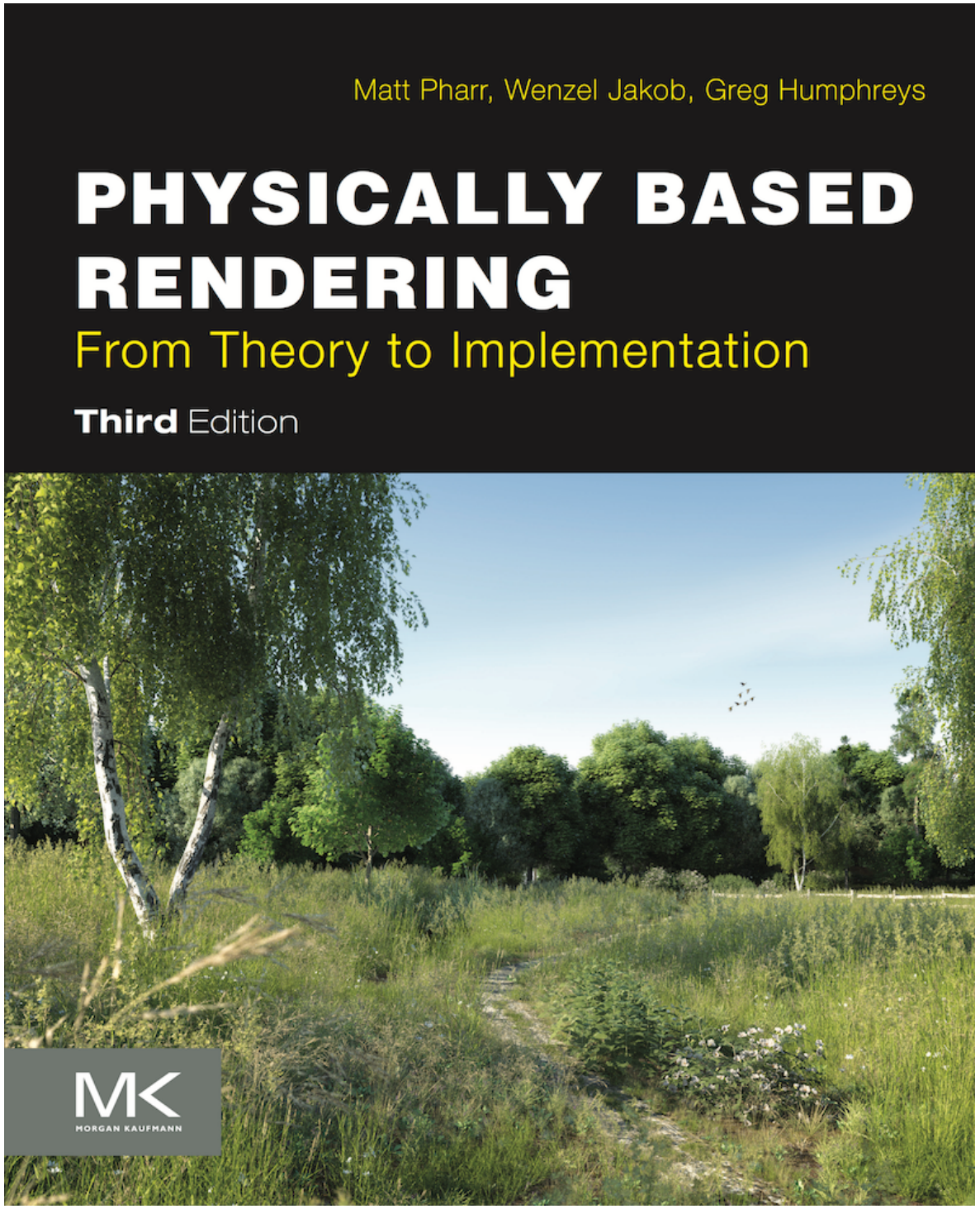
2. Recursively generate
reflected rays (mirror) and
transmitted rays (glass)

Time:
- VAX 11/780 (1979) 74m
- PC (2006) 6s
- GPU (2012) 1/30s



Spheres and Checkerboard
T. Whitted, 1979

PBRT Overview



http://www.pbr-book.org/

Table 1.1: Main Interface Types. Most of `pbrt` is implemented in terms of 10 key abstract base classes, listed here. Implementations of each of these can easily be added to the system to extend its functionality.

Base class	Directory	Section
`Shape`	`shapes/`	3.1
`Aggregate`	`accelerators/`	4.2
`Camera`	`cameras/`	6.1
`Sampler`	`samplers/`	7.2
`Filter`	`filters/`	7.8
`Material`	`materials/`	9.2
`Texture`	`textures/`	10.3
`Medium`	`media/`	11.3
`Light`	`lights/`	12.2
`Integrator`	`integrators/`	1.3.3

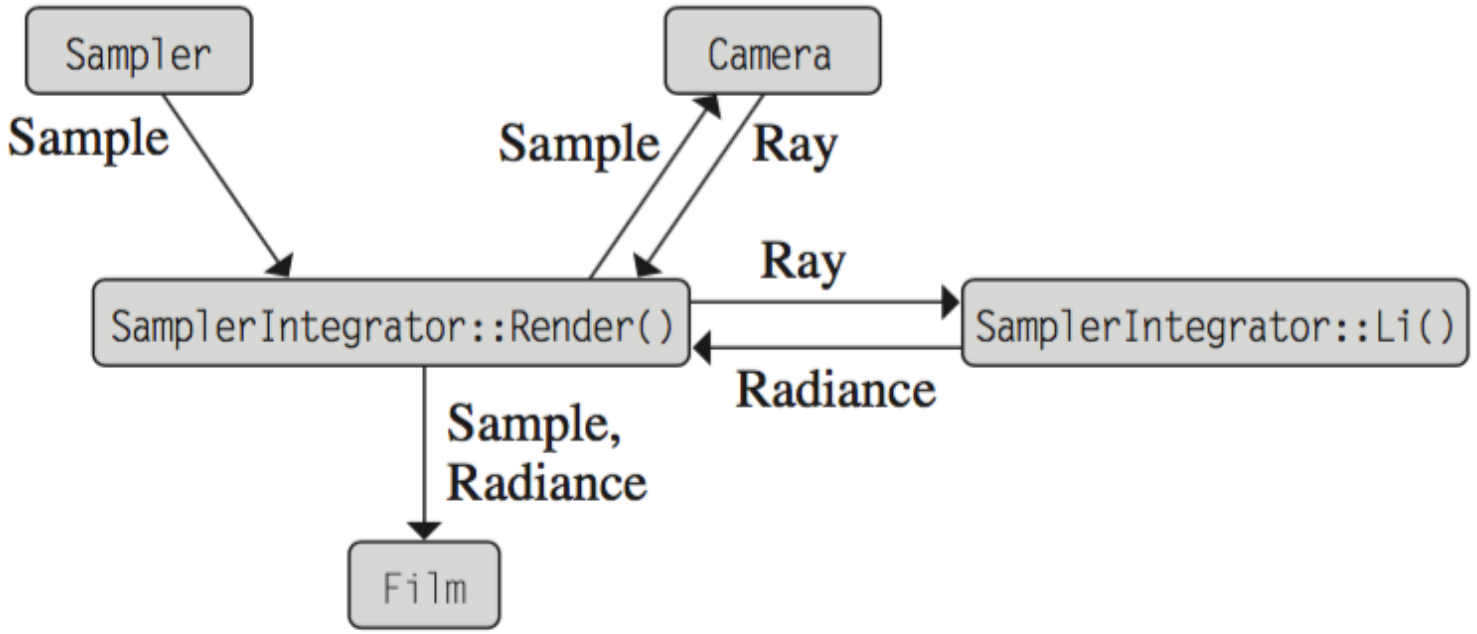


Figure 1.17: Class Relationships for the Main Rendering Loop in the `SamplerIntegrator::Render()` Method in `core/integrator.cpp`. The `Sampler` provides a sequence of sample values, one for each image sample to be taken. The `Camera` turns a sample into a corresponding ray from the film plane, and the `Li()` method implementation computes the radiance along that ray arriving at the film. The sample and its radiance are given to the `Film`, which stores their contribution in an image. This process repeats until the `Sampler` has provided as many samples as are necessary to generate the final image.

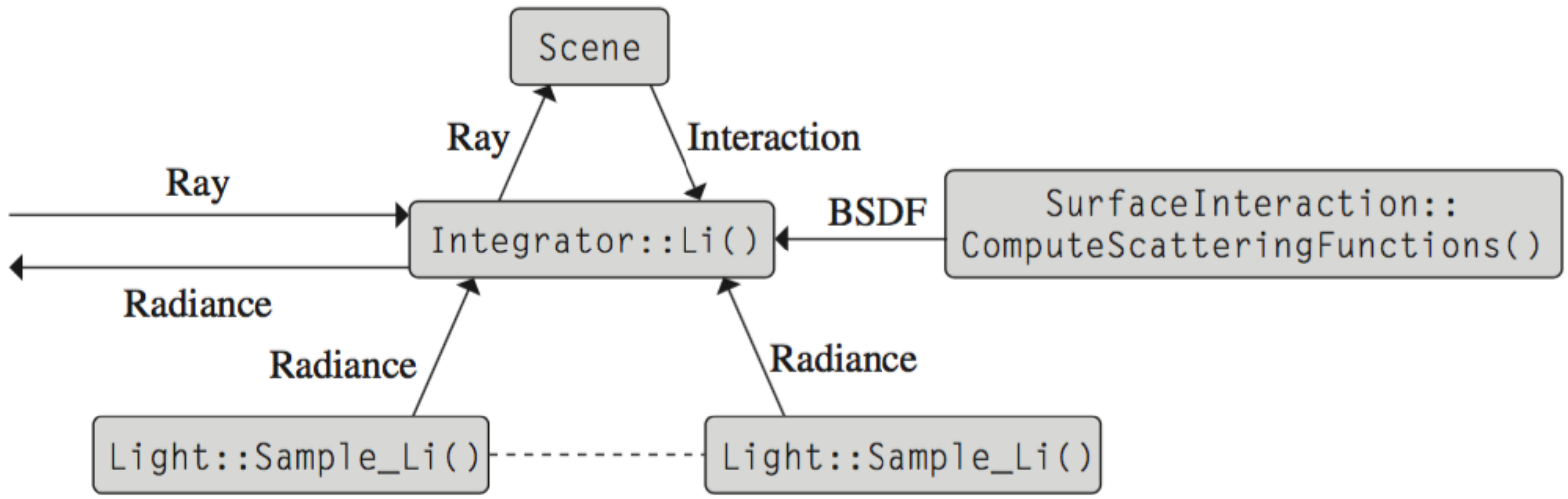
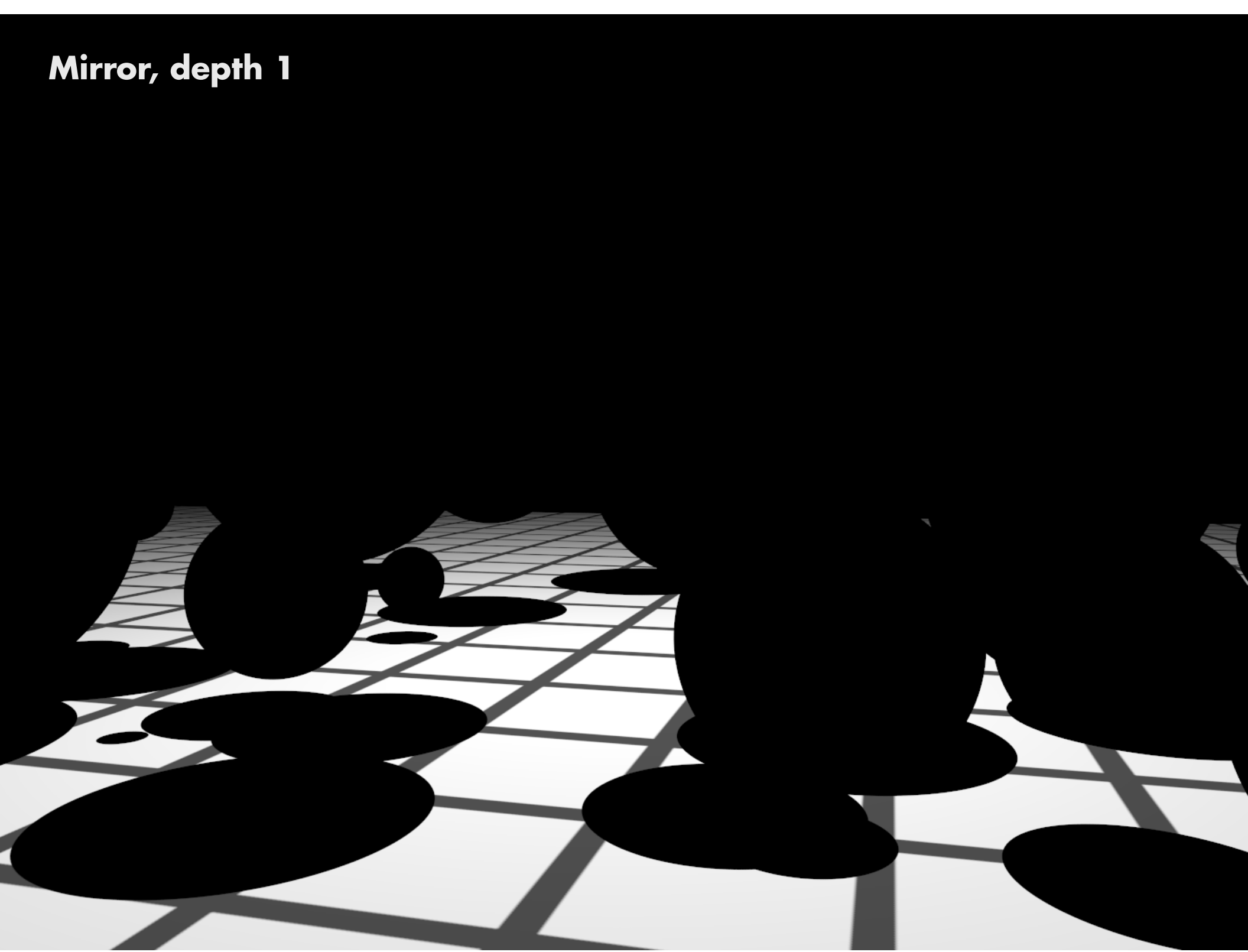
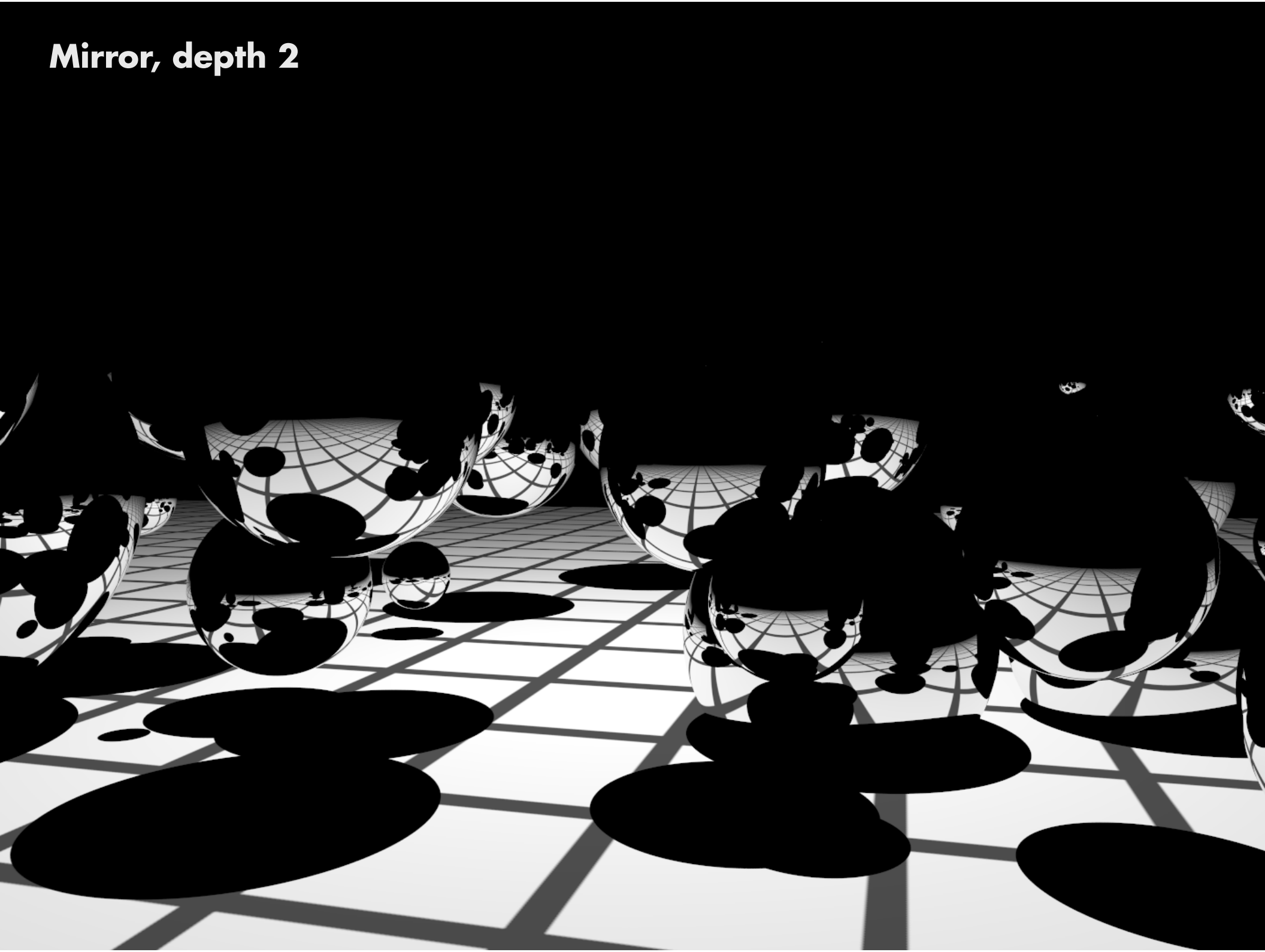


Figure 1.19: Class Relationships for Surface Integration. The main rendering loop in the `SamplerIntegrator` computes a camera ray and passes it to the `Li()` method, which returns the radiance along that ray arriving at the ray's origin. After finding the closest intersection, it computes the material properties at the intersection point, representing them in the form of a BSDF. It then uses the Lights in the Scene to determine the illumination there. Together, these give the information needed to compute the radiance reflected back along the ray at the intersection point.

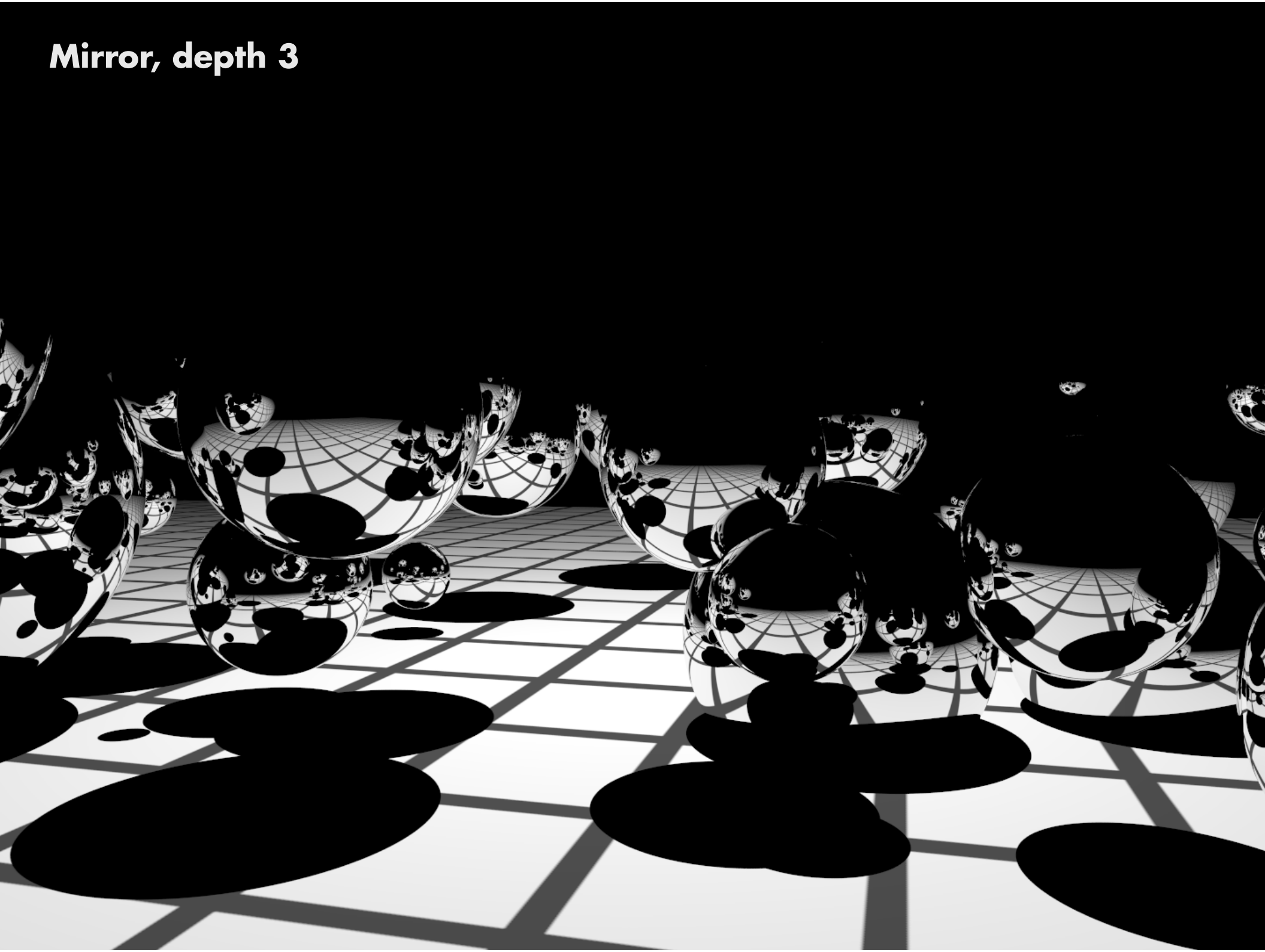
Mirror, depth 1



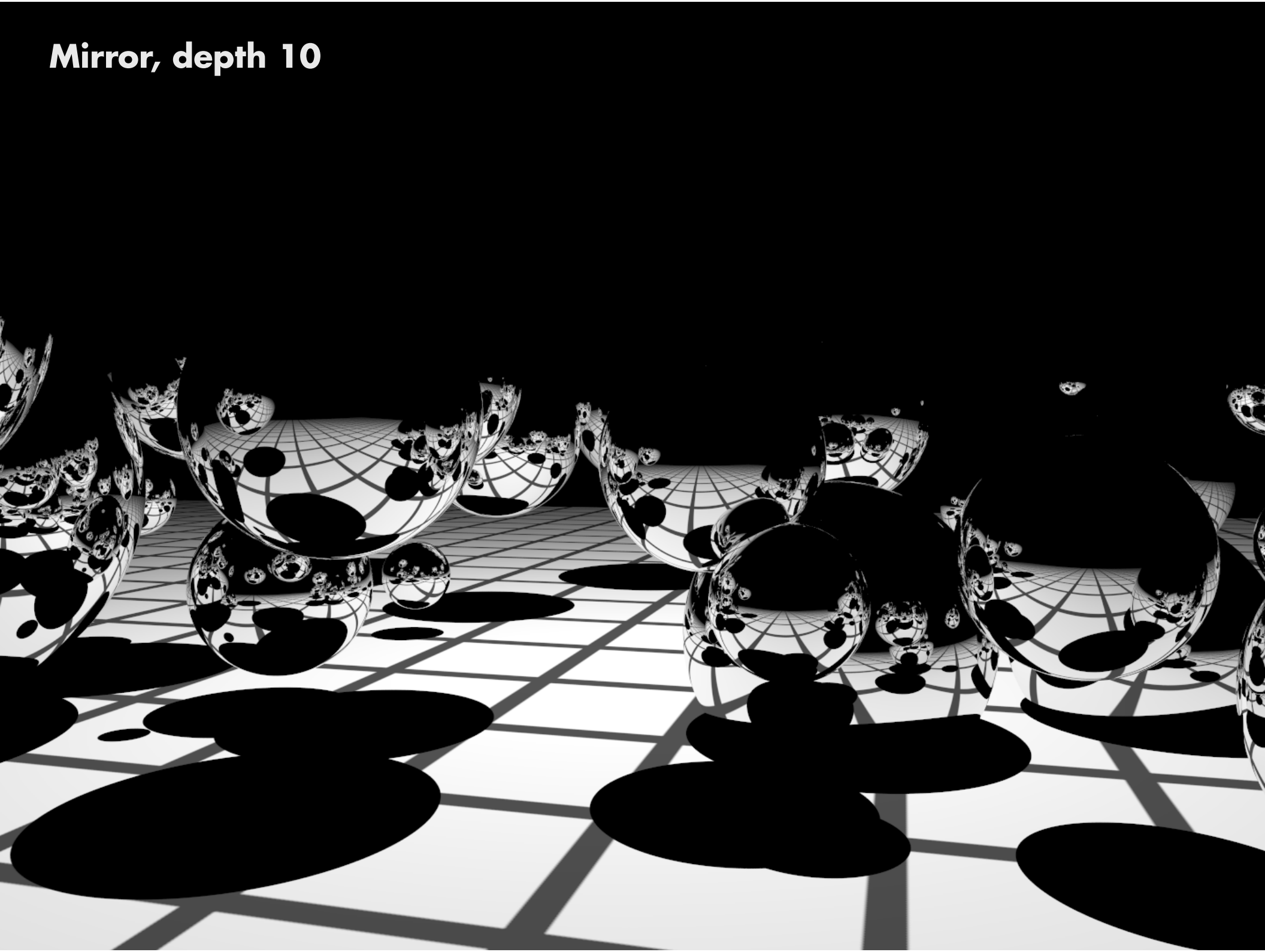
Mirror, depth 2



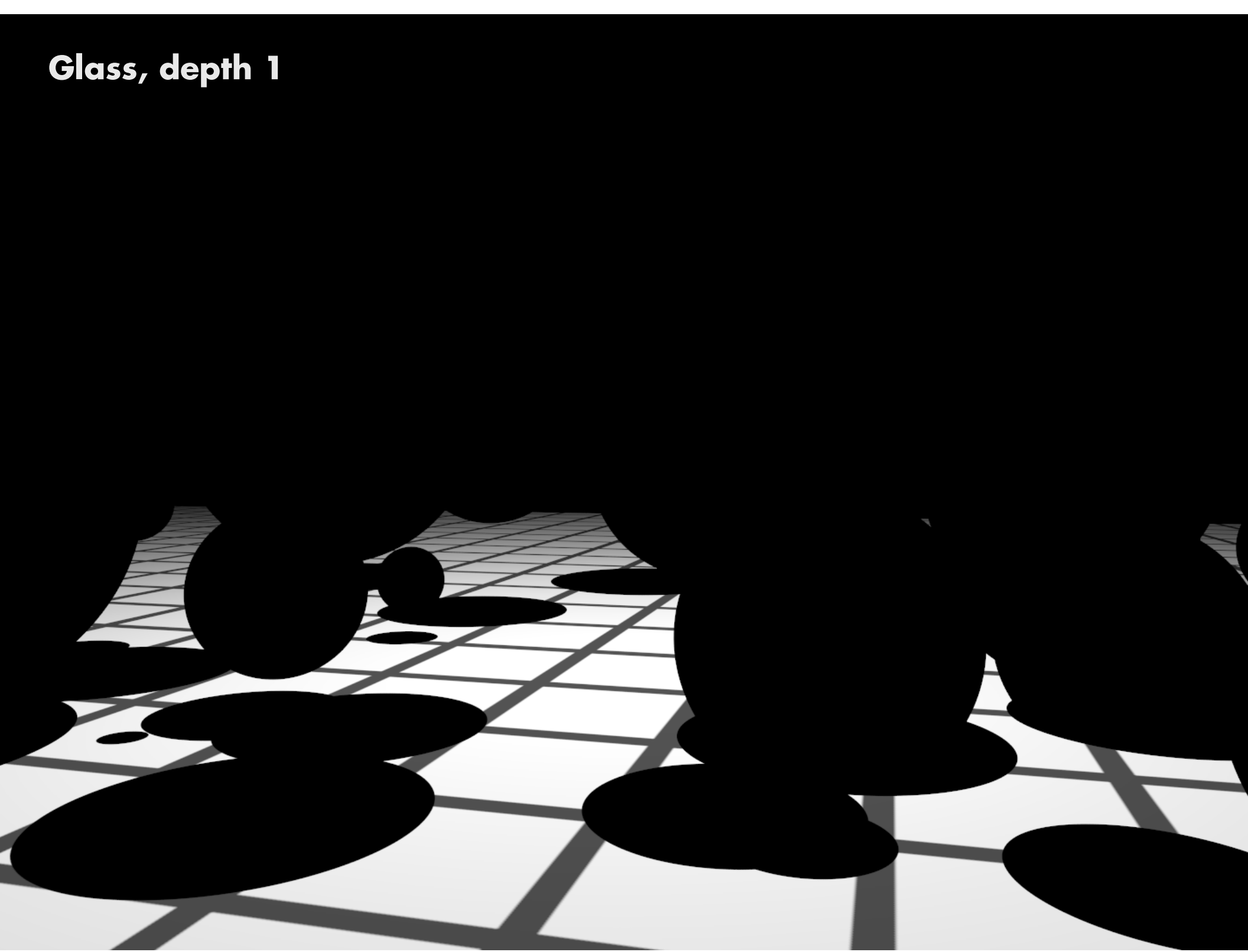
Mirror, depth 3



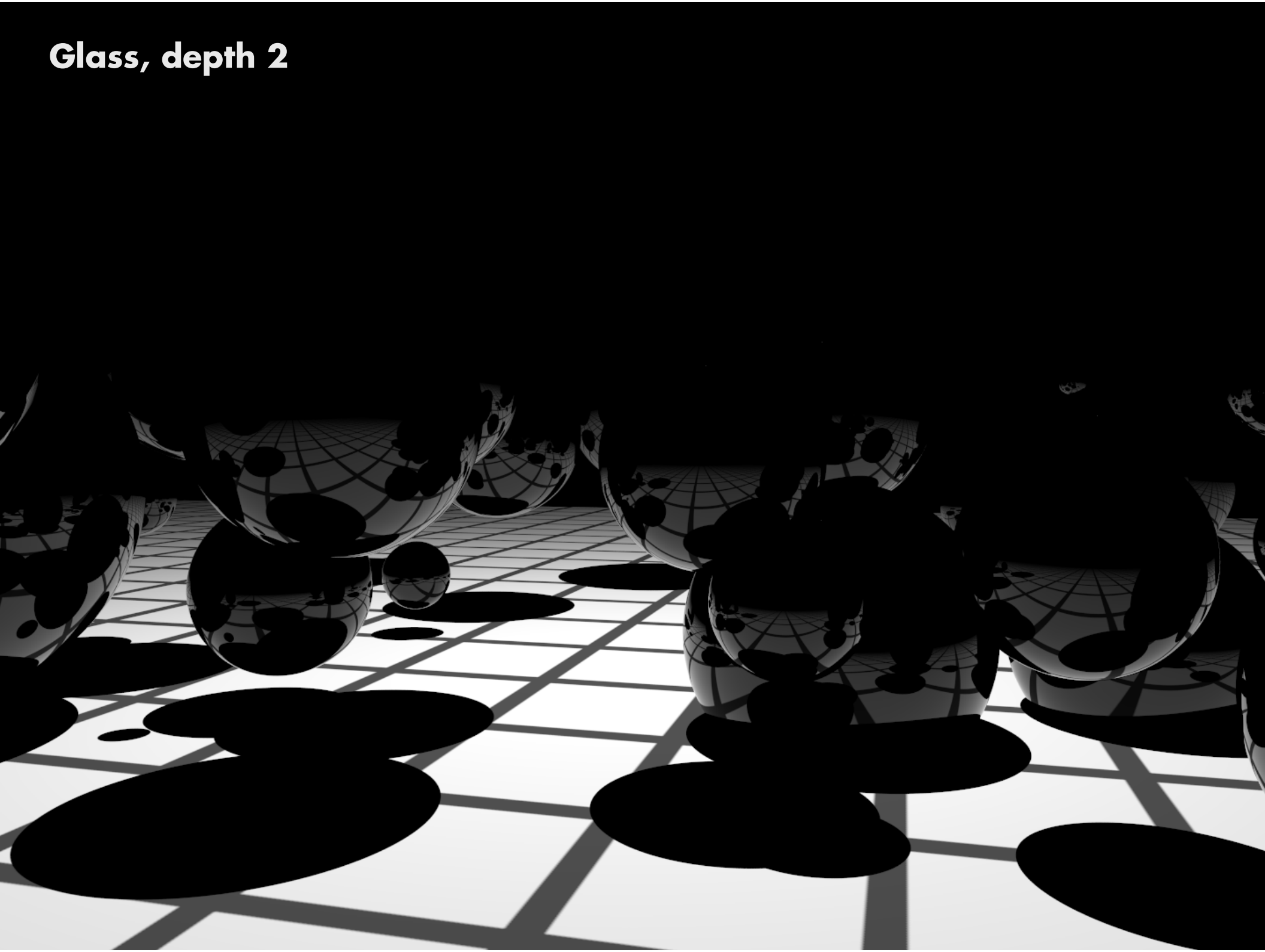
Mirror, depth 10



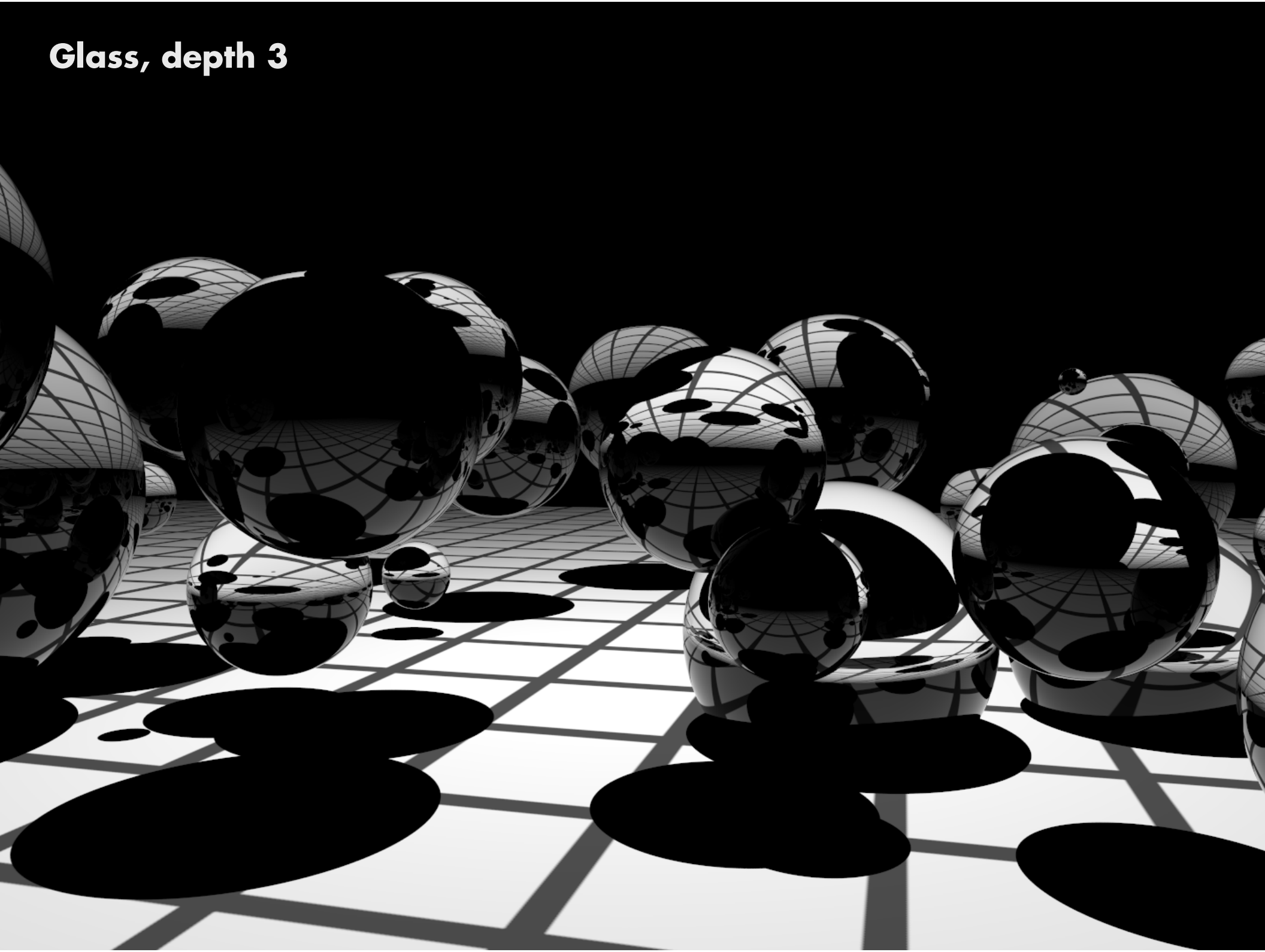
Glass, depth 1



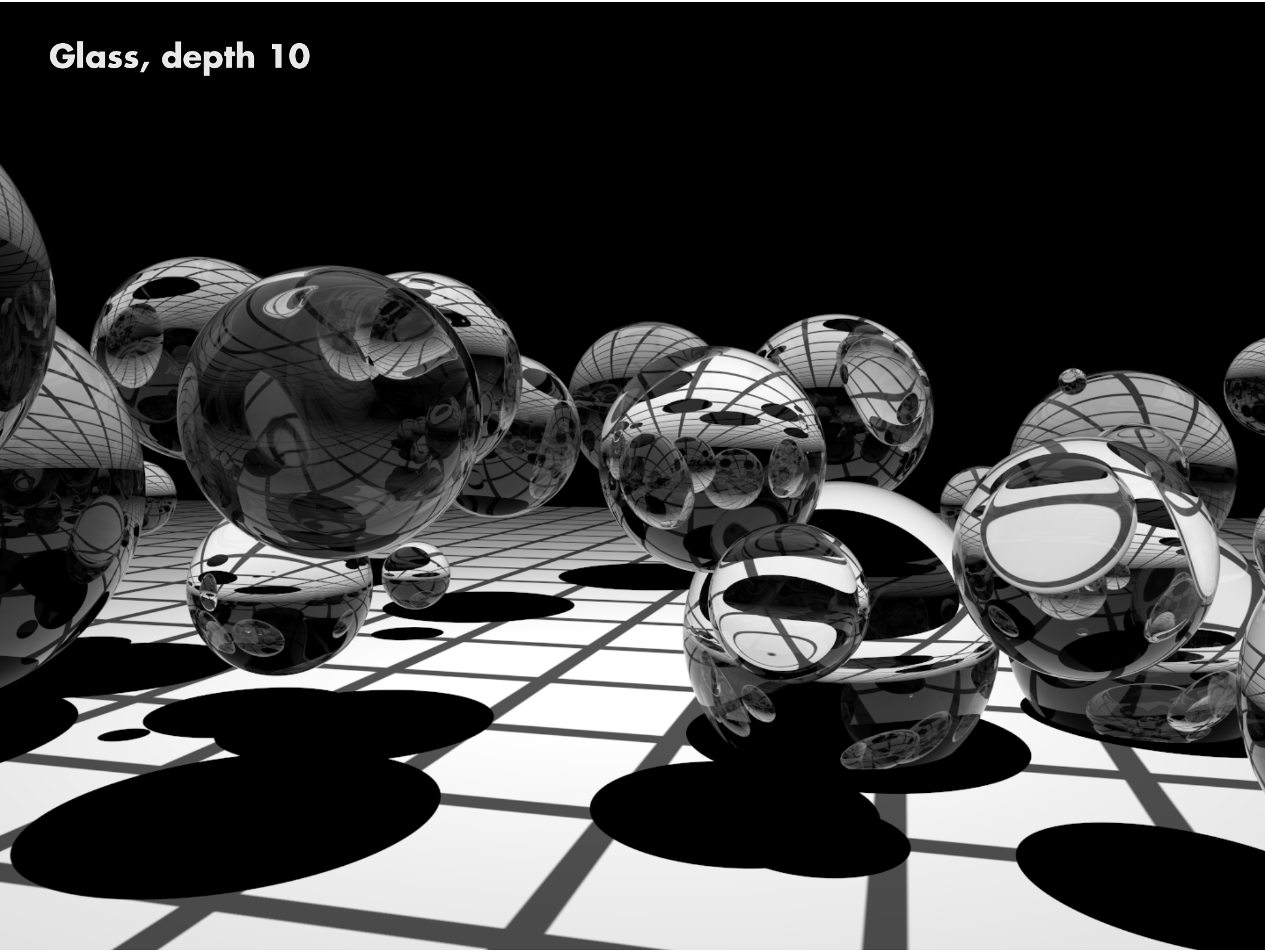
Glass, depth 2



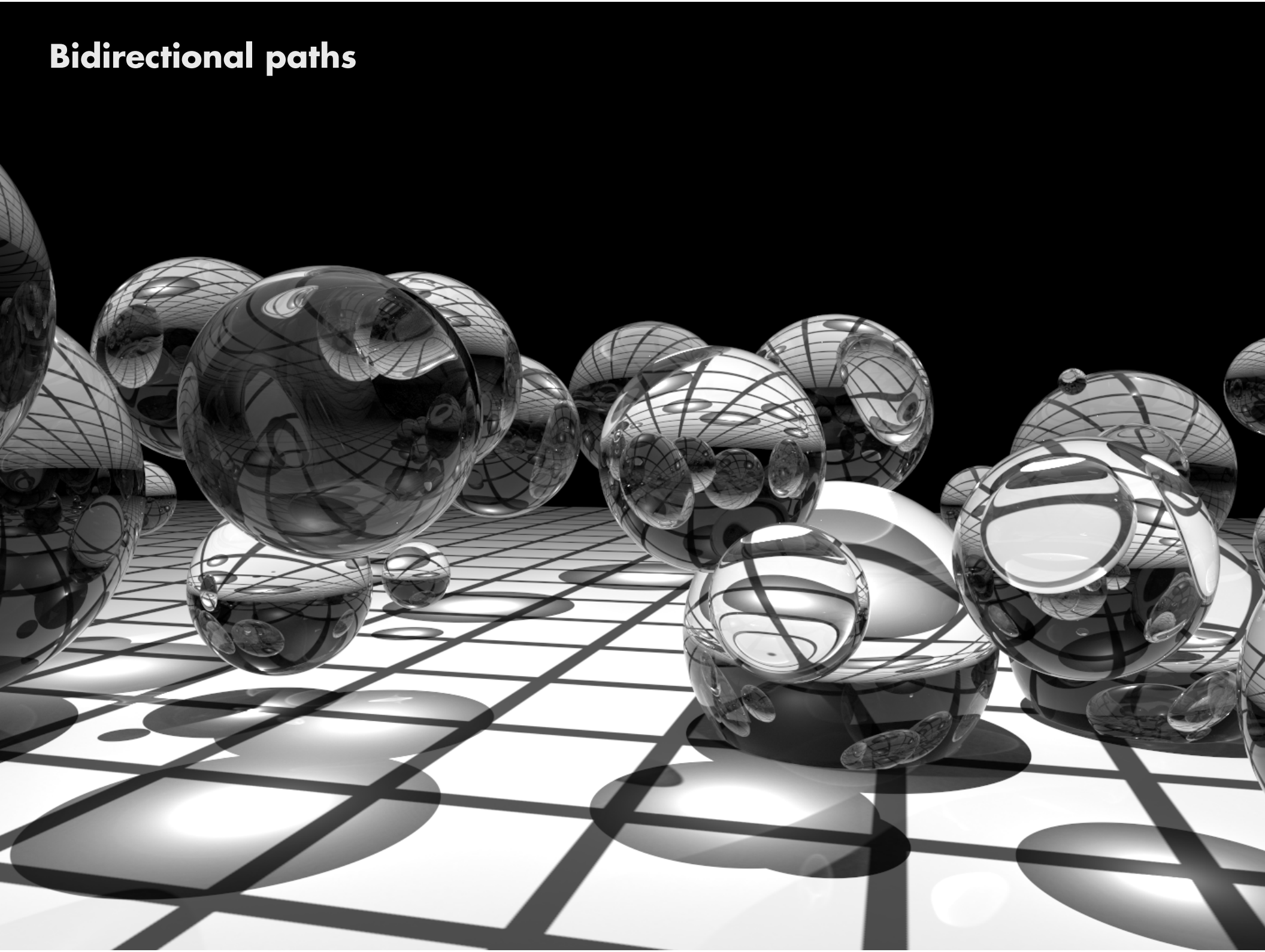
Glass, depth 3



Glass, depth 10



Bidirectional paths



Shape Interface (Simplified)

```
class Shape {
  public:
    Bounds3f ObjectBound() const;
    Bounds3f WorldBound() const;
    bool Intersect(const Ray &ray, Float *tHit,
                   SurfaceInteraction *isect,
                   bool testAlphaTexture) const;
    bool IntersectP(const Ray &ray,
                    bool testAlphaTexture);
    Float Area() const;
    // …
};
```

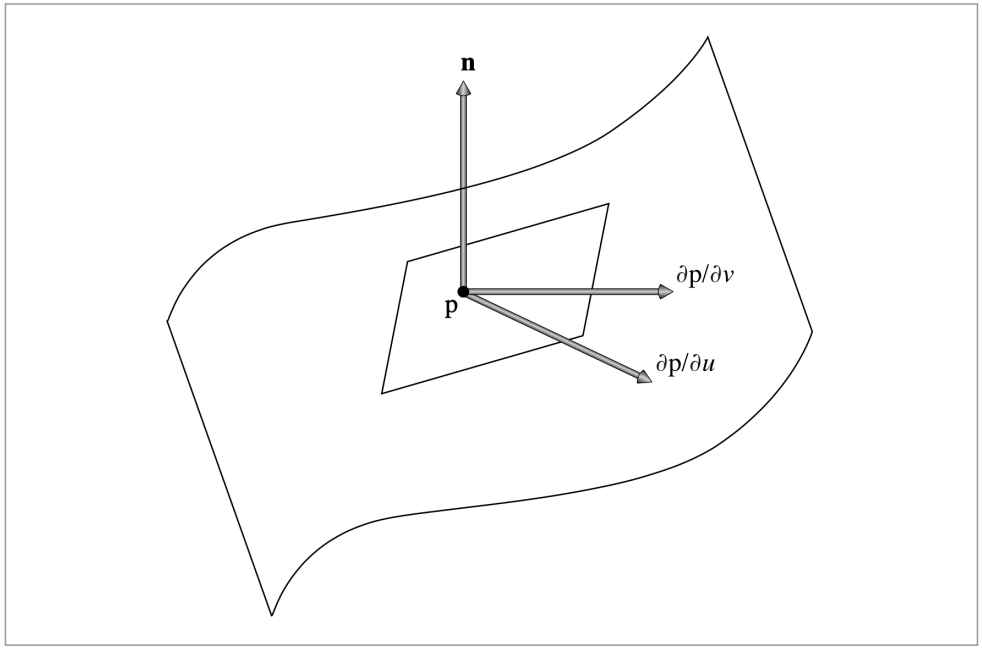
Surface Interaction (Simplified)

```
class SurfaceInteraction {
    Point3f p;
    Normal3f n;

    Point2f uv;
    Vector3f dpdu, dpdv;
    Normal3f dndu, dndv;

    struct {
        Normal3f n;
        Vector3f dpdu, dpdv;
        Normal3f dndu, dndv;
    } shading;

    // …
};
```

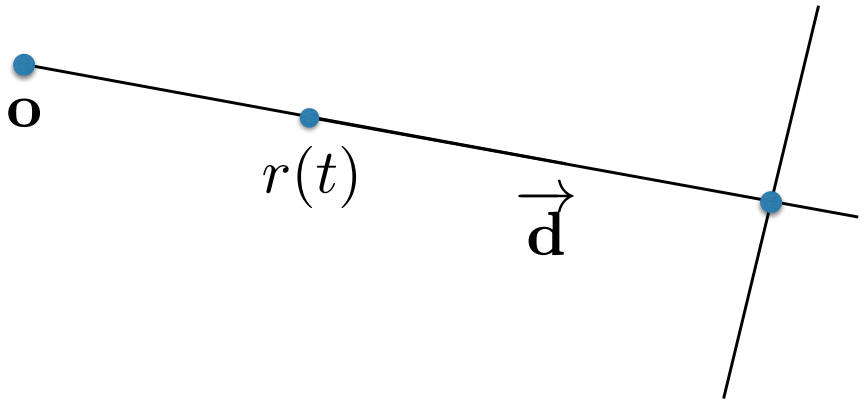


Ray-Surface Intersection

Ray-Plane Intersection

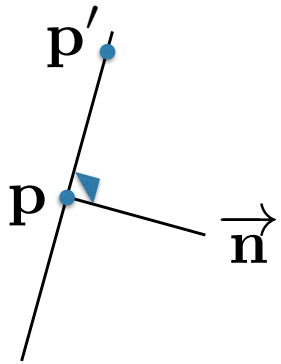
Ray

$$r(t) = \mathbf{o} + t\,\vec{\mathbf{d}}$$



Want *t* where the ray intersects the plane

Plane



$$(\mathbf{p} - \mathbf{p}') \cdot \vec{\mathbf{n}} = 0$$

Ray-Plane Intersection

Ray: $r(t) = \mathbf{o} + t\,\overrightarrow{\mathbf{d}}$

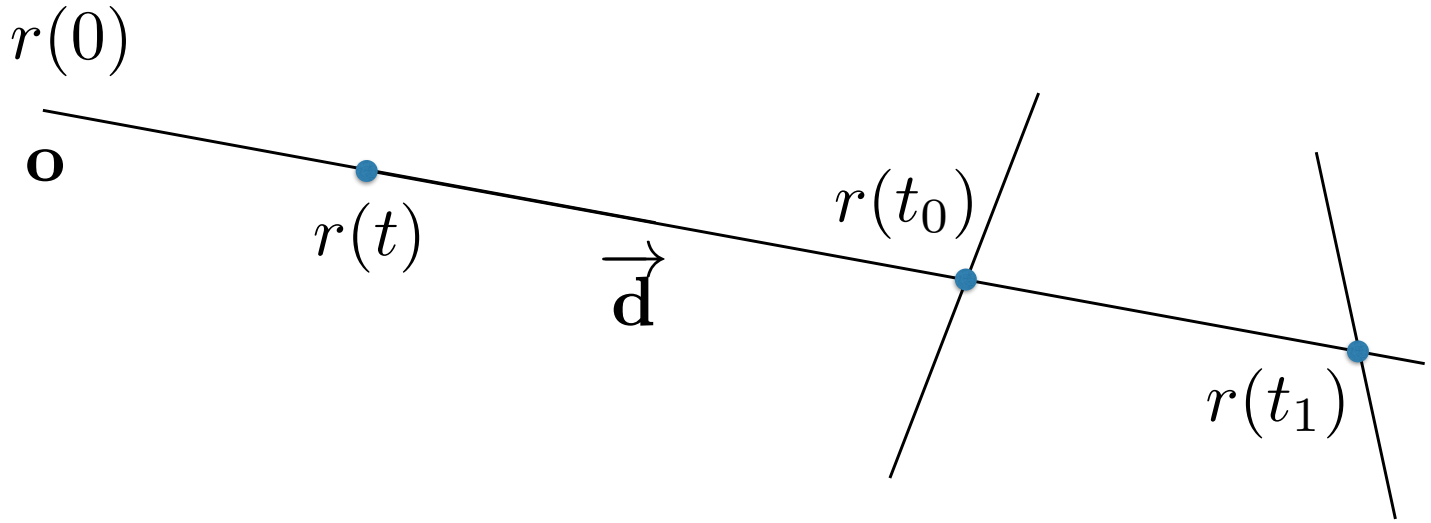
Plane: $(\mathbf{p} - \mathbf{p}') \cdot \overrightarrow{\mathbf{n}} = 0$

Substitute ray equation into plane equation:

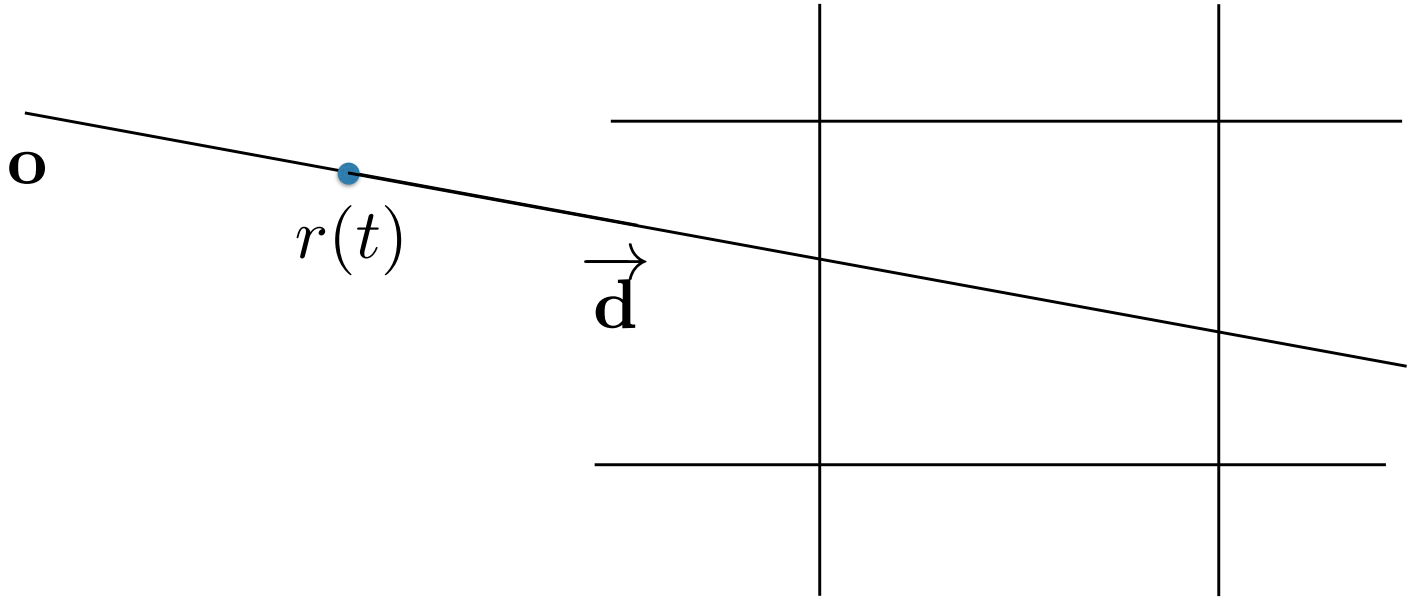
$$(\mathbf{p} - r(t)) \cdot \overrightarrow{\mathbf{n}} = (\mathbf{p} - (\mathbf{o} + t\,\overrightarrow{\mathbf{d}})) \cdot \overrightarrow{\mathbf{n}} = 0$$

$$t = -\frac{(\mathbf{o} - \mathbf{p}) \cdot \overrightarrow{\mathbf{n}}}{\overrightarrow{\mathbf{d}} \cdot \overrightarrow{\mathbf{n}}}$$

Finding The Closest Intersection



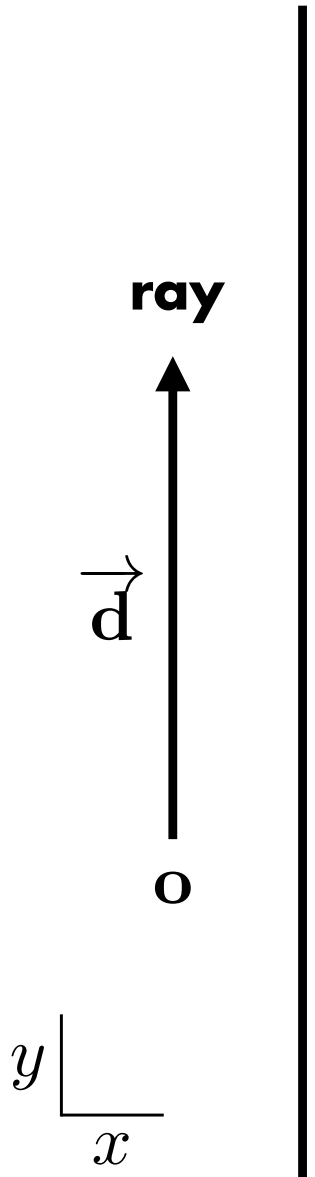
Optimizing Ray-Box



$$t = -\frac{(\mathbf{o} - \mathbf{p}) \cdot \vec{\mathbf{n}}}{\vec{\mathbf{d}} \cdot \vec{\mathbf{n}}}$$

$$t = -\frac{\mathbf{o_x} - \mathbf{p_x}}{\vec{\mathbf{d_x}}}$$

What About Rays Parallel to a Plane?



plane

$$t = -\frac{\mathbf{o_x} - \mathbf{p_x}}{\vec{\mathbf{d_x}}}$$

Math says: $$t = -\frac{\mathbf{o_x} - \mathbf{p_x}}{0} = \pm\infty$$

IEEE Floating Point Standard says:

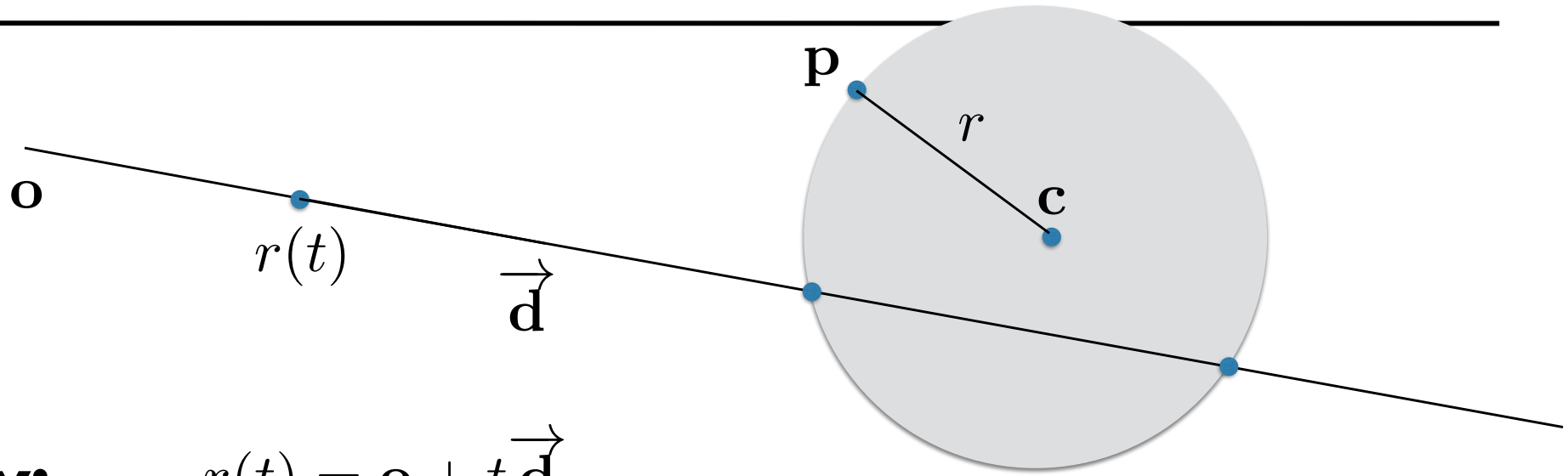
1. **positive num / 0 = +Inf**

   negative num / 0 = -Inf

2. **+Inf > all other floats**

   -Inf < all other floats

Ray-Sphere Intersection



Ray: $r(t) = \mathbf{o} + t\overrightarrow{\mathbf{d}}$

Sphere: $||\mathbf{p} - \mathbf{c}||^2 - r^2 = 0$

$$(\mathbf{o} + t\overrightarrow{\mathbf{d}} - \mathbf{c})^2 - r^2 = 0$$

$$at^2 + bt + c = 0$$

$$a = \overrightarrow{\mathbf{d}} \cdot \overrightarrow{\mathbf{d}}$$

$$b = 2(\mathbf{o} - \mathbf{c}) \cdot \overrightarrow{\mathbf{d}}$$

$$c = ((\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c})) - r^2$$

Ray-Triangle Intersection

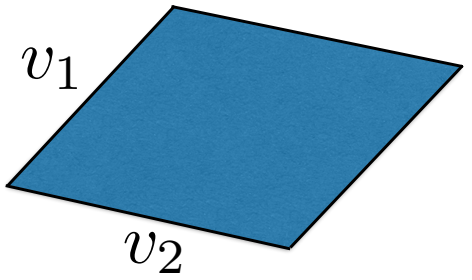
Ray-Triangle Intersection

1. Find ray-plane intersection point using the methods developed previously
2. Test whether the intersection point is inside the triangle

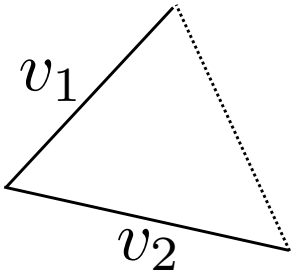
Review: Geometric Building-Blocks

The signed area of the parallelogram given by the vectors $v_1 = (x_1, y_1)$ and $v_2 = (x_2, y_2)$ is given by

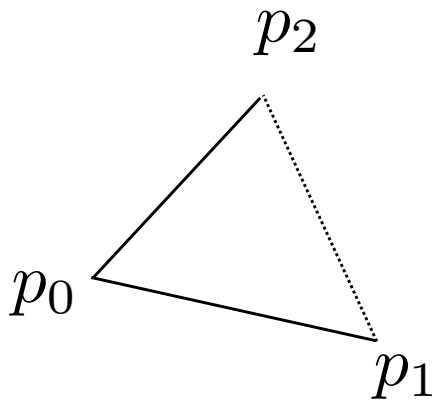
$$\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} = (x_1 y_2) - (x_2 y_1)$$



Half of this area is the area of the triangle determined by the 3 points



Review: Geometric Building-Blocks

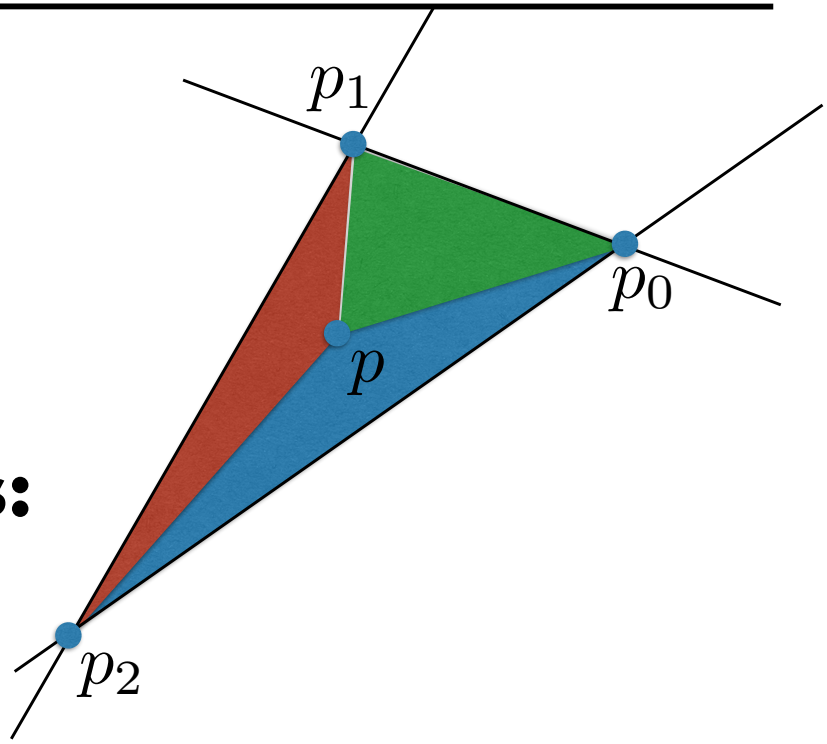


Area of a triangle with vertices $p_0 = (x_0, y_0)$ etc.

$$\frac{1}{2}\begin{vmatrix} x_1 - x_0 & x_2 - x_0 \\ y_1 - y_0 & y_2 - y_0 \end{vmatrix} = \frac{1}{2}((x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0))$$

Barycentric Coordinates

$$a_0(p) = \text{Area}(p_1, p_2, p)$$
$$a_1(p) = \text{Area}(p_2, p_0, p)$$
$$a_2(p) = \text{Area}(p_0, p_1, p)$$



Define barycentric coordinates:

$$b_i = \frac{a_i}{\text{Area}(p_0, p_1, p_2)}, \quad 0 \le i \le 2$$

p is inside the triangle if $b_0 > 0, b_1 > 0, b_2 > 0$

Ray-Triangle Intersection

Points on a plane: $\mathbf{p} = b_0\mathbf{p_0} + b_1\mathbf{p_1} + b_2\mathbf{p_2}$

$$\begin{bmatrix} \mathbf{p_0} & \mathbf{p_1} & \mathbf{p_2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} \mathbf{p} \end{bmatrix}$$

1. Find ray-plane intersection point

2. Test whether that point is inside the triangle

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} \mathbf{p_0} & \mathbf{p_1} & \mathbf{p_2} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{p} \end{bmatrix}$$

Inside iff $b_0 > 0, b_1 > 0, b_2 > 0$

Ray Triangle Intersection

$$\mathbf{o} + t\overrightarrow{\mathbf{d}} = (1 - b_1 - b_2)\mathbf{p_0} + b_1\mathbf{p_1} + b_2\mathbf{p_2}$$

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\overrightarrow{\mathbf{s_1}} \cdot \overrightarrow{\mathbf{e_1}}} \begin{bmatrix} \overrightarrow{\mathbf{s_2}} \cdot \overrightarrow{\mathbf{e_2}} \\ \overrightarrow{\mathbf{s_1}} \cdot \overrightarrow{\mathbf{s}} \\ \overrightarrow{\mathbf{s_2}} \cdot \overrightarrow{\mathbf{d}} \end{bmatrix}$$

Where

$$\overrightarrow{\mathbf{e_1}} = \overrightarrow{\mathbf{p_1}} - \overrightarrow{\mathbf{p_0}}$$
$$\overrightarrow{\mathbf{e_2}} = \overrightarrow{\mathbf{p_2}} - \overrightarrow{\mathbf{p_0}}$$
$$\overrightarrow{\mathbf{s}} = \overrightarrow{\mathbf{o}} - \overrightarrow{\mathbf{p_0}}$$
$$\overrightarrow{\mathbf{s_1}} = \overrightarrow{\mathbf{d}} \times \overrightarrow{\mathbf{e_2}}$$
$$\overrightarrow{\mathbf{s_2}} = \overrightarrow{\mathbf{s}} \times \overrightarrow{\mathbf{e_1}}$$

[Möller and Trumbore 1997]

Contemporary Bathroom Scene

525M triangles

12.9B ray-triangle tests

~26 tests per triangle

3.2B hits (~25% of tests)

~30 minutes with 8 threads

5m 6s in ray/triangle (17%)

Conserve Memory vs. Coherence

```
class Triangle {
  // …
  int *v;
  TriangleMesh *mesh;
};

class TriangleMesh {
  Point3f *P;
  // …
};
```

pbrt Triangle

```
class Triangle {
  // …
  Point3f P[3];
};
```

Alternative Triangle

Ray-Implicit Surface Intersection

Implicit surface

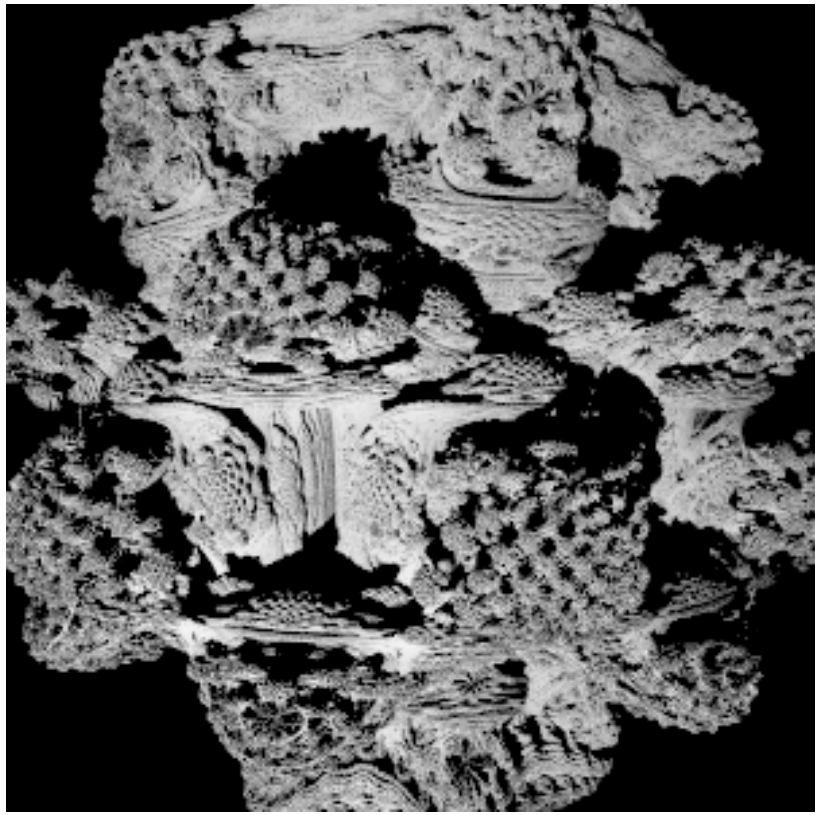
$$f(x, y, z) = 0$$

Substitute ray equation

$$x = o_x + td_x$$
$$y = o_y + td_y$$
$$z = o_z + td_z$$

Univariate root finding

$$f^*(t) = 0$$

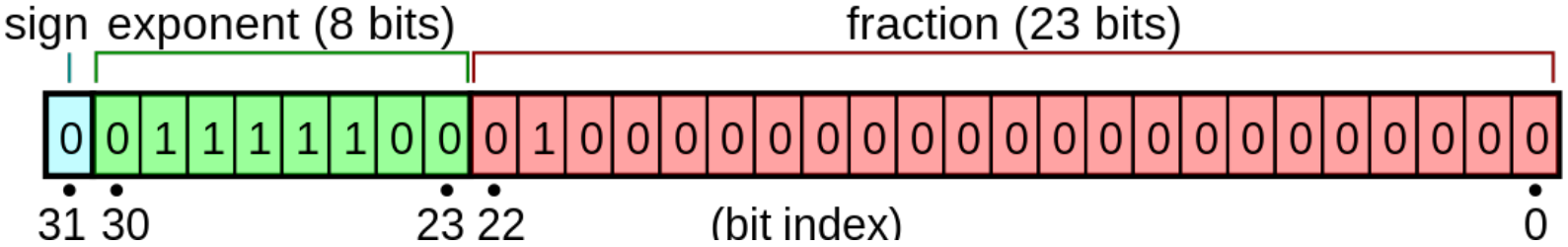


Floating-Point Error

Floating-point Representation

Scientific notation $\pm 1.m \times 2^e$

- **with a fixed sized mantissa (23-bits),**
- **a limited exponent range (8-bits, e-127),**
- **signed bit**

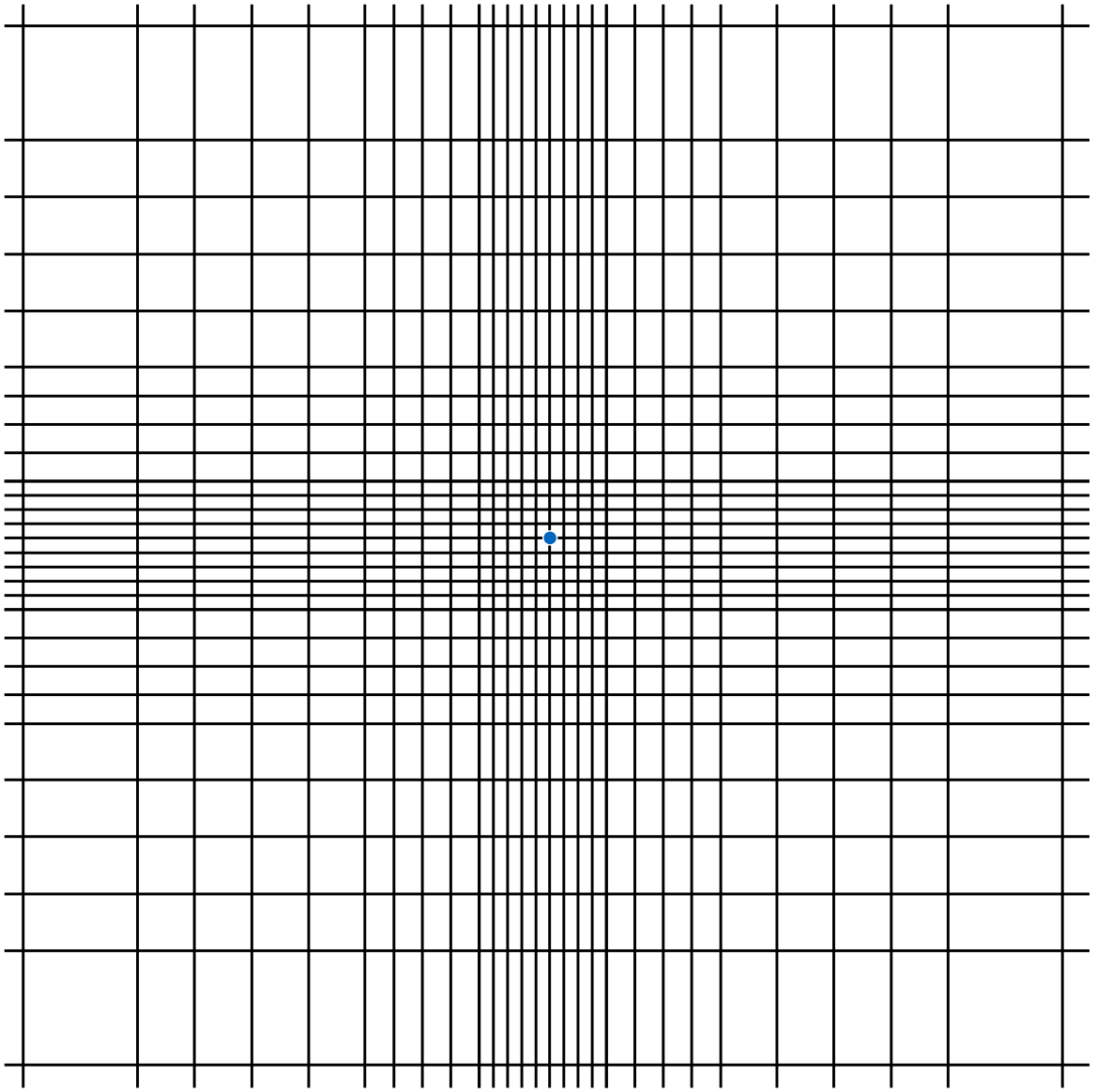


$$2.5 = 1.25 \times 2^1 = 1.01_b \times 2^1$$

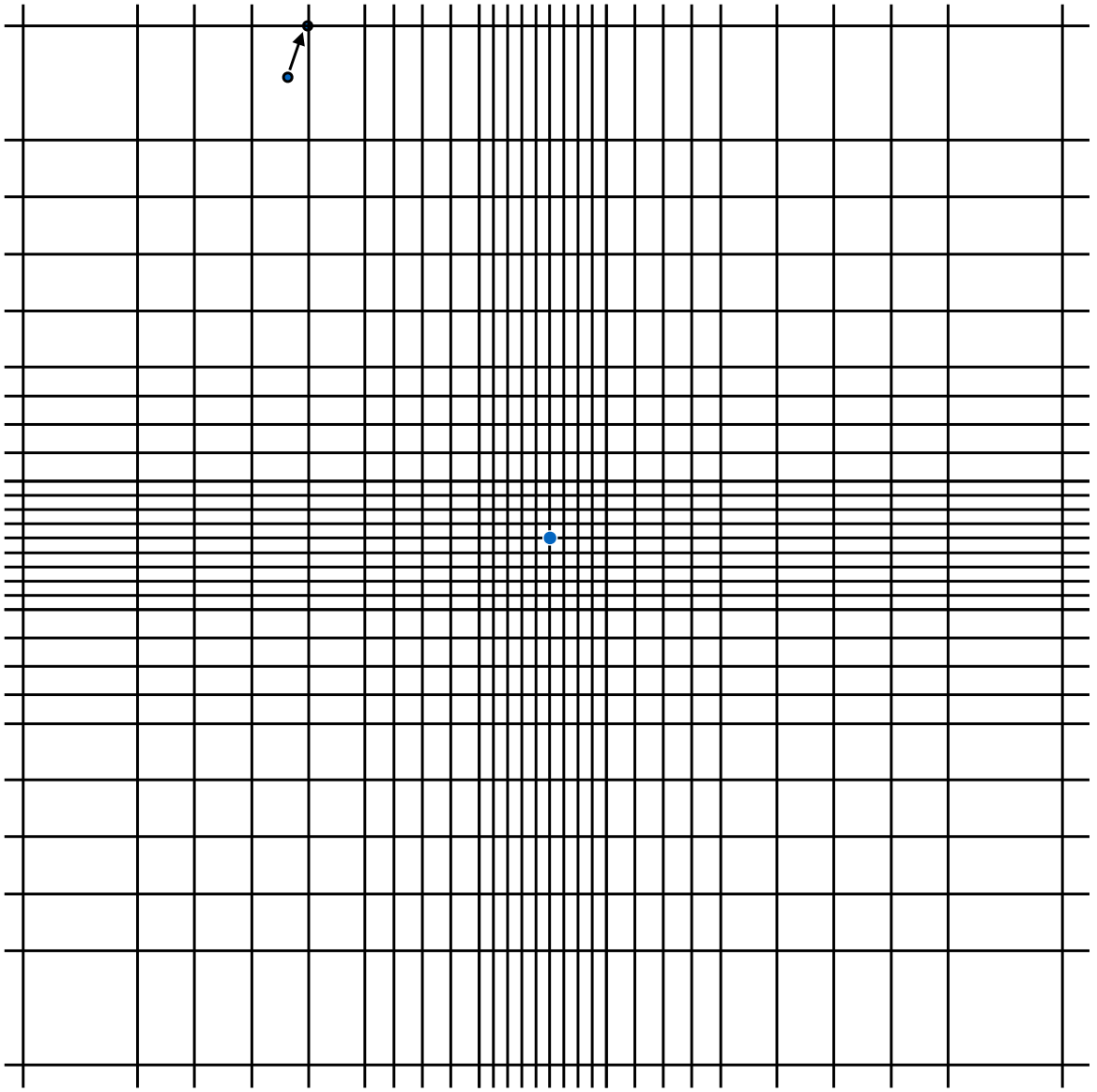
$$1/3 \approx 1.01010101010101010101011_b \times 2^{-2}$$

$$0 = ?$$

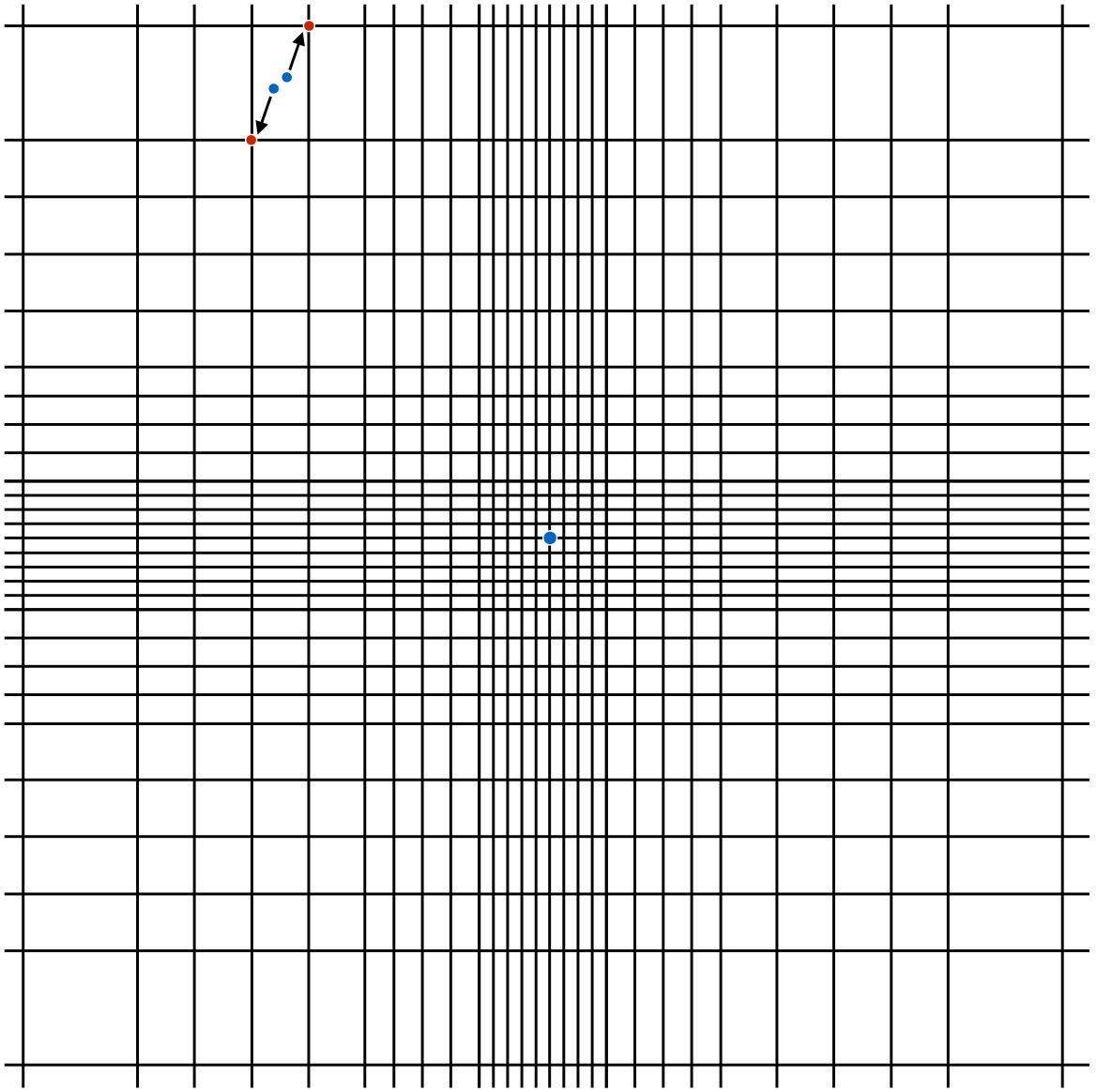
Floating-point Representation



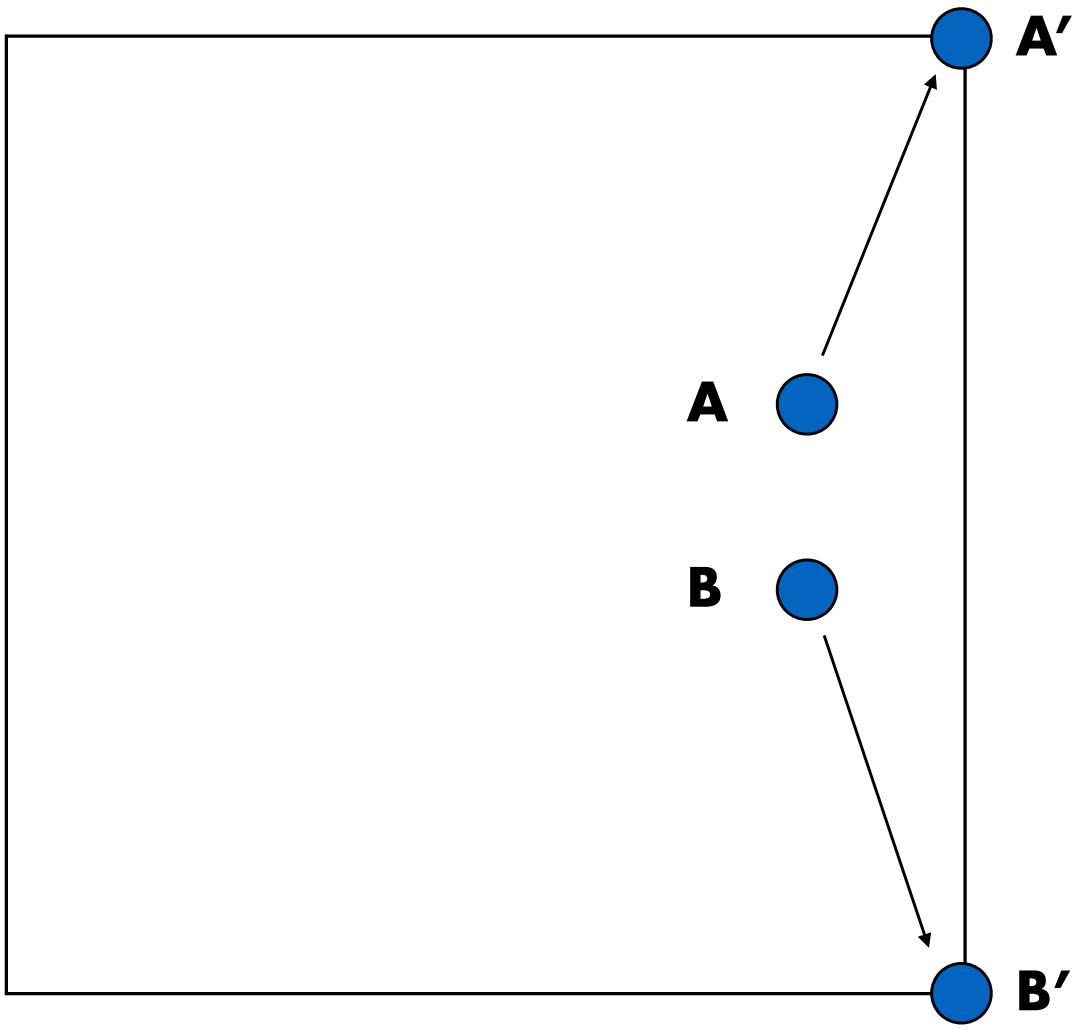
Roundoff Error



Catastrophic Cancellation

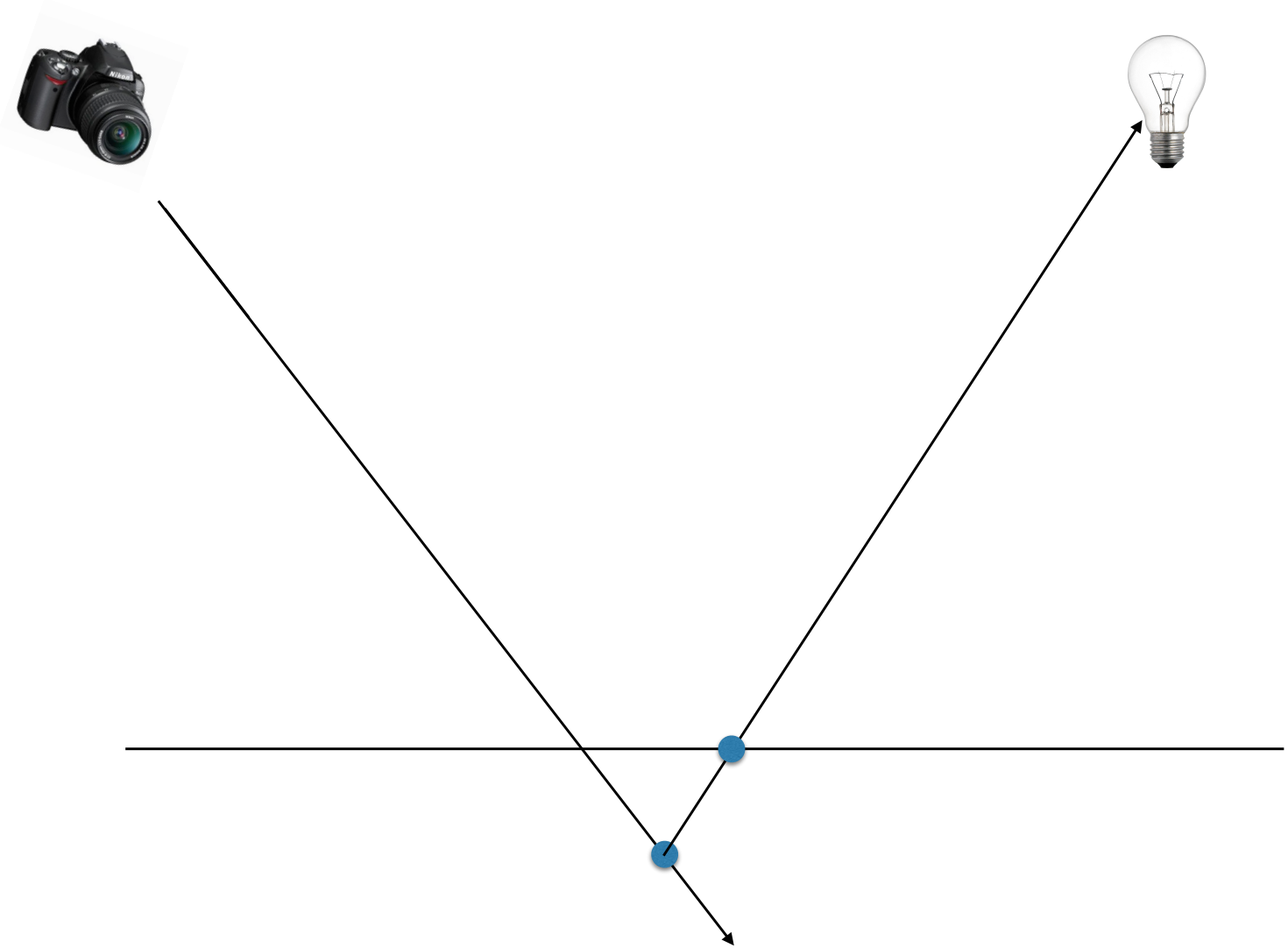


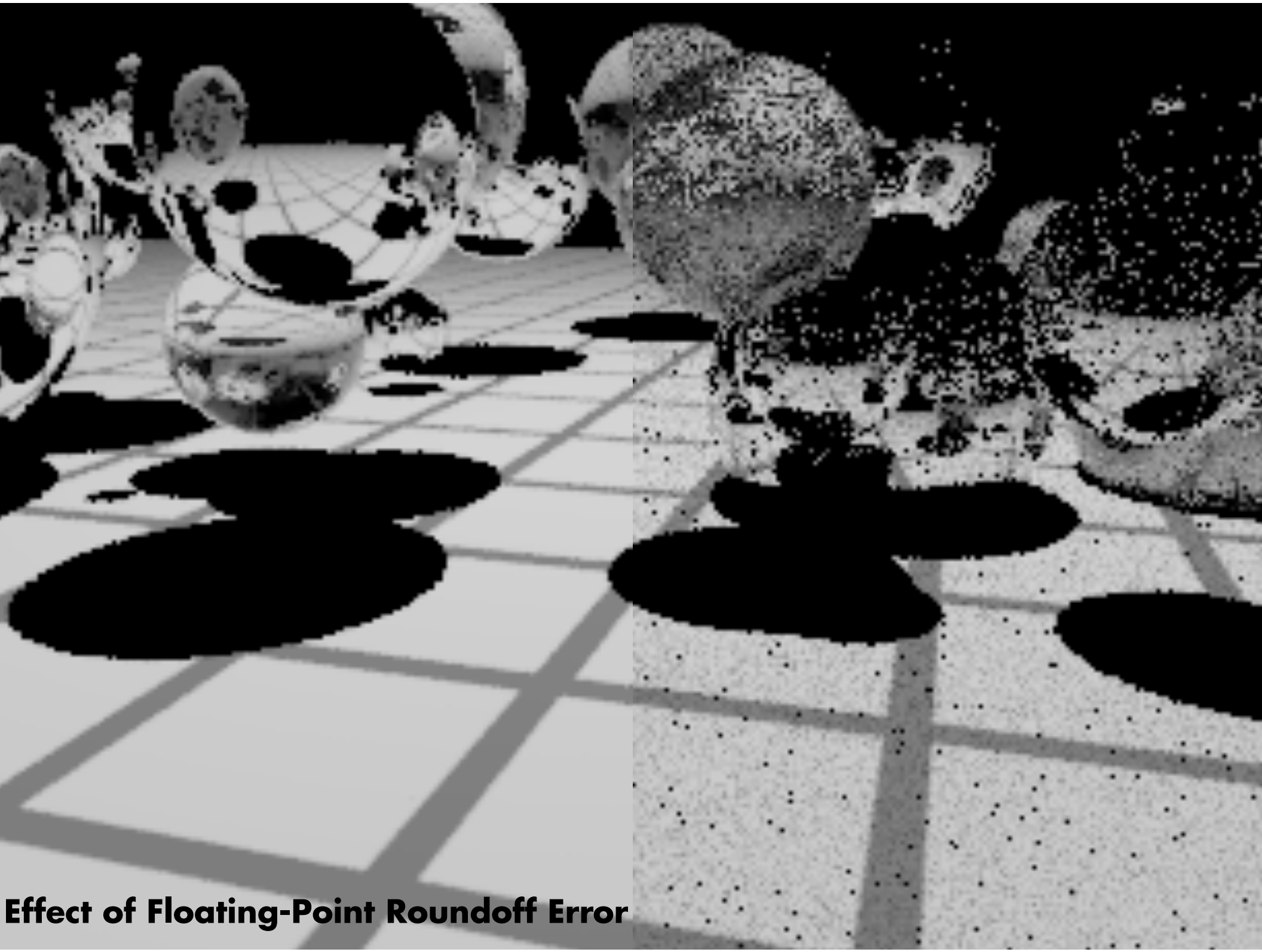
Catastrophic Cancellation



A - B vs A′ - B′ …

Effect of Roundoff Error





Effect of Floating-Point Roundoff Error

Round-off Error Remedies

Use double (fp64) rather than float (fp32)

- Can help, but also avoids the root problem
- Increases memory use, reduces performance

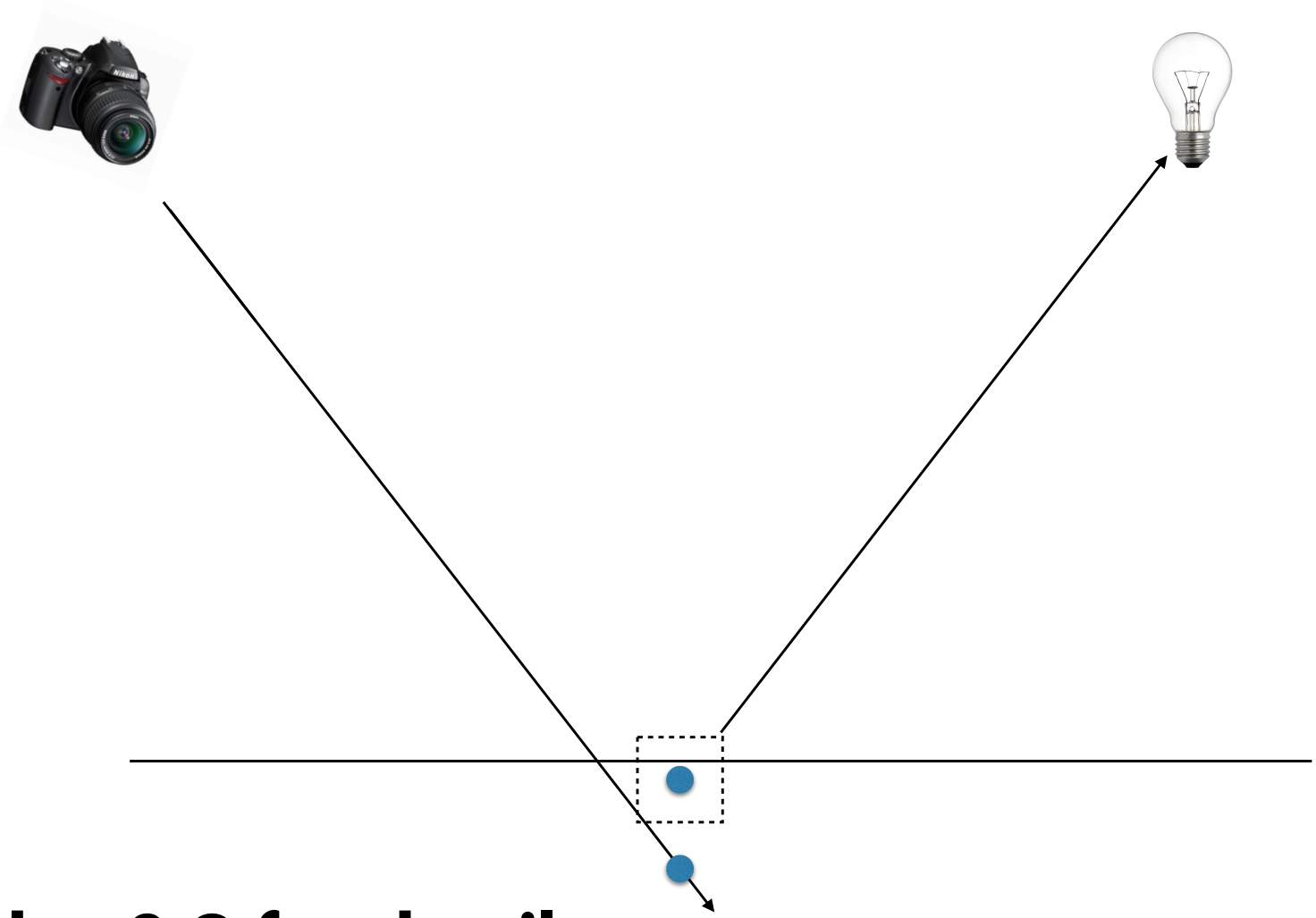
Ignore reintersection with the last object hit

- Only works for flat objects (e.g. triangles)
- No help if model has coincident triangles

Have a *tmin* along ray to ignore close intersections

- What value should *tmin* take?

Refine Intersection Point, Bound Error



See pbrt 3.9 for details