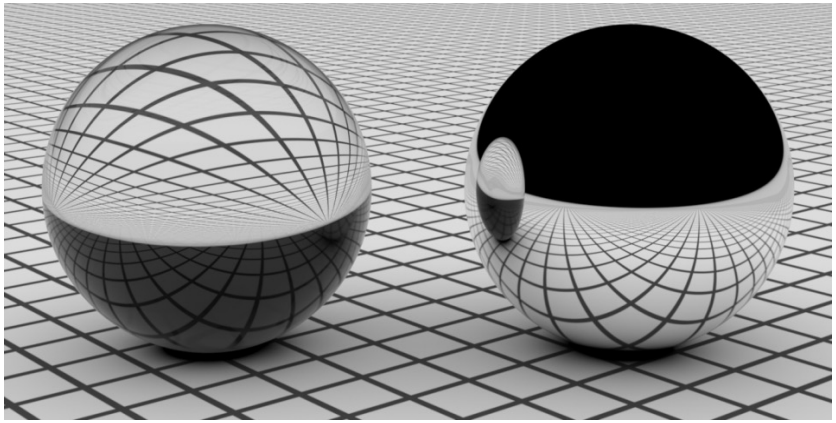
Ray Tracing I: Basics

Today

- Basic algorithms
- Overview of pbrt
- Ray-surface intersection



Next lecture

- Techniques to accelerate ray tracing of large numbers of geometric primitives

Light Rays

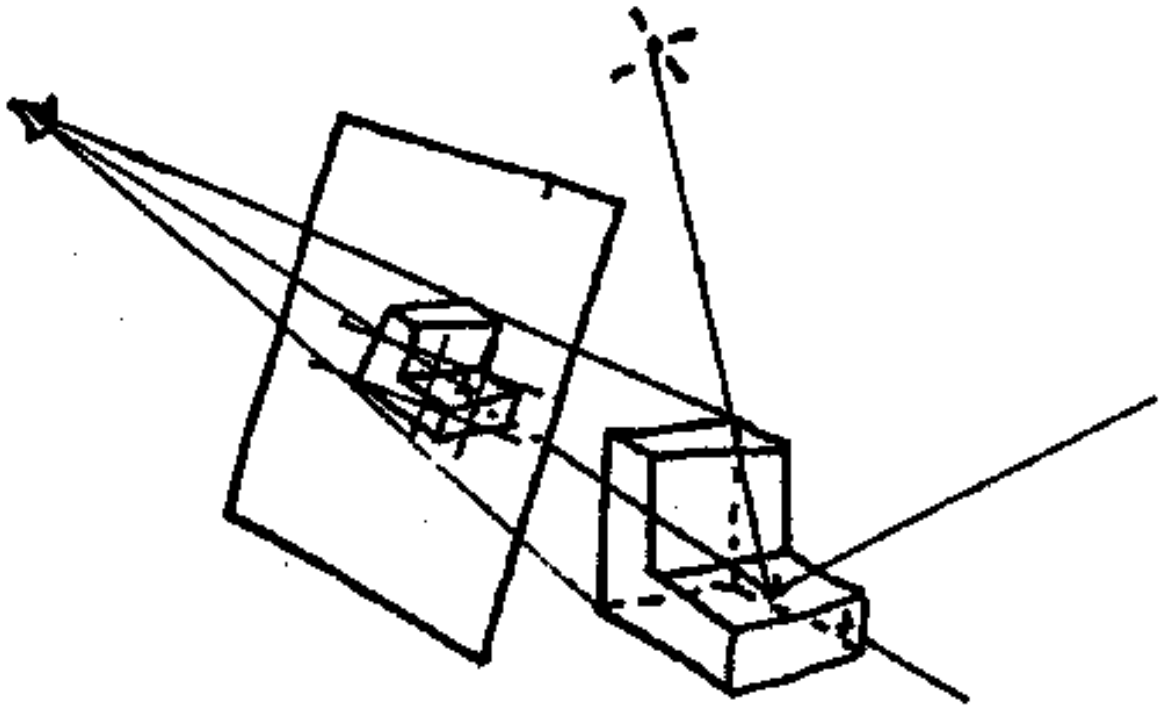
Three ideas about light rays

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

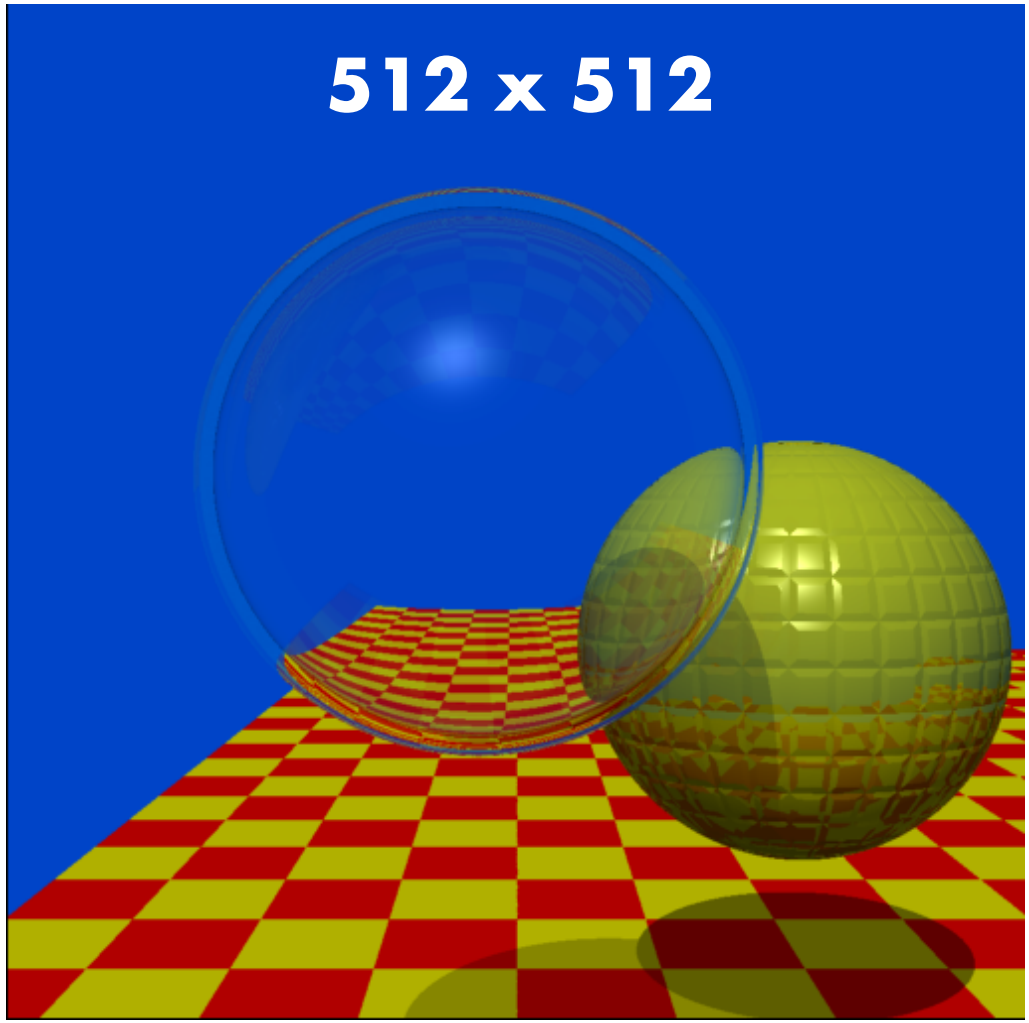
1. Generate an image by casting one ray per pixel
2. Check for shadows by sending a ray to the light



Ray Tracing in Computer Graphics

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

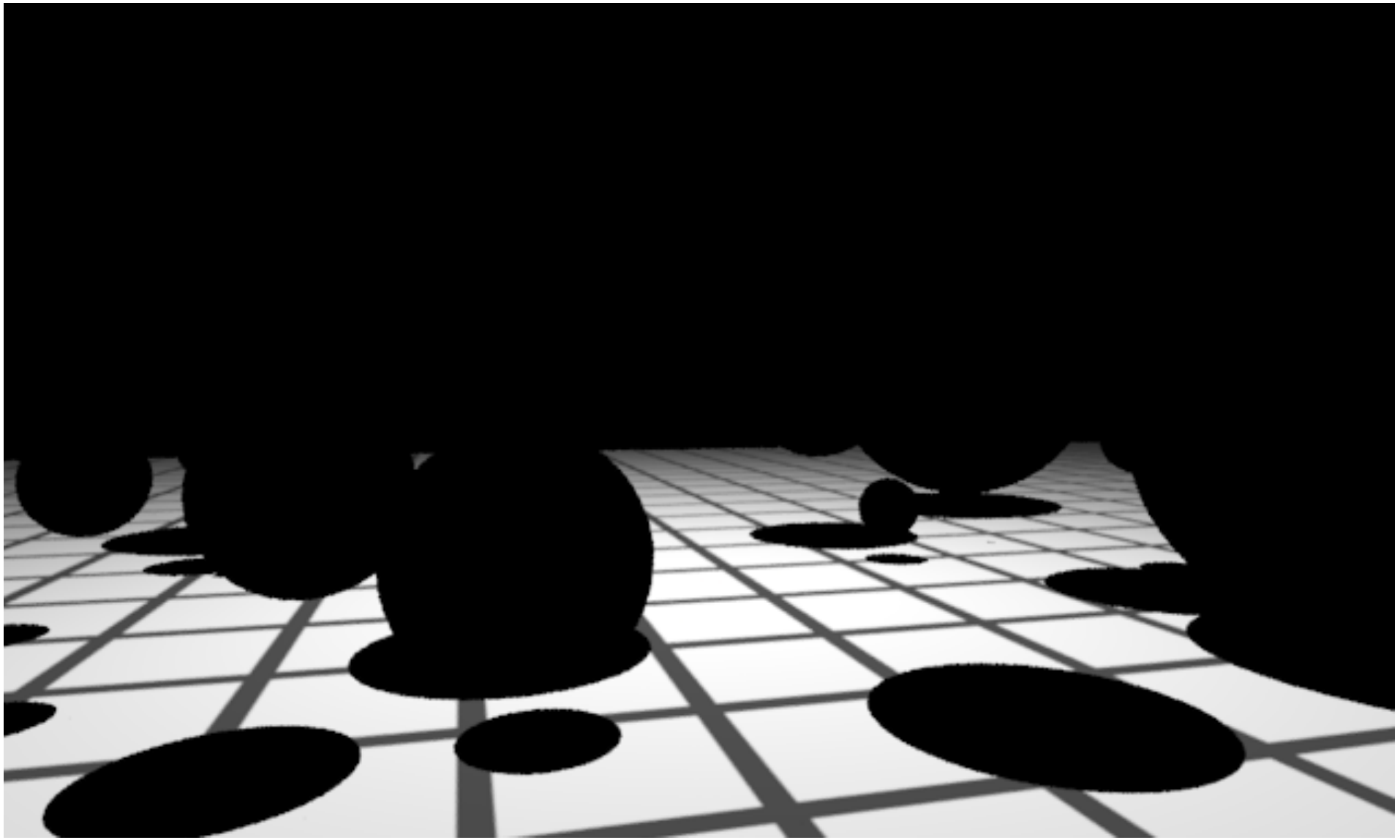
Time:
- **VAX 11/780 (1979) 74m**
- **PC (2006) 6s**
- **GPU (2012) 1/30s**



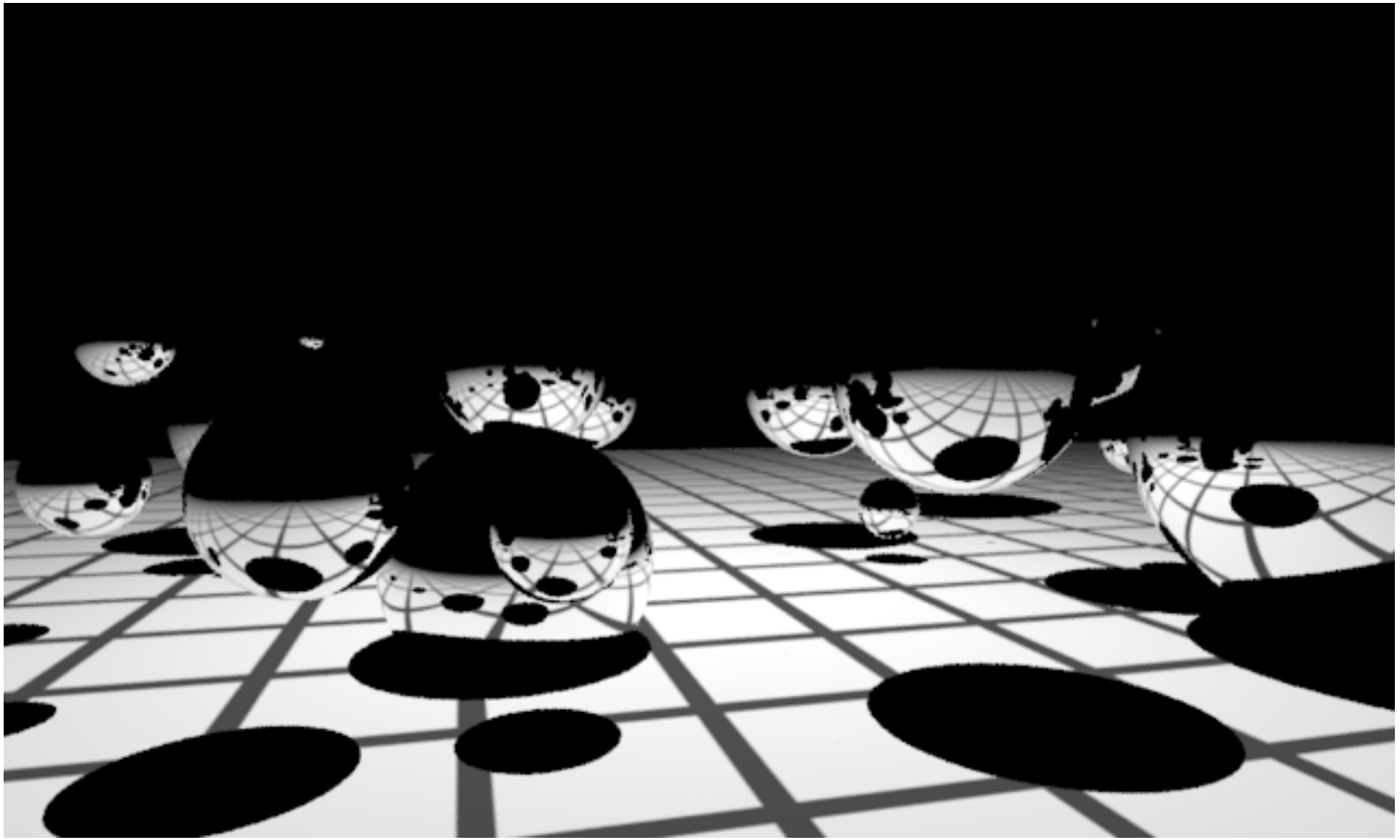
Spheres and Checkerboard, T. Whitted, 1979

Ray Tracing Video

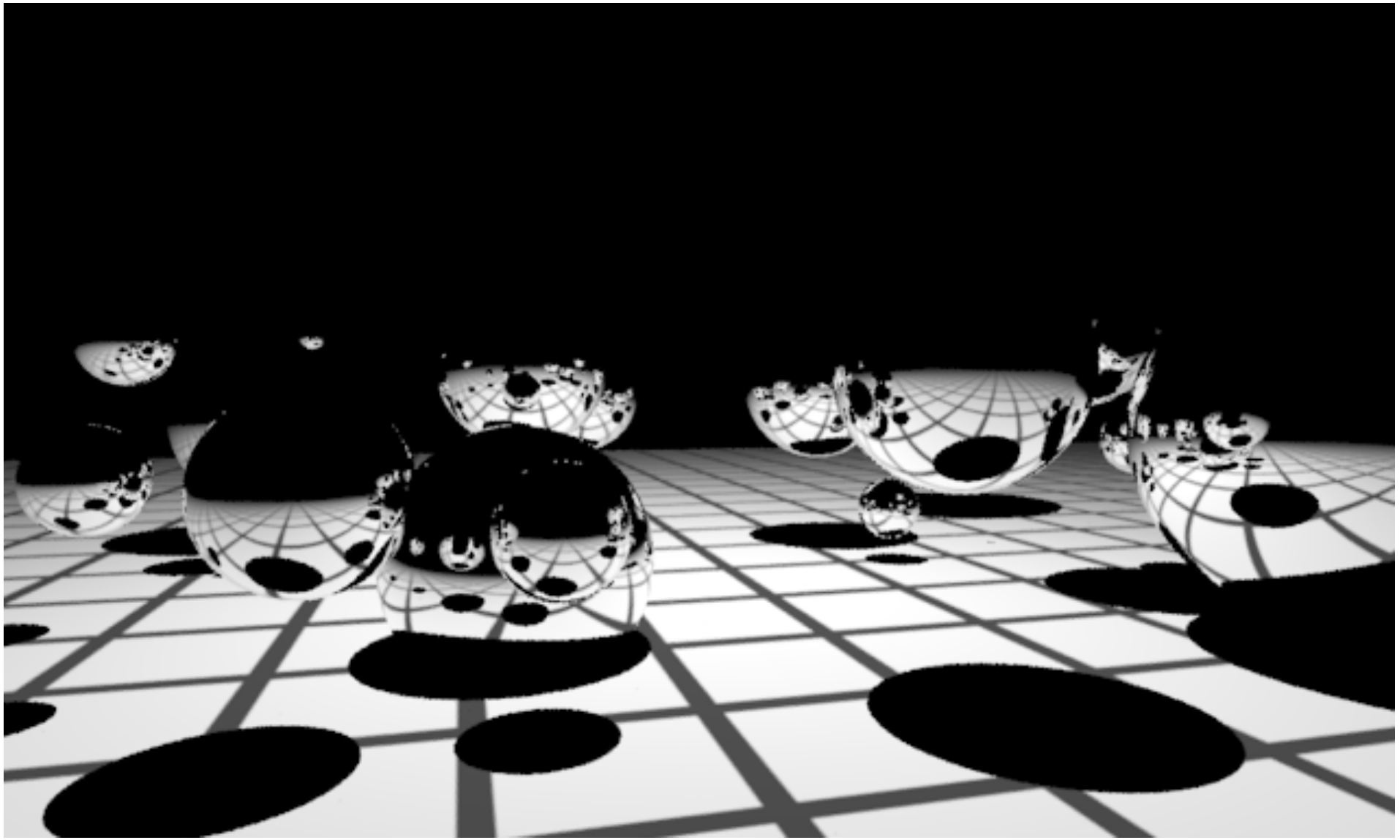
Spheres-over-plane.pbrt (depth=1)



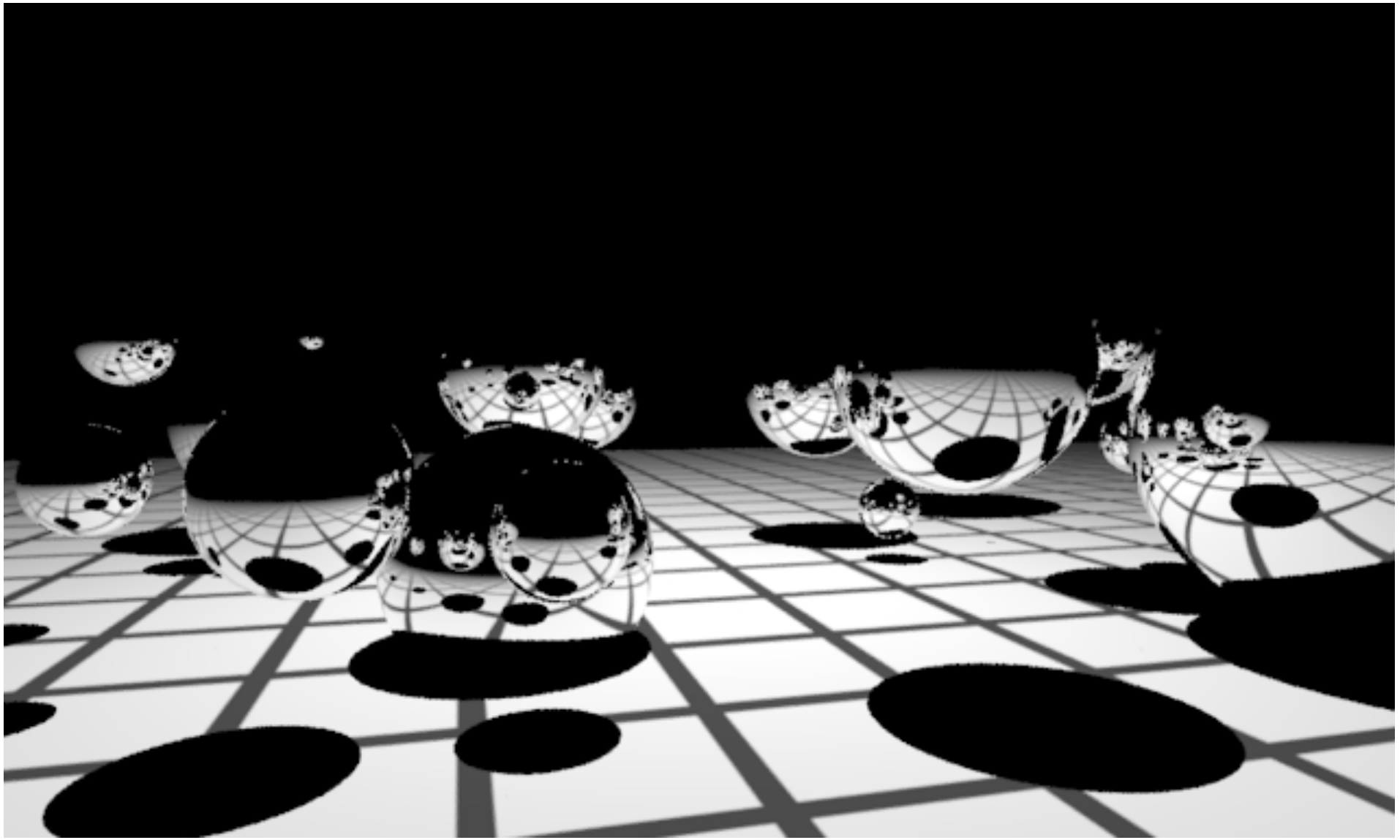
Spheres-over-plane.pbrt (mirror depth=2)



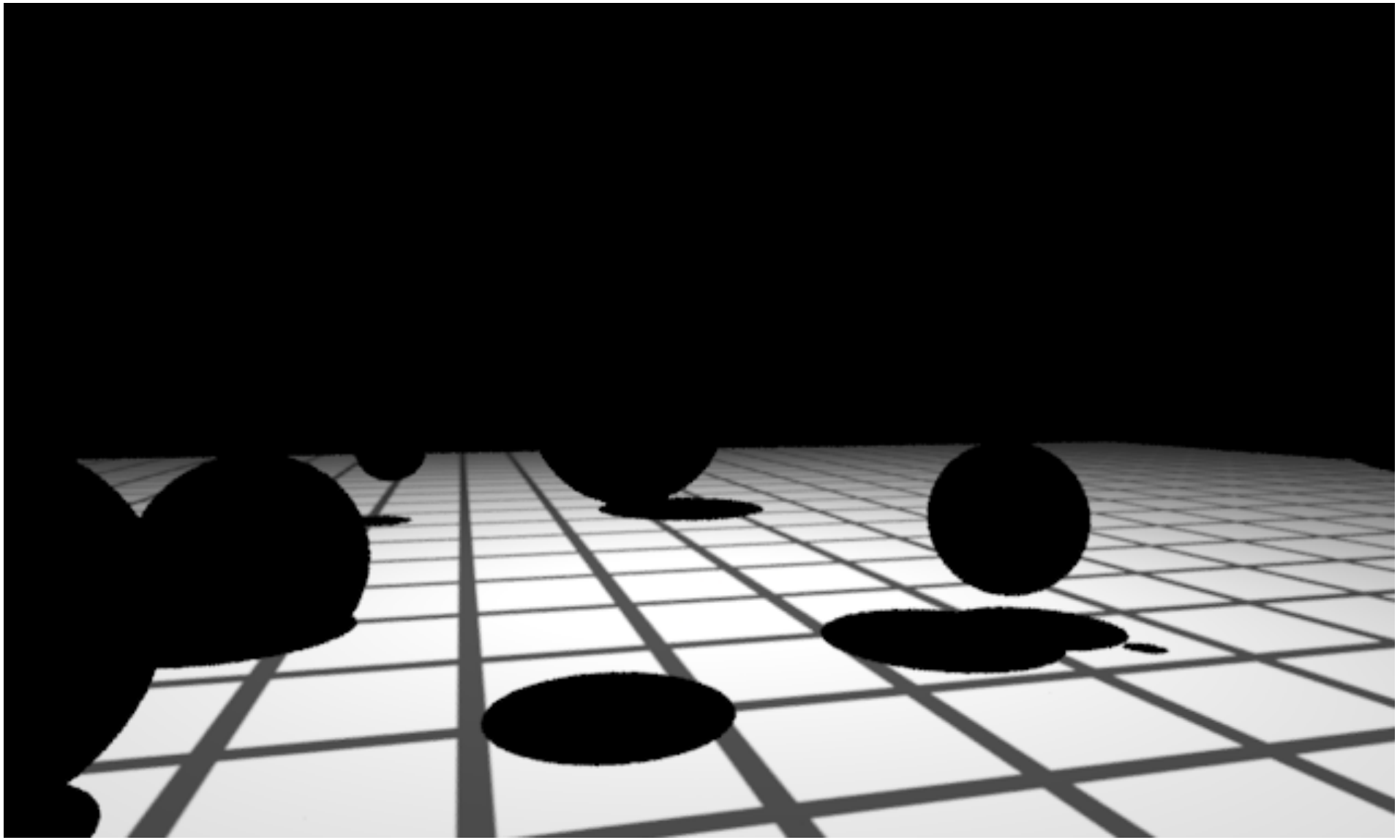
Spheres-over-plane.pbrt (mirror depth=3)



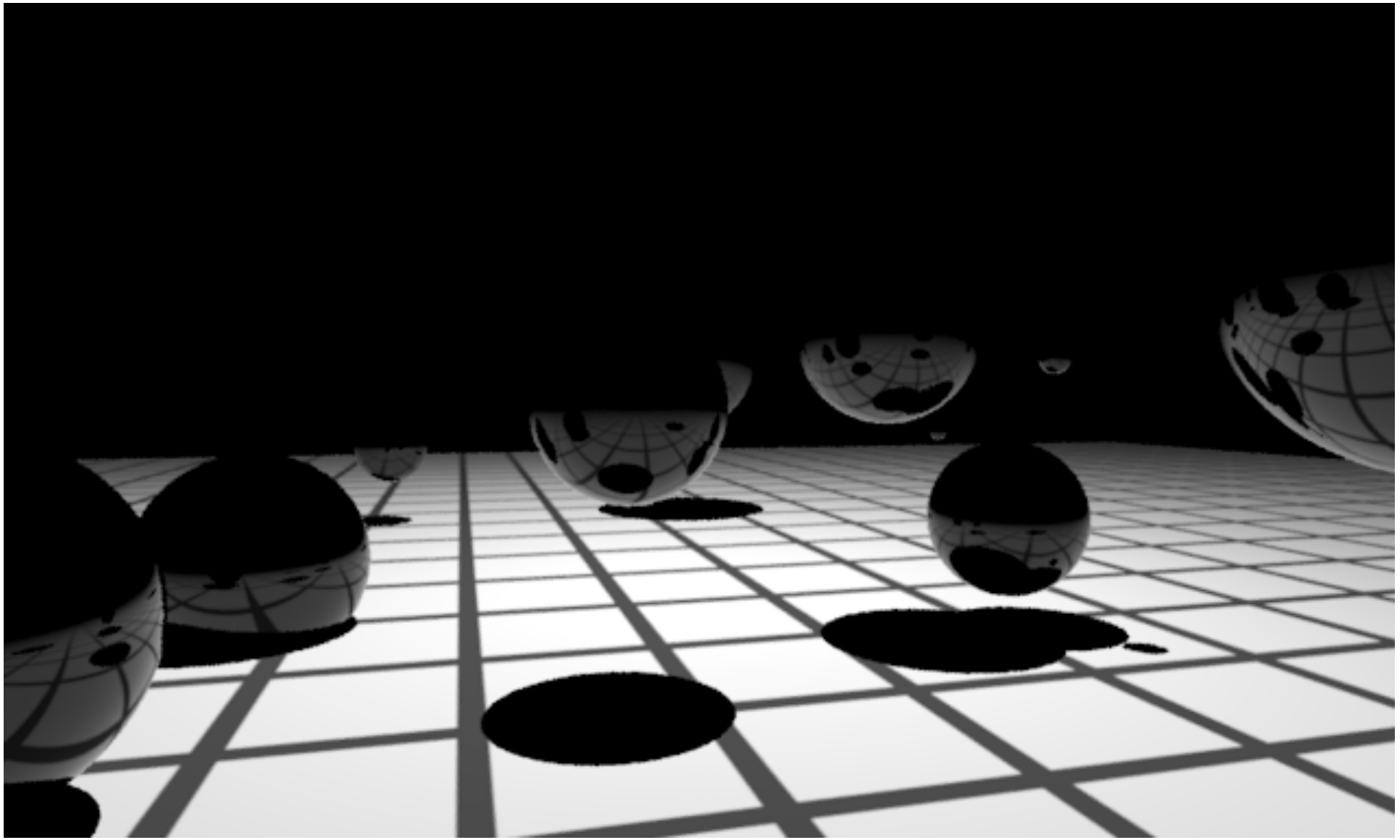
Spheres-over-plane.pbrt (mirror depth=10)



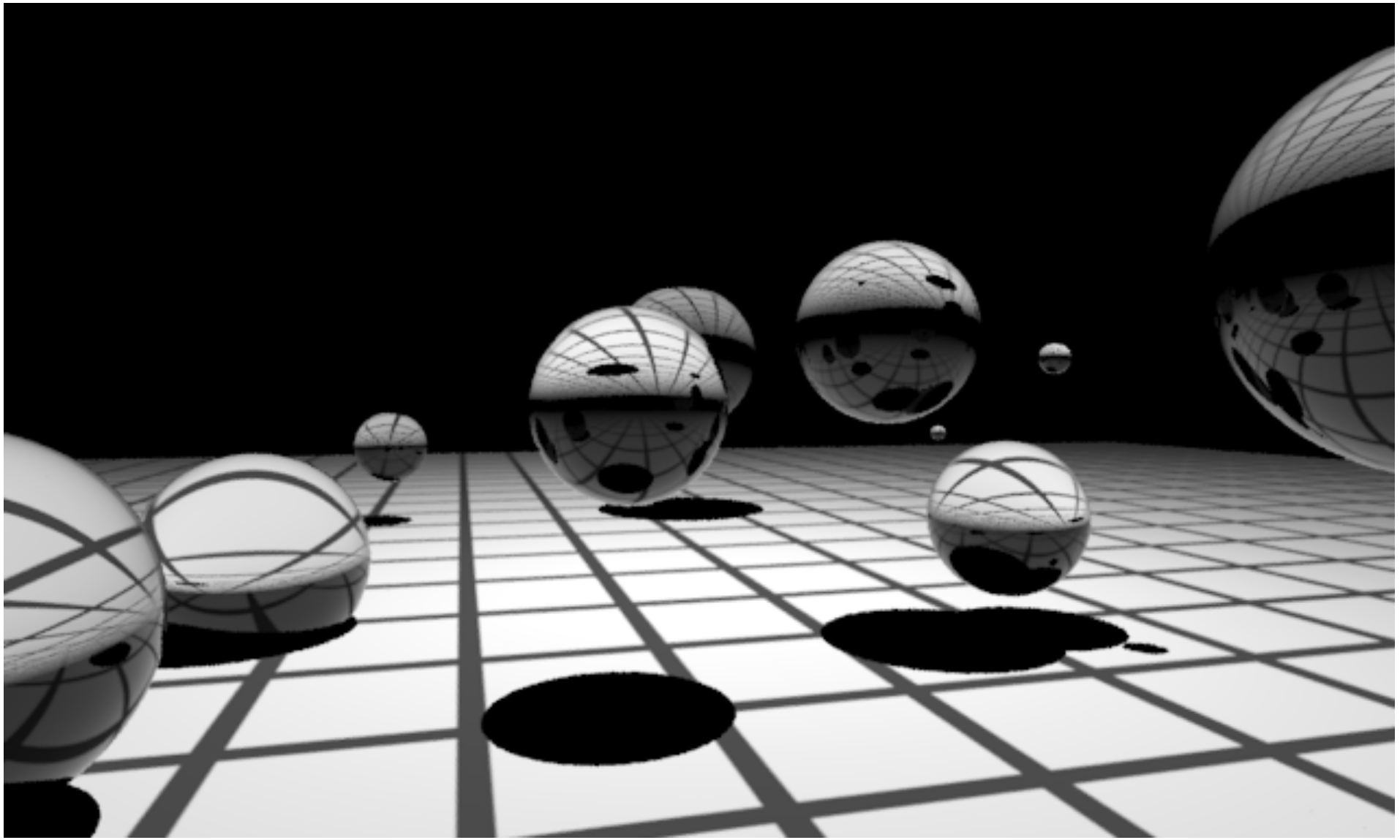
Spheres-over-plane.pbrt (glass depth=1)



Spheres-over-plane.pbrt (glass depth=2)



Spheres-over-plane.pbrt (glass depth=3)



Spheres-over-plane.pbrt (g/m depth=10)

Table 1.1: Main Interface Types. Most of `pbrt` is implemented in terms of 13 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/`	7.7
`Film`	`film/`	7.8
`Material`	`materials/`	9.2
`Texture`	`textures/`	10.3
`VolumeRegion`	`volumes/`	11.3
`Light`	`lights/`	12.1
`Renderer`	`renderers/`	1.3.3
`SurfaceIntegrator`	`integrators/`	Ch. 15 intro
`VolumeIntegrator`	`integrators/`	16.2

PBRT Render Loop

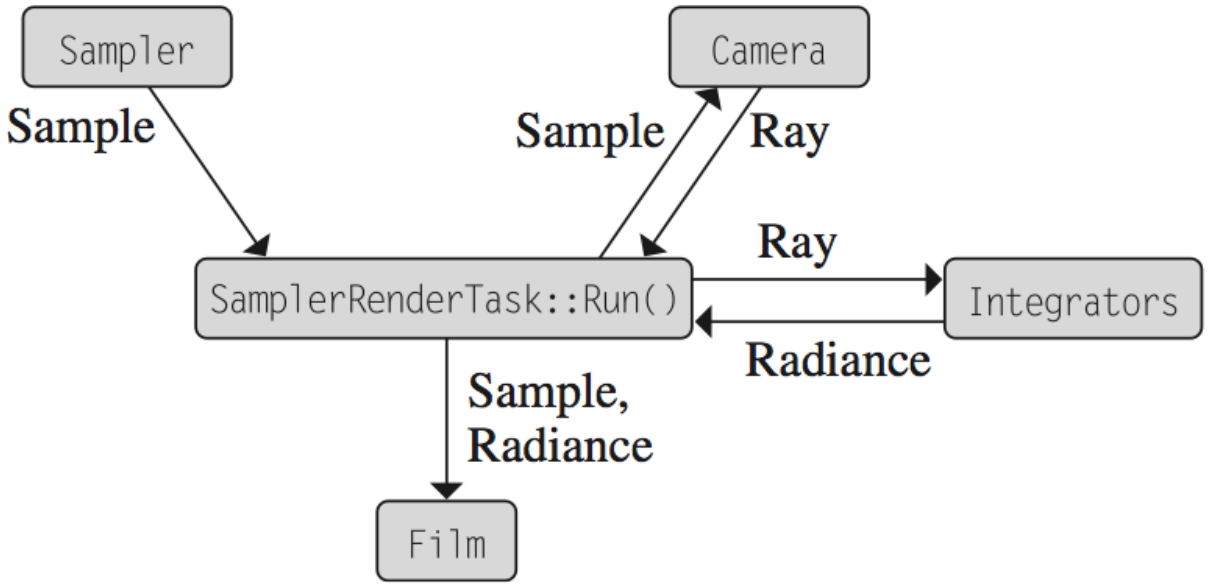
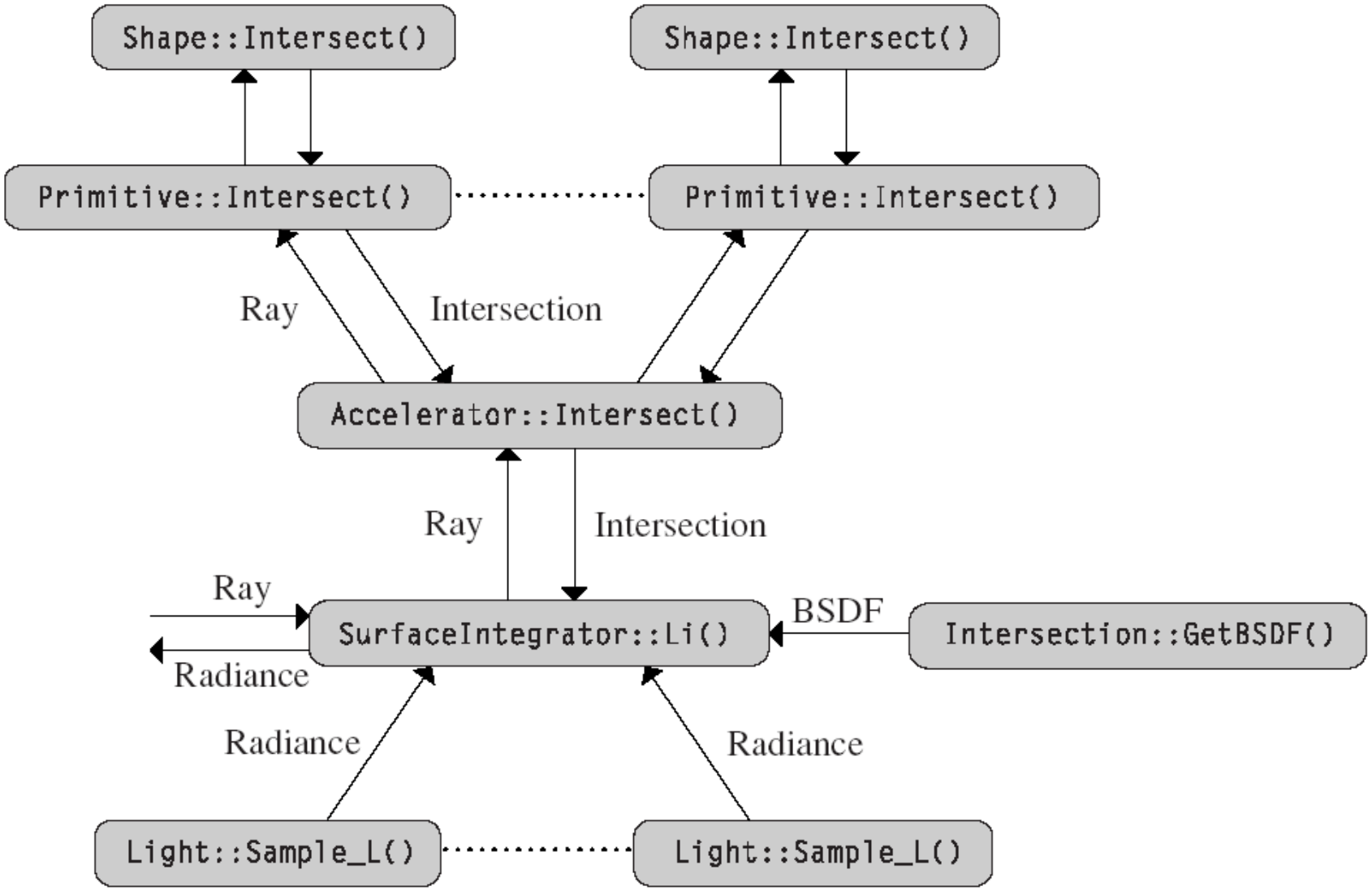


Figure 1.17: Class Relationships for the Main Rendering Loop in the `SamplerRenderer::Render()` Method in `renderers/sample.cpp`. The `Sampler` provides a sequence of sample values, one for each image sample to be taken. The `Camera` turns a sample into a corresponding ray from the film plane, and the `Integrators` compute 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.

PBRT Intersection Methods



PBRT Intergrater

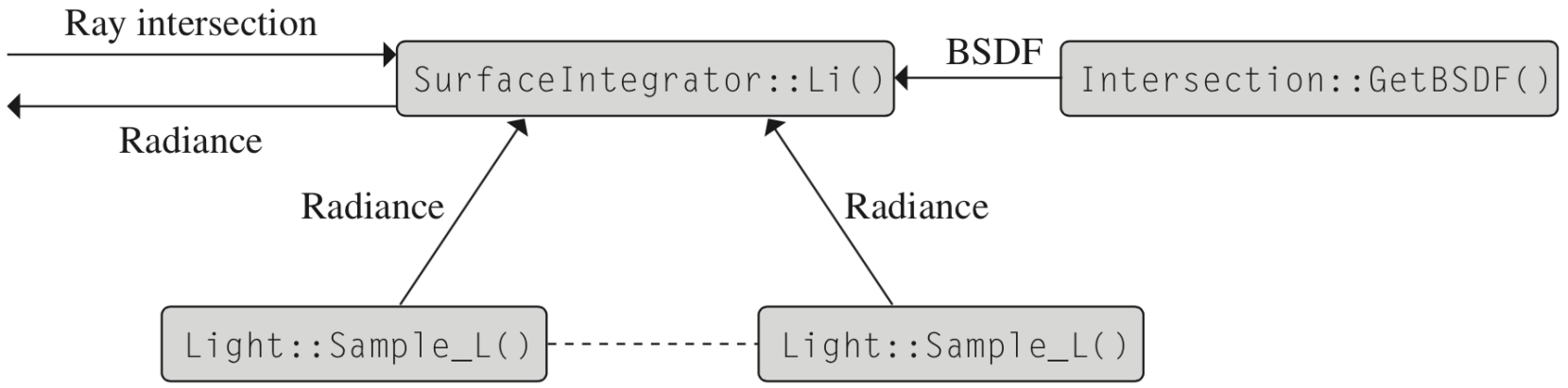


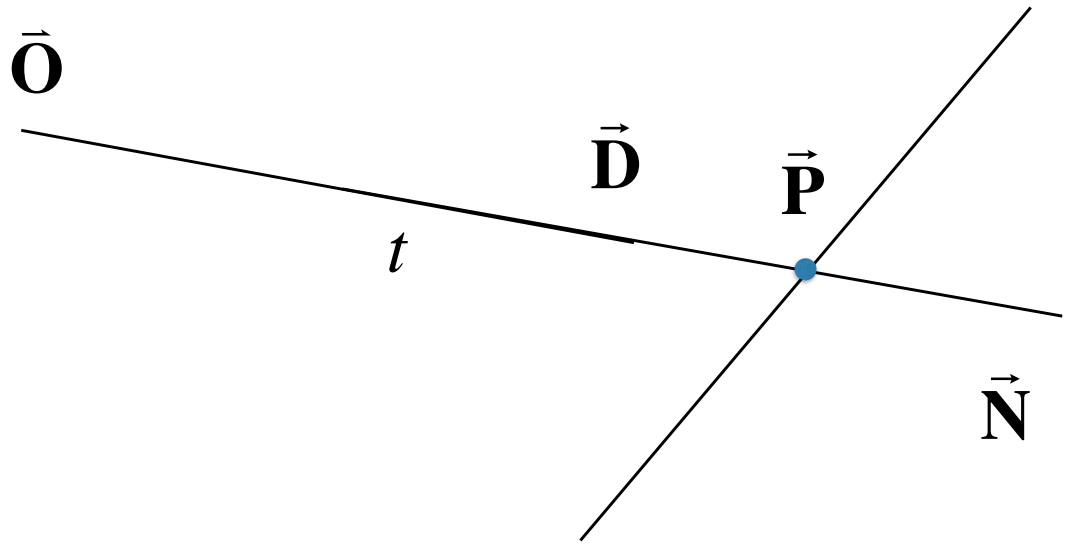
Figure 1.19: Class Relationships for Surface Integration. The main rendering loop passes a camera ray and information about its intersection point to the `SurfaceIntegrator`, which returns the radiance along that ray arriving at the film plane. The integrator finds the material properties at the intersection point in the form of a `BSDF` and 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.

Ray-Surface Intersection

Ray-Plane Intersection

Ray: $\vec{\mathbf{P}} = \vec{\mathbf{O}} + t\,\vec{\mathbf{D}}$

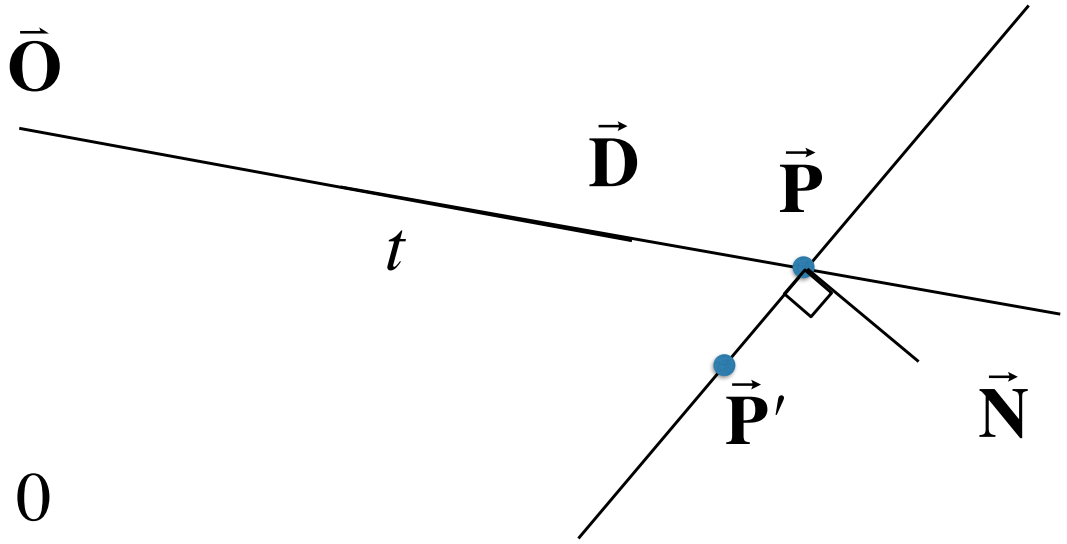
$0 \le t < \infty$



Ray-Plane Intersection

Ray: $\vec{\mathbf{P}} = \vec{\mathbf{O}} + t\,\vec{\mathbf{D}}$

$0 \le t < \infty$



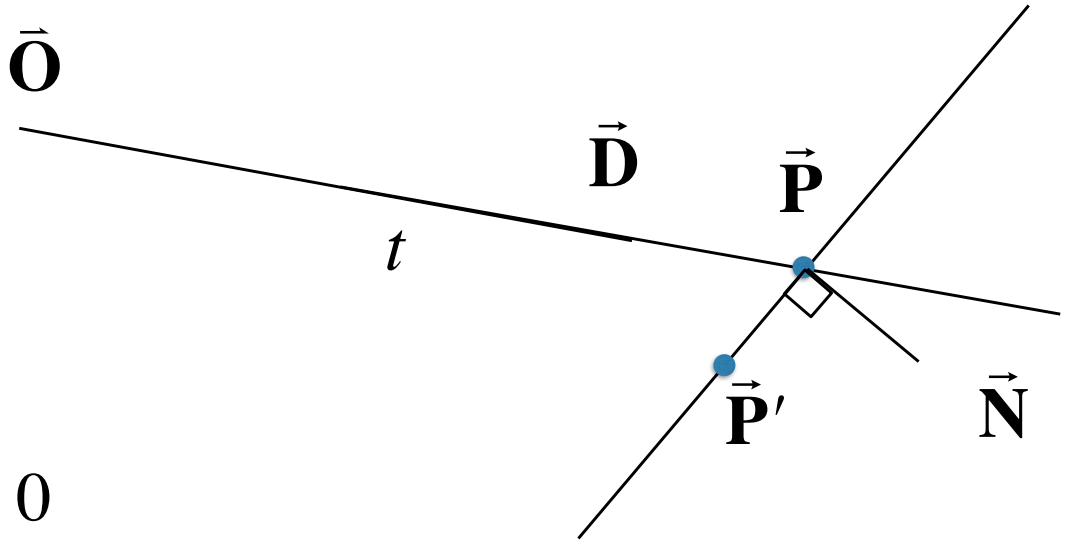
Plane: $(\vec{\mathbf{P}} - \vec{\mathbf{P}}') \bullet \vec{\mathbf{N}} = 0$

$ax + by + cz + d = 0$

Ray-Plane Intersection

Ray: $\vec{\mathbf{P}} = \vec{\mathbf{O}} + t\,\vec{\mathbf{D}}$

$0 \le t < \infty$



Plane: $(\vec{\mathbf{P}} - \vec{\mathbf{P}}') \bullet \vec{\mathbf{N}} = 0$

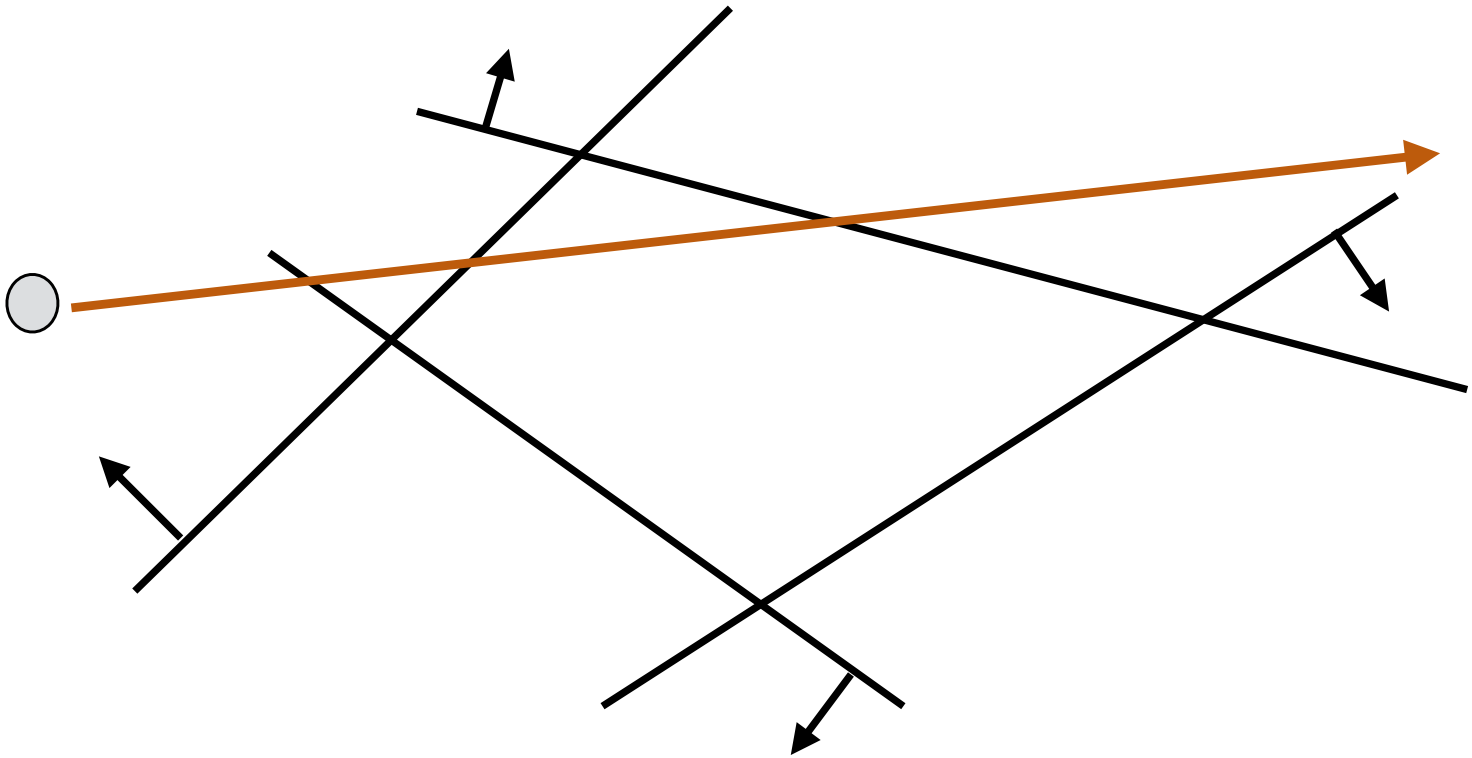
$ax + by + cz + d = 0$

Solve for intersection

$$(\vec{\mathbf{P}} - \vec{\mathbf{P}}') \bullet \vec{\mathbf{N}} = (\vec{\mathbf{O}} + t\vec{\mathbf{D}} - \vec{\mathbf{P}}') \bullet \vec{\mathbf{N}} = 0$$

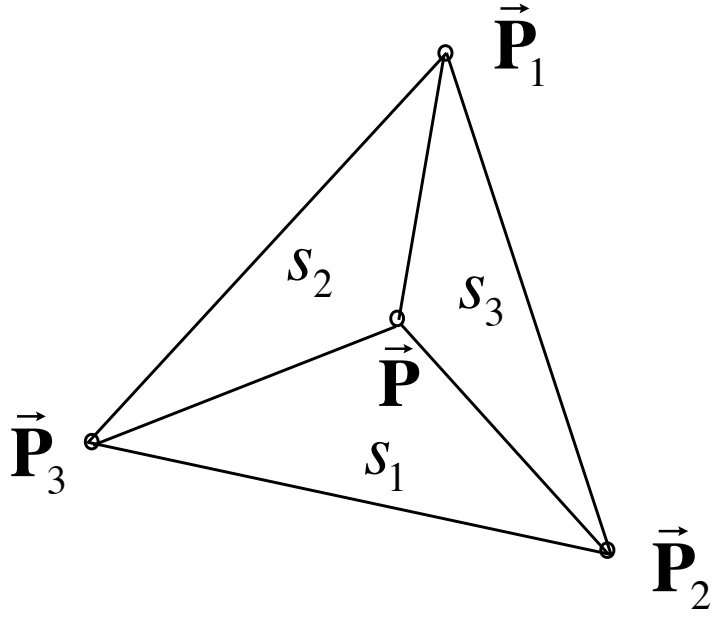
$$t = -\frac{(\vec{\mathbf{O}} - \vec{\mathbf{P}}') \bullet \vec{\mathbf{N}}}{\vec{\mathbf{D}} \bullet \vec{\mathbf{N}}}$$

Optimize Ray-Convex Polyhedra?



Polyhedra defined as the intersection of N half-planes

Triangles



Barycentric coordinates

$$\vec{\mathbf{P}} = s_1\vec{\mathbf{P}}_1 + s_2\vec{\mathbf{P}}_2 + s_3\vec{\mathbf{P}}_3$$

Inside triangle criteria

$$s_1 = \text{area}(\triangle \mathbf{P P_2 P_3}) / \text{area}(\triangle \mathbf{P_1 P_2 P_3})$$

$$s_2 = \text{area}(\triangle \mathbf{P_1 P P_3}) / \text{area}(\triangle \mathbf{P_1 P_2 P_3})$$

$$s_3 = \text{area}(\triangle \mathbf{P_1 P_2 P}) / \text{area}(\triangle \mathbf{P_1 P_2 P_3})$$

$$0 \le s_1 \le 1$$

$$0 \le s_2 \le 1$$

$$0 \le s_3 \le 1$$

$$s_1 + s_2 + s_3 = 1$$

Triangle Inside Test

Test whether a point is inside the triangle

$$\vec{\mathbf{P}} = s_1\vec{\mathbf{P}}_1 + s_2\vec{\mathbf{P}}_2 + s_3\vec{\mathbf{P}}_3$$

$$\begin{bmatrix}\mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3\end{bmatrix}\begin{bmatrix}s_1\\ s_2\\ s_3\end{bmatrix} = \begin{bmatrix}\mathbf{P}\end{bmatrix}$$

$$\begin{bmatrix}s_1\\ s_2\\ s_3\end{bmatrix} = \begin{bmatrix}\mathbf{P}_1 & \mathbf{P}_2 & \mathbf{P}_3\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{P}\end{bmatrix}$$

Moller-Trumbore Algorithm

$$\vec{\mathbf{O}} + t\vec{\mathbf{D}} = (1 - b_1 - b_2)\vec{\mathbf{P}}_0 + b_1\vec{\mathbf{P}}_1 + b_2\vec{\mathbf{P}}_2$$

Where:

$$\begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \frac{1}{\vec{\mathbf{S}}_1 \bullet \vec{\mathbf{E}}_1} \begin{bmatrix} \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{E}}_2 \\ \vec{\mathbf{S}}_1 \bullet \vec{\mathbf{S}} \\ \vec{\mathbf{S}}_2 \bullet \vec{\mathbf{D}} \end{bmatrix}$$

$$\vec{\mathbf{E}}_1 = \vec{\mathbf{P}}_1 - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{E}}_2 = \vec{\mathbf{P}}_2 - \vec{\mathbf{P}}_0$$

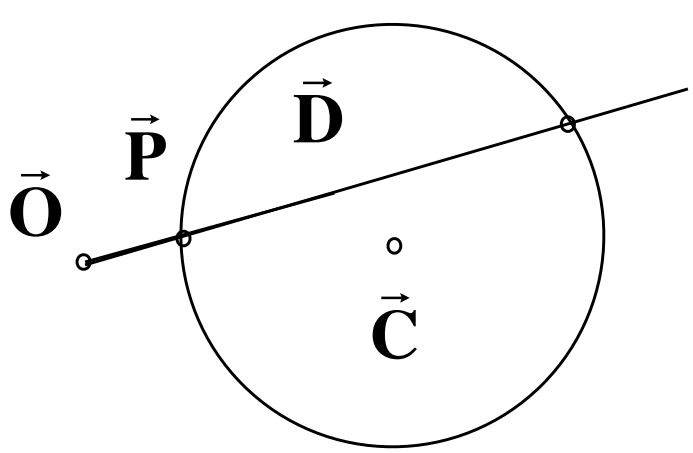
$$\vec{\mathbf{S}} = \vec{\mathbf{O}} - \vec{\mathbf{P}}_0$$

$$\vec{\mathbf{S}}_1 = \vec{\mathbf{D}} \times \vec{\mathbf{E}}_2$$

$$\vec{\mathbf{S}}_2 = \vec{\mathbf{S}} \times \vec{\mathbf{E}}_1$$

Cost = (1 div, 27 mul, 17 add)

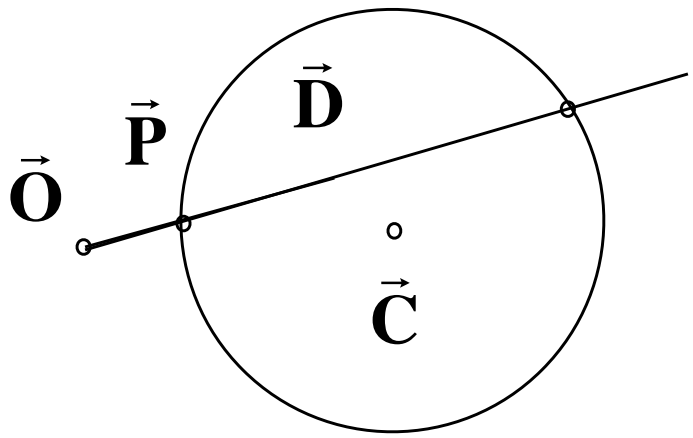
Ray-Sphere Intersection



Ray: $\vec{P} = \vec{O} + t\vec{D}$

Sphere: $\left(\vec{P} - \vec{C}\right)^2 - R^2 = 0$

Ray-Sphere Intersection



Ray: $\vec{\mathbf{P}} = \vec{\mathbf{O}} + t\vec{\mathbf{D}}$

Sphere: $\left(\vec{\mathbf{P}} - \vec{\mathbf{C}}\right)^2 - R^2 = 0$

$$\left(\vec{\mathbf{O}} + t\,\vec{\mathbf{D}} - \vec{\mathbf{C}}\right)^2 - R^2 = 0$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

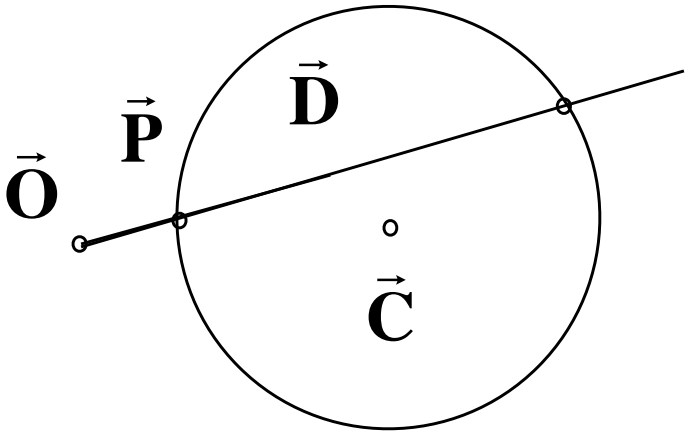
$$at^2 + bt + c = 0$$

$$a = \vec{\mathbf{D}}^2$$

$$b = 2(\vec{\mathbf{O}} - \vec{\mathbf{C}}) \bullet \vec{\mathbf{D}}$$

$$c = (\vec{\mathbf{O}} - \vec{\mathbf{C}})^2 - R^2$$

Ray-Sphere Intersection

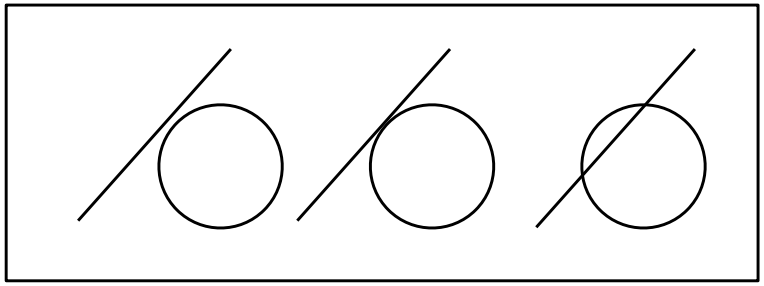


Ray: $\vec{\mathbf{P}} = \vec{\mathbf{O}} + t\vec{\mathbf{D}}$

Sphere: $\left(\vec{\mathbf{P}} - \vec{\mathbf{C}}\right)^2 - R^2 = 0$

$$\left(\vec{\mathbf{O}} + t\,\vec{\mathbf{D}} - \vec{\mathbf{C}}\right)^2 - R^2 = 0$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$



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

$$a = \vec{\mathbf{D}}^2$$

$$b = 2(\vec{\mathbf{O}} - \vec{\mathbf{C}}) \bullet \vec{\mathbf{D}}$$

$$c = (\vec{\mathbf{O}} - \vec{\mathbf{C}})^2 - R^2$$

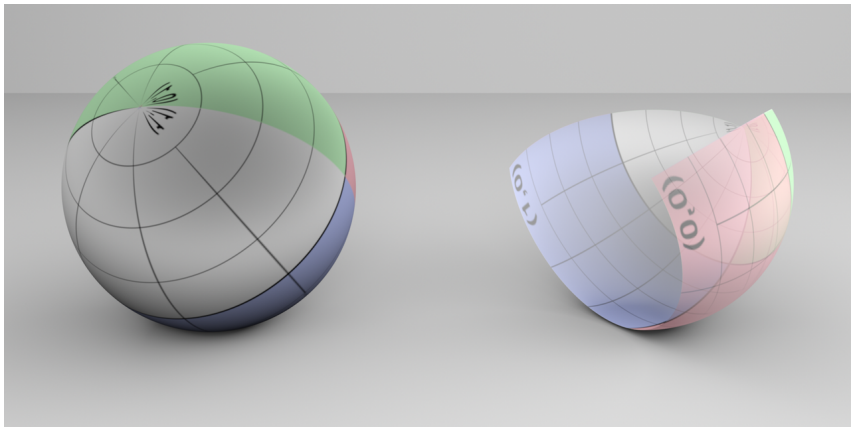
Shapes

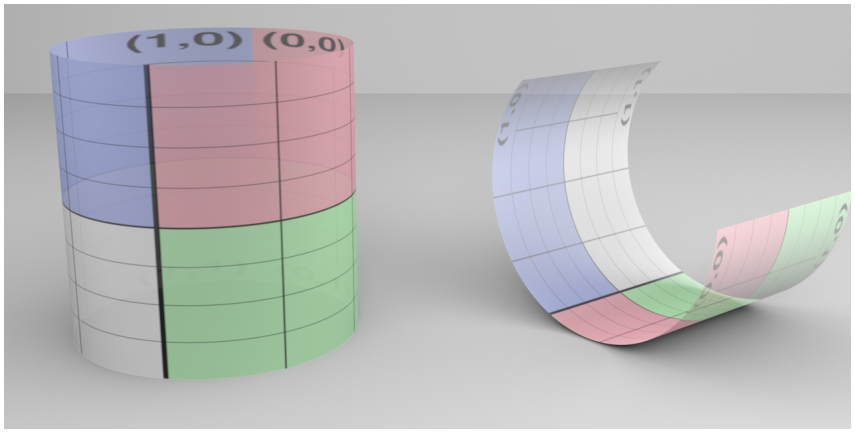
Shapes in pbrt

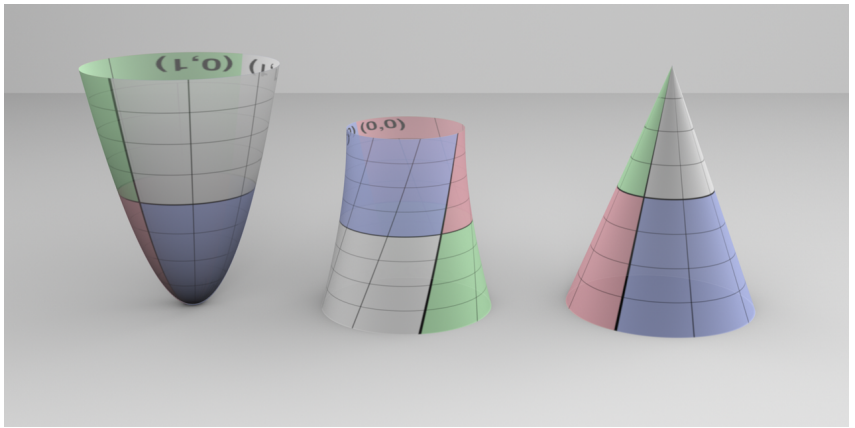
```
class Shape {
public:
    // Shape interface
    virtual BBox ObjectBound() const = 0;
    virtual BBox WorldBound() const;
    virtual bool CanIntersect() const;
    virtual void Refine(vector<Reference<Shape>> &refined) const;
    virtual bool Intersect(const Ray &ray, float *tHit,
                           float *rayEpsilon, DifferentialGeometry *dg) const;
    virtual bool IntersectP(const Ray &ray) const;
    virtual void GetShadingGeometry(const Transform &obj2world,
                                    const DifferentialGeometry &dg,
                                    DifferentialGeometry *dgShading) const {
        *dgShading = dg;
    }
    virtual float Area() const;

    // Shape Public Data
    const Transform *ObjectToWorld, *WorldToObject;
};
```

Quadrics



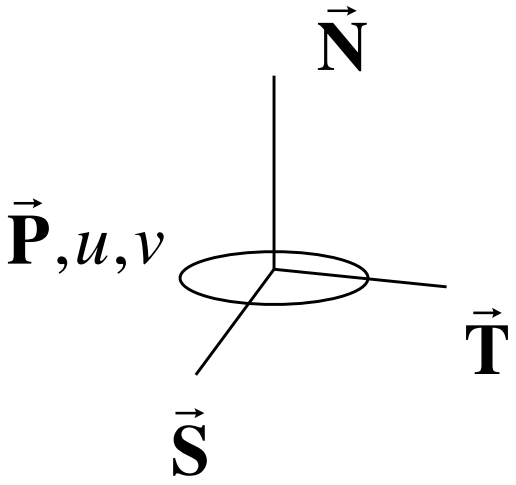




Geometric Methods: Normals

e.g. Sphere

$$\vec{\mathbf{N}} = \vec{\mathbf{P}} - \vec{\mathbf{C}}$$



$$x = \sin\theta \cos\phi$$
$$y = \sin\theta \sin\phi$$
$$z = \cos\theta$$
$$\vec{\mathbf{P}} = (x, y, z)$$

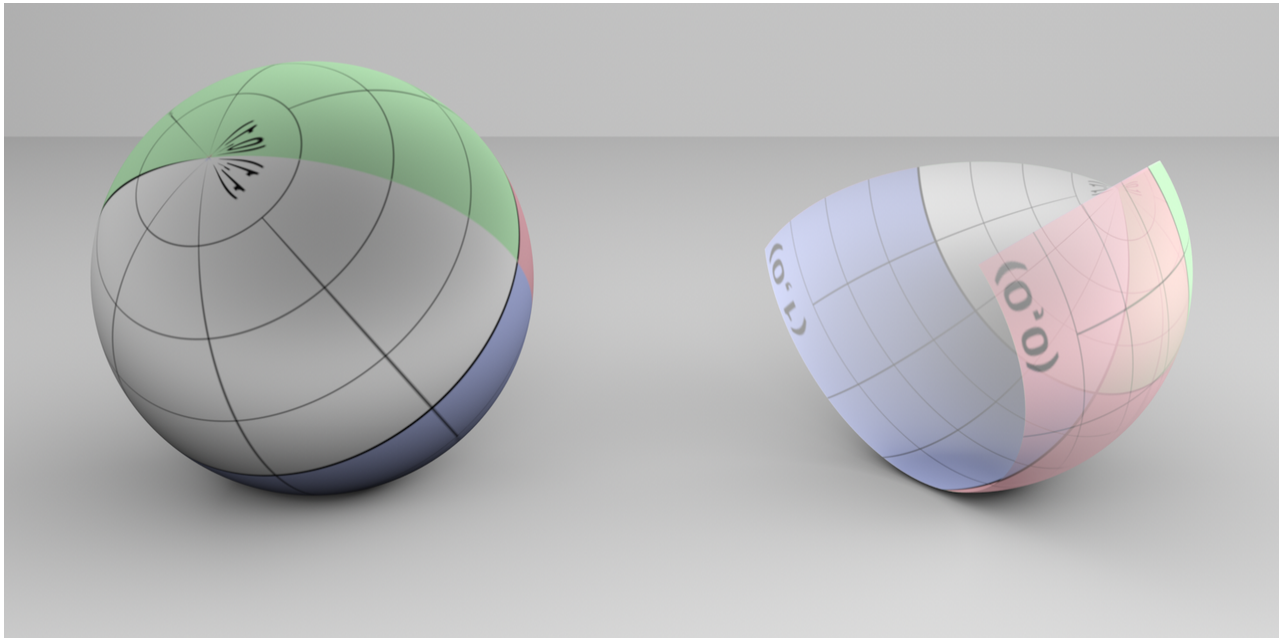
$$\frac{\partial \vec{\mathbf{P}}}{\partial \theta} = (\cos\theta \cos\phi, \cos\theta \sin\phi, -\sin\theta)$$

$$\frac{\partial \vec{\mathbf{P}}}{\partial \phi} = (-\sin\theta \sin\phi, \sin\theta \cos\phi, 0)$$

$$\vec{\mathbf{N}} = \frac{\partial \vec{\mathbf{P}}}{\partial \theta} \times \frac{\partial \vec{\mathbf{P}}}{\partial \phi}$$

Geometric Methods: Parameters

e.g. Sphere



$$x = \sin\theta\cos\phi$$
$$y = \sin\theta\sin\phi$$
$$z = \cos\theta$$

$$\phi = \tan^{-1}(x, y)$$
$$\theta = \cos^{-1} z$$

$$\theta = \theta_{\min} + v(\theta_{\max} - \theta_{\min})$$
$$\phi = u\phi_{\max}$$

$$v = (\theta - \theta_{\min})/(\theta_{\max} - \theta_{\min})$$
$$u = \phi/\phi_{\max}$$

Differential Geometry Representation

```
struct DifferentialGeometry {
    // Filled in by Shape::Intersect()
    Point p;
    Normal nn;
    float u, v;
    const Shape *shape;
    Vector dpdu, dpdv;
    Normal dndu, dndv;
};
```

Ray-Implicit Surface Intersection

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

$$x = x_0 + x_1 t$$

$$y = y_0 + y_1 t$$

$$z = z_0 + z_1 t$$

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

1. Substitute ray equation
2. Find positive, real roots

Univariate root finding

- Newton's method
- Regula-falsi
- Interval methods
- ...

Ray-Algebraic Surface Intersection

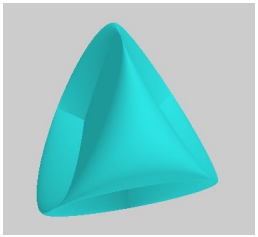
$$p_n(x,y,z) = 0$$

$$x = x_0 + x_1 t$$

$$y = y_0 + y_1 t$$

$$z = z_0 + z_1 t$$

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



Degree n

- **Linear: Plane**
- **Quadric: Spheres, ...**
- **Quartic: Tori**

Polynomial root finding

- **Quadratic, cubic, quartic**
- **...**