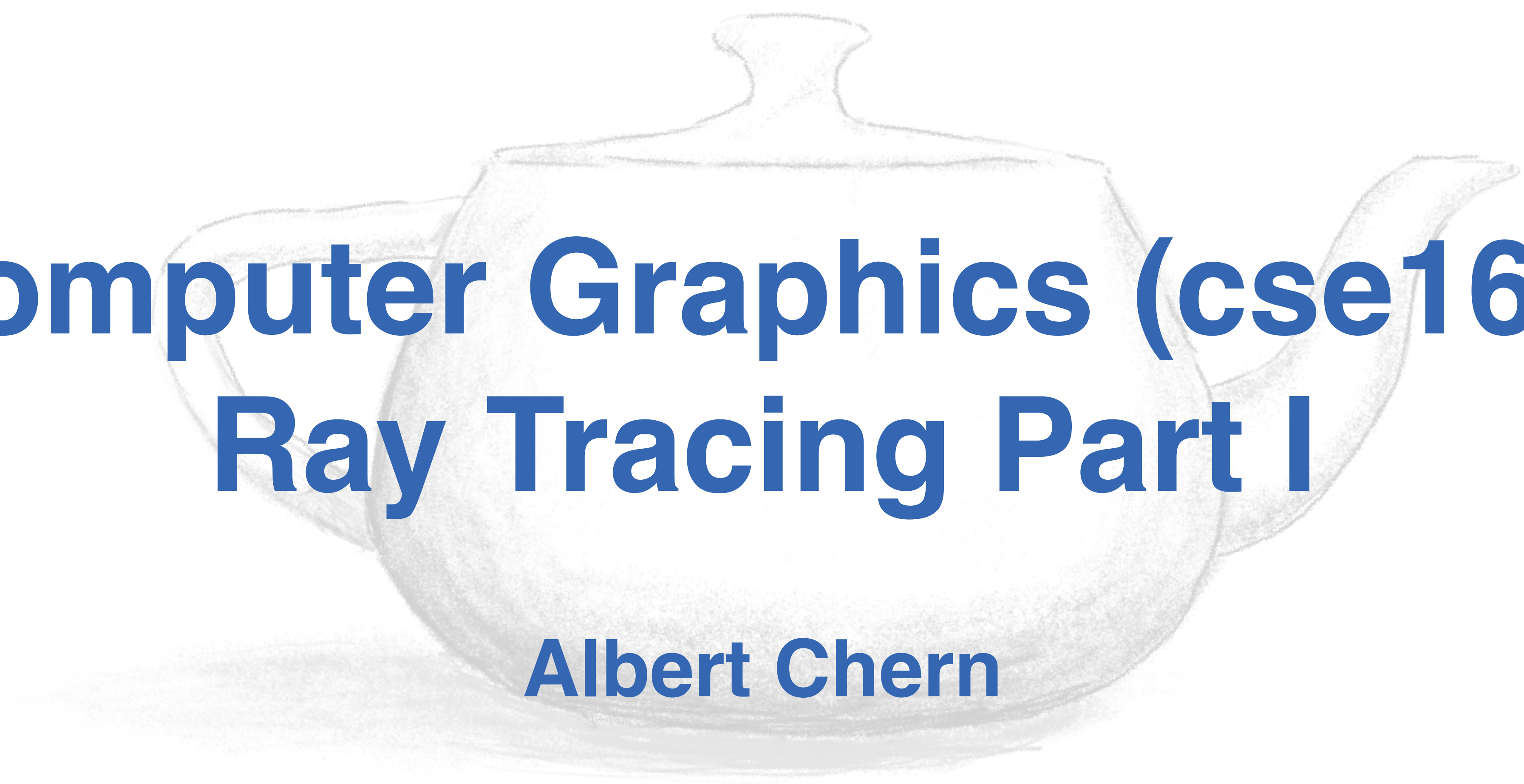




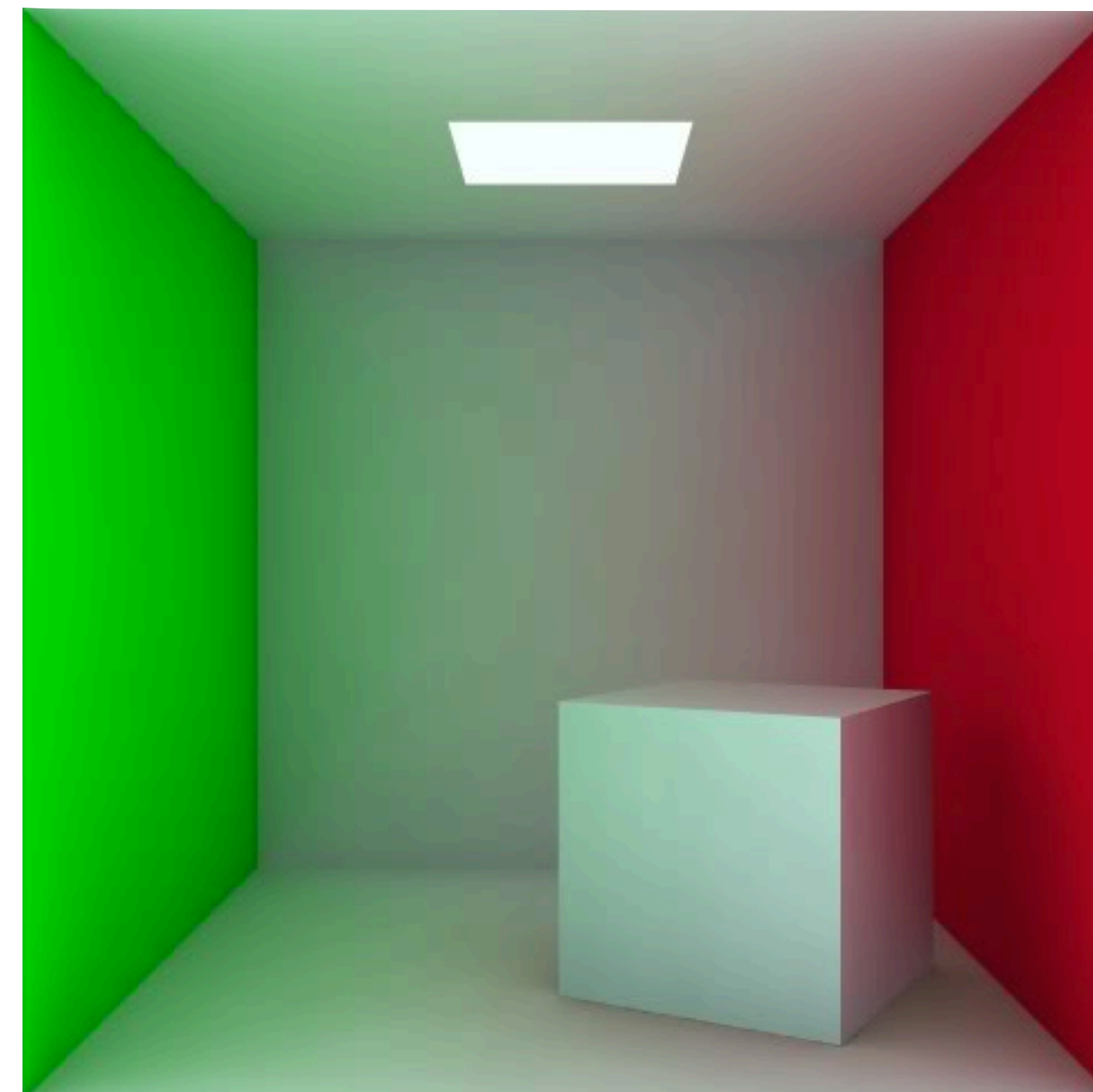
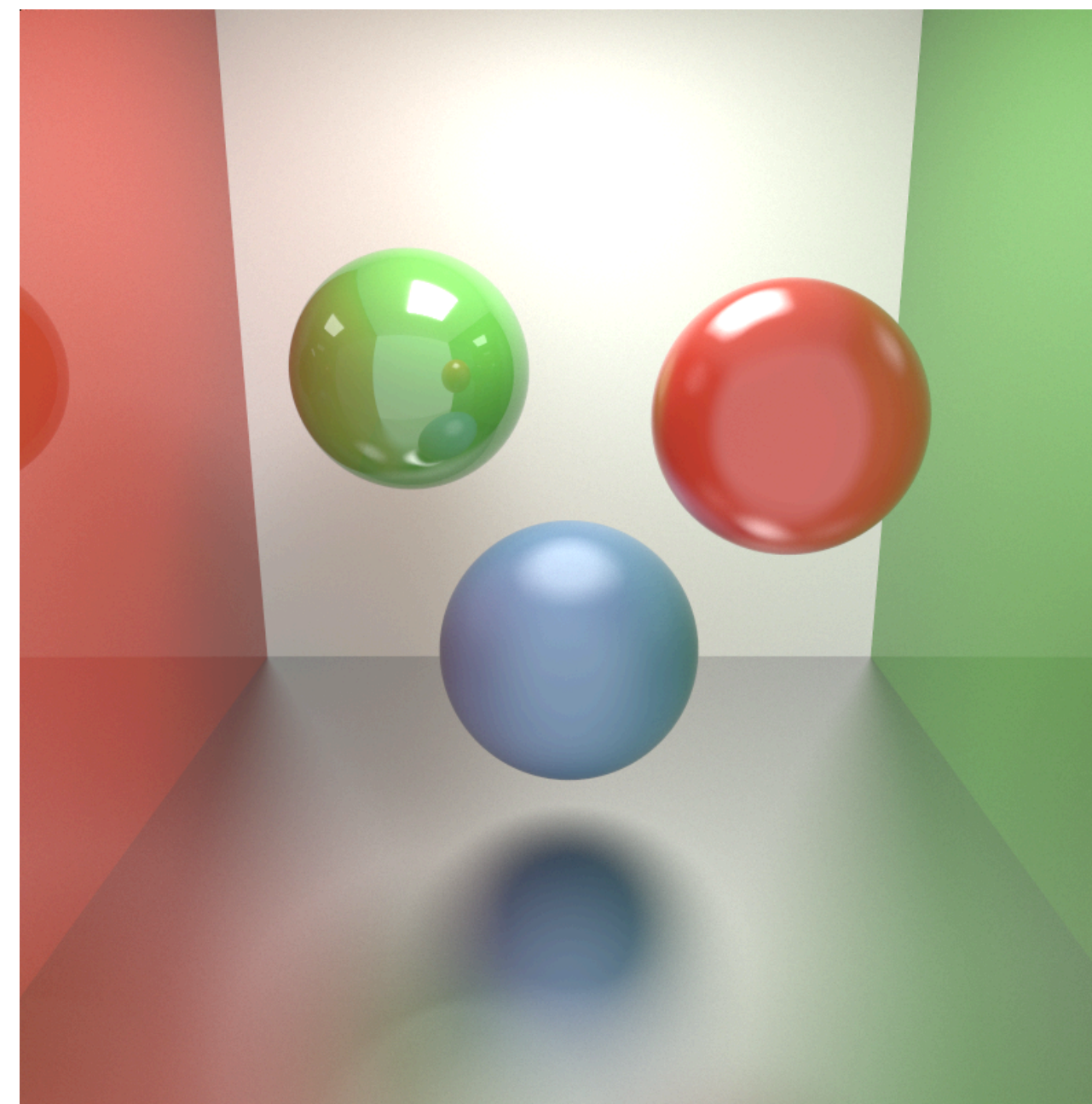
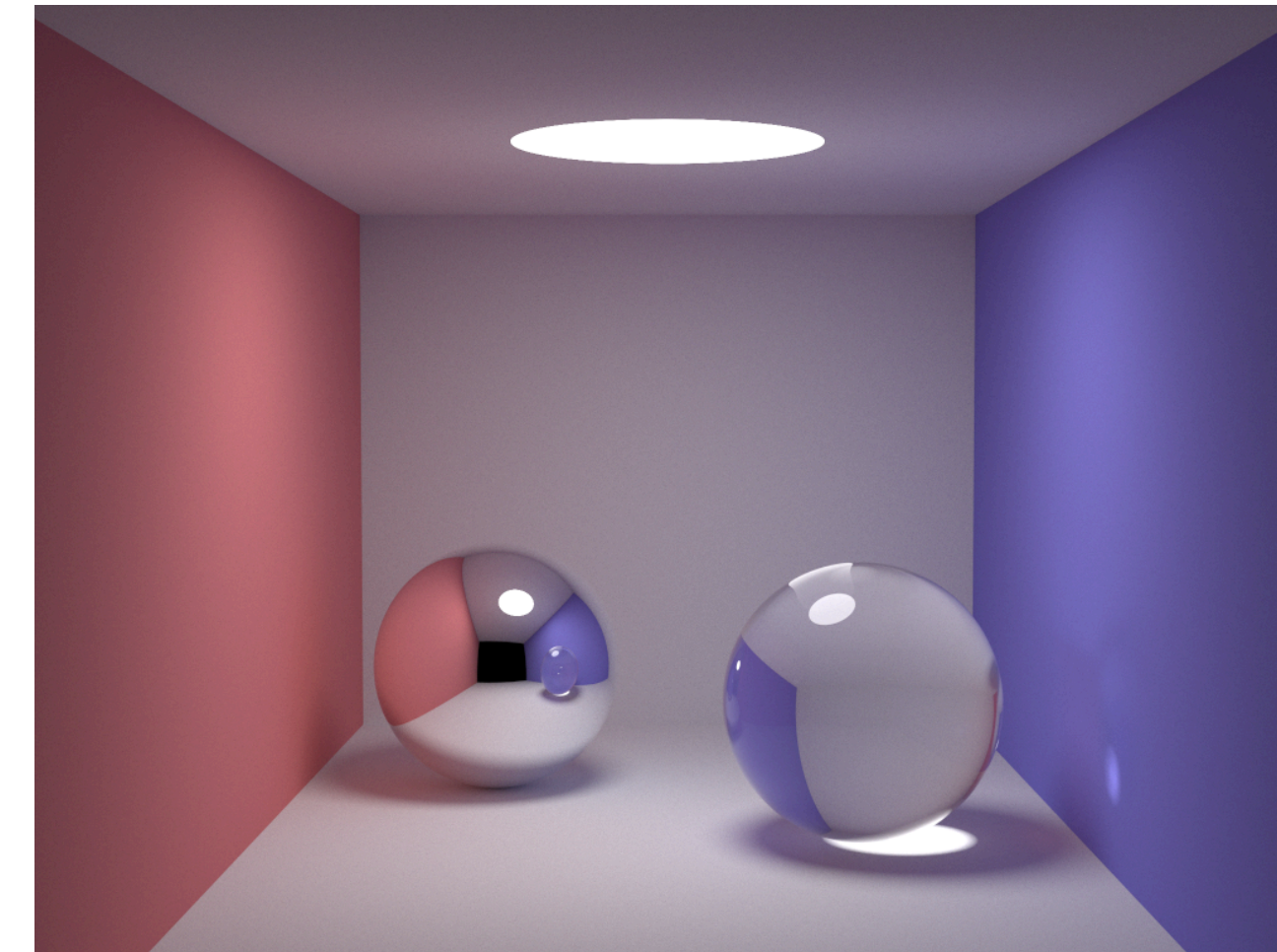
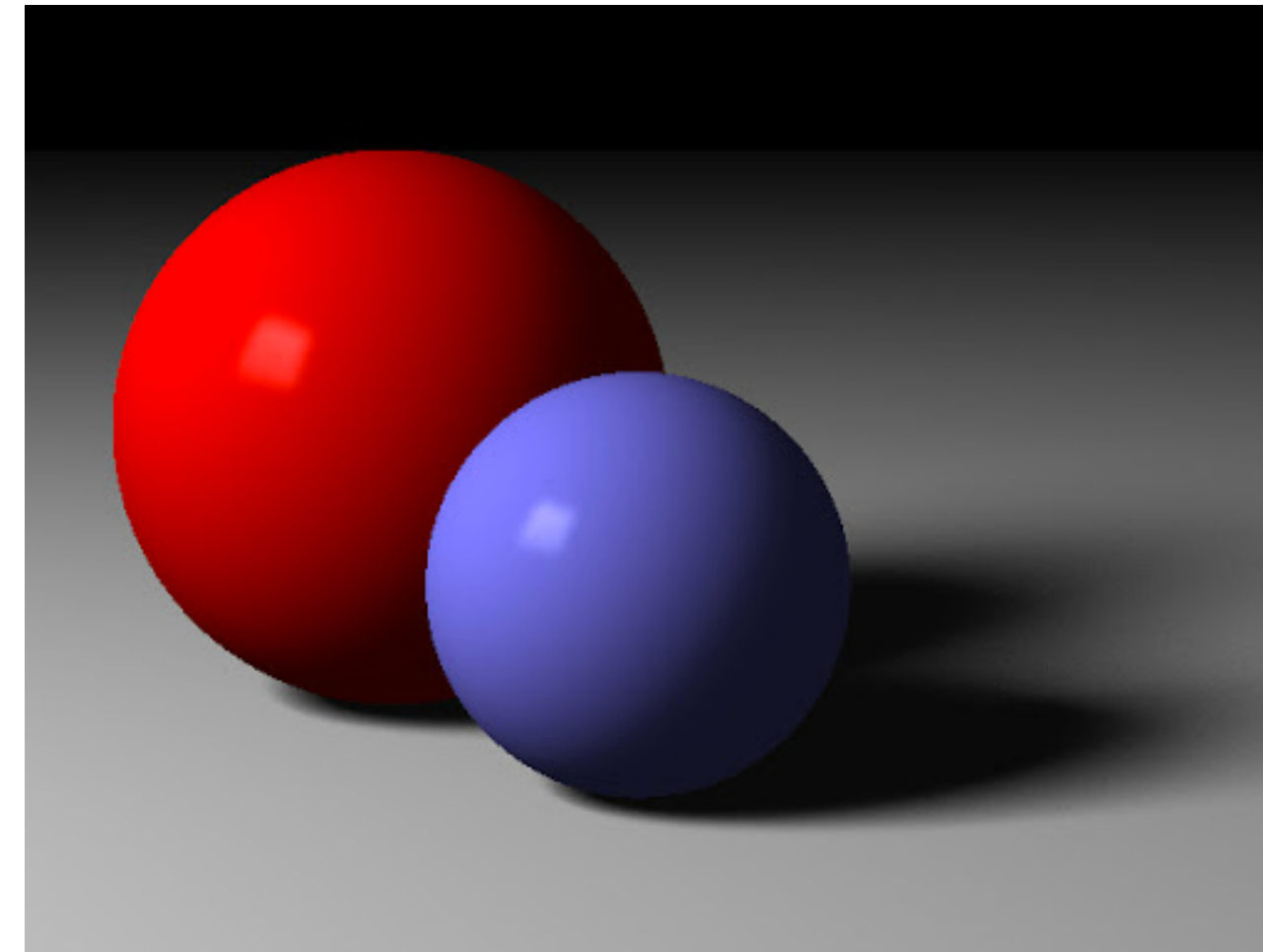
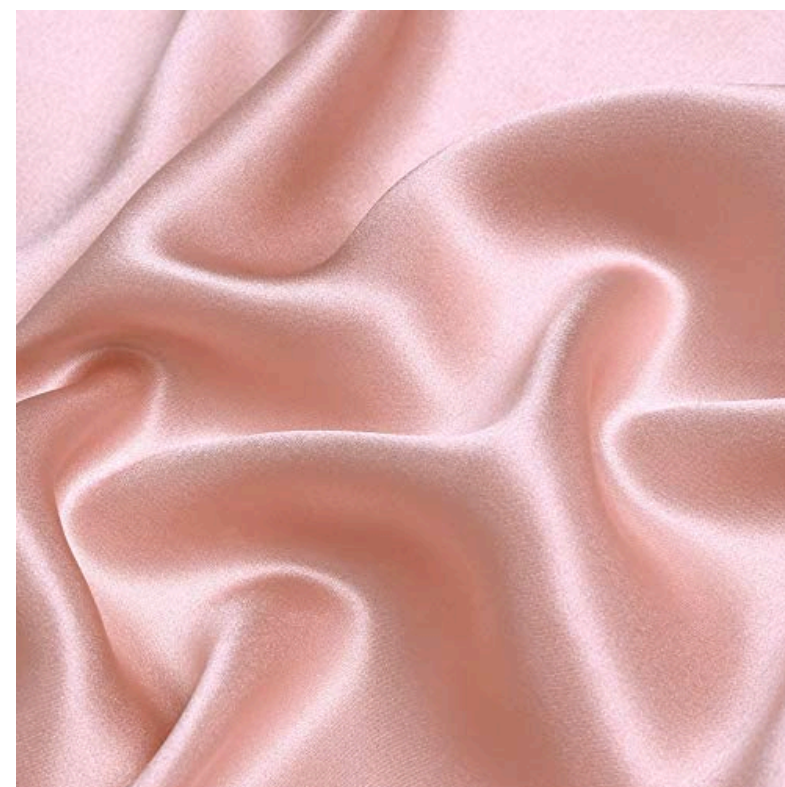
Computer Graphics (cse167)

Ray Tracing Part I

Albert Chern

Rendering

- Effects for realistic images
 - ▶ (Soft) shadows
 - ▶ Reflections (mirror and glossy)
 - ▶ Transparent (water, glass)
 - ▶ Inter-reflections (color bleeding)
 - ▶ Realistic materials
- Difficult in OpenGL pipeline
- Easy in raytracing framework



Outline

- Basic ray casting
- Shadow and reflection
- Ray-surface intersection

Rasterization v.s. Ray Casting

- Two different approaches to turn 3D scenes into pixels

- ▶ Rasterization

For each geometry in the scene

For each pixel from the screen

IntersectionTest(geometry, pixel)

Store fragments into a Z-buffer

EndFor

EndFor

Keep the nearest fragments in each pixel

Shade each pixel

- ▶ Ray casting

For each pixel from the screen

For each geometry in the scene

IntersectionTest(geometry, pixel)

Keep the nearest intersection

EndFor

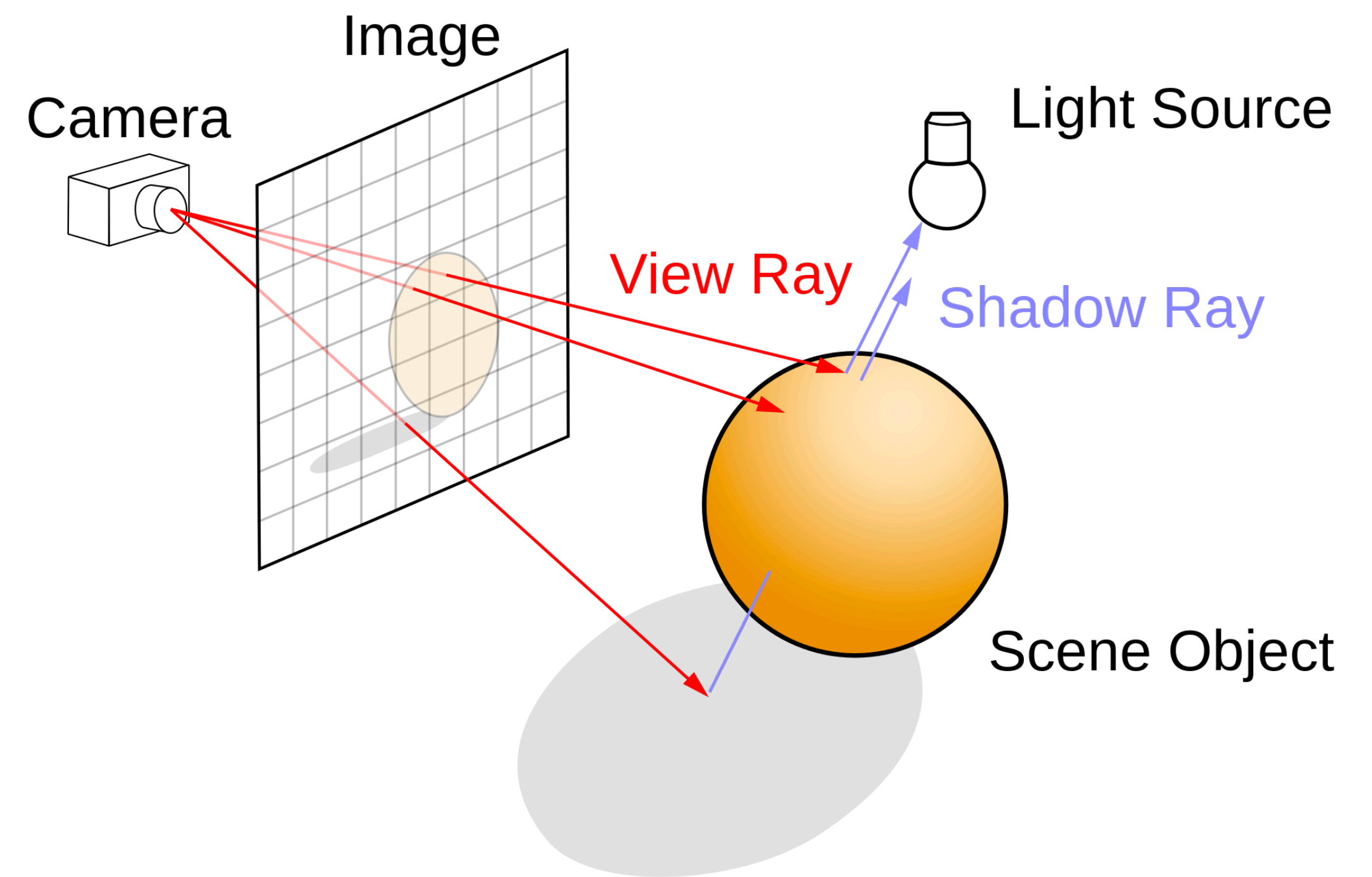
Shade the pixel

EndFor

- These two approaches give the same images as long as the shading models are the same

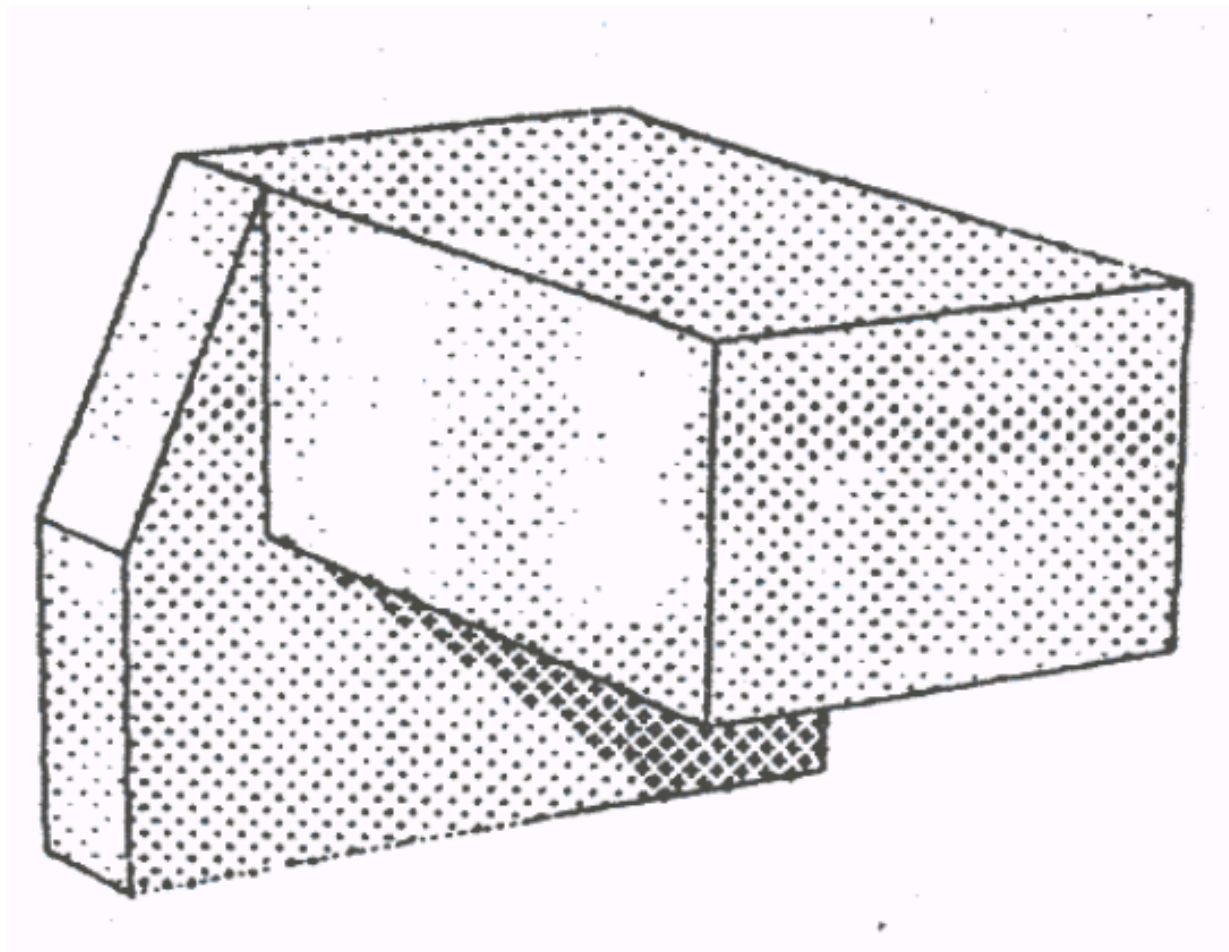
Rasterization v.s. Ray Casting

- These two approaches give the same images as long as the shading models are the same
- Shadows etc are much easier in ray casting
- OpenGL is a rasterizer. Rasterizer pipeline is fixed, easy to optimize in hardware for realtime rendering

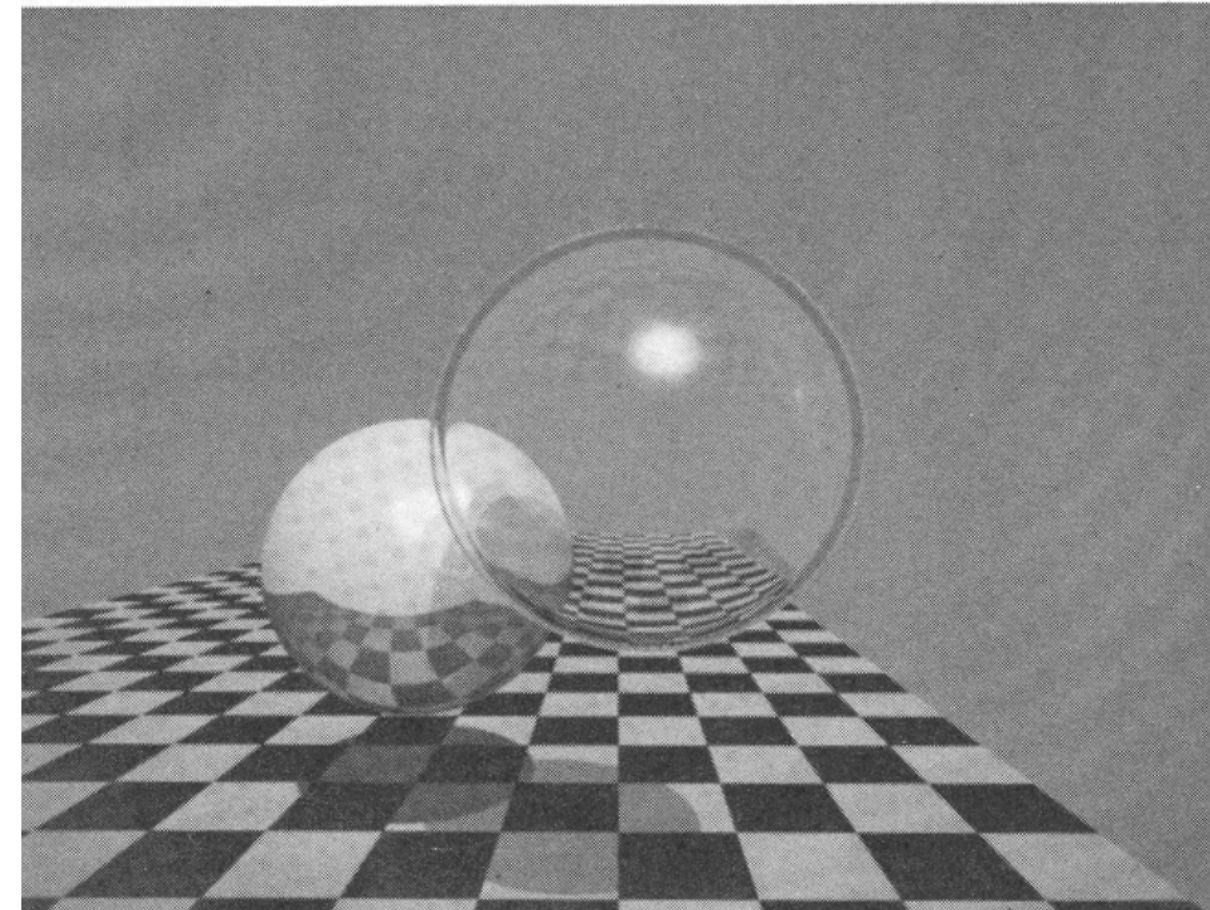


History of ray casting/tracing

- Appel 1968



- Whitted 1980 recursive ray tracing



- Lots of work on photorealism, accelerations.



- Real time ray tracing

- ▶ 2009 Nvidia OptiX API
- ▶ 2020 PlayStation5, Xbox series X and S

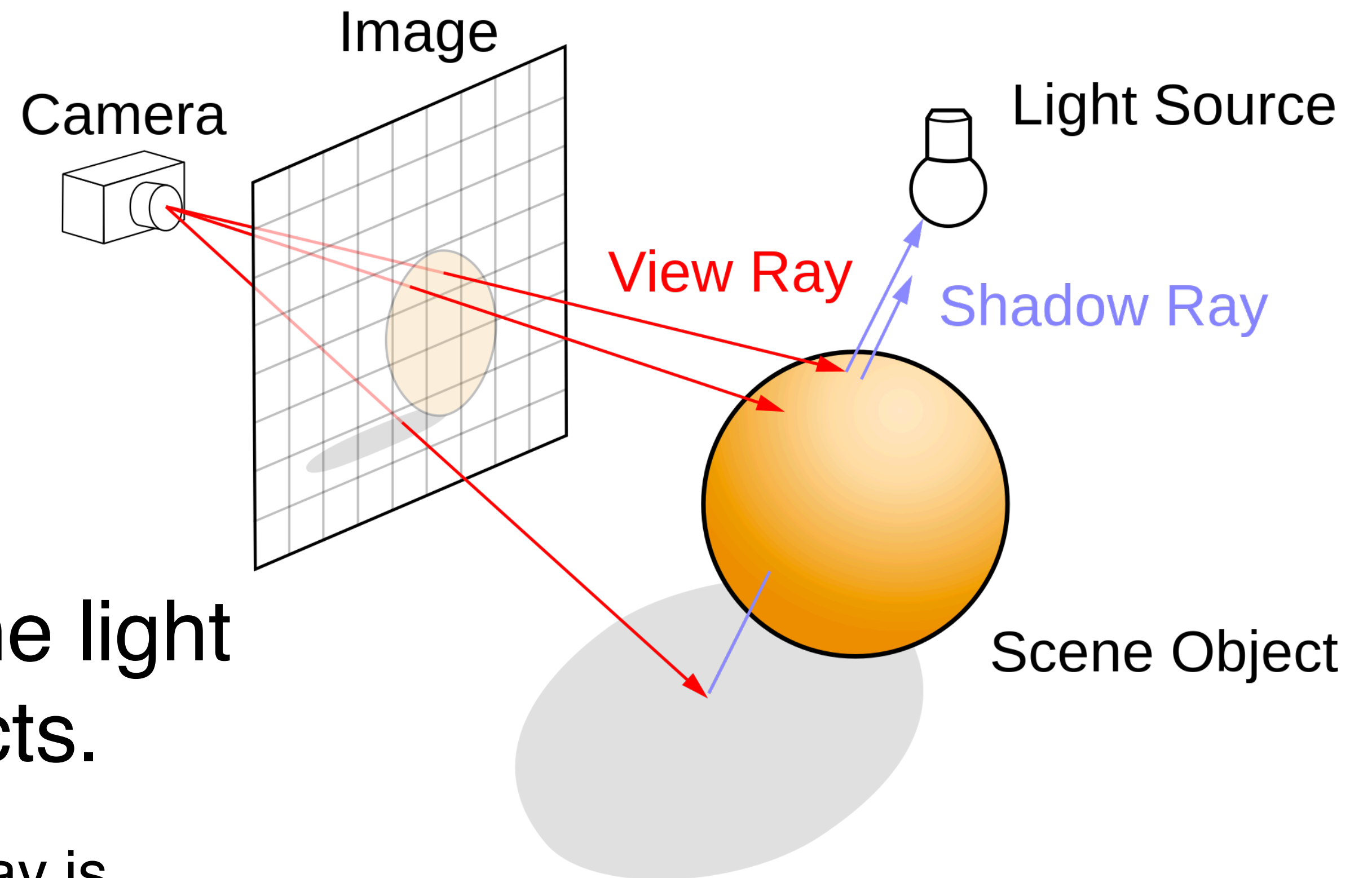


Outline code

```
Image Raytrace(Camera cam, Scene scene, int width, int height){
    Image image = new Image(width,height);
    for (int i=0; i<height; i++)
        for (int j=0; j<width; j++){
            Ray ray = RayThruPixel( cam, i, j );
            Intersection hit = Intersect( ray, scene );
            image[i][j] = FindColor( hit );
        }
    return image;
}
```

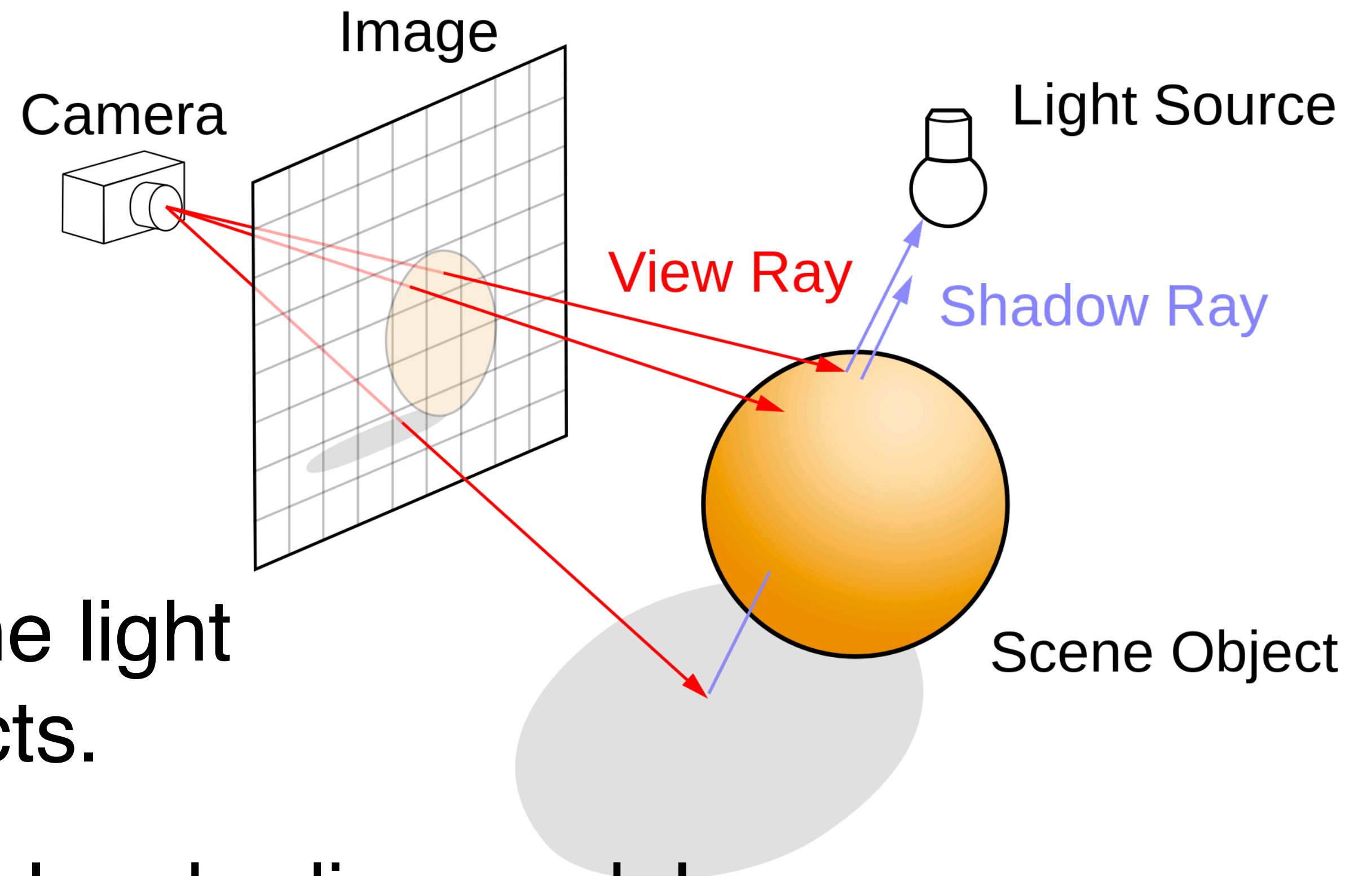
Shading, shadow and mirror reflection

- Take the closest hit for each ray.
- Shade the ray according to a shading model.
- Shade the ray only when the secondary ray connecting to the light source does not hit other objects.
 - ▶ To avoid self-shadowing, the secondary ray is shot off slightly above the hitting point.
(This is more straightforward than the rasterization-based shadow casting)



Shading, shadow and mirror reflection

- Take the closest hit for each ray.
- Shade the ray according to a shading model.
- Shade the ray only when the secondary ray connecting to the light source does not hit other objects.
- In addition to the diffuse+specular shading model we can include a recursive specular reflection



Recursive ray tracing

ForEach pixel

- ▶ Trace the primary eye ray
 - ▶ Find intersection
- ▶ Trace secondary shadow ray to all lights
 - ▶ Color = Visible ? IlluminationModel : 0 ;
- ▶ Trace reflected ray
 - ▶ Color += reflectivity * (Color of the reflected ray)
 - ▶ Recursion might never stop, so set a max recursion depth

Detail: RayThruPixel

Outline code

```
Image Raytrace(Camera cam, Scene scene, int width, int height){  
    Image image = new Image(width,height);  
    for (int i=0; i<height; i++)  
        for (int j=0; j<width; j++){  
            Ray ray = RayThruPixel( cam, i, j );  
            Intersection hit = Intersect( ray, scene );  
            image[i][j] = FindColor( hit );  
        }  
    return image;  
}
```

Ray

- A **ray** is described by a point $\mathbf{p}_0 \in \mathbb{R}^3$ and a direction $\mathbf{d} \in \mathbb{R}^3$.
- Mathematically, the ray is a continuous set of points parametrized as

$$\mathbf{p}(t) = \mathbf{p}_0 + t\mathbf{d} \quad t > 0$$

Camera

- A **camera** is described by

$$\mathbf{eye} \in \mathbb{R}^3 \quad \mathbf{center} \in \mathbb{R}^3 \quad \mathbf{up} \in \mathbb{R}^3$$

which can also be rewritten in terms of a coordinate frame

$$\mathbf{eye} \in \mathbb{R}^3 \quad \mathbf{u} \in \mathbb{R}^3 \quad \mathbf{v} \in \mathbb{R}^3 \quad \mathbf{w} \in \mathbb{R}^3$$

$$\mathbf{w} = \frac{\mathbf{eye} - \mathbf{center}}{|\mathbf{eye} - \mathbf{center}|} \quad \mathbf{u} = \frac{\mathbf{up} \times \mathbf{w}}{|\mathbf{up} \times \mathbf{w}|} \quad \mathbf{v} = \mathbf{w} \times \mathbf{u}$$

Ray through pixel

- Other relevant parameters:

window resolution $width$ $height$

field of view (angle) fov_x fov_y

$$\tan\left(\frac{fov_x}{2}\right) = \tan\left(\frac{fov_y}{2}\right) \frac{width}{height}$$

- Given camera $\mathbf{eye} \in \mathbb{R}^3$ $\mathbf{u} \in \mathbb{R}^3$ $\mathbf{v} \in \mathbb{R}^3$ $\mathbf{w} \in \mathbb{R}^3$

and pixel $i \in \{0, \dots, height - 1\}$

$j \in \{0, \dots, width - 1\}$

the ray is given by... (next page, see whiteboard for derivation)

Ray through pixel

$$\mathbf{p}_0 = \text{eye} \quad \mathbf{d} = \frac{\alpha \mathbf{u} + \beta \mathbf{v} - \mathbf{w}}{|\alpha \mathbf{u} + \beta \mathbf{v} - \mathbf{w}|}$$

where

$$\alpha = \tan\left(\frac{fov_x}{2}\right) \left(\frac{(j+0.5) - \frac{\text{width}}{2}}{\frac{\text{width}}{2}} \right)$$
$$\beta = \tan\left(\frac{fov_y}{2}\right) \left(\frac{(i+0.5) - \frac{\text{height}}{2}}{\frac{\text{height}}{2}} \right)$$

Detail: Ray-Scene Intersect

Outline code

```
Image Raytrace(Camera cam, Scene scene, int width, int height){  
    Image image = new Image(width,height);  
    for (int i=0; i<height; i++)  
        for (int j=0; j<width; j++){  
            Ray ray = RayThruPixel( cam, i, j );  
            Intersection hit = Intersect( ray, scene );  
            image[i][j] = FindColor( hit );  
        }  
    return image;  
}
```

Ray-scene intersection

```
Intersection Intersect(Ray ray, Scene scene){
    Distance mindist = INFINITY;
    Object hitobject = NULL;
    foreach (object in scene){ // Find closest intersection; test all objects
        Distance t = Intersect(ray, object);
        if (t>0 && t< mindist){ // closer than previous closest object
            mindist = t;
            hitobject = object;
        }
    }
    return IntersectionInfo(ray, mindist, hitobject);
}
```

Ray-object intersection

```
Intersection Intersect(Ray ray, Scene scene){
    Distance mindist = INFINITY;
    Object hitobject = NULL;
    foreach (object in scene){ // Find closest intersection; test all objects
        Distance t = Intersect(ray, object);
        if (t>0 && t< mindist){ // closer than previous closest object
            mindist = t;
            hitobject = object;
        }
    }
    return IntersectionInfo(ray, mindist, hitobject);
}
```

- Compute the intersection (in terms of the parameter t for the ray) between the ray and a primitive.

Ray-object intersection

- Compute the intersection (in terms of the parameter t for the ray) between the ray and a primitive.
- We will study ray-sphere (and ray-ellipsoids) and ray-triangle on whiteboard.

Next time

- Lighting

