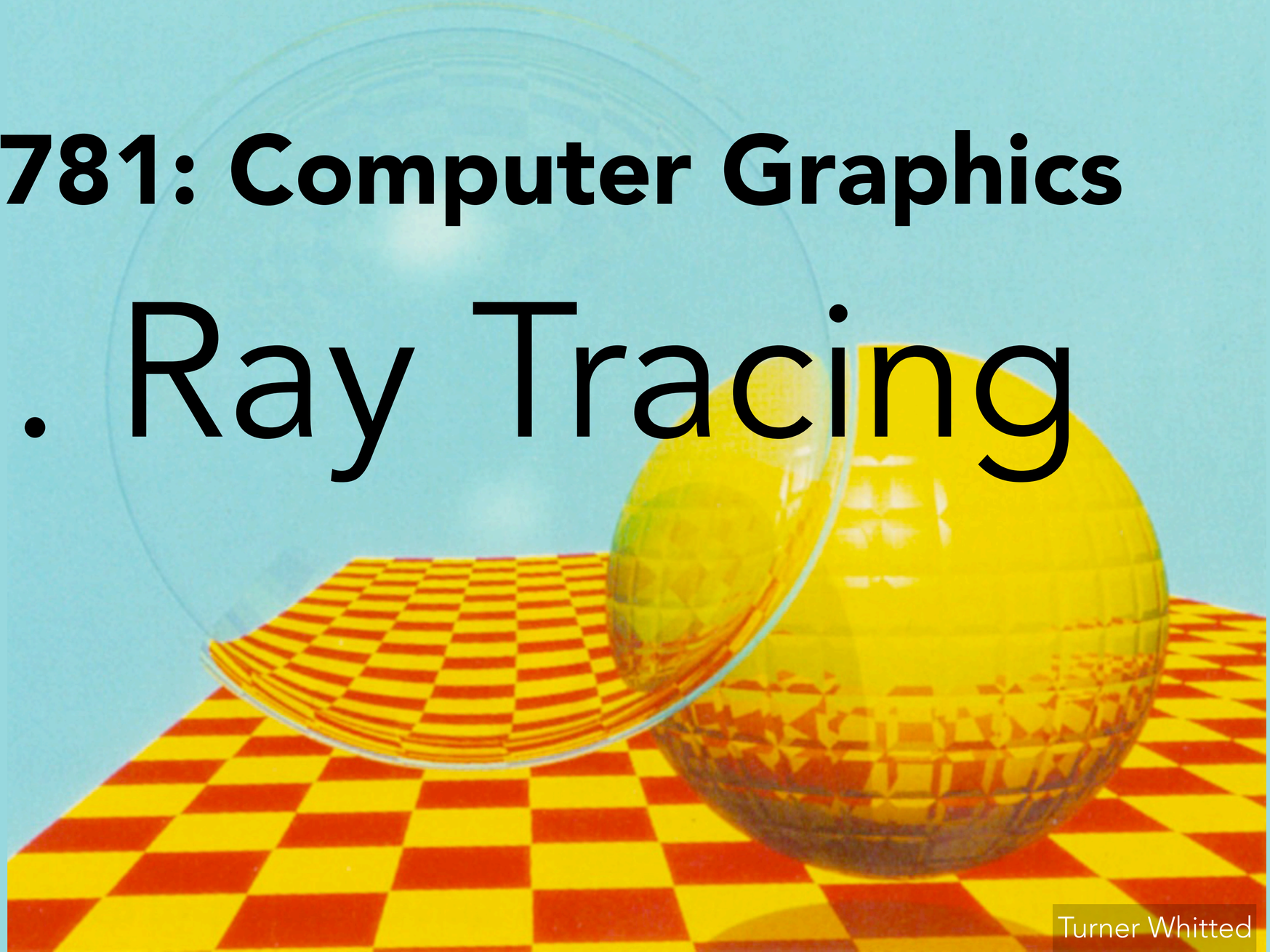
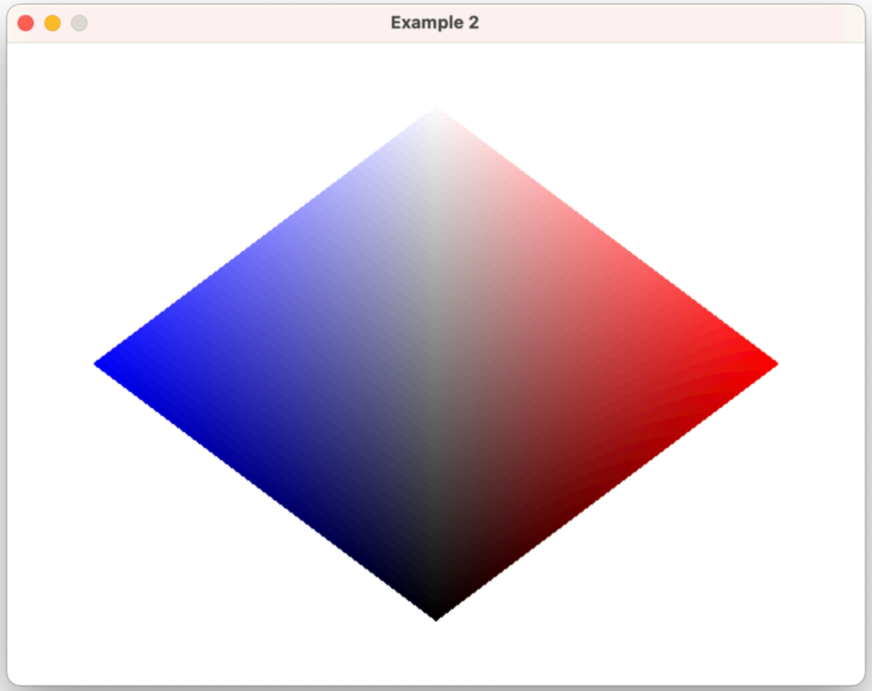
COL781: Computer Graphics

13. Ray Tracing



Assignment 1

```
R::Rasterizer r;
...
R::Object shape = r.createObject();
r.setVertexAttribs(shape, 0, 4, vertices);
r.setVertexAttribs(shape, 1, 4, colors);
r.setTriangleIndices(shape, 2, triangles);
while (!r.shouldQuit()) {
    r.clear(vec4(1.0,1.0,1.0, 1.0));
    r.useShaderProgram(program);
    r.drawObject(shape);
    r.show();
}
...
```

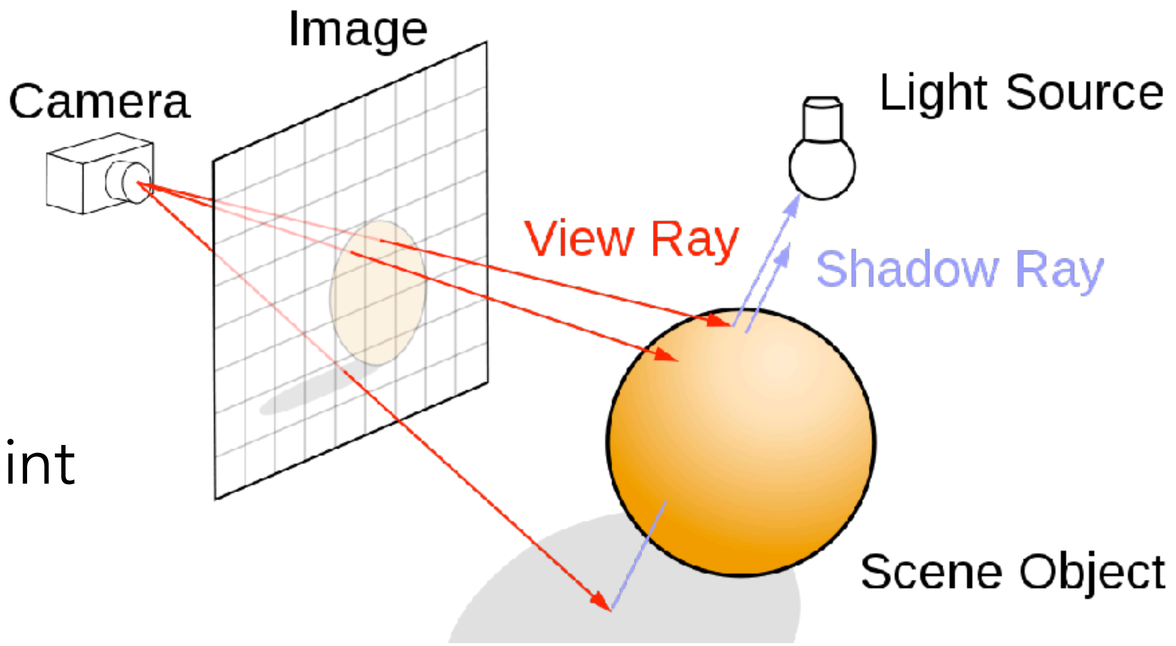


Assignment 1 groups issues?

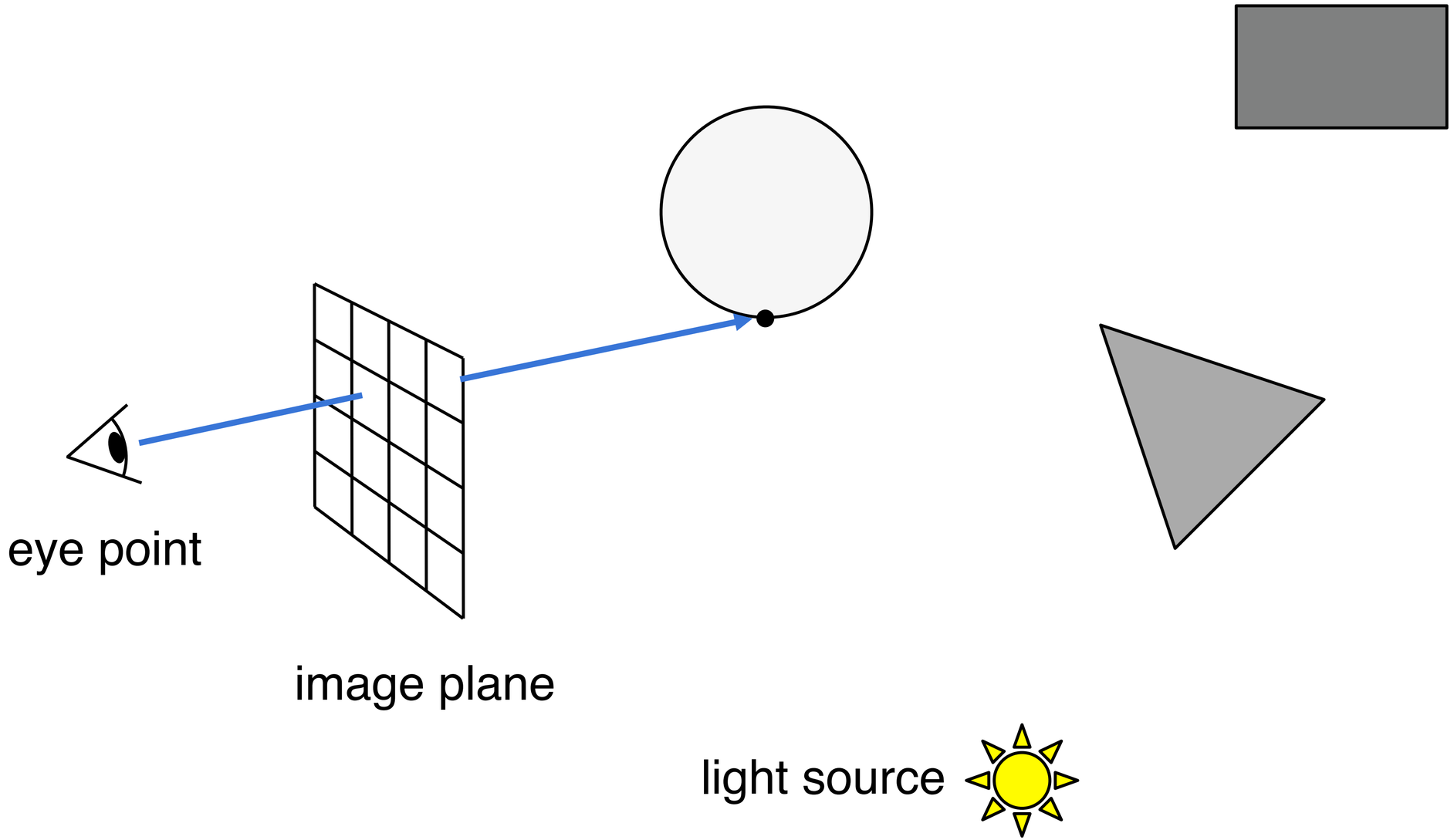
Ray tracing

For each sample:

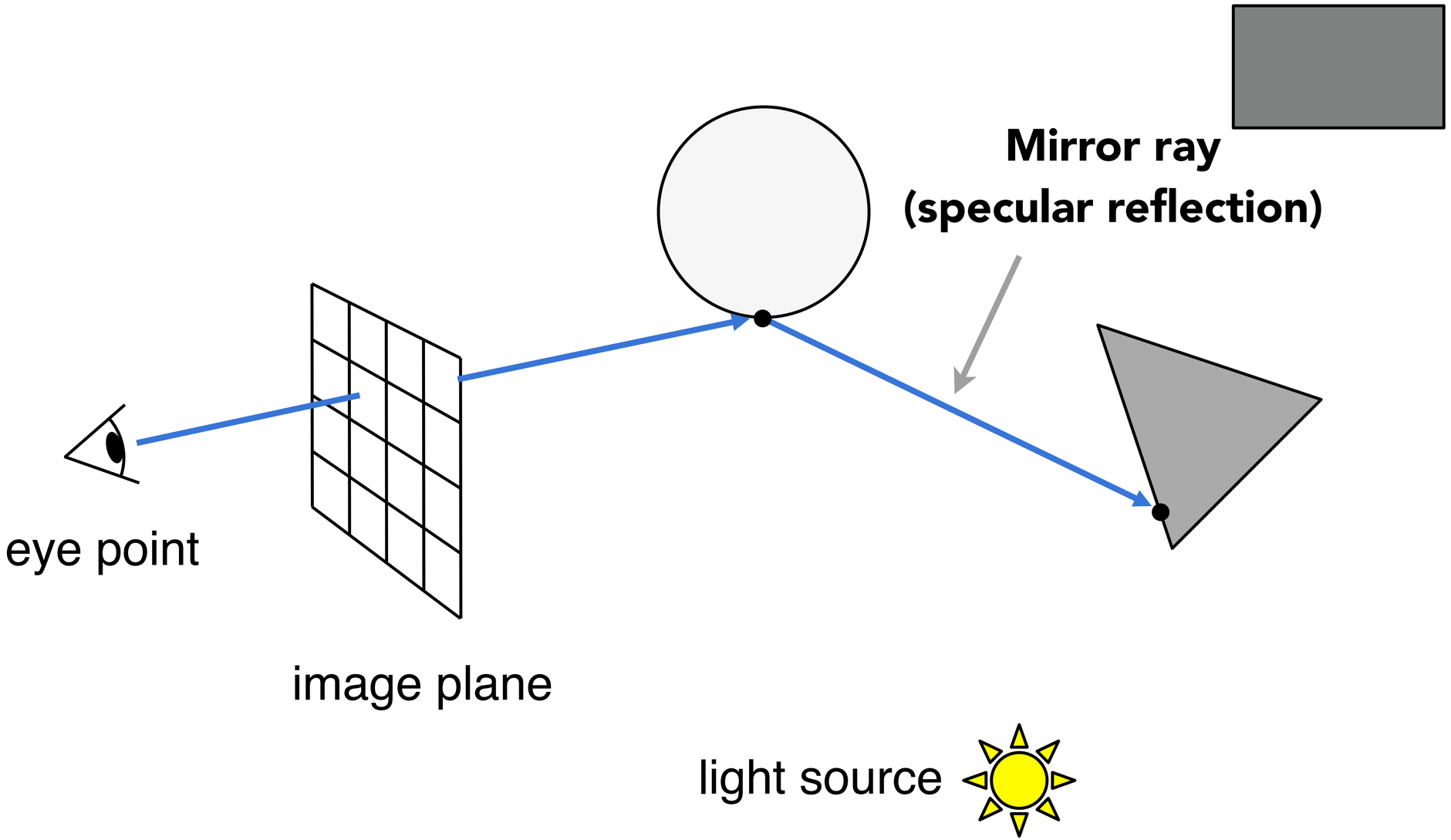
- Shoot a ray into the scene
- Find the closest intersection
- Get shaded colour at intersection point
- Set sample colour to it



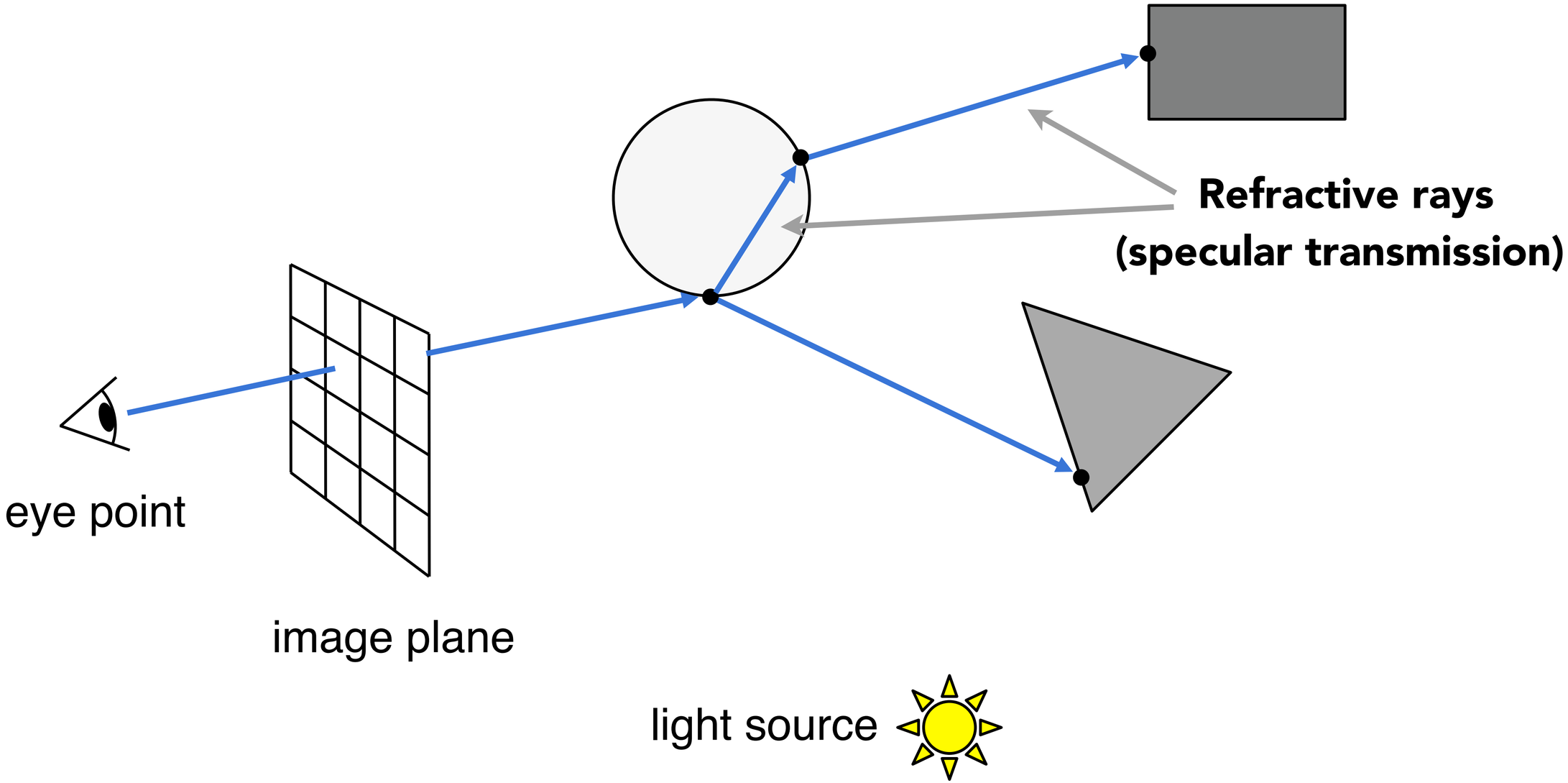
Weirder pixel→ray mappings



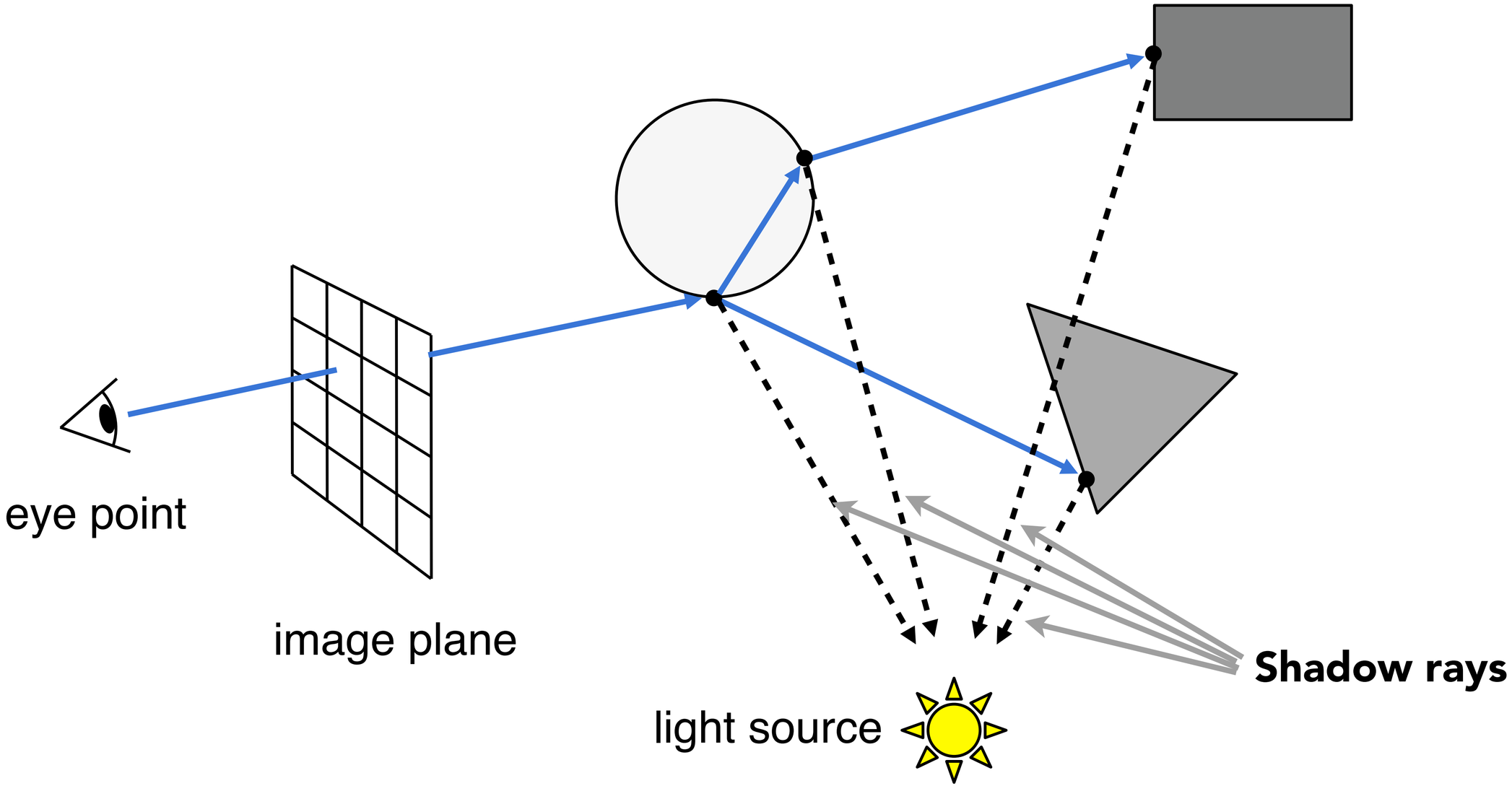
Kanazawa and Ng



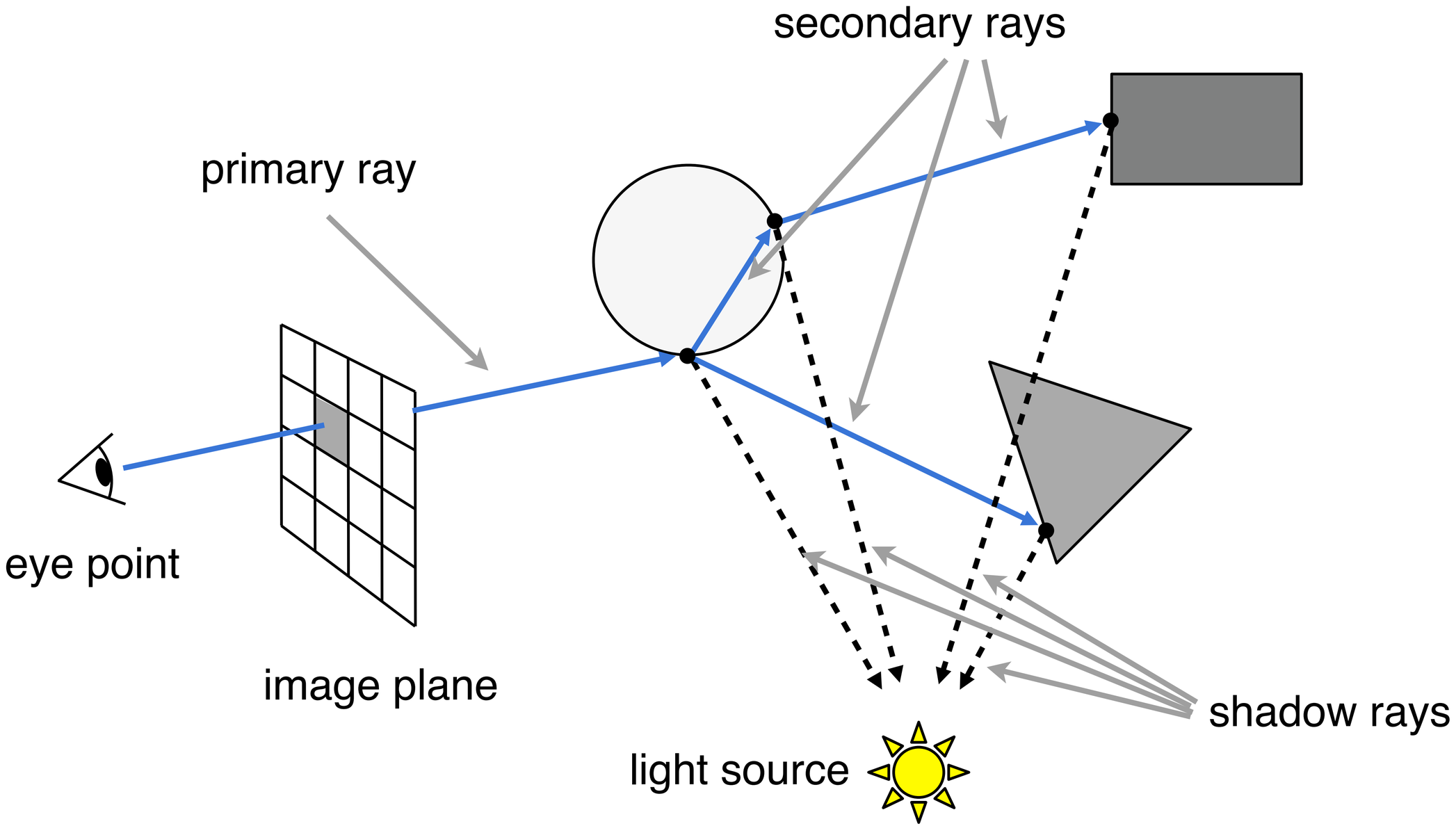
Kanazawa and Ng



Kanazawa and Ng



Kanazawa and Ng



Kanazawa and Ng

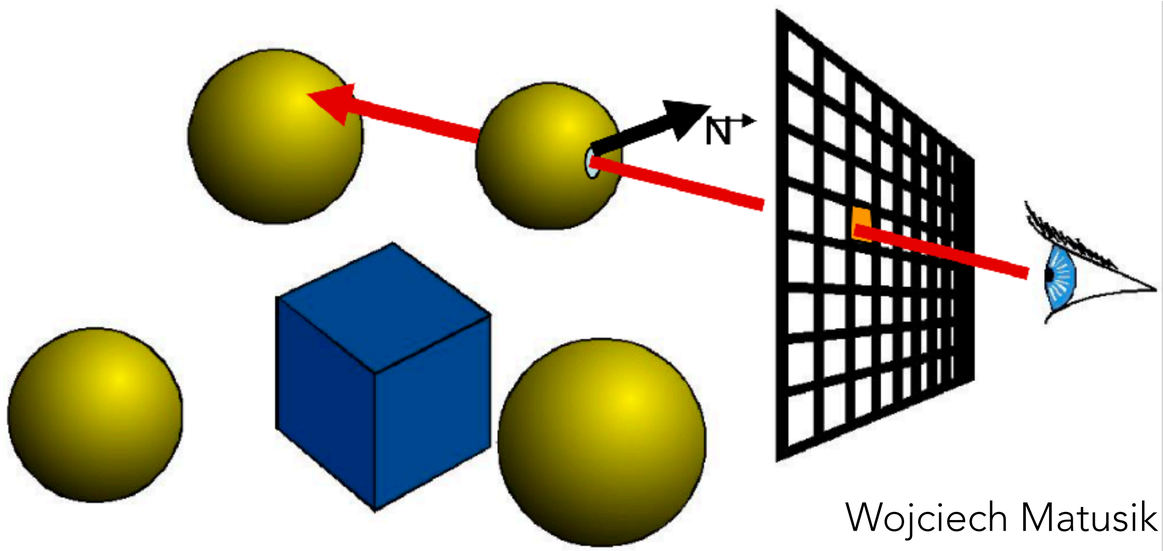
Ray-surface intersection

Each sample (x, y) in the image corresponds to a ray $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$ in the world

Find closest positive intersection: min $t > 0$

Return info needed for shading:

- Position **p**
- Normal **n**
- Object ID / material properties



Wojciech Matusik

Ray-sphere intersection

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

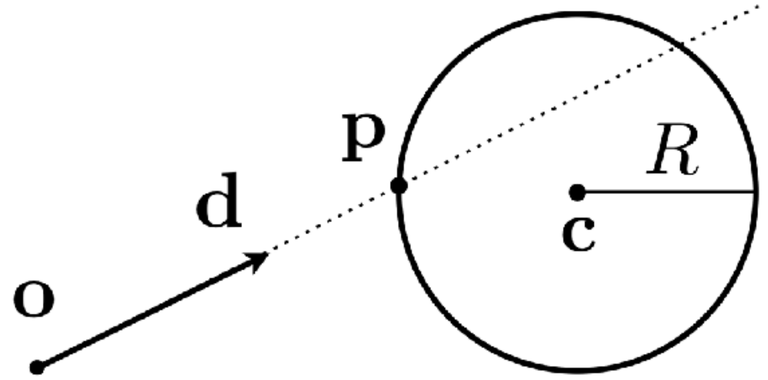
Sphere equation: $\|\mathbf{p} - \mathbf{c}\|^2 = R^2$

$$\|(\mathbf{o} - \mathbf{c}) + t\mathbf{d}\|^2 = R^2$$

$$\|\mathbf{d}\|^2 t^2 + 2\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})t + \|\mathbf{o} - \mathbf{c}\|^2 - R^2 = 0$$

Solve for t, keep smallest positive root (if any)

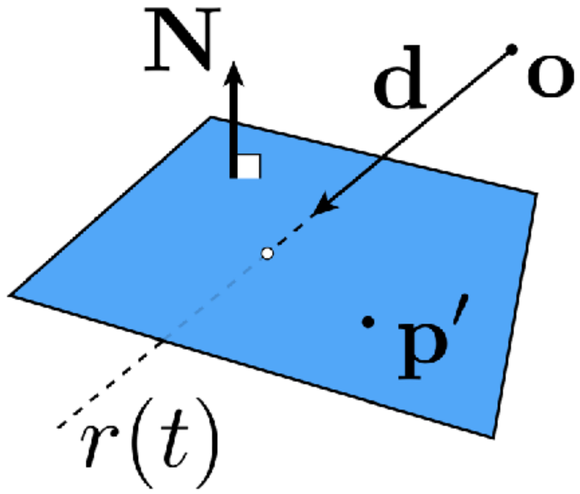
- Position: $\mathbf{p} = \mathbf{r}(t)$
- Normal: $\mathbf{n} = (\mathbf{p} - \mathbf{c})/\|\mathbf{p} - \mathbf{c}\|$



Ray-plane intersection

Plane equation: $\mathbf{n} \cdot (\mathbf{p} - \mathbf{p}_0) = 0$

any known point on the plane

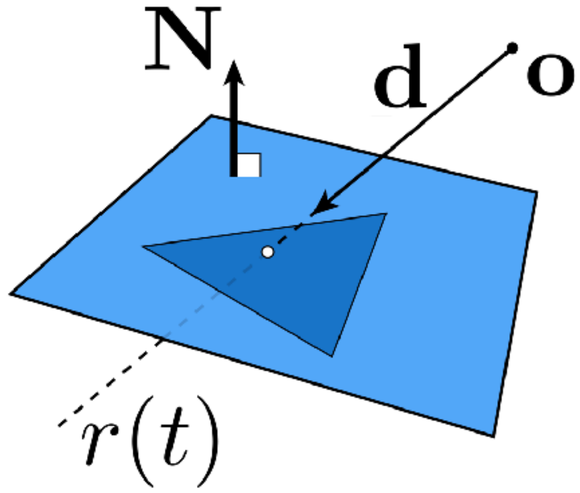


$$\mathbf{n} \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{p}_0) = 0$$

$$t = (\mathbf{n} \cdot (\mathbf{p}_0 - \mathbf{o}))/(\mathbf{n} \cdot \mathbf{d})$$

Ray-triangle intersection

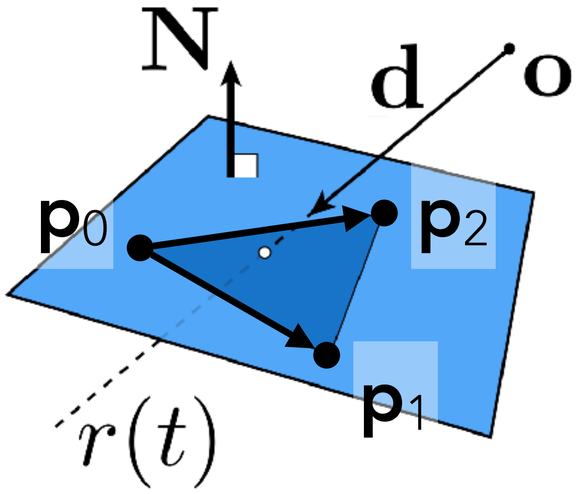
Intersect ray with plane, then check if it is inside triangle?



A better way: Any point on the plane is

$$\mathbf{p} = \mathbf{p}_0 + b_1(\mathbf{p}_1 - \mathbf{p}_0) + b_2(\mathbf{p}_2 - \mathbf{p}_0)$$
$$= (1-b_1-b_2)\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{p}_2$$

$$\mathbf{o} + t\mathbf{d} = \mathbf{p}_0 + b_1(\mathbf{p}_1 - \mathbf{p}_0) + b_2(\mathbf{p}_2 - \mathbf{p}_0)$$



3 equations in 3 unknowns:

$$\begin{bmatrix} -\mathbf{d} & \mathbf{p}_1 - \mathbf{p}_0 & \mathbf{p}_2 - \mathbf{p}_0 \end{bmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \mathbf{o} - \mathbf{p}_0$$

Solve to get t, b_1, b_2. For what values of b_1, b_2 is the point inside the triangle?

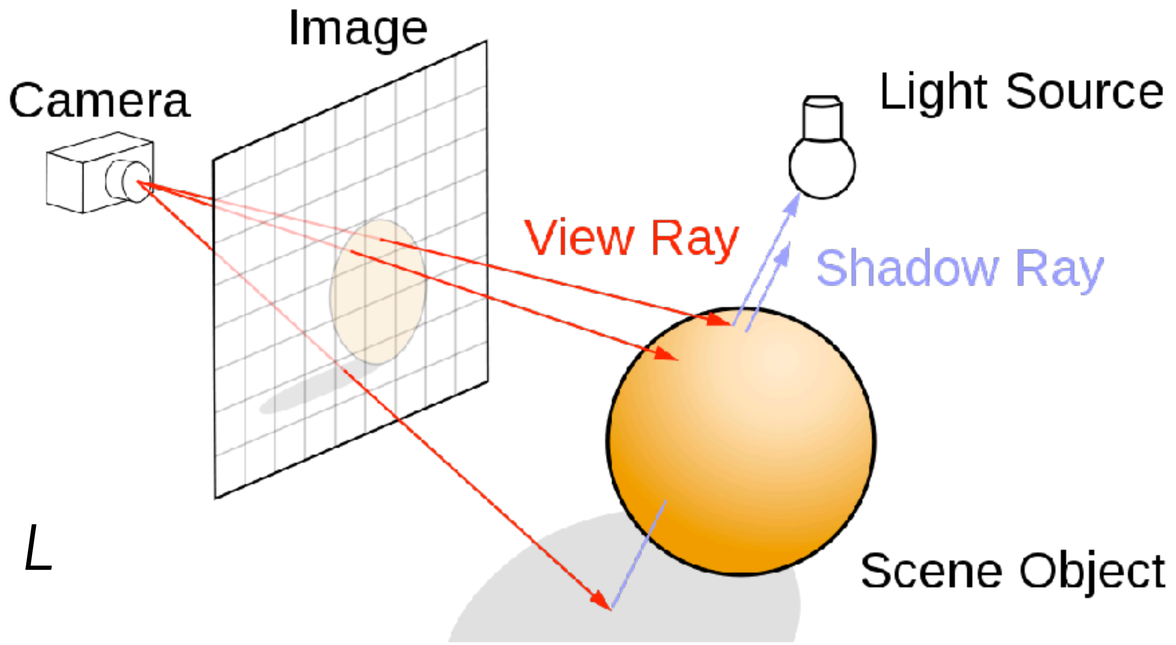
Shading the intersection point

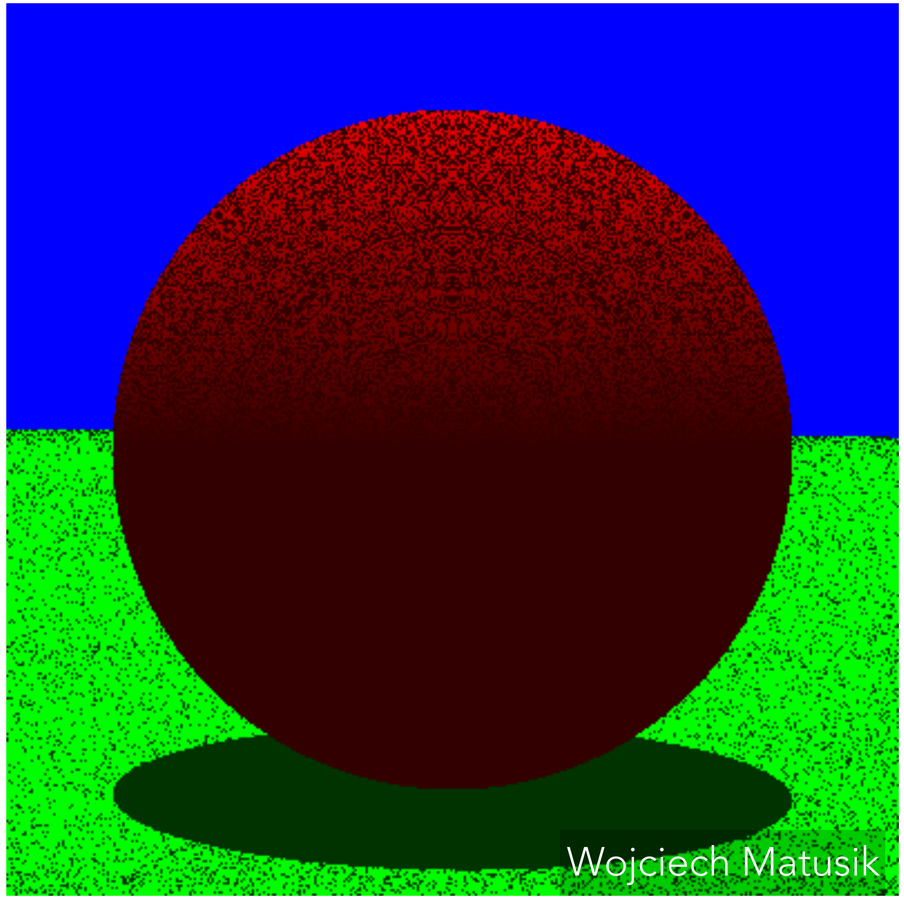
Once we know **p**, we can apply any desired shading model e.g. Blinn-Phong:

$$L = k_a I_a + \sum (k_d I_i \max(0, \mathbf{n} \cdot \boldsymbol{\ell}_i) + k_s I_i \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p)$$

Light should only be included if it is visible from **p**

- Shoot a "shadow ray" $\mathbf{p} + t\boldsymbol{\ell}_i$ towards light source
- If no intersection closer than distance to light source, add its contribution to L



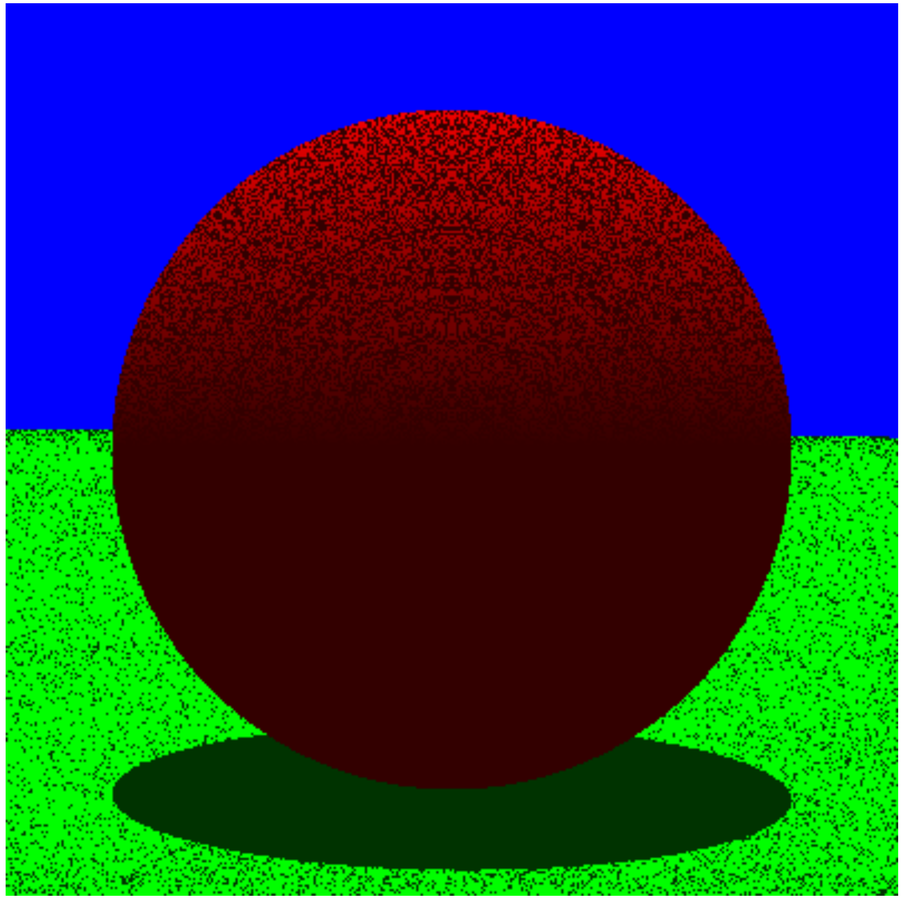


"**Shadow acne**"

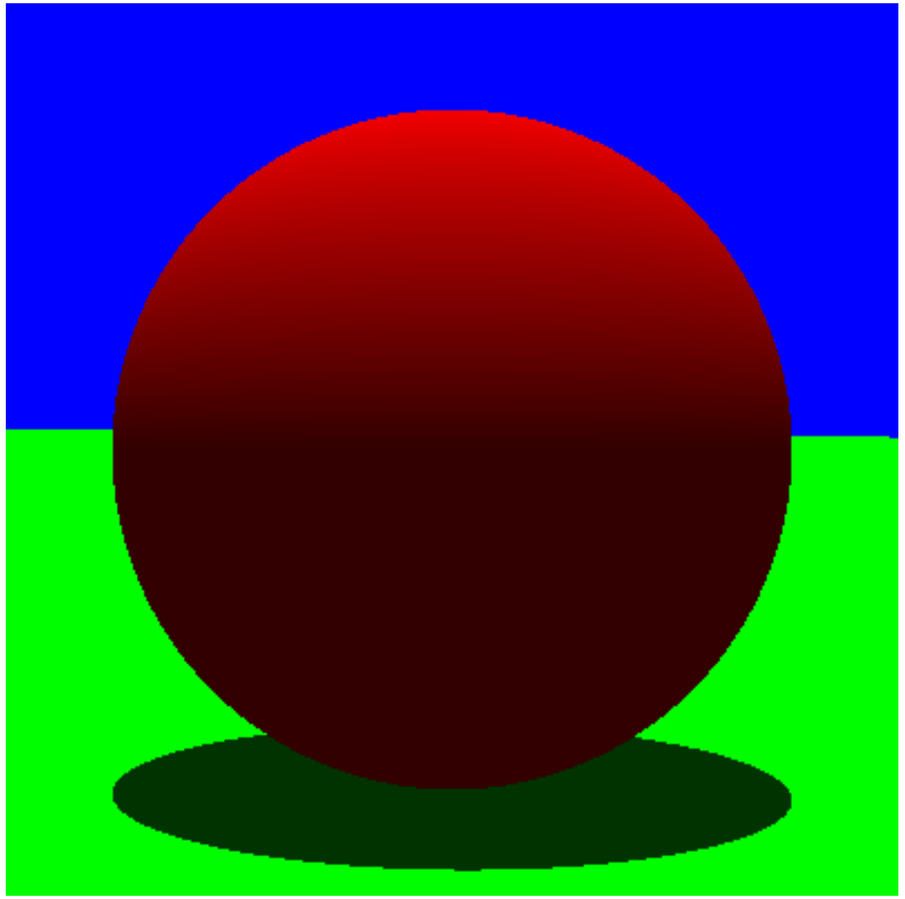
Why does this happen?

- **p** lies on surface
- **p** + $t\boldsymbol{\ell}$ intersects surface at $t = 0$
- Floating-point arithmetic may give e.g. $t = 0.000001$
- **p** thinks it's shadowed… by itself!

Solution: Pick a small positive ε ("bias") and ignore intersections with $t < \varepsilon$.

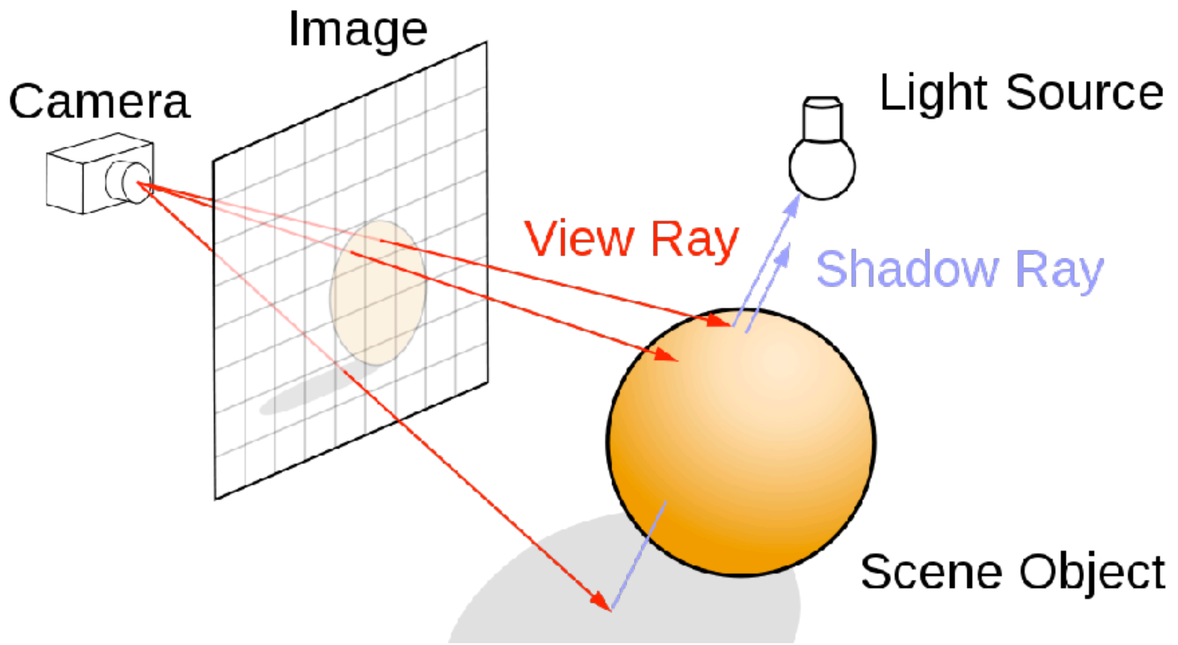


Without bias



With bias

Ray tracing



For each sample (*x*, *y*):

ray = makeRay(*camera*, *x*, *y*)

hit = castRay(*ray*, *scene*)

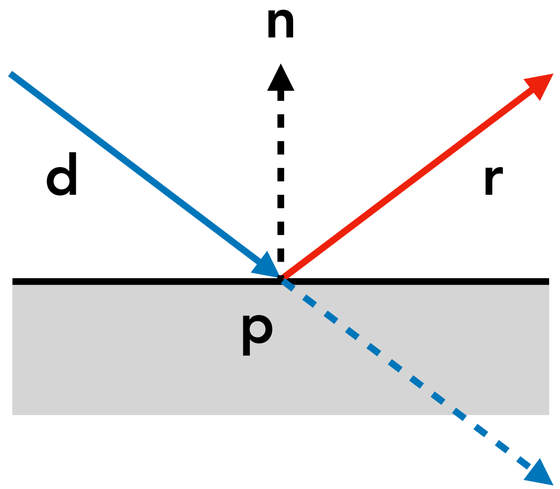
color = shade(*hit*, *scene*)

} *color* = traceRay(*ray*, *scene*)

framebuffer[*x*, *y*] = *color*

Reflection

$$\mathbf{r} = \mathbf{d} - 2\mathbf{d}_{\text{normal}}$$
$$= \mathbf{d} - 2(\mathbf{n} \cdot \mathbf{d})\mathbf{n}$$



Perfect mirror:

$$L = \text{traceRay}(\mathbf{p}, \mathbf{r}, \textit{scene})$$

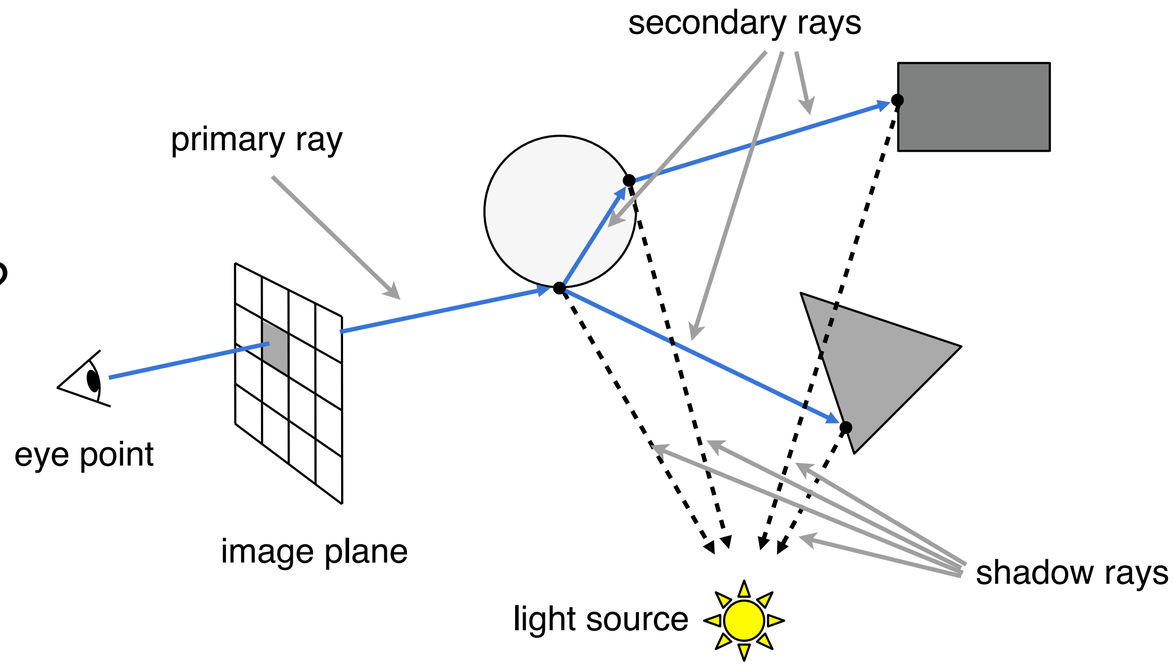
Reflective surface:

Actually, k_r depends on $\mathbf{n} \cdot \mathbf{d}$ in real life…

$$L = [\ldots\text{Blinn-Phong}\ldots] + k_r\,\text{traceRay}(\mathbf{p}, \mathbf{r}, \textit{scene})$$

Again, don’t forget ray bias!

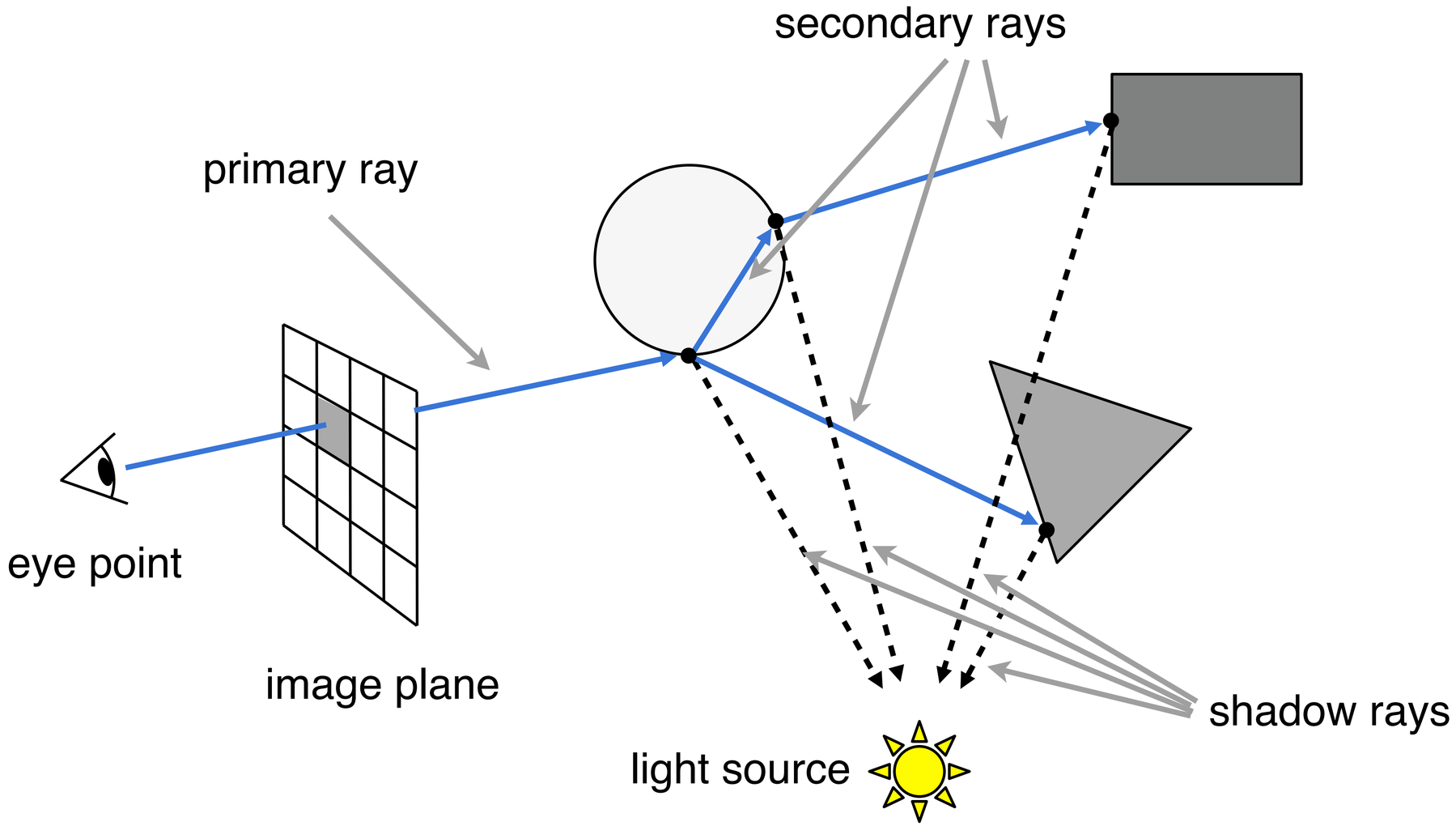
Interpretation 1: At the hit point, we are trying to find out what colour would be seen, i.e. light from which direction would be reflected towards the camera?



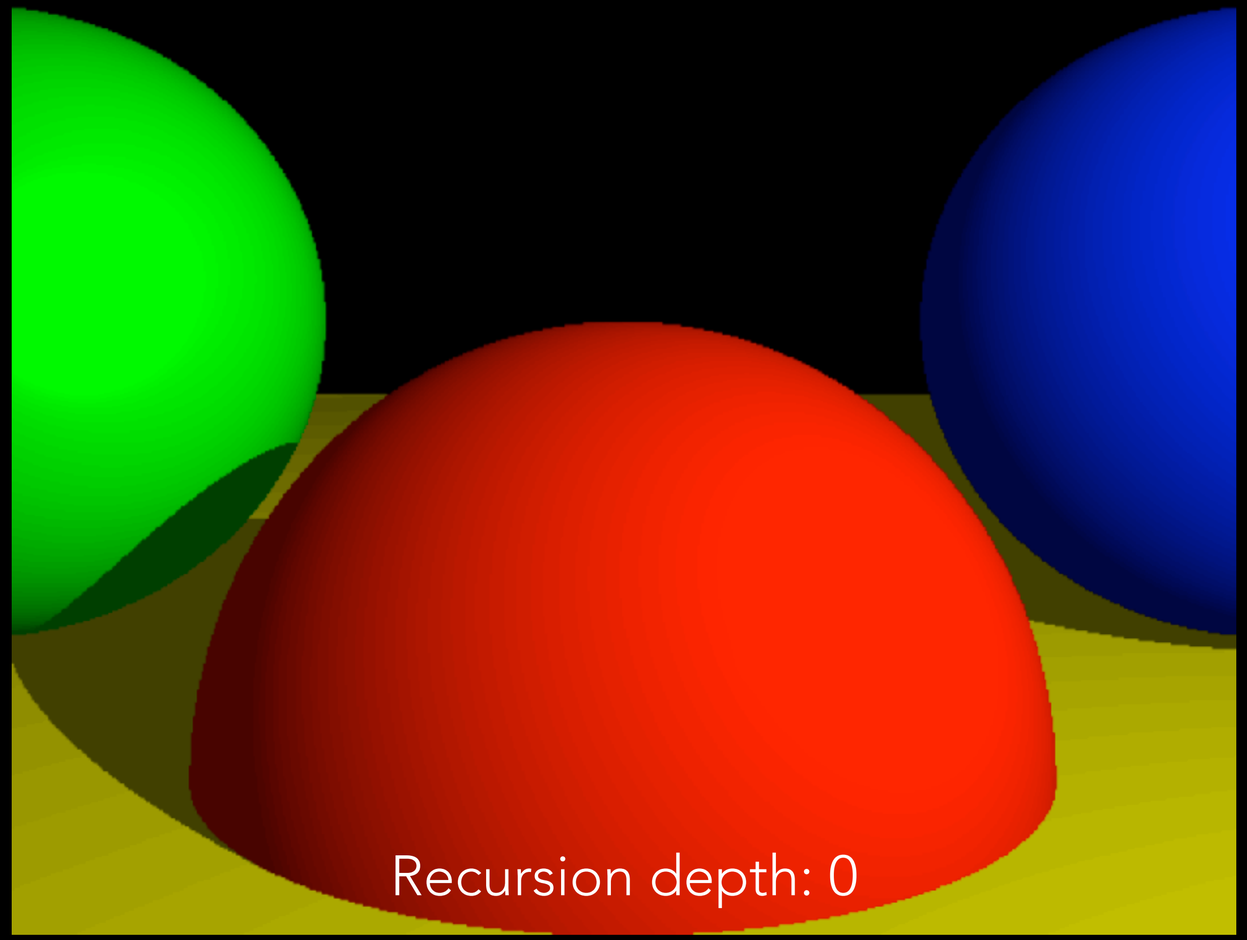
Interpretation 2: Instead of sending out rays from the light source until they get scattered into the camera, we are sending out rays from the camera to gather light from the light source.

Luckily, light transport is **reciprocal** (look up “Helmholtz reciprocity principle”) so both interpretations are the same!

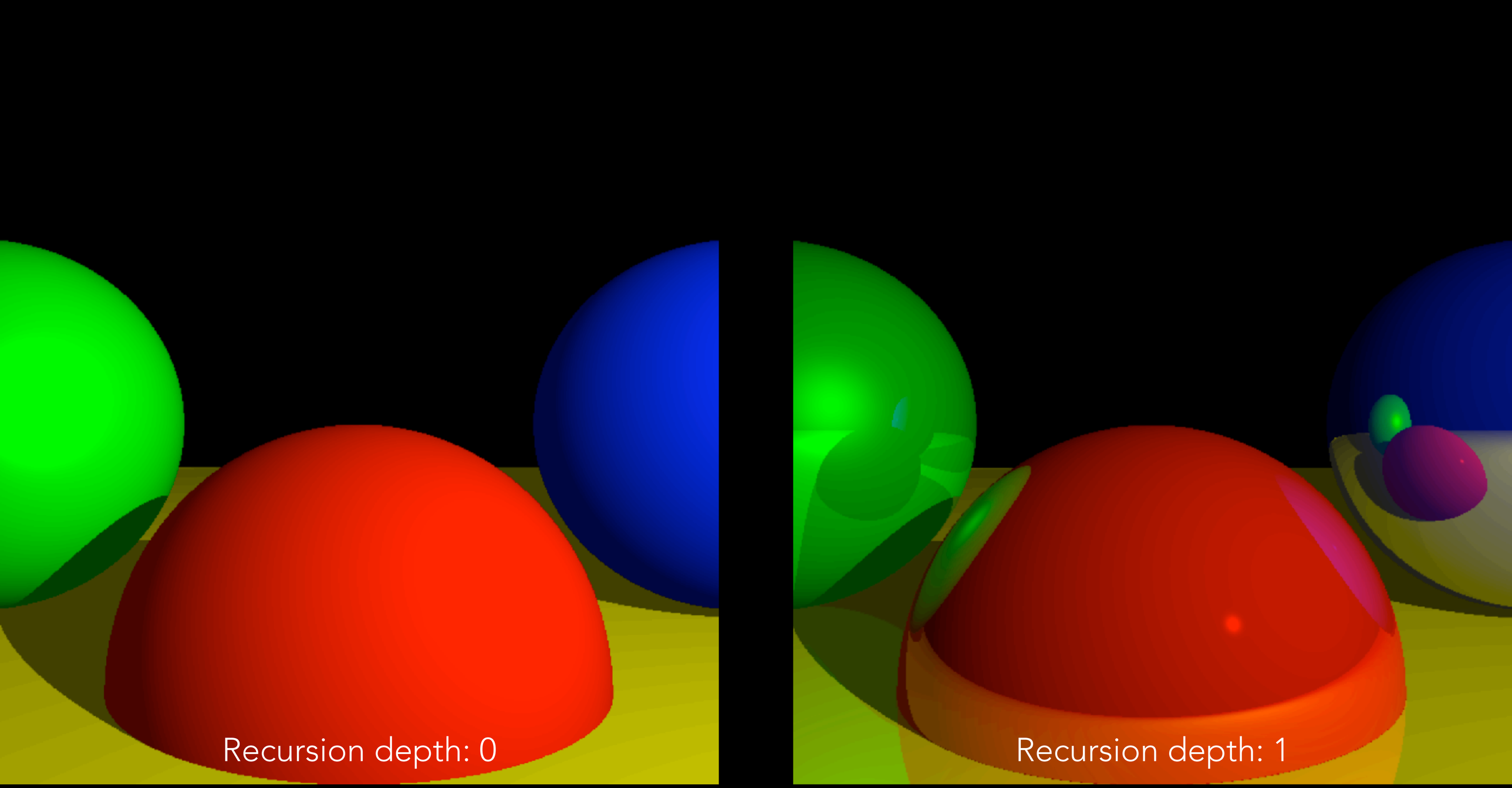
Recursive ray tracing

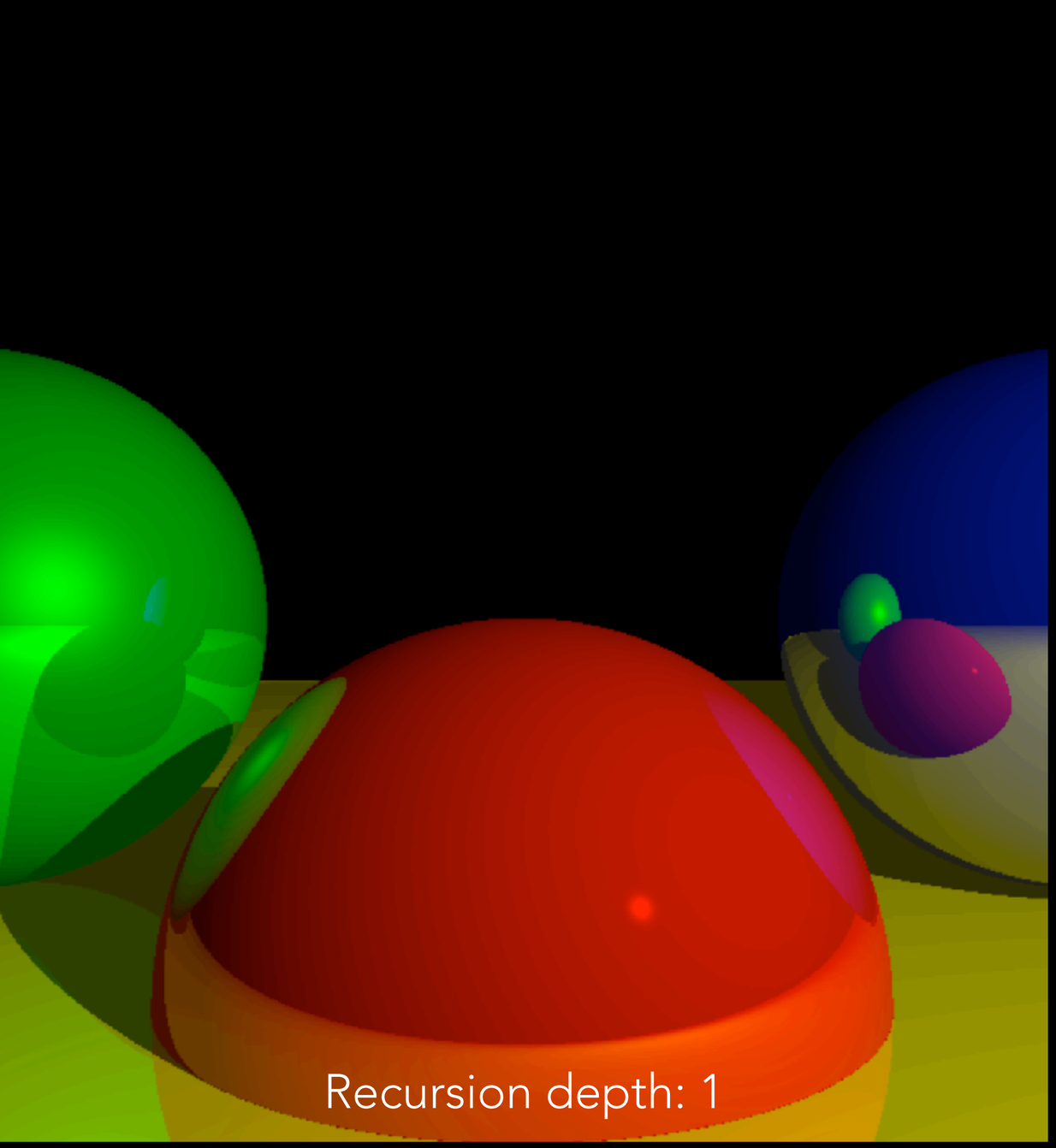


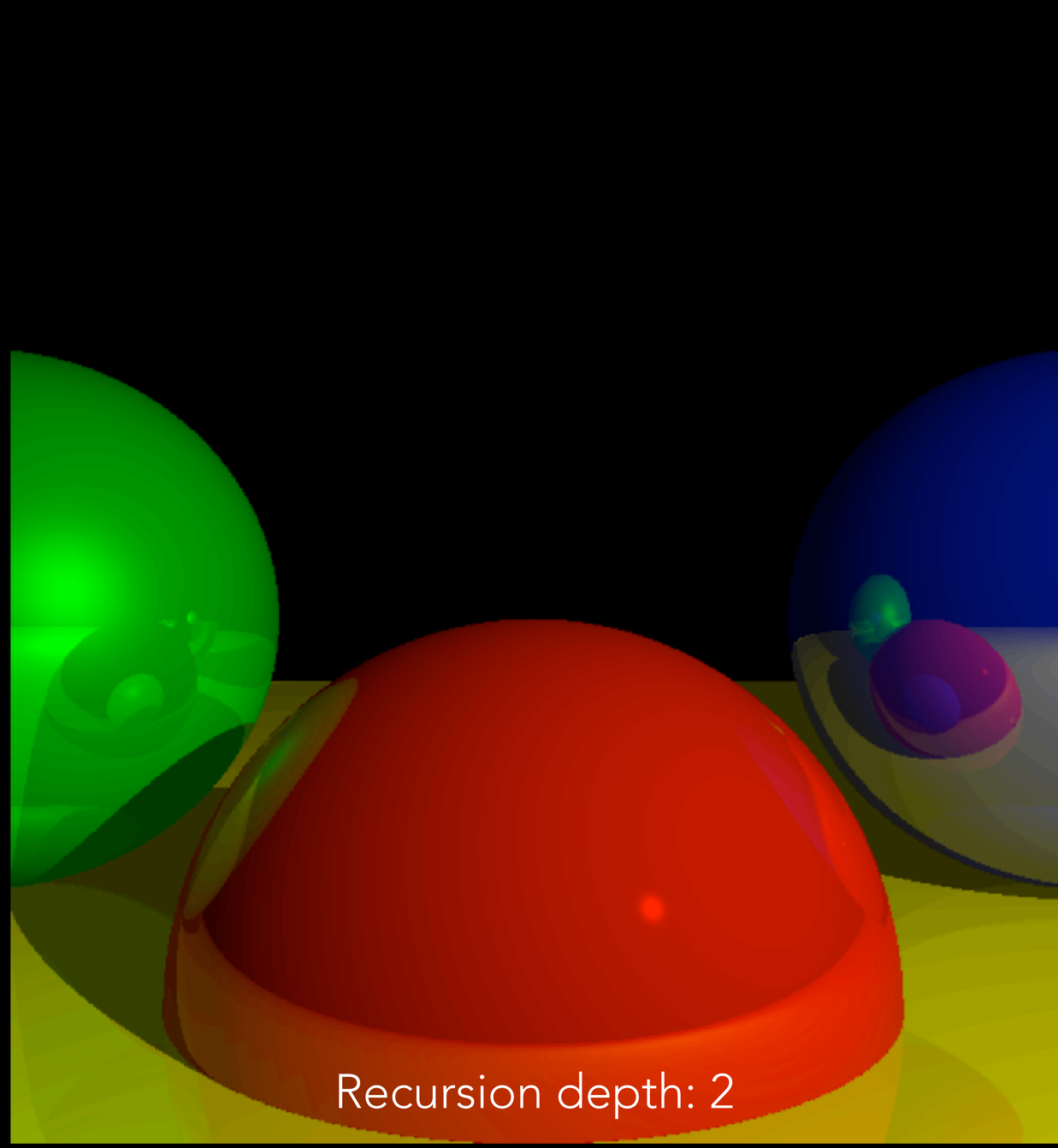
When to terminate the recursion?

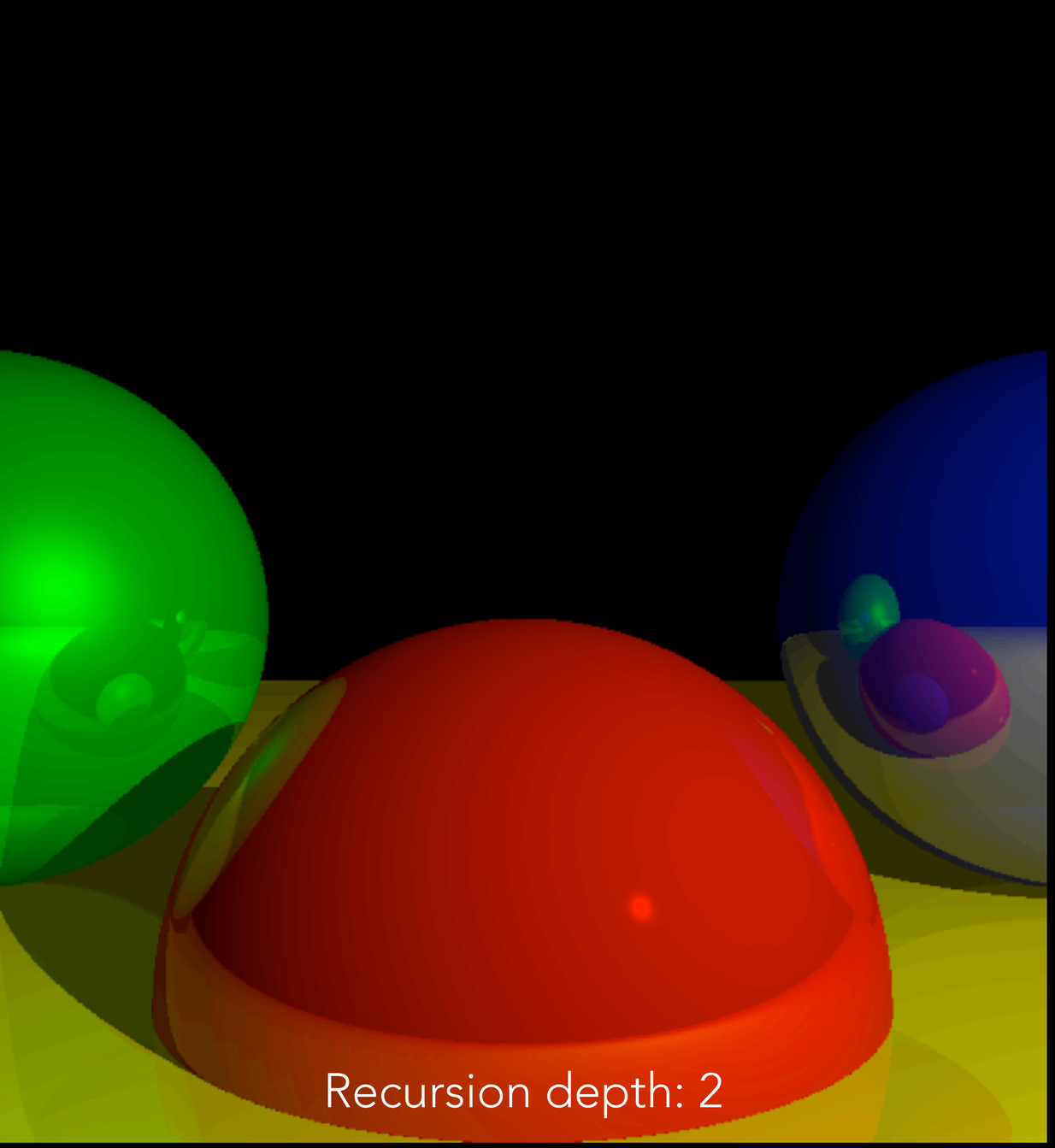


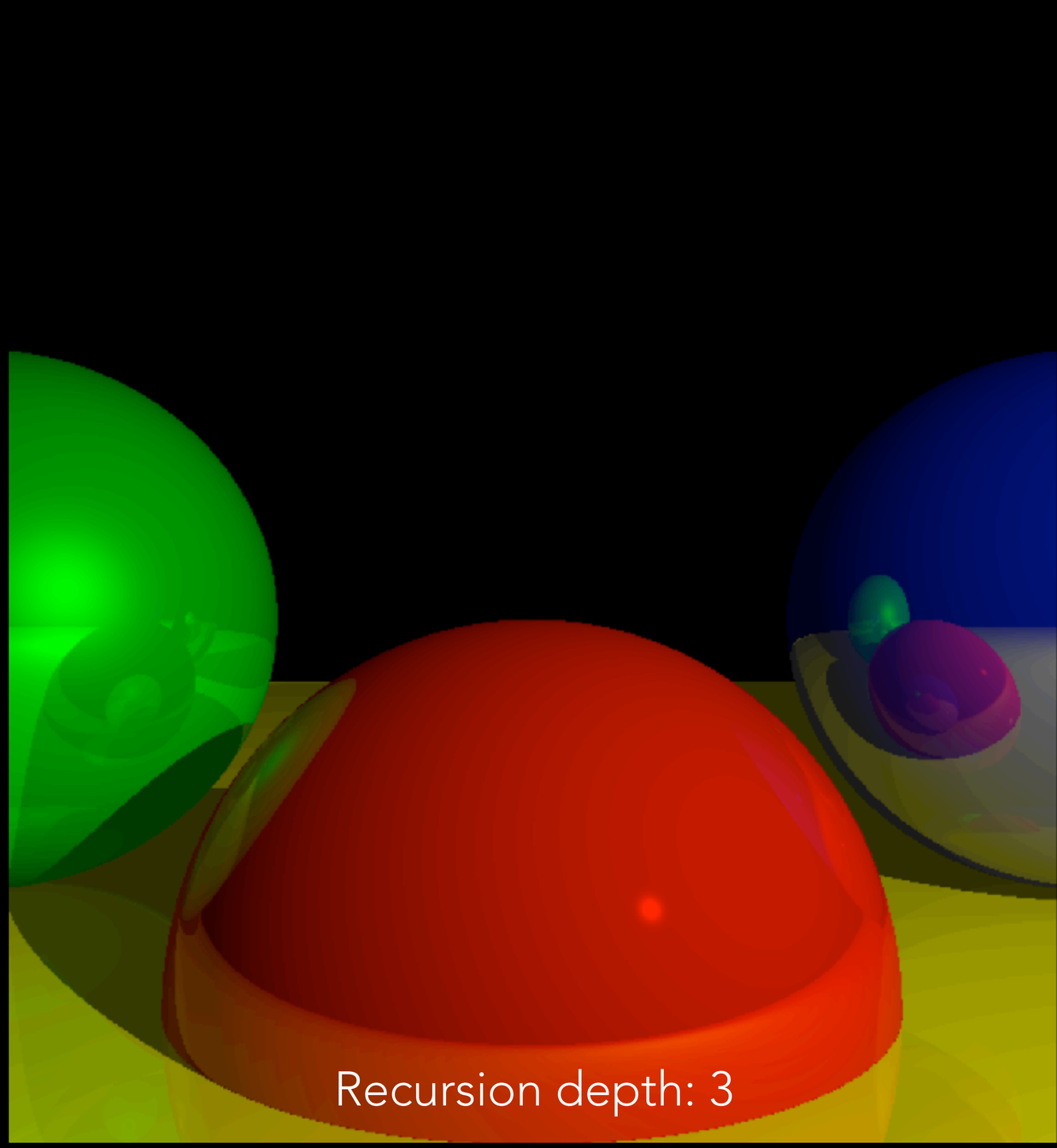
mikedoesweb.com


Recursion depth: 0
Recursion depth: 1


Recursion depth: 1


Recursion depth: 2

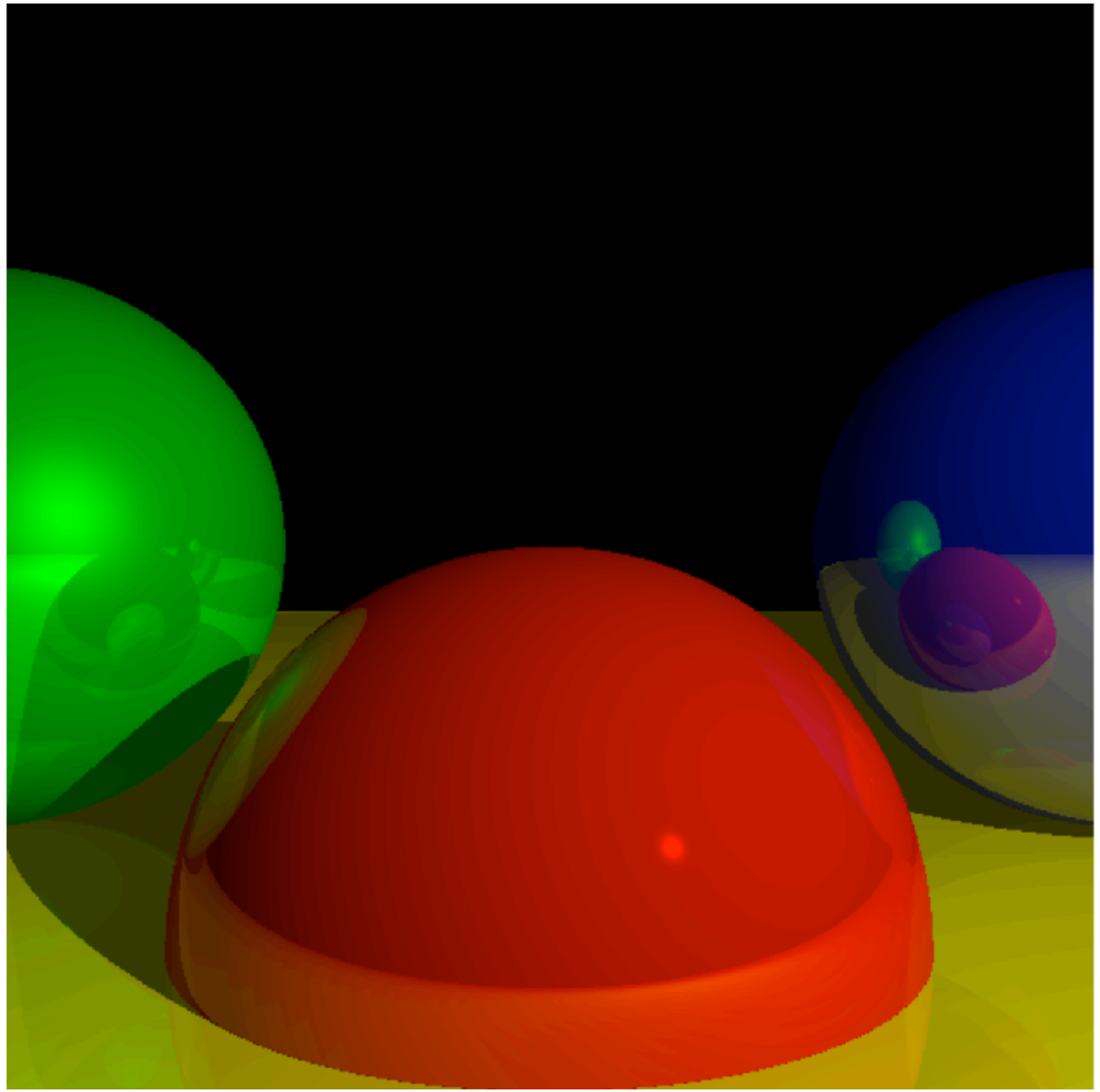

Recursion depth: 2


Recursion depth: 3

When to terminate the recursion?

- **Fixed:** stop if recursion depth > max
- **Adaptive:** stop if contribution of ray to final pixel colour < threshold
- …or whichever comes first

On termination, return… diffuse colour? background colour? black?



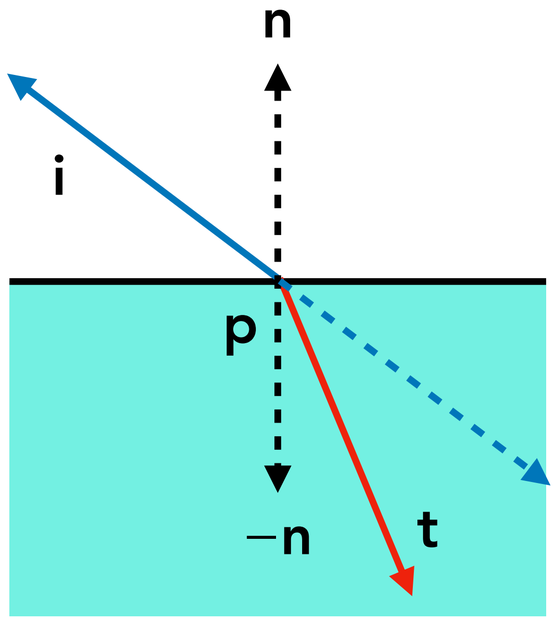
Transmission

Suppose ray passes from one material to another with different indices of refraction.

Snell's law: $\eta_i \sin \theta_i = \eta_t \sin \theta_t$

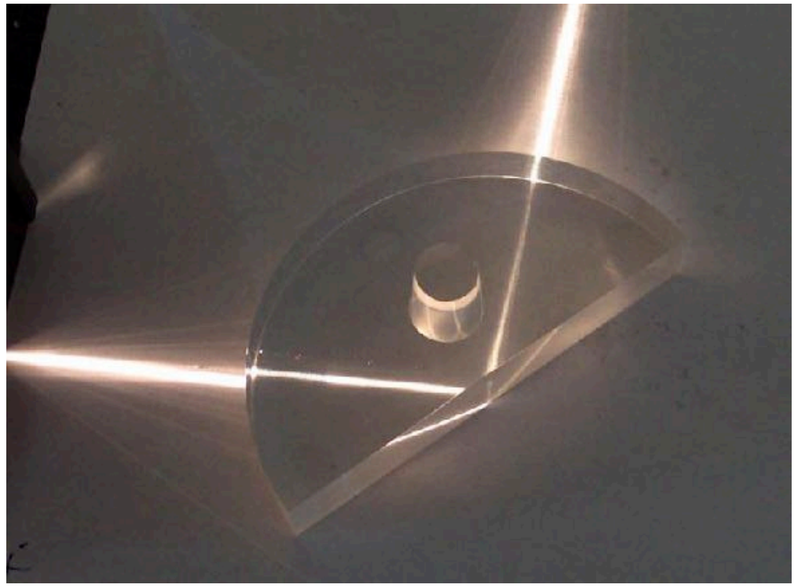
$$\mathbf{t} = \left(\eta_r (\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - \eta_r^2 \left(1 - (\mathbf{n} \cdot \mathbf{i})^2\right)} \right) \mathbf{n} - \eta_r \mathbf{i}$$

where $\eta_r = \eta_i / \eta_t$.



$$L = [\ldots] + k_t \, \text{traceRay}(\mathbf{p}, \mathbf{t}, \textit{scene})$$

If quantity inside square root is negative (sin $\theta_t > 1$): total internal reflection instead



Homework problem

Suppose you have a specialized, optimized code for intersecting any ray $\mathbf{o} + t\mathbf{d}$ with specific complicated shape S.

If the shape is transformed by some matrix $\mathbf{A}$, is there an easy way to intersect the transformed shape with a ray without designing a new optimized code?