EDAN30 Photorealistic Computer Graphics

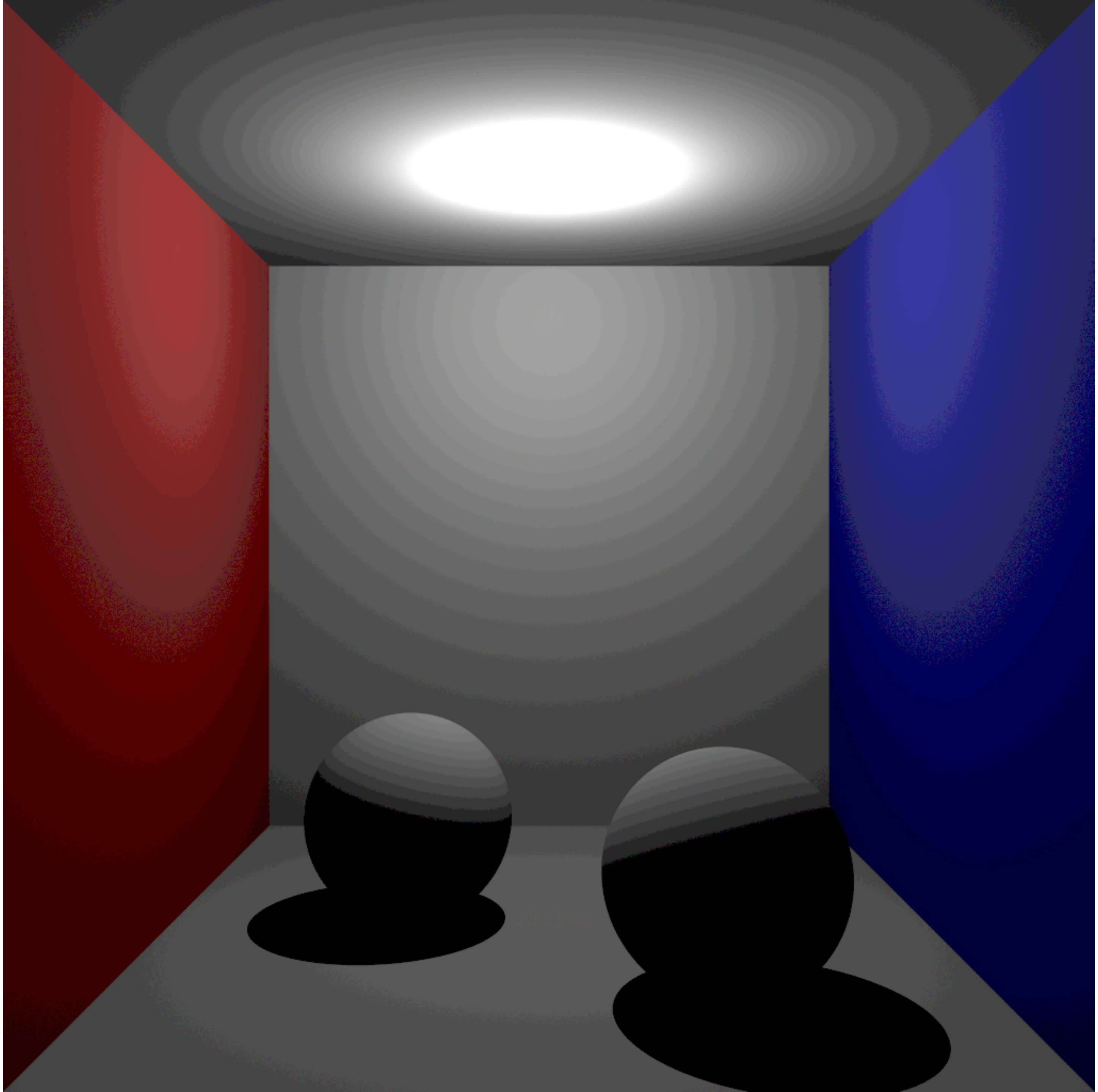
Seminar 3

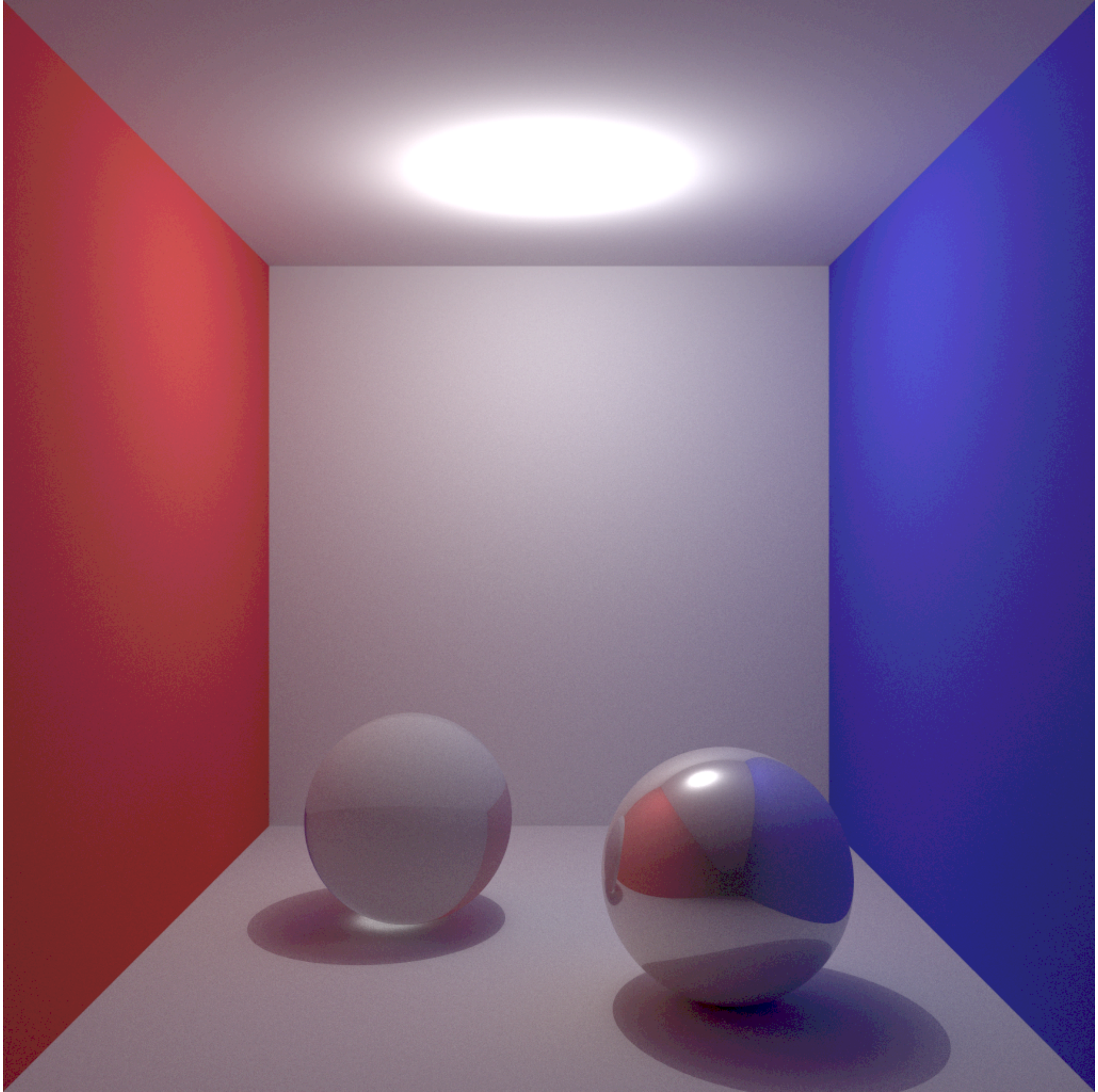
Path Tracing

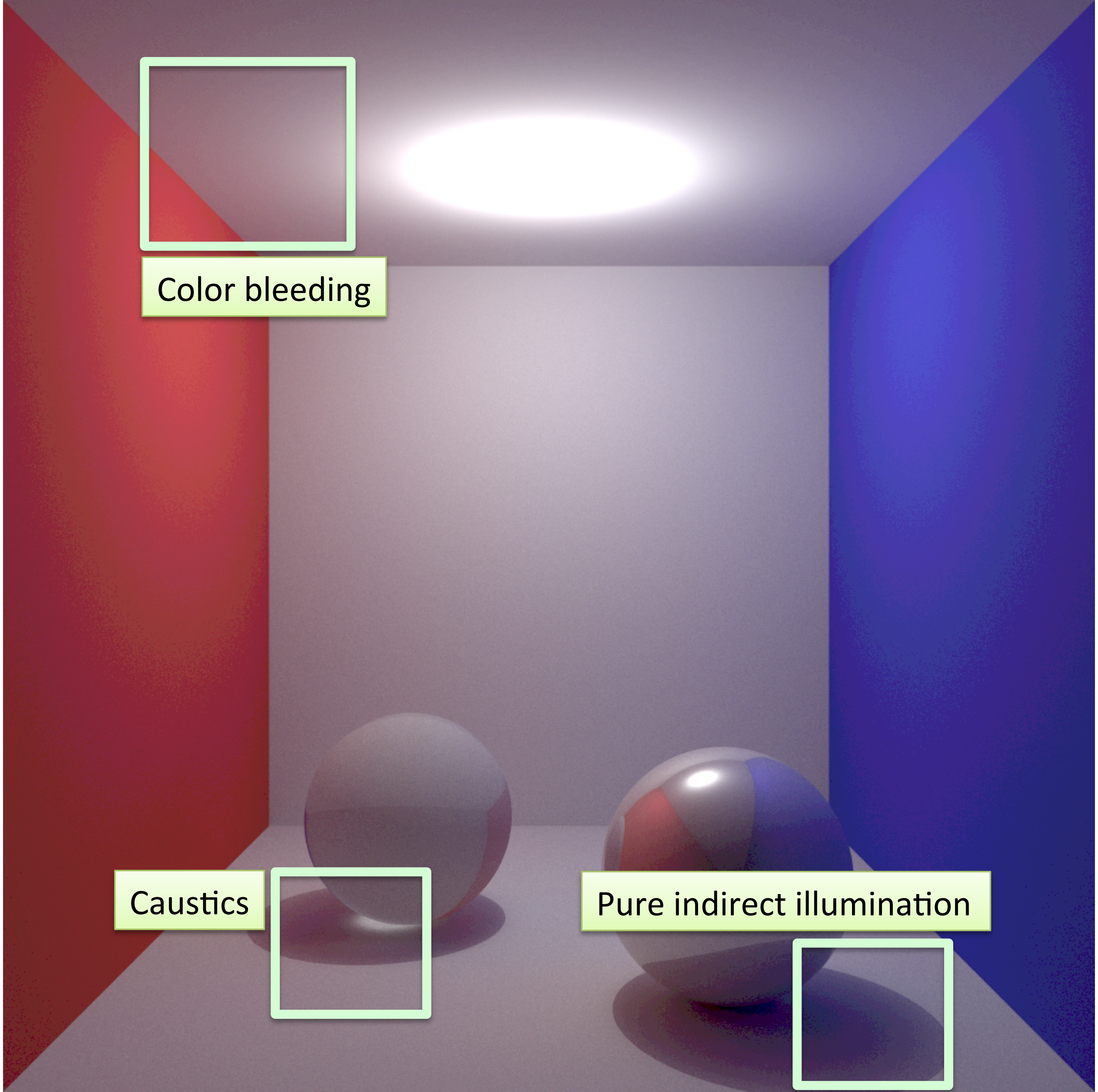
Rasmus Barringer, PhD student (rasmus@cs.lth.se)

Goal

- Render realistic lighting!

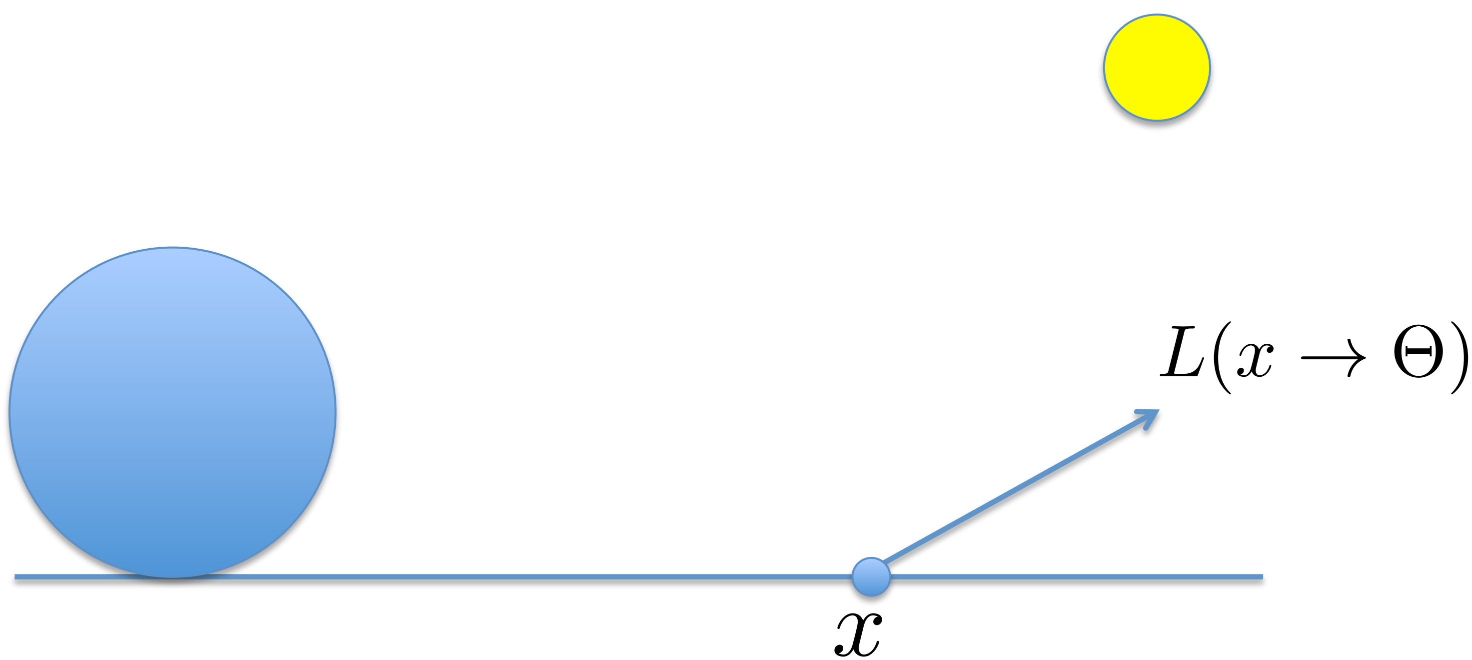





Color bleeding
Caustics
Pure indirect illumination

The rendering equation

$$L(x \to \Theta) = L_{direct}(x \to \Theta) + L_{indirect}(x \to \Theta)$$

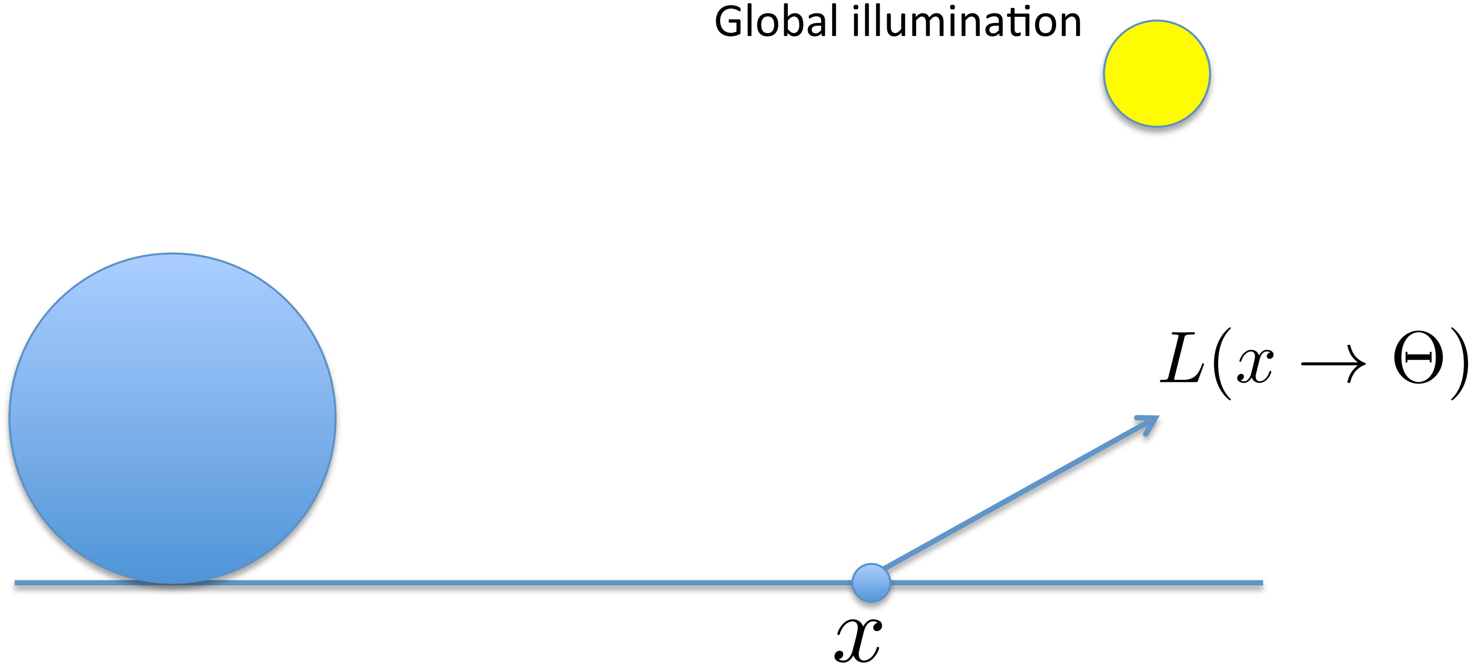


The rendering equation

$$L(x \to \Theta) = L_{direct}(x \to \Theta) + L_{indirect}(x \to \Theta)$$

Local illumination

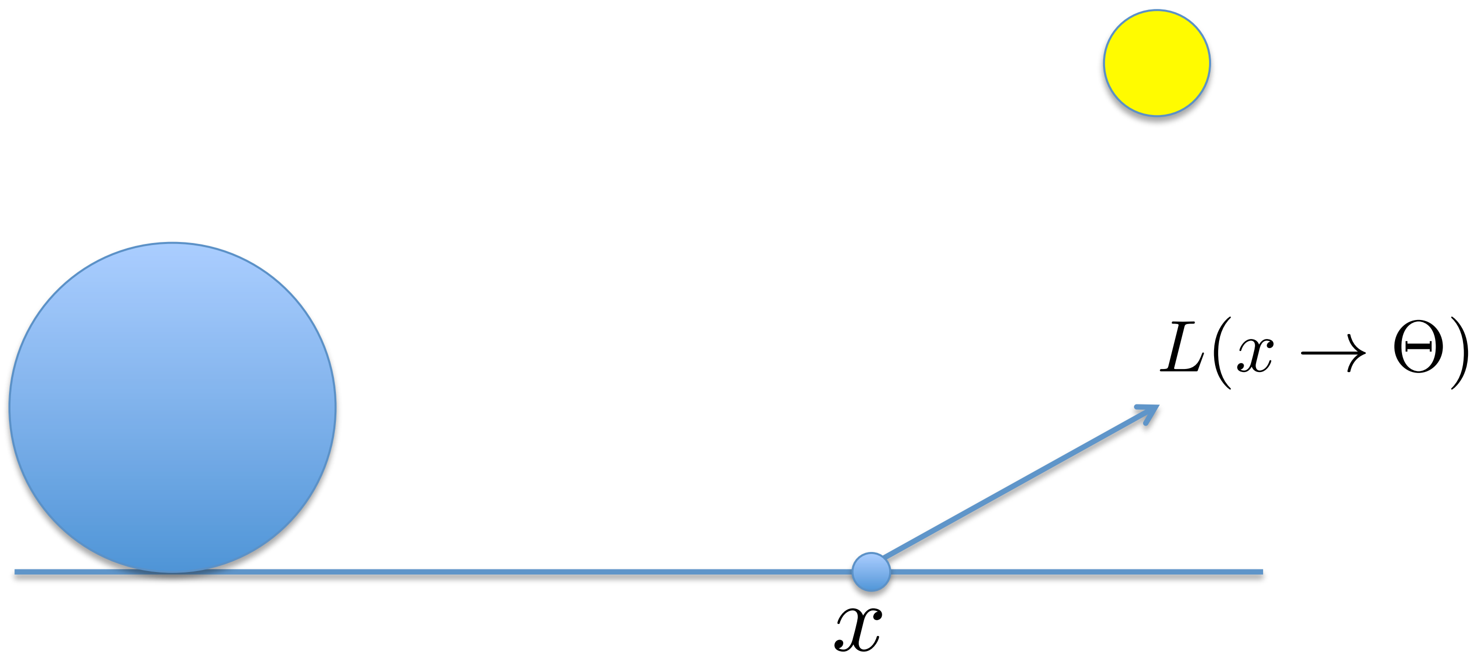
Global illumination



The rendering equation

$$L_{indirect}(x \rightarrow \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)\, \mathrm{d}\omega_{\Psi}$$

$$L_{direct}(x \rightarrow \Theta) = L_{direct}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$



The rendering equation

$$L_{indirect}(x \to \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi) \, \mathrm{d}\omega_{\Psi}$$

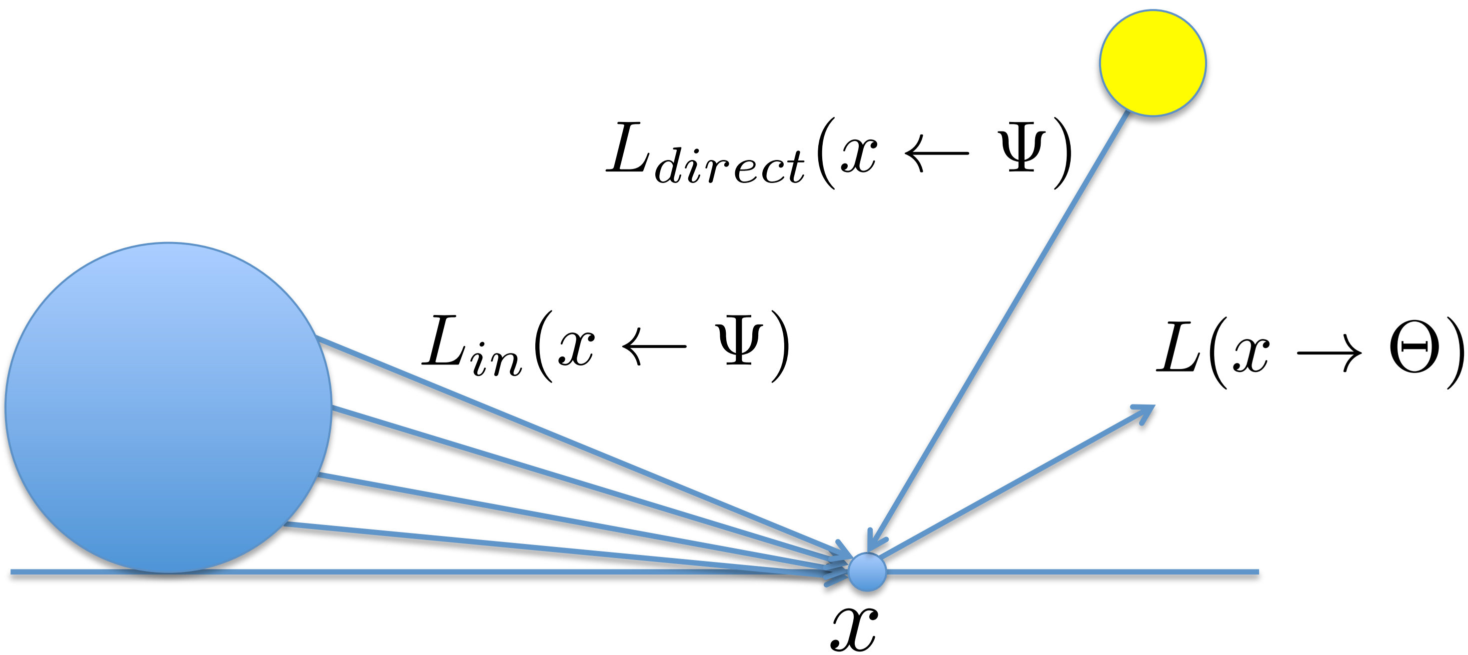
- Integral over the hemisphere!
- Need to use Monte Carlo sampling.

$$L_{indirect}(x \to \Theta) \approx \frac{1}{n} \sum_{i=1}^{n} \frac{L_{in}(x \leftarrow \Psi_i) f_r(x, \Psi_i \leftrightarrow \Theta) cos(N_x, \Psi_i)}{p(\Psi_i)}$$

The rendering equation

$$L_{indirect}(x \rightarrow \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)\, \mathrm{d}\omega_{\Psi}$$

$$L_{direct}(x \rightarrow \Theta) = L_{direct}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

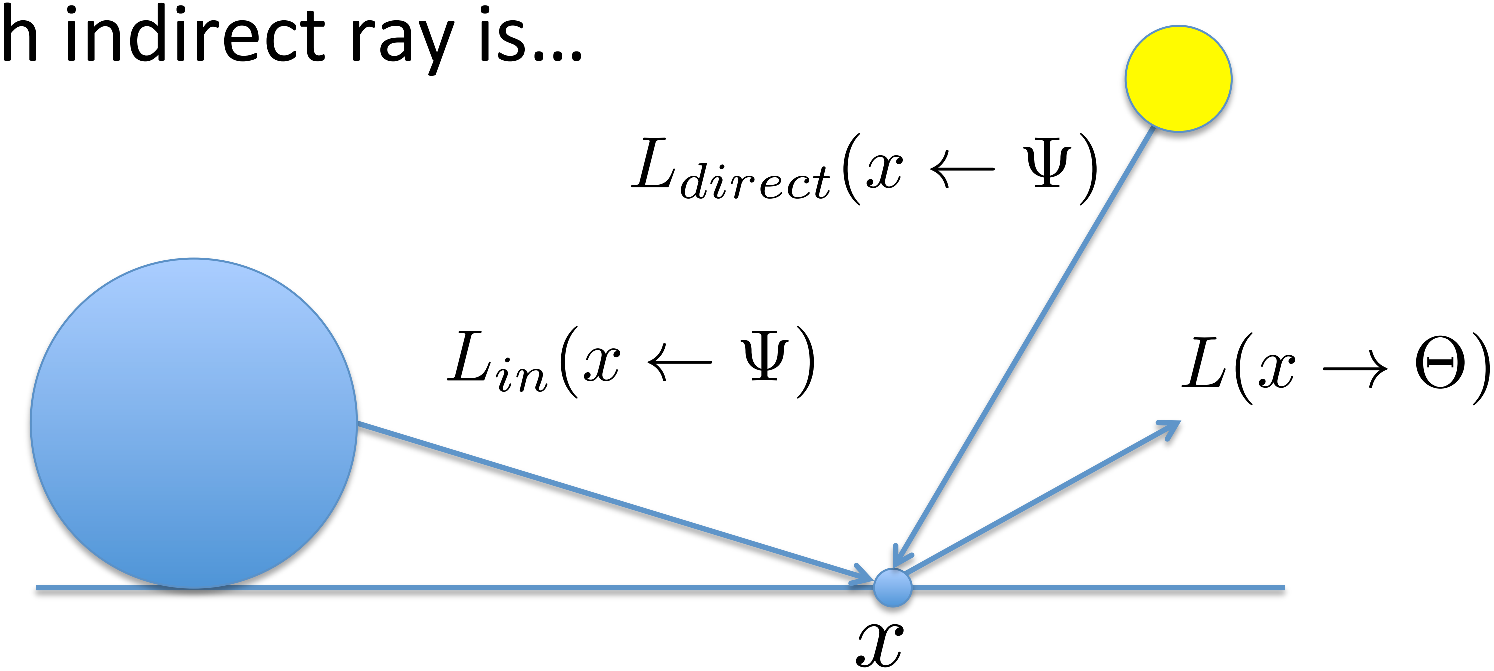


The rendering equation

$$L_{indirect}(x \to \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi) \, \mathrm{d}\omega_\Psi$$

$$L_{direct}(x \to \Theta) = L_{direct}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

Each indirect ray is…

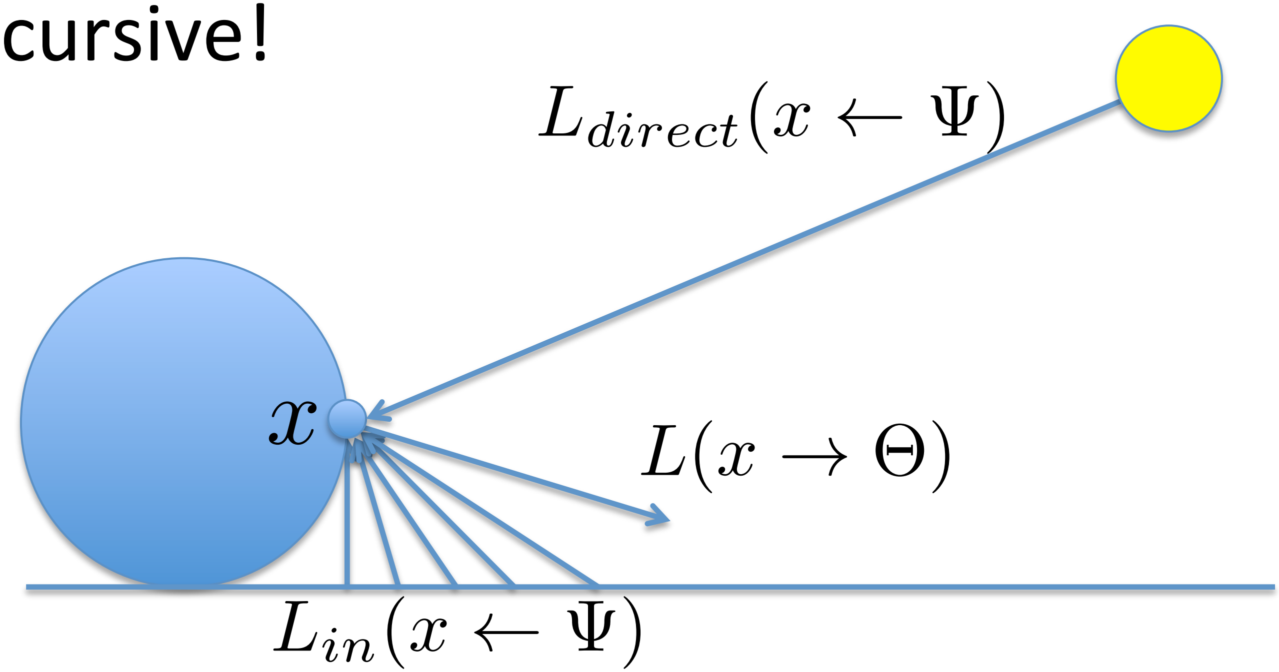


The rendering equation

$$L_{indirect}(x \rightarrow \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)\, \mathrm{d}\omega_{\Psi}$$

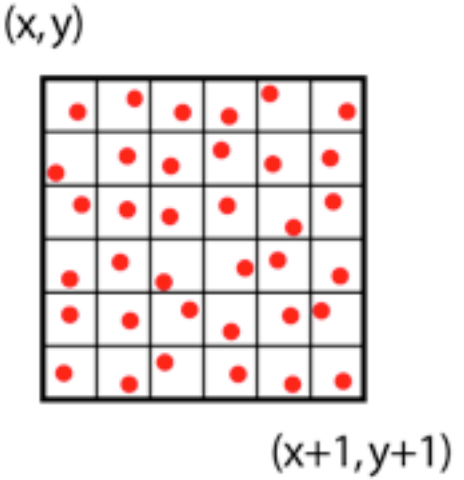
$$L_{direct}(x \rightarrow \Theta) = L_{direct}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

…recursive!



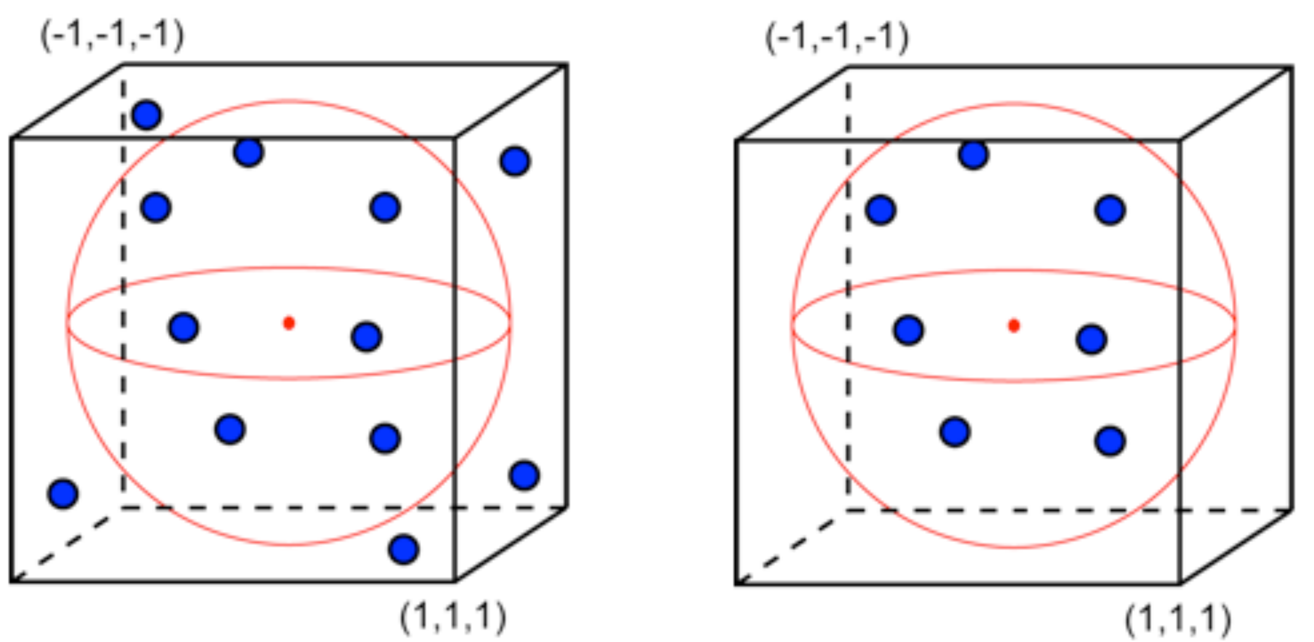
Path tracing

- Want to avoid n^k rays after k bounces (each ray contributing less to the image).
- In path tracing, we trace n rays through each pixel and randomize its path.
- Sample indirect illumination in a single random direction.
  - Noisy with few samples per pixel!



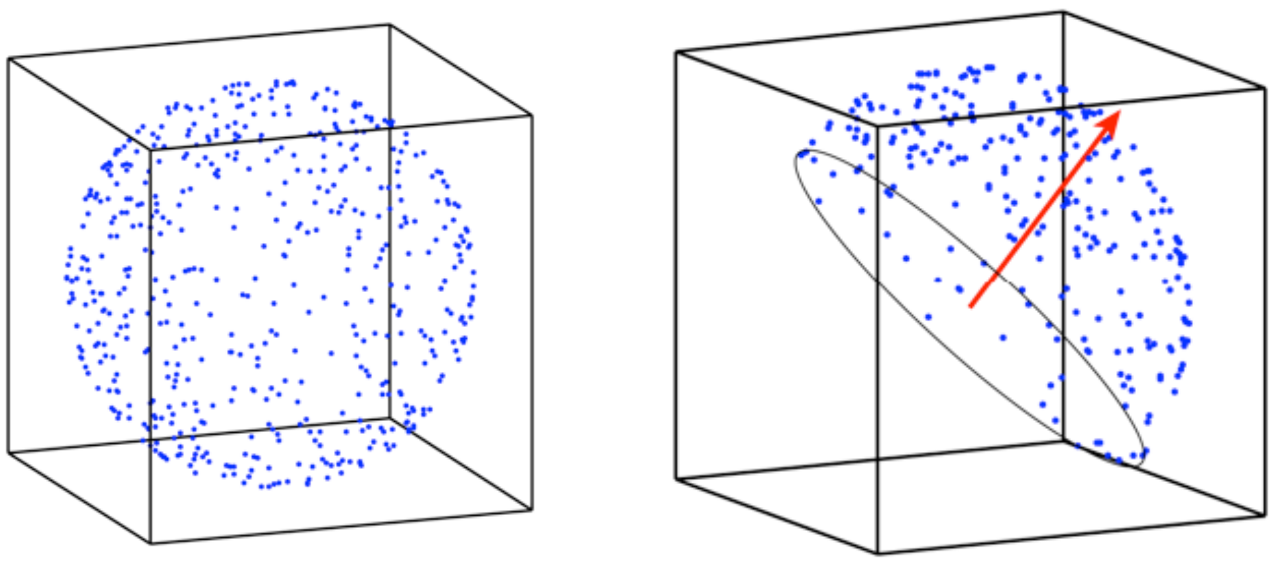
Random direction

- Rejection sampling:
  – Sample uniformly in the unit cube.
  – Discard samples outside the sphere.



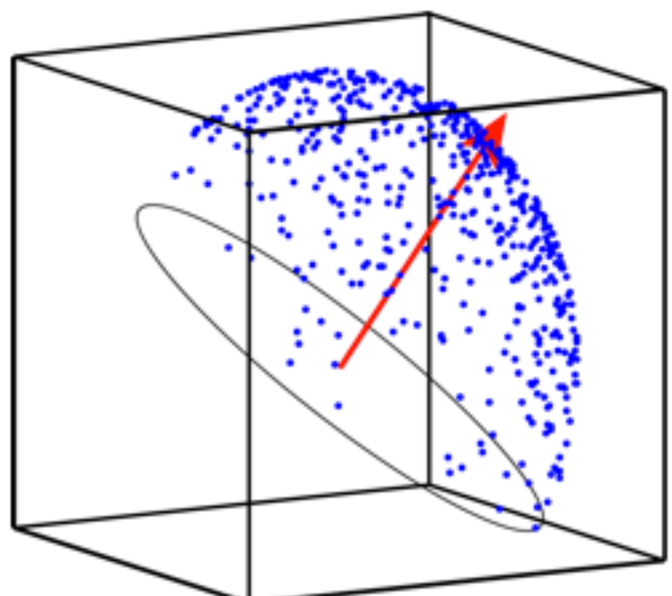
Random direction

- Normalize to get samples on the sphere.
- Discard samples behind the normal.

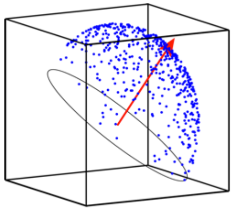


Importance sampling

- We can do smarter sampling to increase efficiency (reduce noise).
- In the assignment you will do cosine-weighted importance sampling.



Cosine weighted sampling



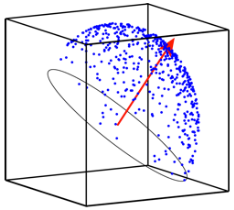
- Rendering equation:

$$L_{indirect}(x \rightarrow \Theta) = \int_{\Omega} L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)\, \mathrm{d}\omega_{\Psi}$$

- Point sampled rendering equation:

$$L_{indirect}(x \rightarrow \Theta) \approx \frac{1}{n} \sum_{i=1}^{n} \frac{L_{in}(x \leftarrow \Psi_i) f_r(x, \Psi_i \leftrightarrow \Theta) cos(N_x, \Psi_i)}{p(\Psi_i)}$$

Cosine weighted sampling



- Point sampled rendering equation:

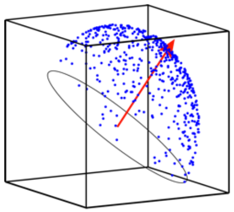
$$\frac{1}{n}\sum_{i=1}^{n}\frac{L_{in}(x \leftarrow \Psi_i) f_r(x, \Psi_i \leftrightarrow \Theta) cos(N_x, \Psi_i)}{p(\Psi_i)}$$

- Get equal contribution of each ray by setting :

$$p(\Psi_i) = k \cdot cos(N_x, \Psi_i)$$

- Eliminates the cosine term!

Cosine weighted sampling



- Cumulative Distribution Function:

$$F(\theta,\phi) = \int_0^{\phi}\int_0^{\theta} p(\theta,\phi) sin(\theta)\,\mathrm{d}\theta\mathrm{d}\phi = \frac{\phi}{2\pi}(1 - cos^2(\theta))$$

- Separate:

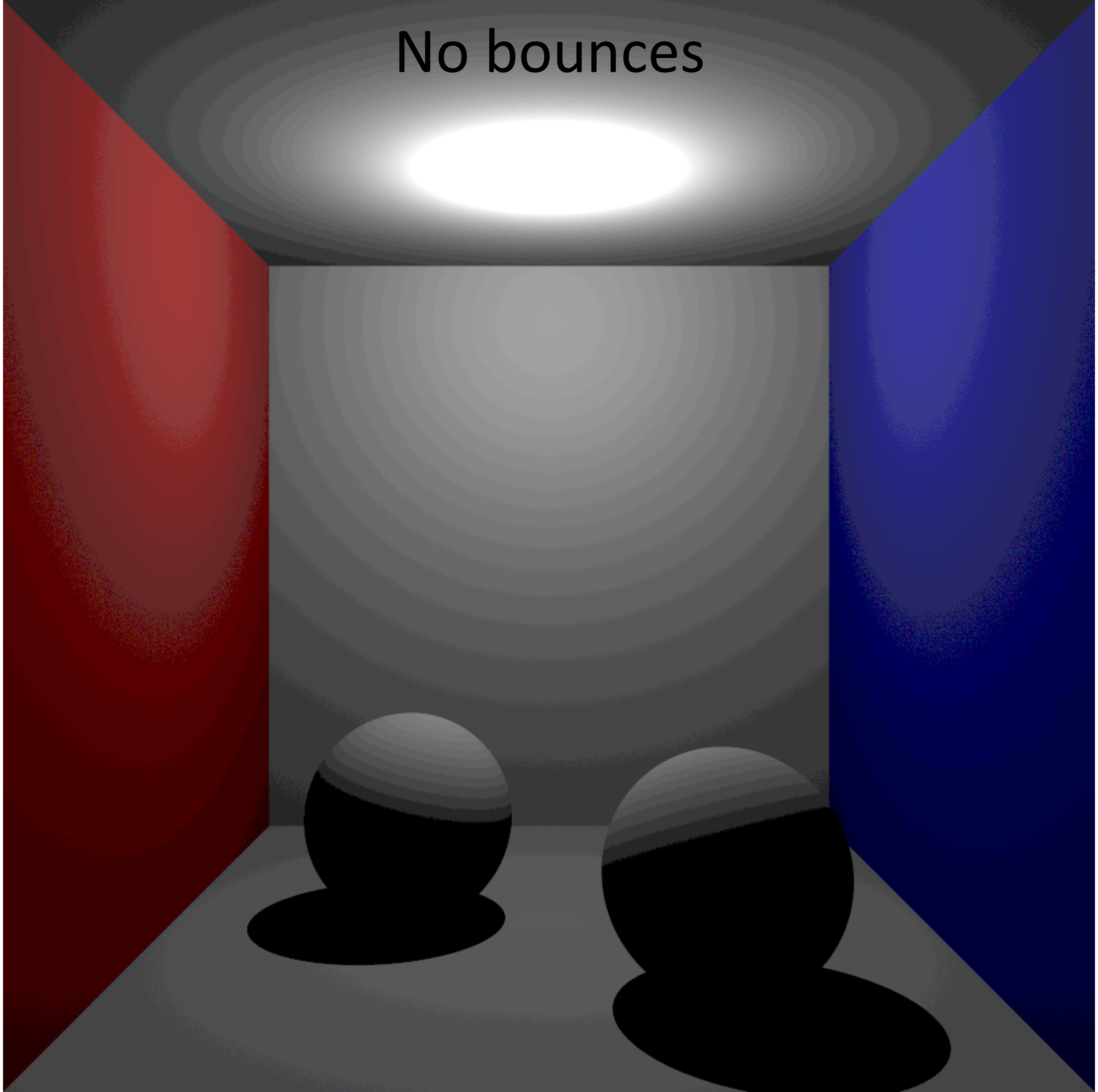
$$\begin{aligned} F_\theta &= 1 - cos^2(\theta) \\ F_\phi &= \frac{\phi}{2\pi} \end{aligned}$$

- Solve:

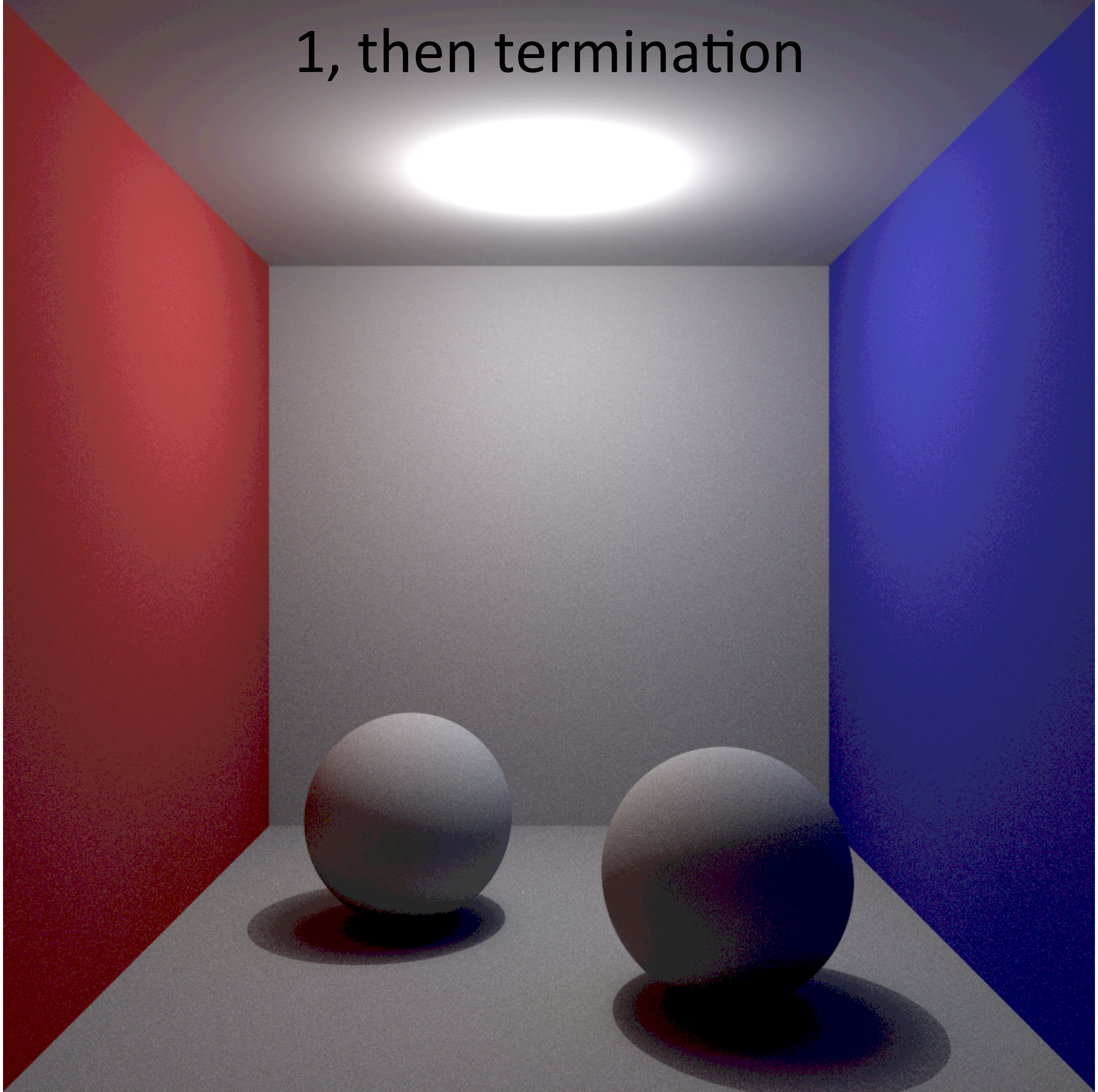
$$\begin{aligned} \theta &= cos^{-1}\sqrt{1 - u_\theta} \\ \phi &= 2\pi u_\phi \end{aligned}$$

How many indirect recursions/ bounces?

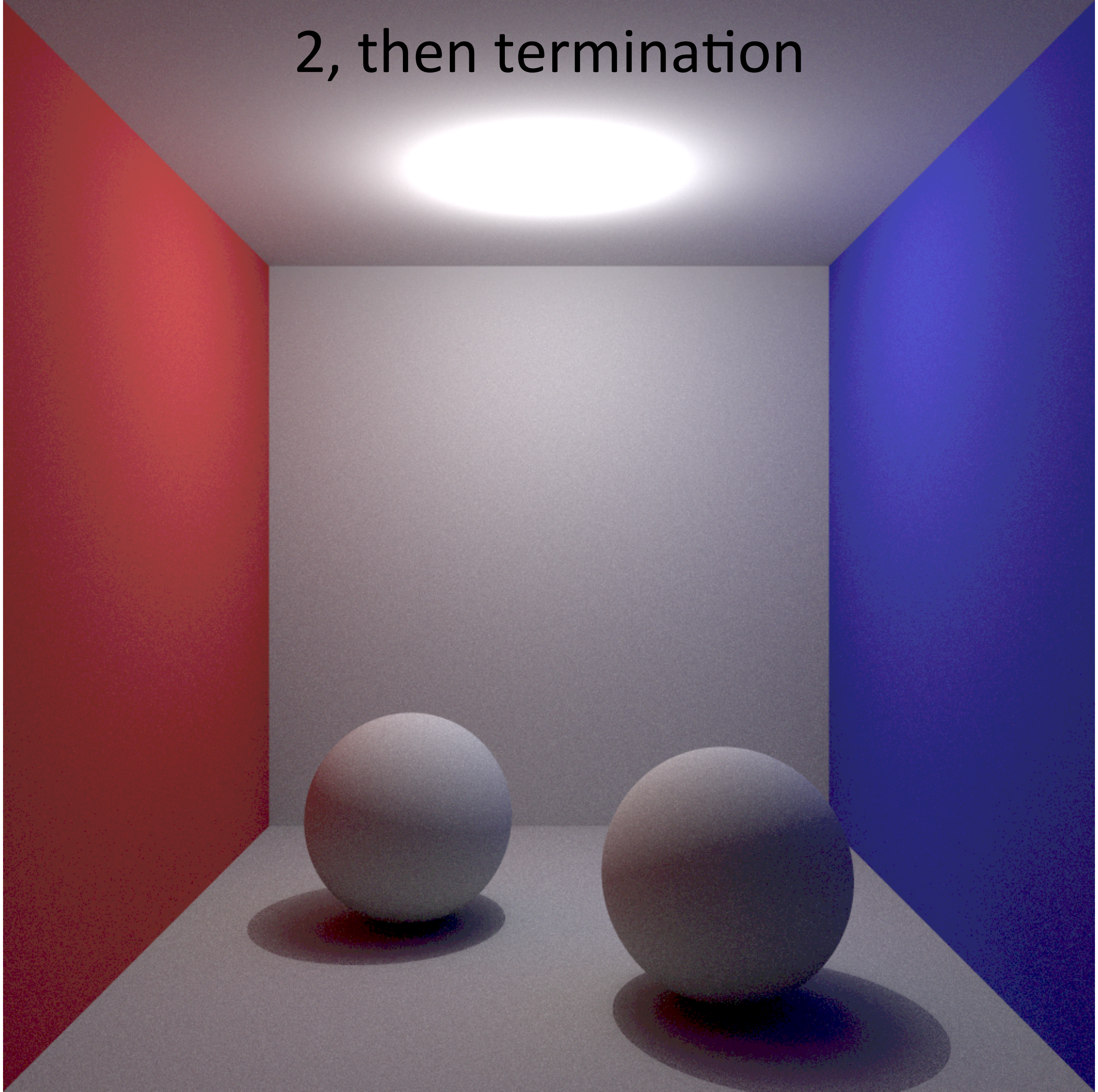
No bounces



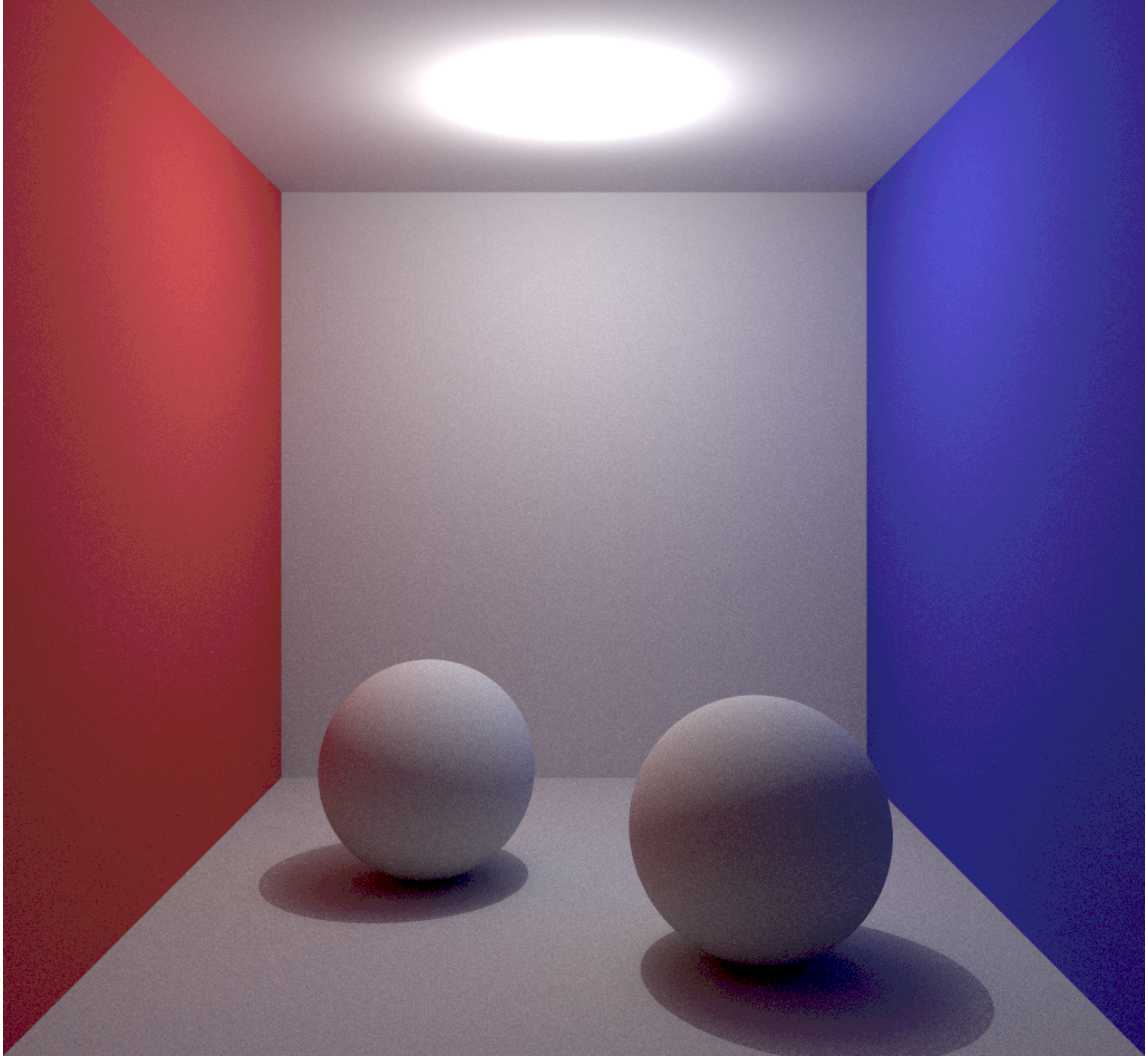
1, then termination

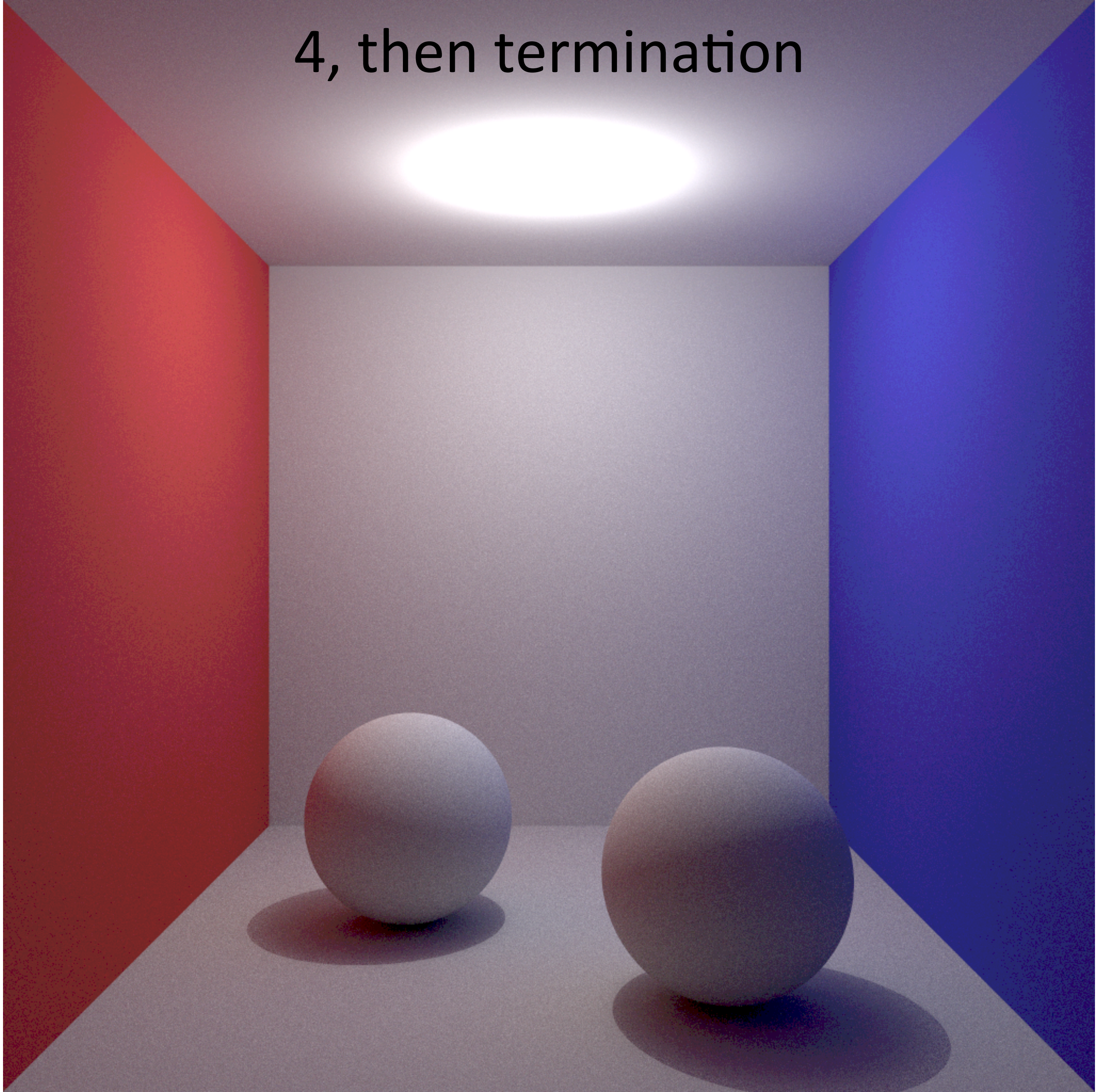


2, then termination



3, then termination




4, then termination

Termination criterion

- Fixed depth termination: biased.
  - Image wont converge to correct solution with more samples per pixel.
- Russian roulette termination: unbiased.
  - Image will eventually be correct.
- We don’t want bias!

Russian roulette

- Noise instead of bias!
- No fixed depth cut-off.
- Absorption (termination) probability α.
- If not absorbed:
  - Multiply contribution with 1/(1-α).

Russian roulette

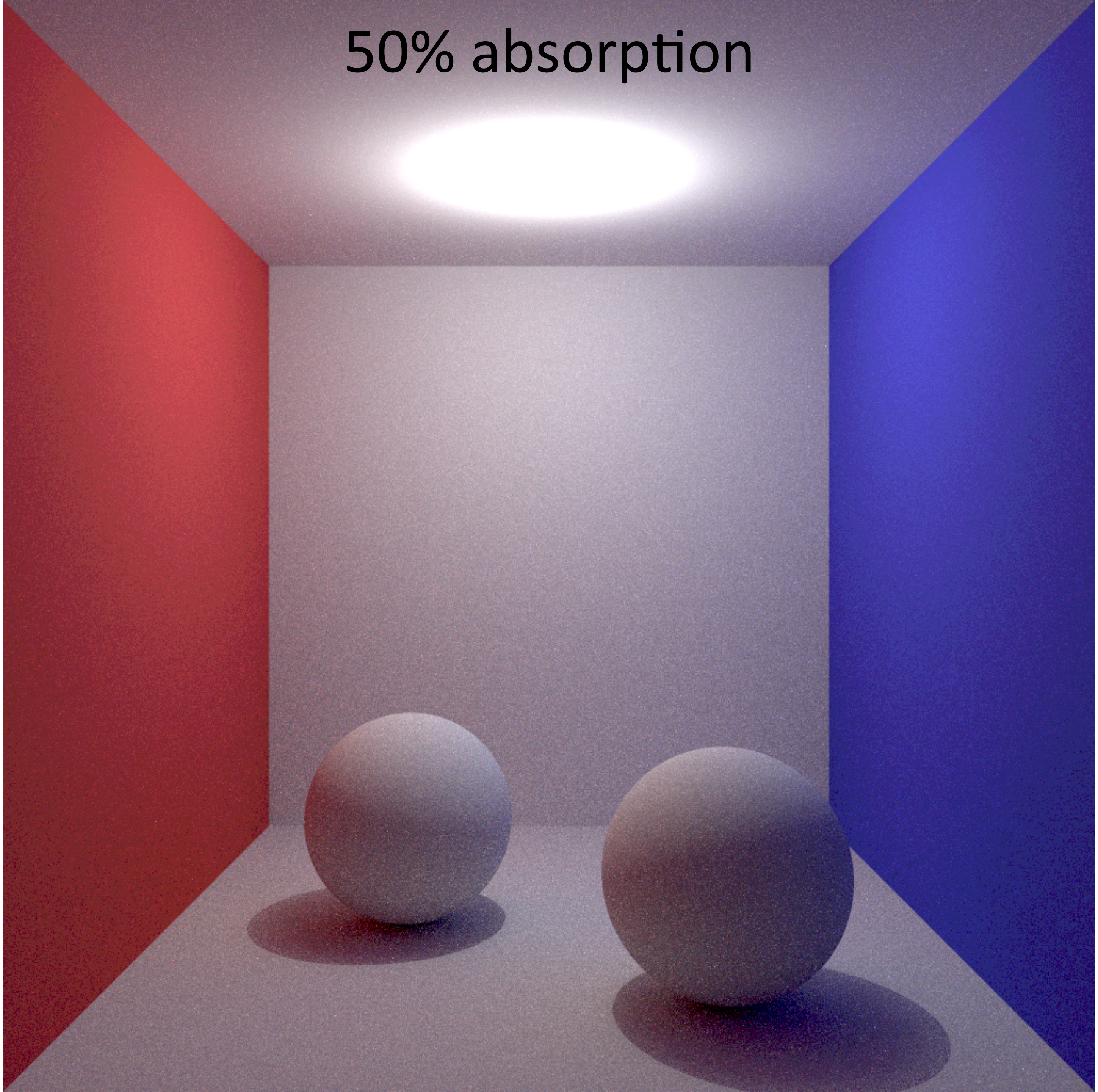
Example:

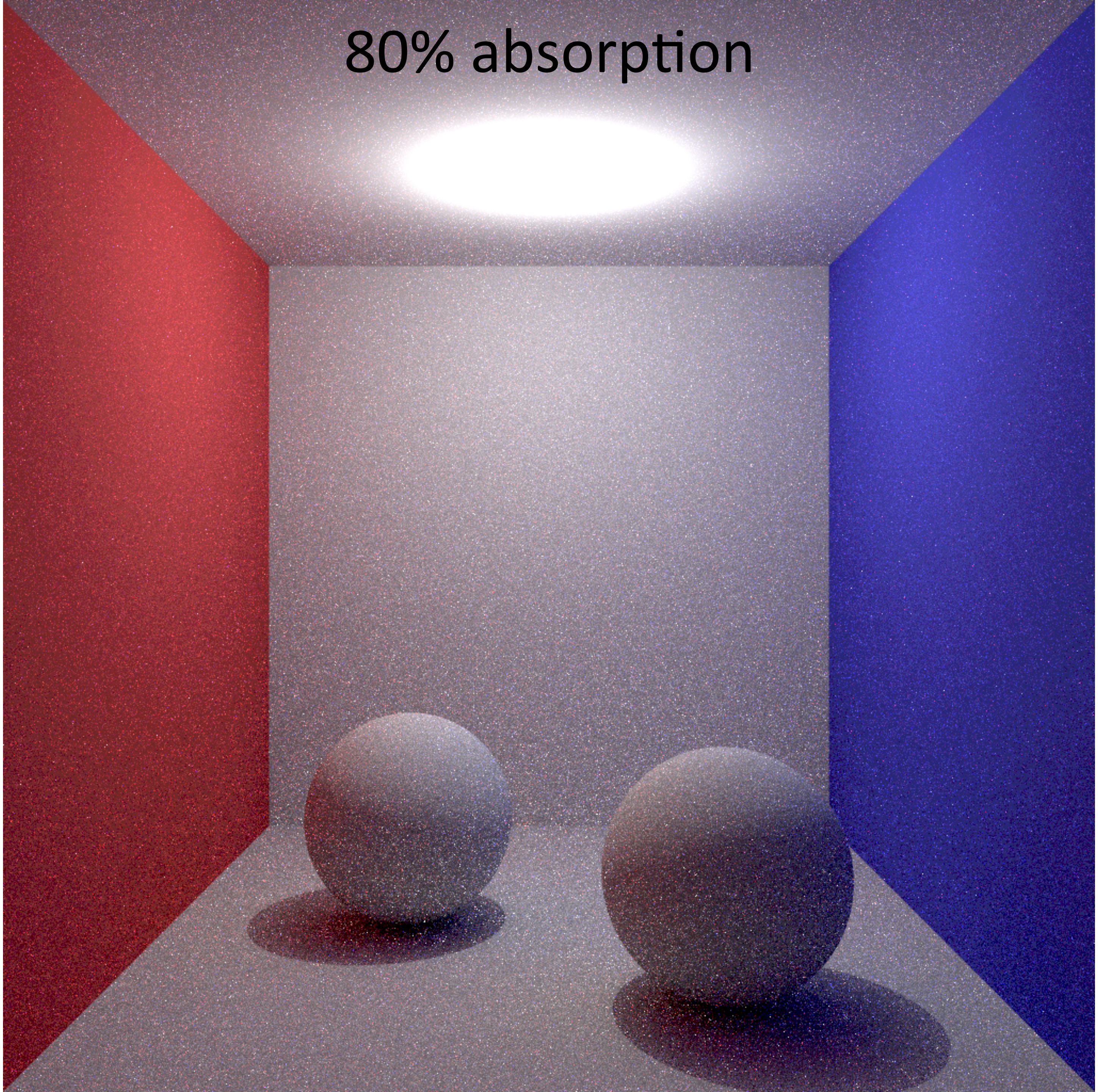
- Trace 1000 rays against white background.
- Absorption probability 0.1.
- 0.1*1000 rays get absorbed (black).
- 0.9*1000 rays lives (white).

$$color = \frac{1}{n}(0.1 \cdot 1000 \cdot (0,0,0) + \frac{0.9 \cdot 1000}{1 - 0.1} \cdot (1,1,1))$$

$$color = (1,1,1)$$

50% absorption

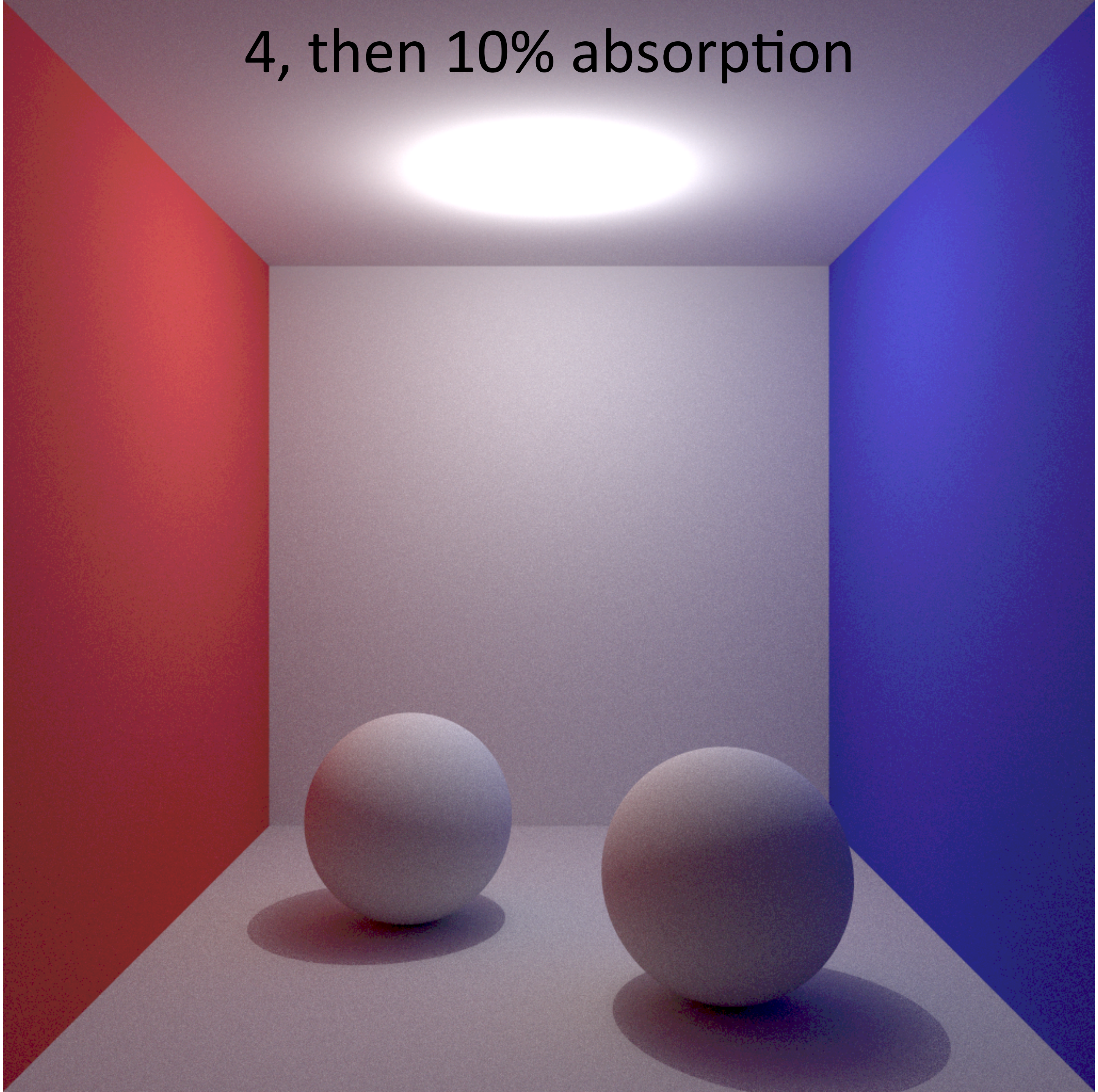



80% absorption

Russian roulette

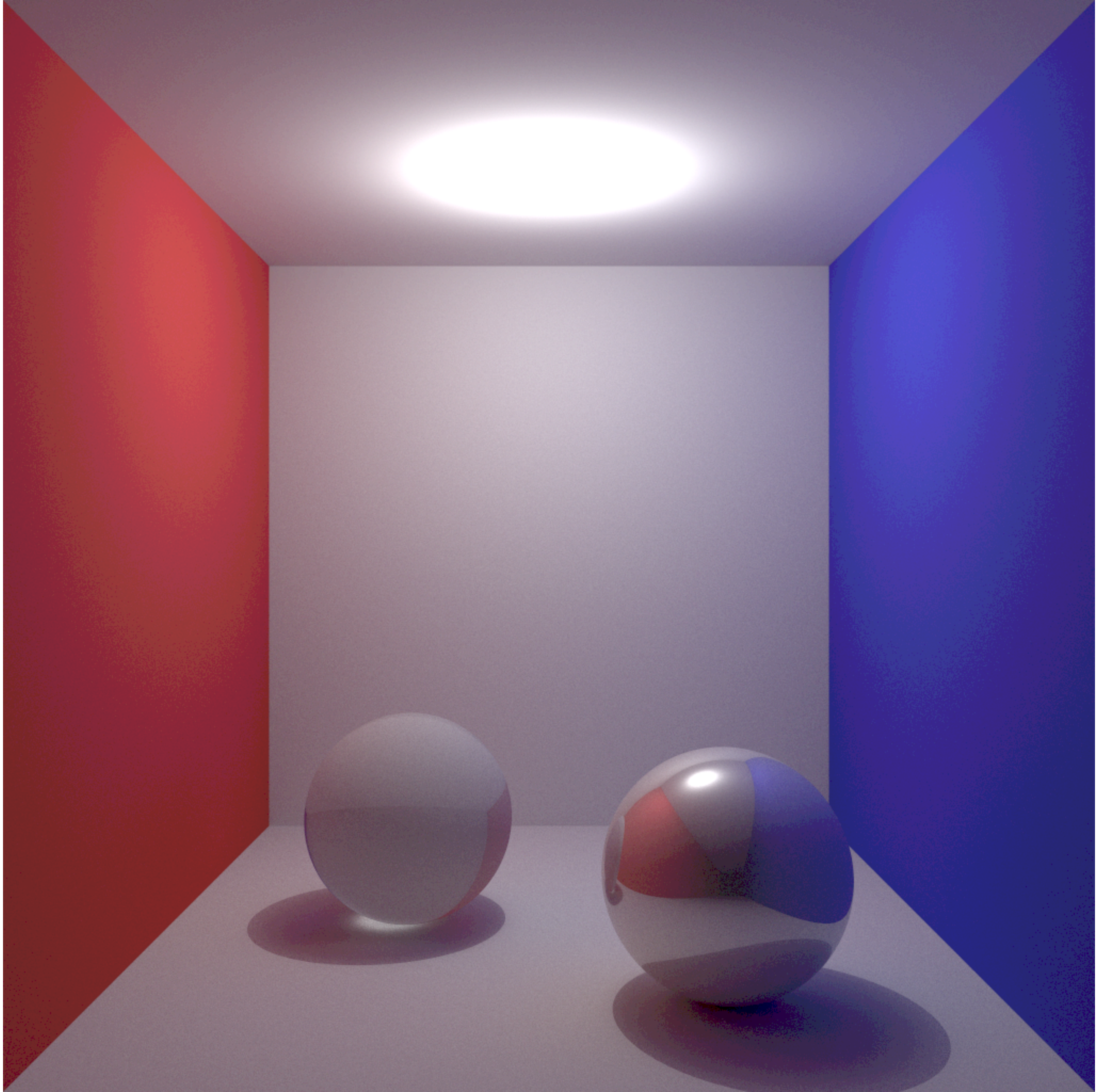
- Good to force a few recursions before starting roulette.
  - Reduces noise!

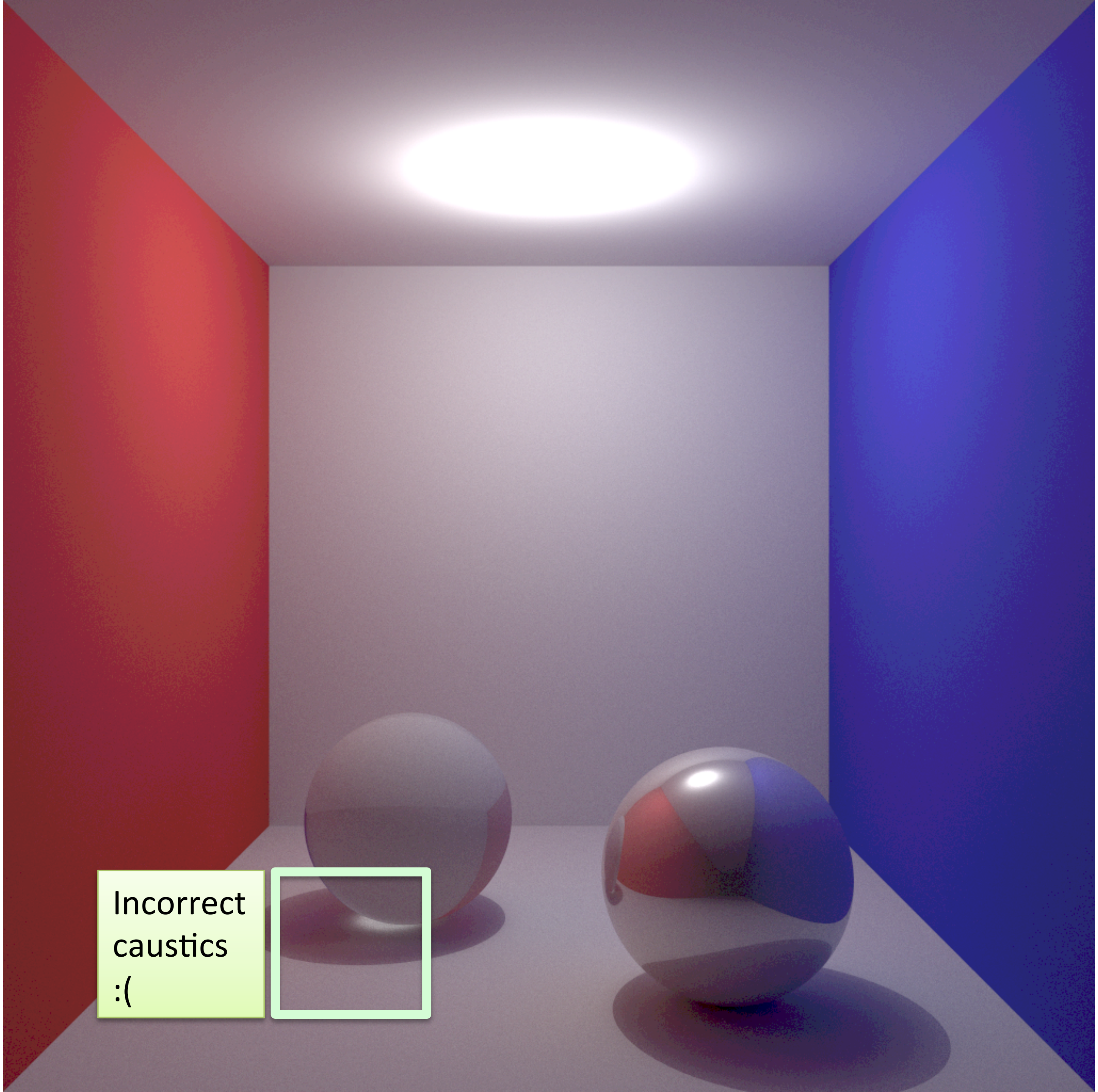
4, then 10% absorption



Reflection and refraction

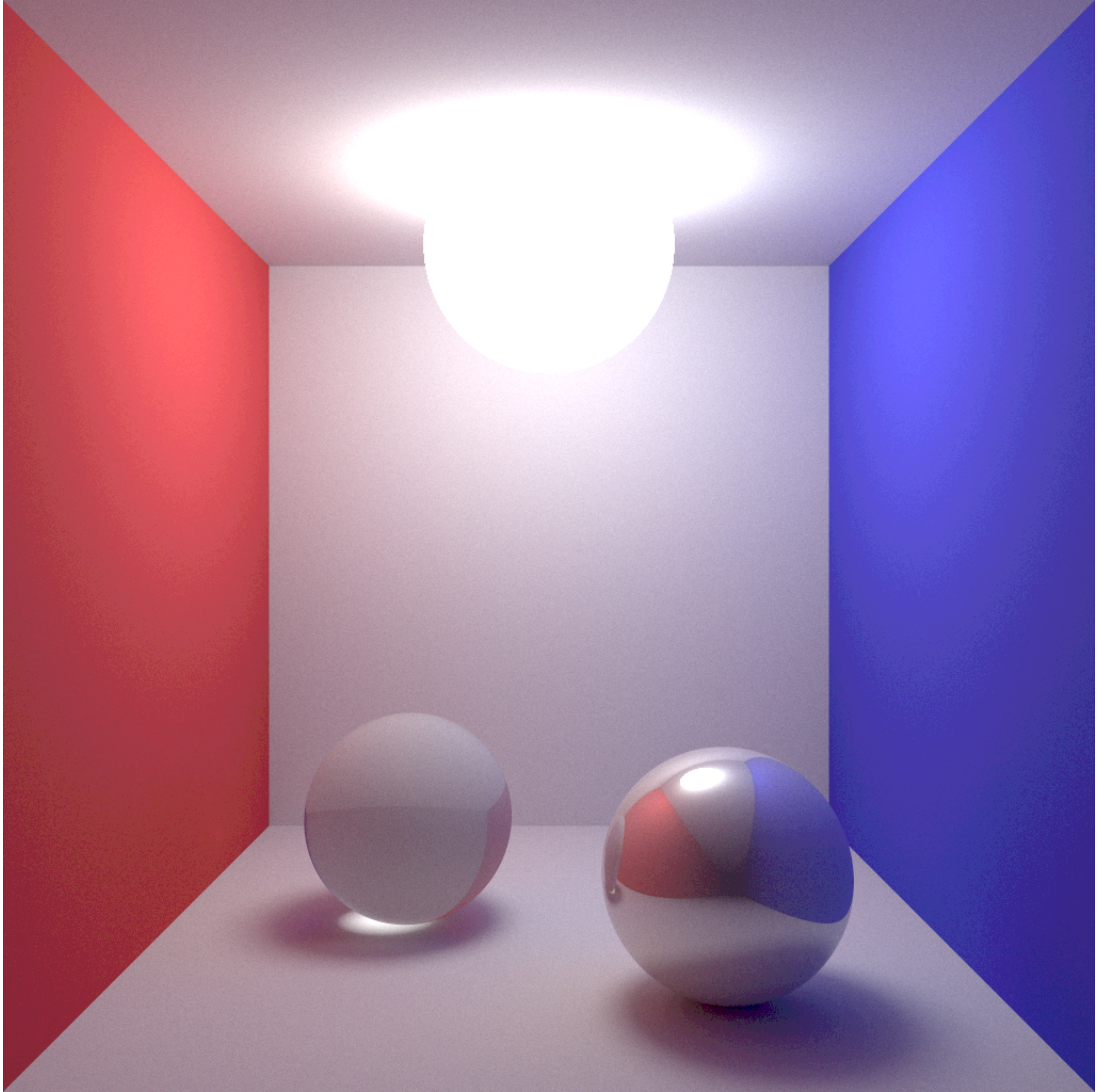
- Almost like previous assignments.
- Reflectivity, R, and Transparency, T.
  - $R+T < 1$
- Use russian roulette to pick one:
  - Reflection (probability R).
  - Refraction (probability T).
  - Direct and indirect illumination (probability 1-R-T).

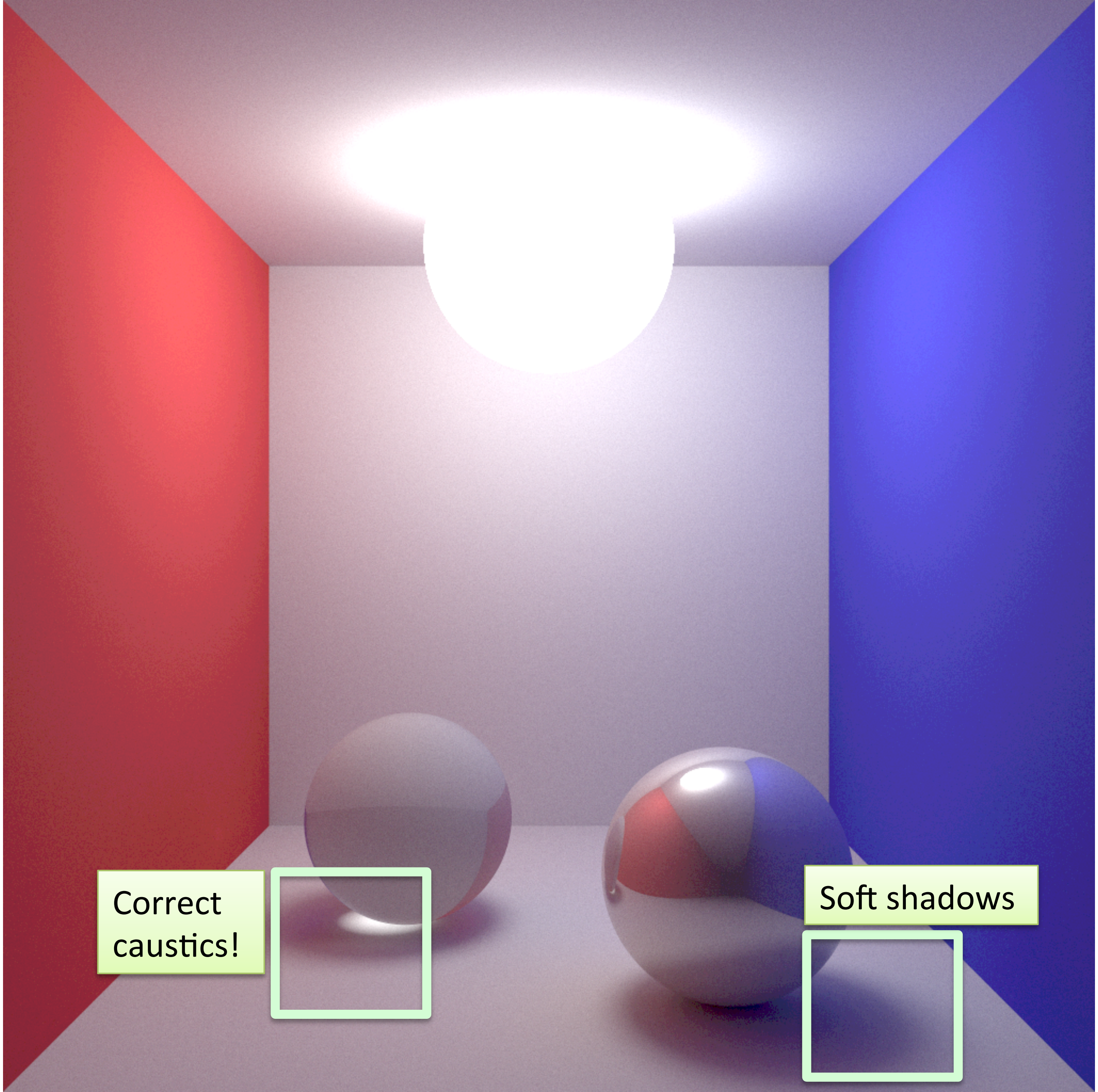



Incorrect
caustics
:(

Area lights

- There are no point lights in real life.
- Area lights provides softer shadows.
- Easy to implement.
  - Sphere with emissive material!
- The light only contributes to the image if a ray actually hits the light source.
  - Size matters.




Correct caustics!
Soft shadows

Sky light

- Remove the walls and roof.
- Use the background as light source.

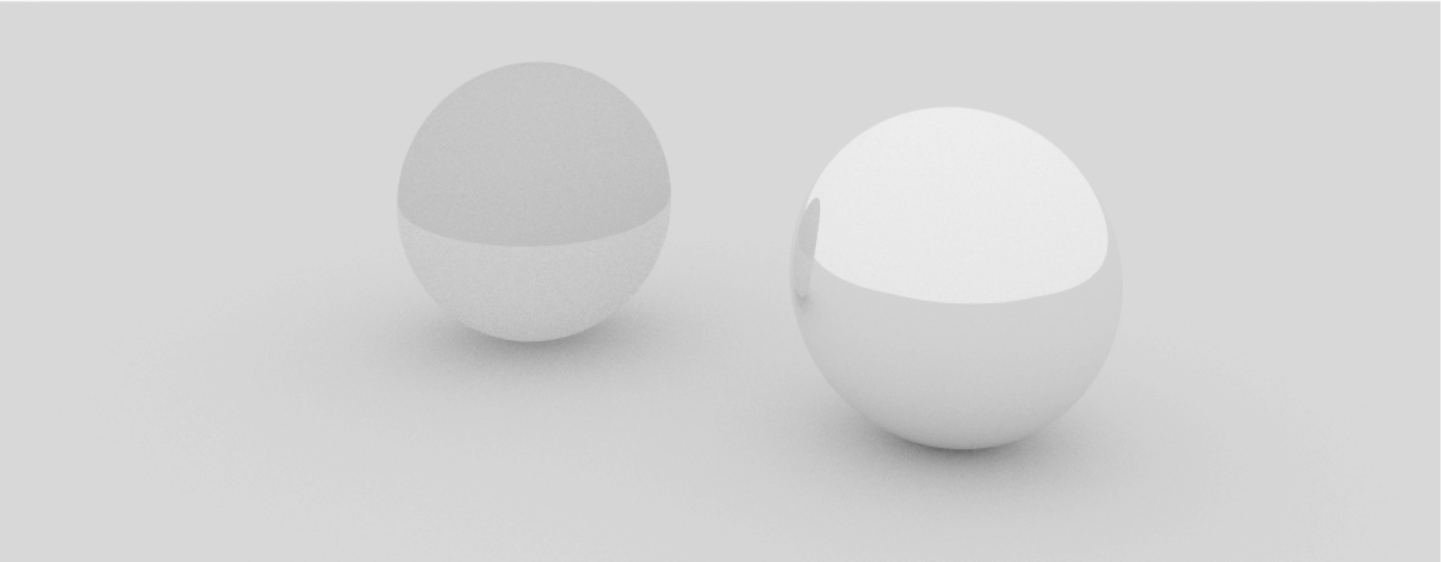


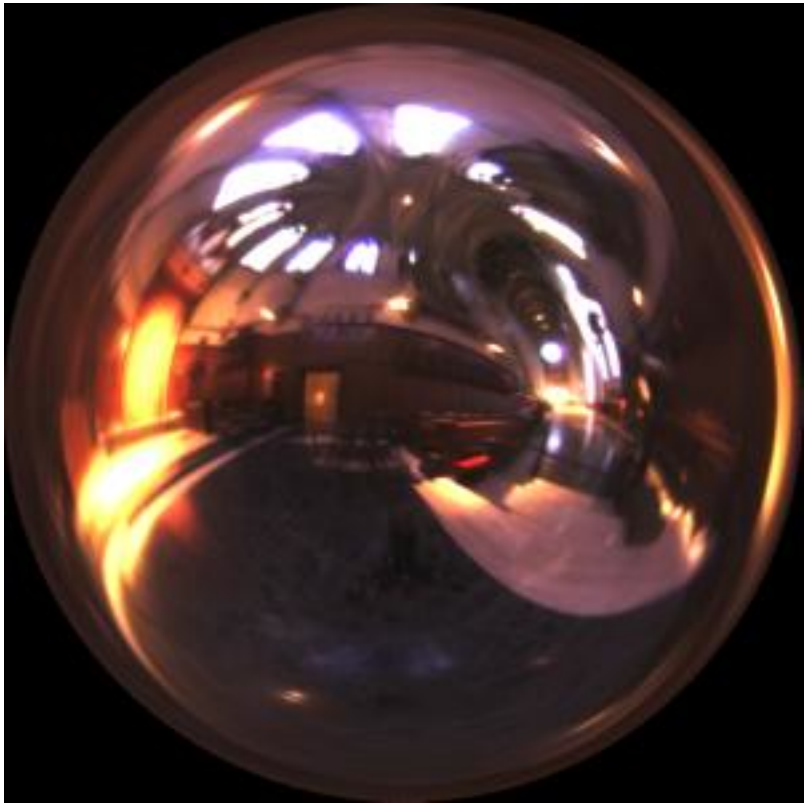
Image based lighting

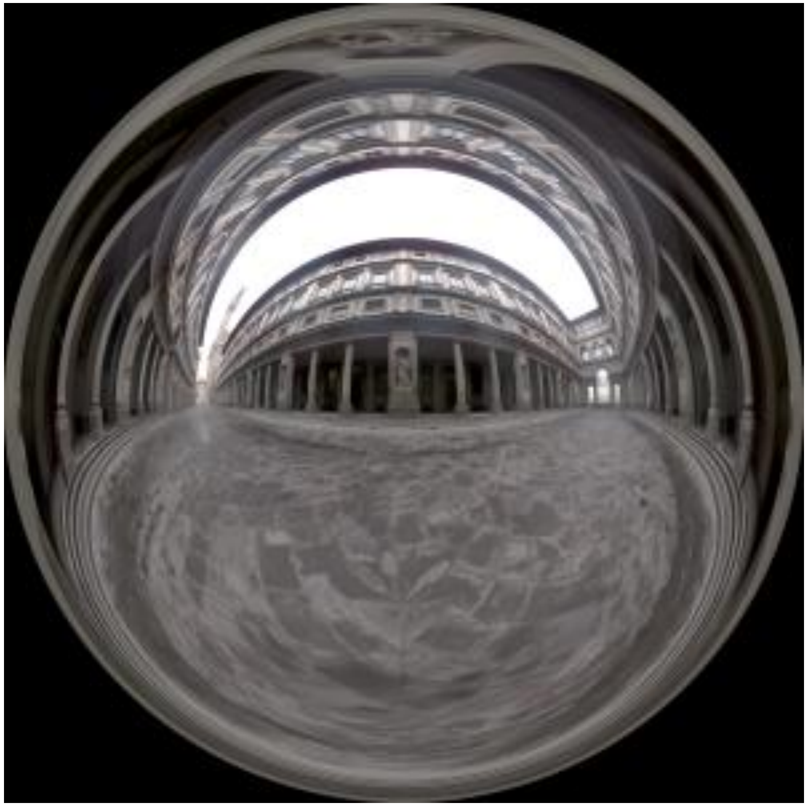
- Instead of background color – use light probe.
- More interesting images.
- Makes the objects blend into the environment.

Creating a light probe

- Light probes are often created by photographing a real world scene.
  - Can also be pre rendered.
- Two pictures of a mirrored ball at ninety degrees of separation.
- Spherically encoded.
  - Center of the image is straight forward, the circumference of the image is straight backwards.

Examples

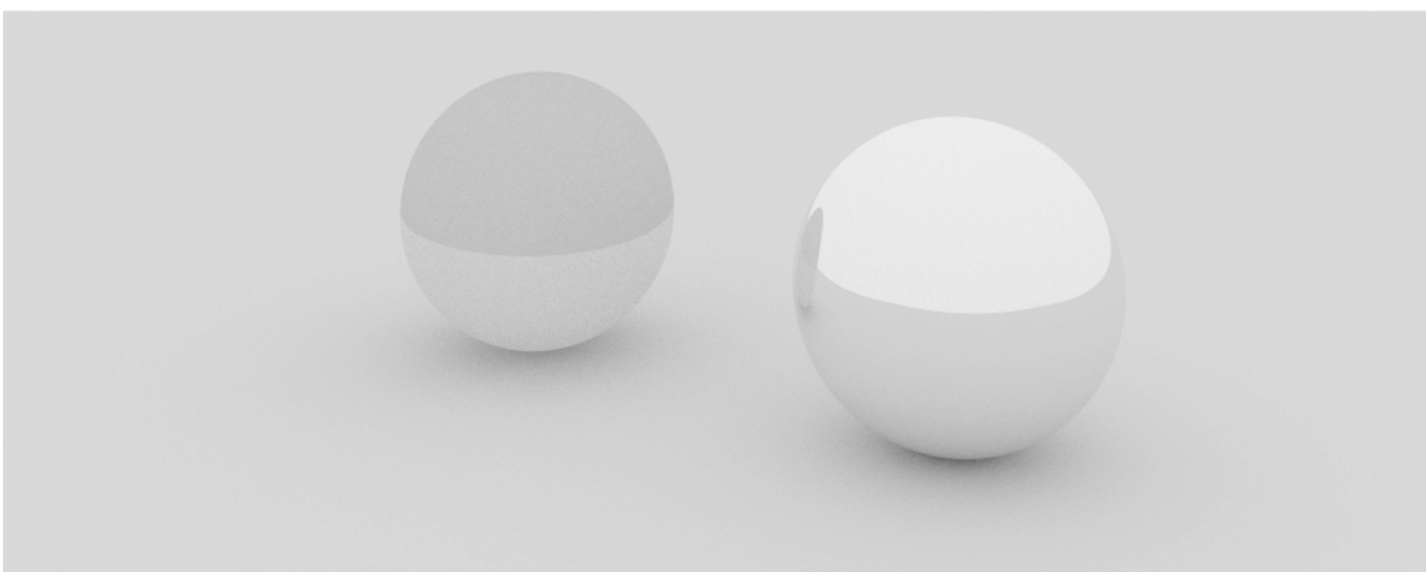




From www.pauldebevec.com/Probes

Portable Float Map

- To store a light probe we use PFM.
  - We used the more advanced OpenEXR format last year but it was difficult to compile at some platforms.
- Basically a uncompressed image format where each color channel is 32-bit float.
  - High Dynamic Range!



Loading and sampling a lightprobe

- Support in the framework:
  - LightProbe(const std::string& filename);
- Sample radiance in direction:
  - Color getRadiance(const Vector3D& d) const;
- Use for rays that misses geometry (hits the background).
  - Automatically becomes a light source!

Multicore support

- Distribute work among multiple CPU cores.
- Ray tracing can generally compute each ray independent of each other.
  - Lends itself well to parallelization.
- More suitable to distribute tiles or rows to avoid scheduling overhead.

OpenMP

- Parallel computeImage:

```
int lines = 0;

#pragma omp parallel for
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        Color c = tracePixel(x,y);
        mImage->setPixel(x,y,c);
    }
    #pragma omp critical
    {
        lines++;

        if (lines % (height/20) == 0 || lines == height)
            std::cout << (100*lines/height) << ”%" << std::endl;
    }
}
```

The end