Rendering: Path Tracing I

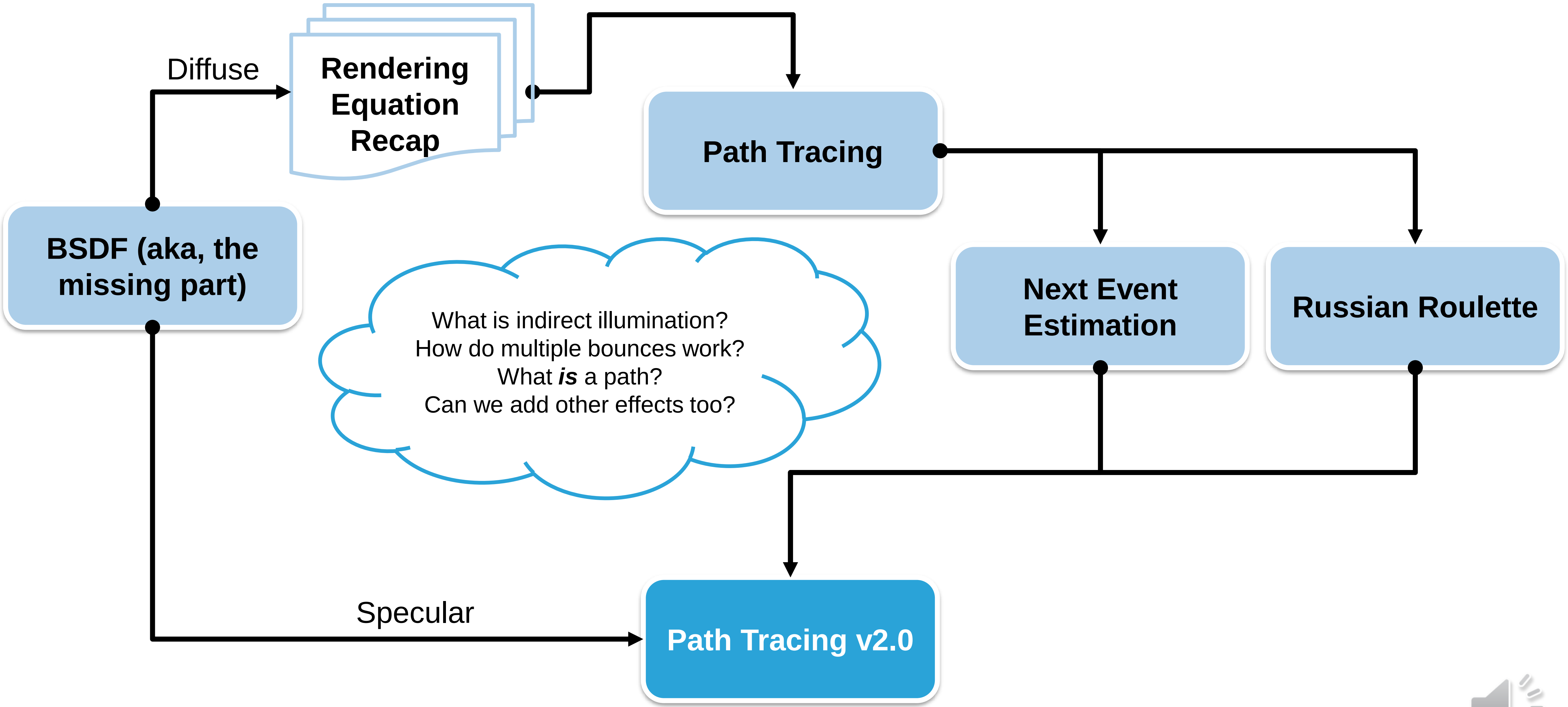
Bernhard Kerbl

Research Division of Computer Graphics
Institute of Visual Computing & Human-Centered Technology
TU Wien, Austria

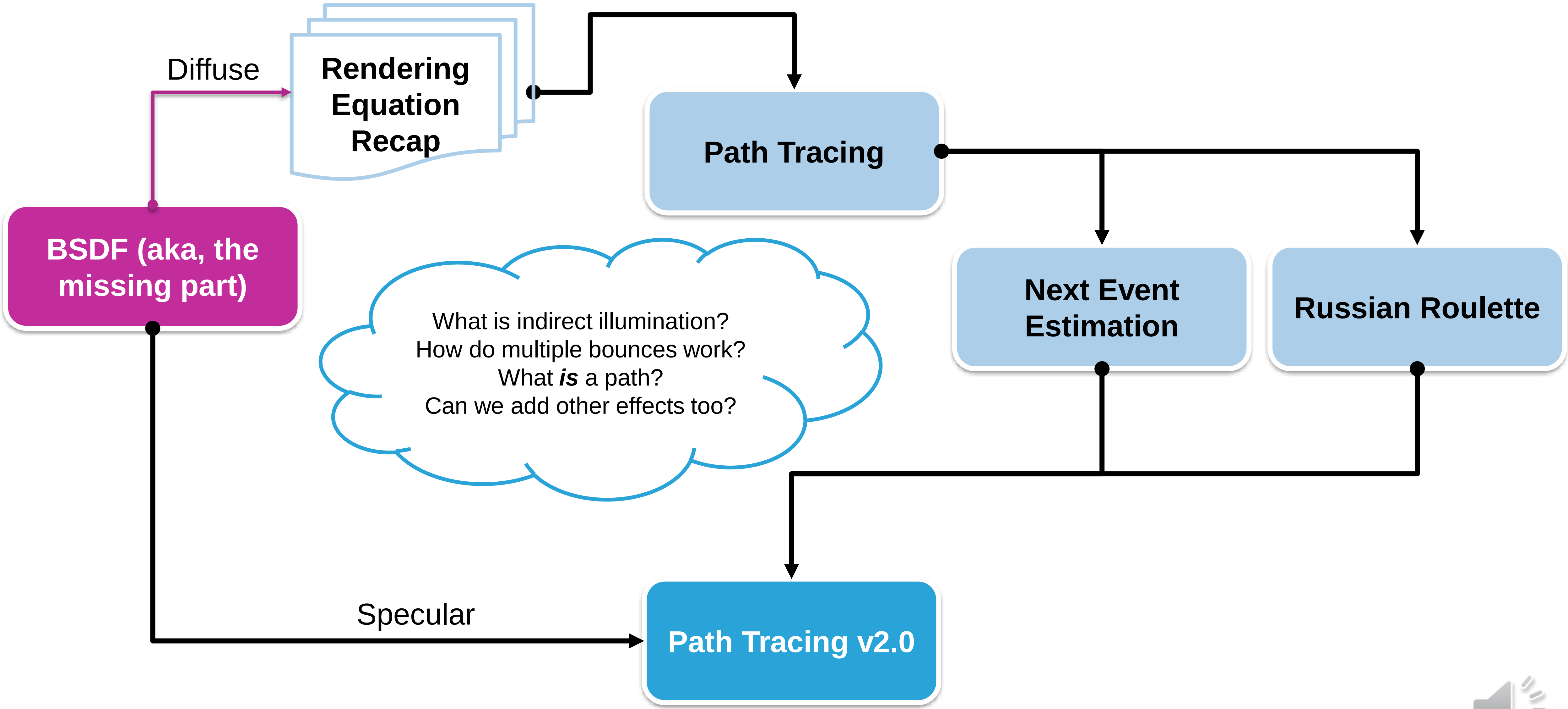
Today’s Goal

- Add the last missing piece, the BSDF (simple version)
- Finally, we will generate some great-looking images by putting together all the things we learned:
  - Light Physics
  - Monte Carlo Integration
  - The Rendering Equation
  - The Path Tracing Algorithm
- We will also check out ways to make the procedure fast and stable

Today's Roadmap



Today's Roadmap



The Missing Part of the Rendering Equation

$$L_e(x,v) = E(x,v) + \int_\Omega \boxed{f_r(x,\omega \to v)} L_i(x,\omega)\cos(\theta_x)\,\mathrm{d}\omega$$

- Bidirectional Scattering Distribution Function (BSDF)
- Describes the light transport properties of the material
- So far, we avoided this term or replaced it with constant factors
- Can model reflections, refractions, volumetric scattering…

Bidirectional Reflectance Distribution Function (BRDF)

- Considers only the **reflection** of incoming light onto a surface
  - The BRDF is a limited instance of the full BSDF (e.g., no transparency)
  - Good for starting out, complex materials need full BSDF
  - More on that in another lecture

- A BRDF function $f_r(x, \omega_i \rightarrow \omega_o)$ with input directions ω_i, ω_o
  - uses convention: ω_i and ω_o are assumed to point away from x
  - How much irradiance from ω_i is reflected as radiance to ω_o at x?

Bidirectional Reflectance Distribution Function (BRDF)

- "How much irradiance from ω_i is reflected as radiance to ω_o at x?"

- $f_r(x, \omega_i \rightarrow \omega_o) = \frac{dL_i(x, \omega_o)}{dE_i(x, \omega_i)} = \frac{dL_i(x, \omega_o)}{L_i(x, \omega_i) \cos_\theta(\omega_i)\, d\omega_i}$

$$L_e(x, v) = E(x, v) + \int_\Omega f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x)\, \mathrm{d}\omega$$

- **Helmholtz reciprocity**: $f_r(x, \omega_i \rightarrow \omega_o) = f_r(x, \omega_o \rightarrow \omega_i)$

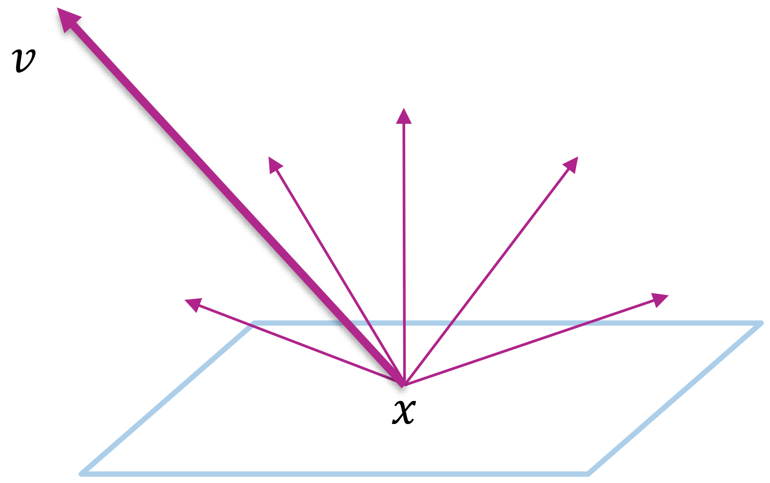
- **Conserves energy**: $\int_\Omega f_r(x, \omega \rightarrow v) \cos\theta\, d\omega \leq 1\ \forall\, v$

Condition for Energy Conservation

- Why must the BRDF f_r fulfill $\int_\Omega f_r(x, \omega \to v) \cos_\theta(\omega)\, d\omega \leq 1$?

- Intuitive interpretation with **reciprocity**: Shine a laser light along $-v$ onto x. We must have $\int_\Omega f_r(x, v \to \omega) \cos_\theta(\omega)\, d\omega \leq 1$

- If we find a direction v for which this is not true, it means we would reflect more light than is coming in (furnace test!)

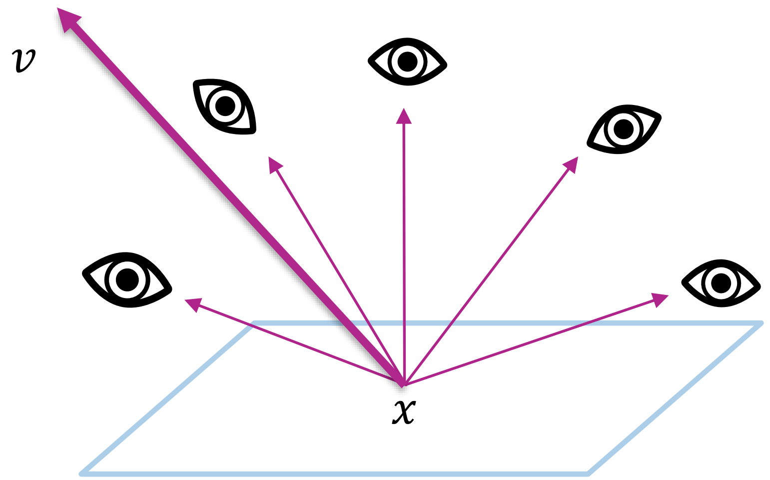


Condition for Energy Conservation

- Why must the BRDF f_r fulfill $\int_\Omega f_r(x, \omega \to v) \cos_\theta(\omega)\, d\omega \leq 1$?

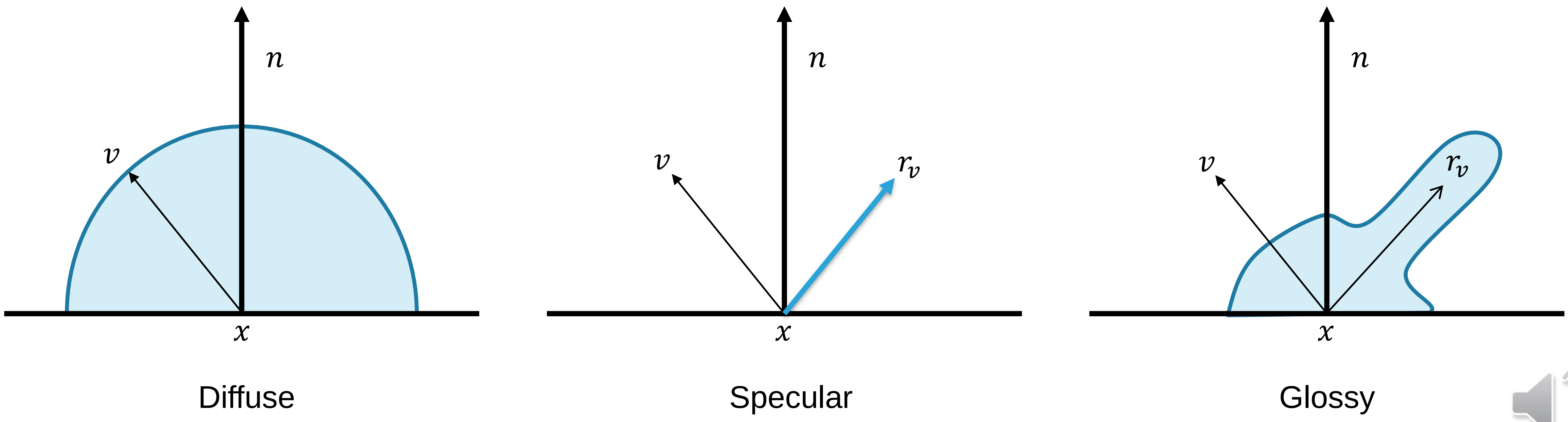
- Intuitive interpretation with **reciprocity**: Shine a laser light along $-v$ onto x. We must have $\int_\Omega f_r(x, v \to \omega) \cos_\theta(\omega)\, d\omega \leq 1$

- If we find a direction v for which this is not true, it means we would reflect more light than is coming in (furnace test!)



BRDF Types

- We usually distinguish three basic BRDF types
  - Perfectly diffuse (light is scattered equally in/from all directions)
  - Perfectly specular (light is reflected in/from exactly one direction)
  - Glossy (mixture of the other two, stronger reflectance around r_v)



BRDF Types

- We usually distinguish three basic BRDF types
  - Perfectly diffuse (light is scattered equally in/from all directions)
  - Perfectly specular (light is reflected in/from exactly one direction)
  - Glossy (mixture of the other two, stronger reflectance around r_v)



Sampling the BRDF

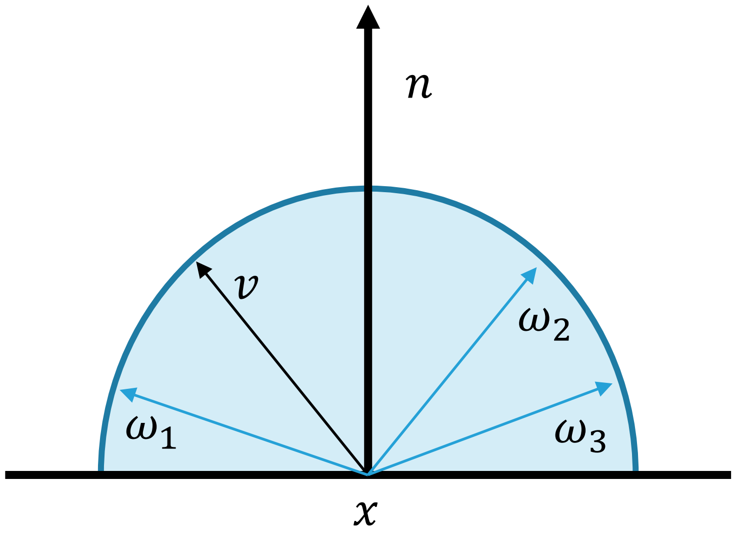
- Before, we considered the BRDF value and sampling of ω separately

- For implementation, it makes a lot of sense to combine them
  - $f_r(x, \omega \rightarrow v)$ depends only on x, v and next ray direction ω
  - Rendering equation: we can't predict L_i, but $f_r(x, \omega \rightarrow v)$ and $\cos\theta$
  - Our renderings will converge faster if the distribution of ω actually matches the shape of $f_r(x, \omega \rightarrow v)\cos\theta$ (**importance sampling**!)
  - If we put the BRDF in charge of choosing our ω, we can make it sample a distribution that directly matches $f_r(x, \omega \rightarrow v)\cos\theta$
  - This actually makes things cleaner in code

How to Handle Diffuse BRDFs

- Diffuse materials reflect same amount of light in/from all directions

- $f_r(x, \omega \rightarrow v) = \frac{\rho}{\pi}\ \forall\ v, \omega \angle n < \frac{\pi}{2}$
  - ρ = amount of reflected light
  - $\rho \leq 1$ in r, g, b

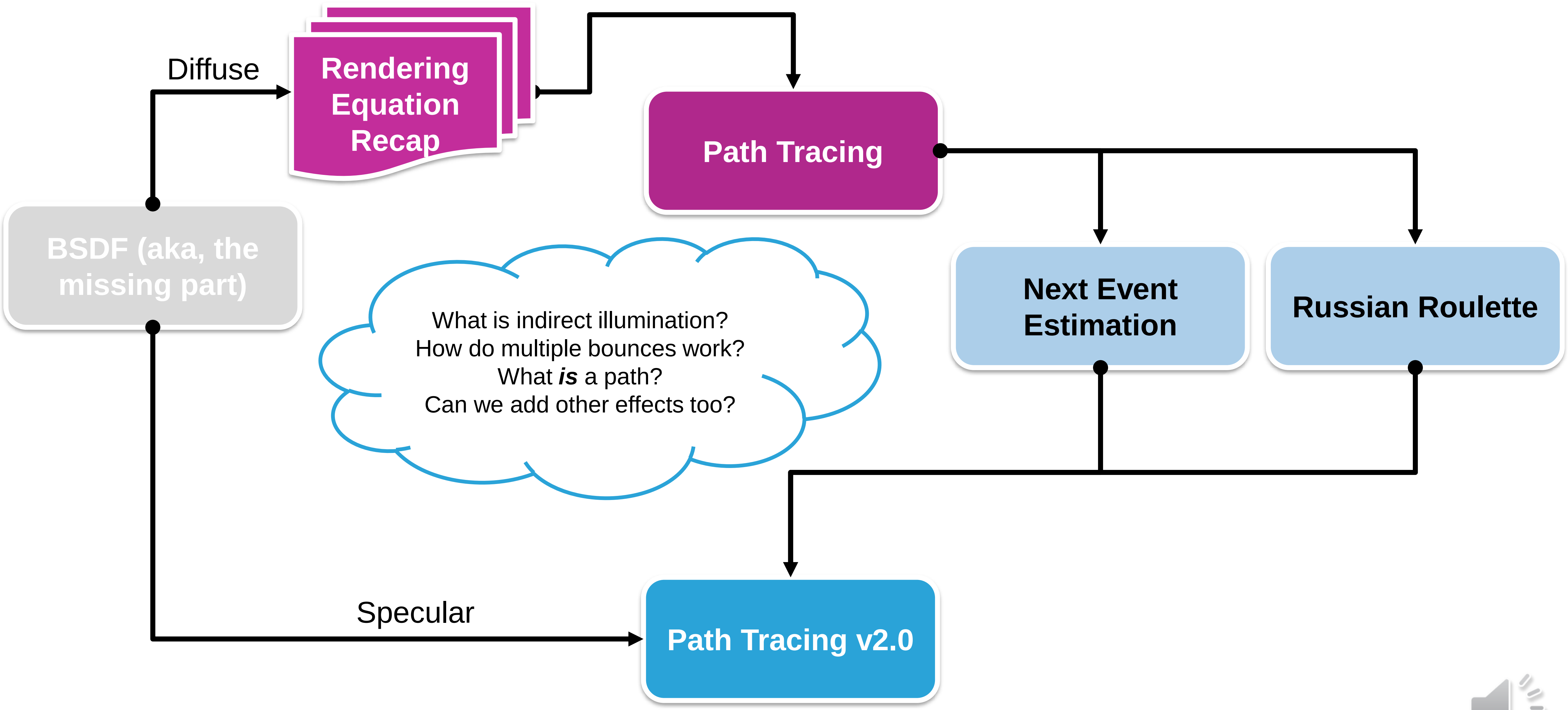


- Importance sampling $f_r(x, \omega \rightarrow v) \cos\theta$ → use $p(\omega) \propto \frac{\rho \cos\theta}{\pi}$
  - Making it a valid PDF leads to $p(\omega) = \frac{\cos\theta}{\pi}$
  - From previous exercise: it's cosine-weighted hemisphere sampling!

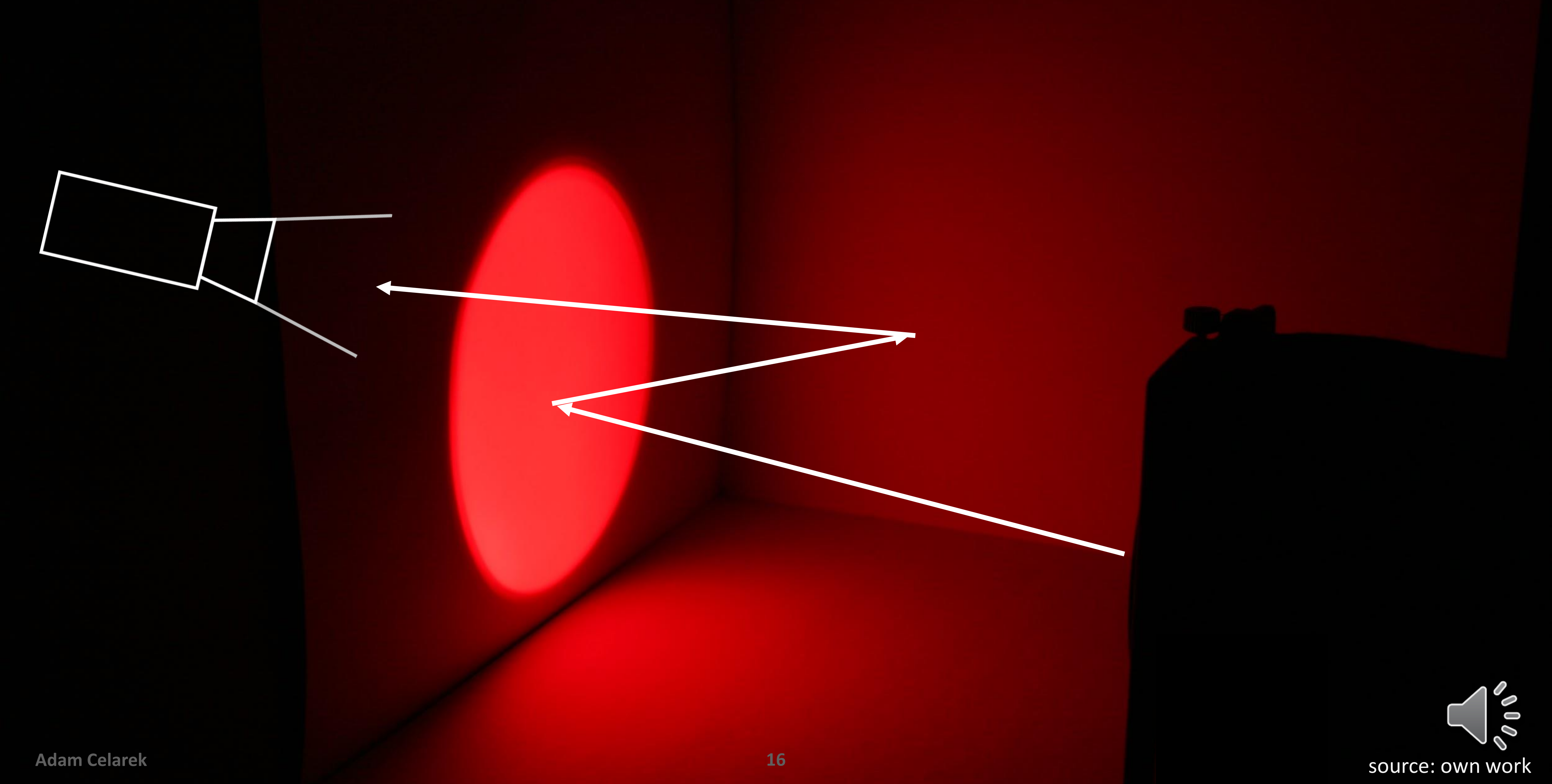
How to Implement Diffuse BRDFs

- Method **sample(v)**: generate a cosine-weighted sample
- Method **evaluate(a, b)**: if $a, b \angle n < \frac{\pi}{2}$, return $f_r(x, b \to a) = \frac{\rho}{\pi}$
- Method **pdf(ω)** : return the proper $p(\omega)$ for the passed sample
- Combine them into unit that takes care of handling diffuse materials
- Use terms as before. Abstracts the importance sampling away!

Today's Roadmap



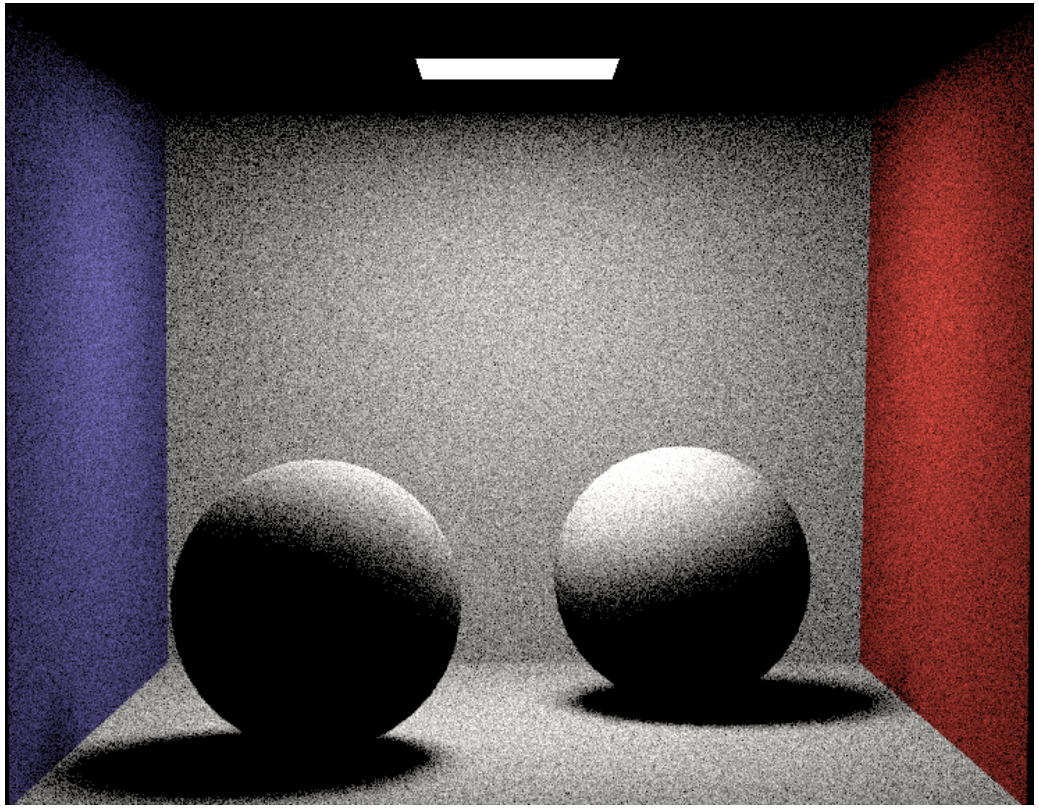
Things get interesting if we look at indirect illumination

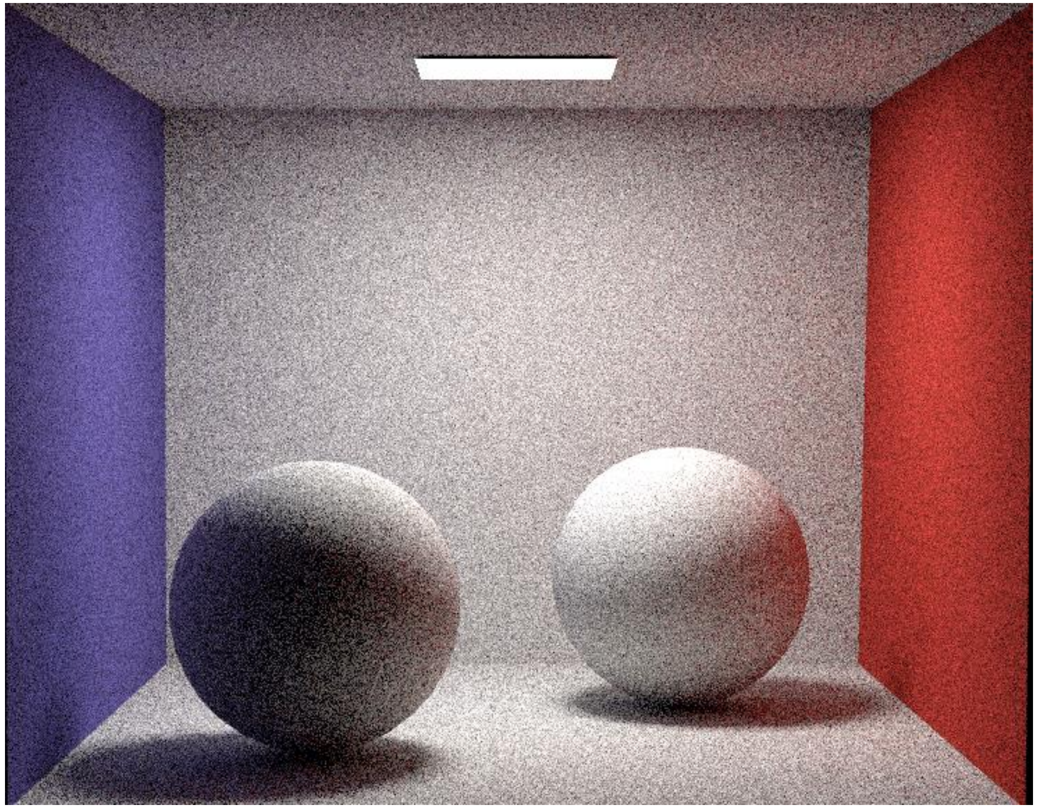


source: own work

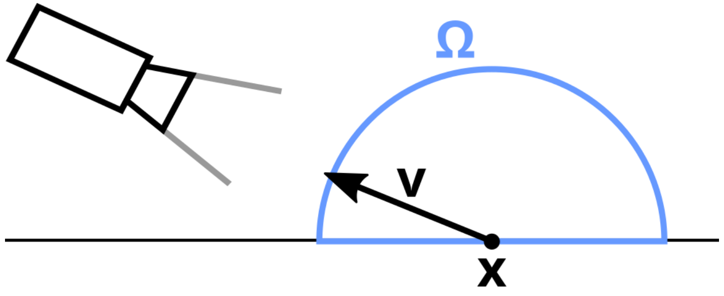
Indirect Illumination

- Difficult in real-time graphics – comes naturally in path tracing!





Recursive Rendering Equation, Recap



Material, modelled by the BRDF

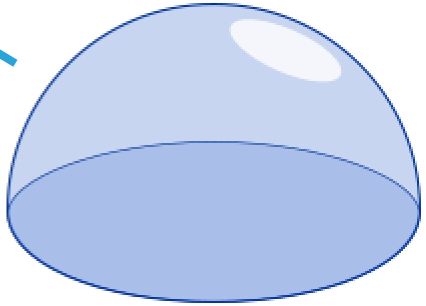
Light from direction ω

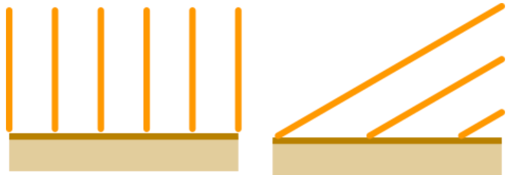
Solid angle

$$L_e(x,v) = E(x,v) + \int_\Omega f_r(x,\omega \to v) L_i(x,\omega) \cos(\theta_x)\, \mathrm{d}\omega$$

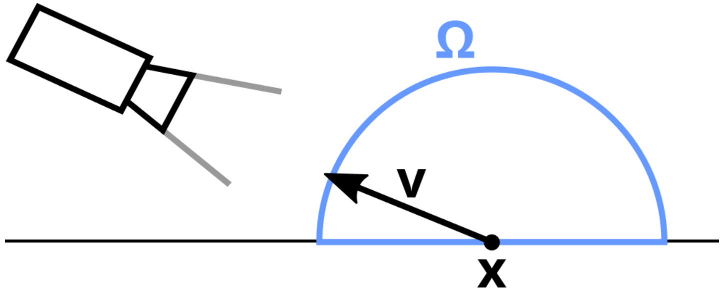
Light going in direction v

Light emitted from x in direction v





Recursive Rendering Equation, Recap



Material, modelled by the BRDF

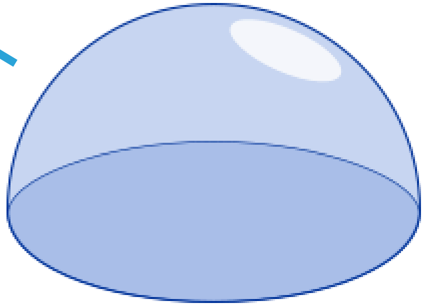
Evaluate light from direction ω **recursively**

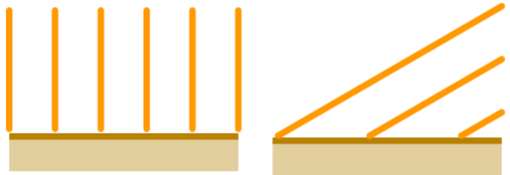
Solid angle

$$L_e(x, v) = E(x, v) + \int_\Omega f_r(x, \omega \rightarrow v) L_i(x, \omega) \cos(\theta_x)\, \mathrm{d}\omega$$

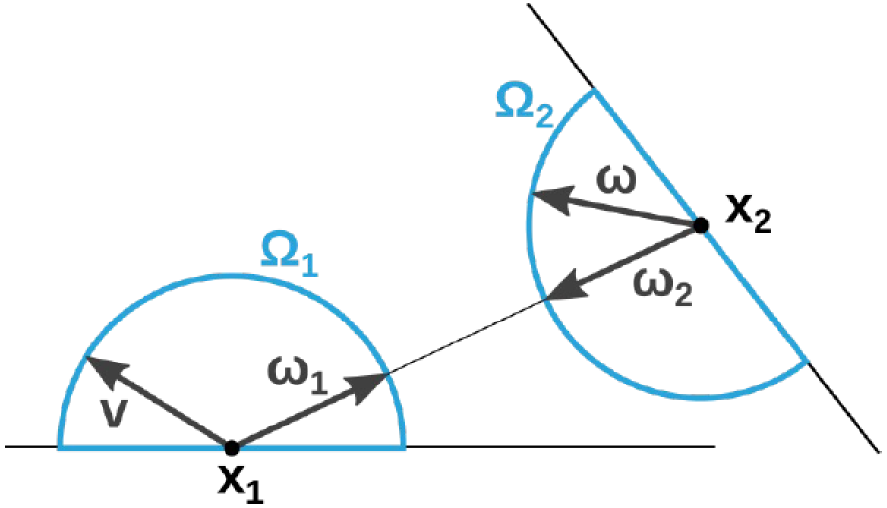
Light going in direction v

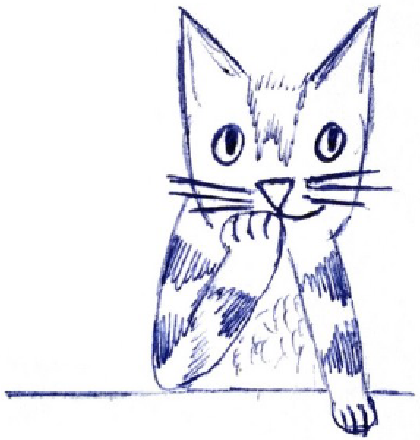
Light emitted from x in direction v





- To get the next bounce, we just evaluate this function recursively





$$L(x_1 \to v) = E(x_1 \to v) + \int_{\Omega_1} f_r(x_1, \omega_1 \to v)\, L(x_1 \leftarrow \omega_1) \cos(\theta_x)\, d\omega_1$$

$$L(x_1 \leftarrow \omega_1) = L(x_2 \to \omega_2) \quad !$$

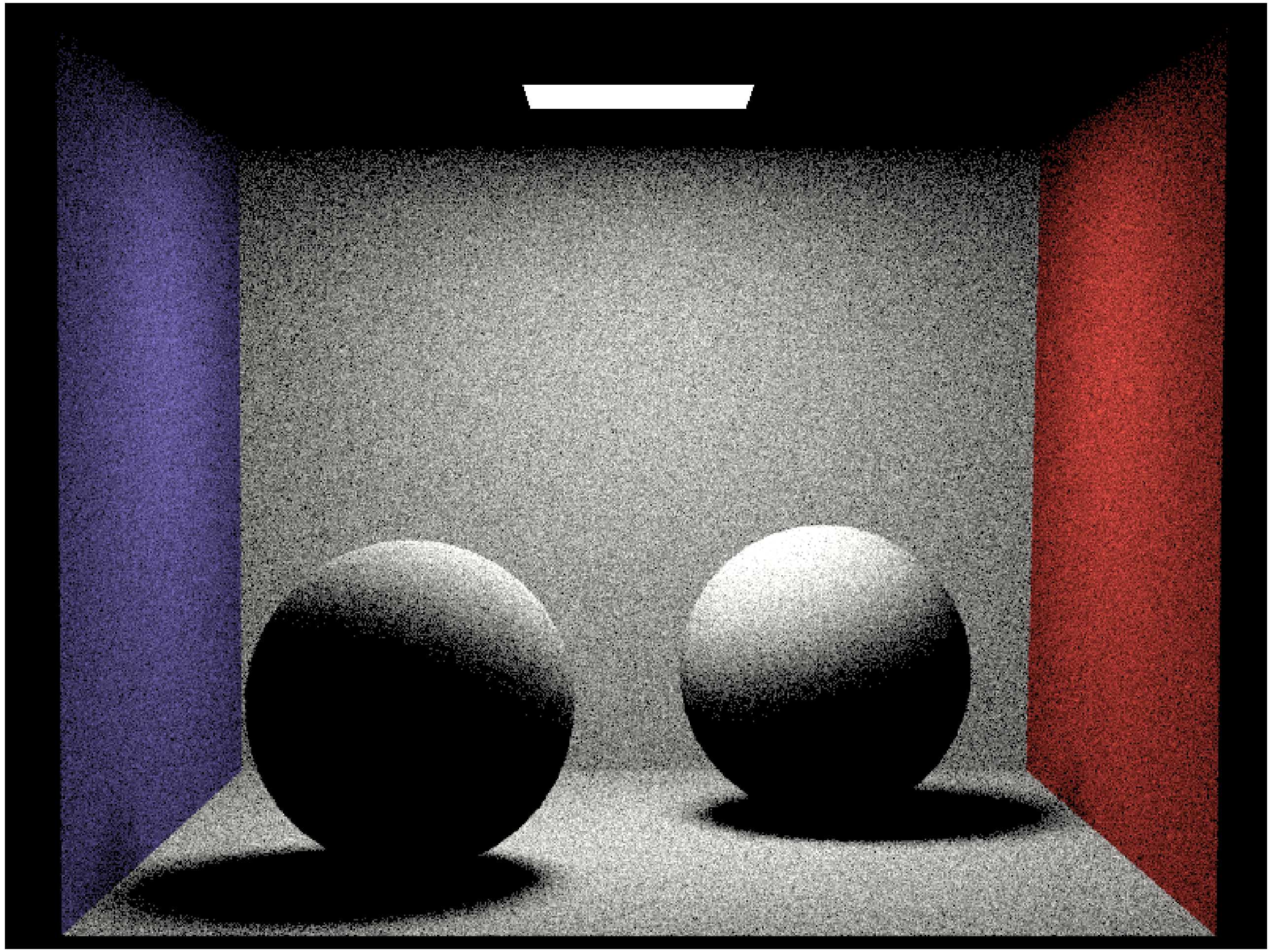
$$L(x_2 \to \omega_2) = E(x_2 \to \omega_2) + \int_{\Omega_2} f_r(x_2, \omega \to \omega_2)\, L(x_2 \leftarrow \omega) \cos(\theta_x)\, d\omega$$

Implementing the Rendering Equation

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= maxDepth) return emitted;        <- Recursion limit

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    Ray wo = BRDFsample(brdf, -ray);             <- Diffuse BRDF
    float pdf = BRDFpdf(brdf, wo);
    Color brdfValue = BRDFevaluate(brdf, -ray, wo);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, wo, depth + 1);   <- Recursion
    return emitted + brdfValue * indirect * cosTheta(its, wo) / pdf;
}
```

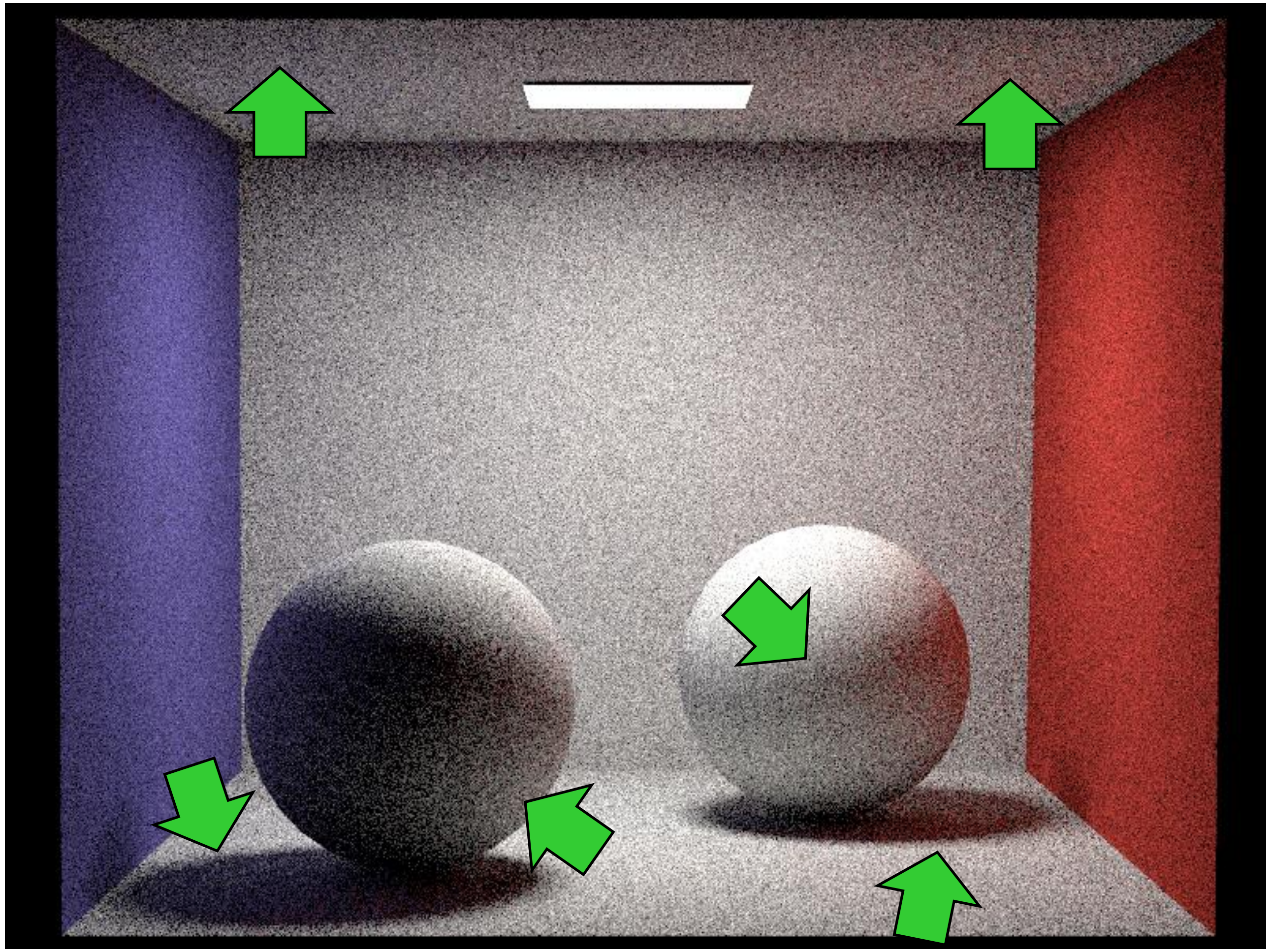
One Bounce

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 1) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    Ray wo = BRDFsample(brdf, -ray);
    float pdf = BRDFpdf(brdf, wo);
    Color brdfValue = BRDFevaluate(brdf, -ray, wo);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, wo, depth + 1);
    return emitted + brdfValue * indirect * cosTheta(its, wo) / pdf;
}
```



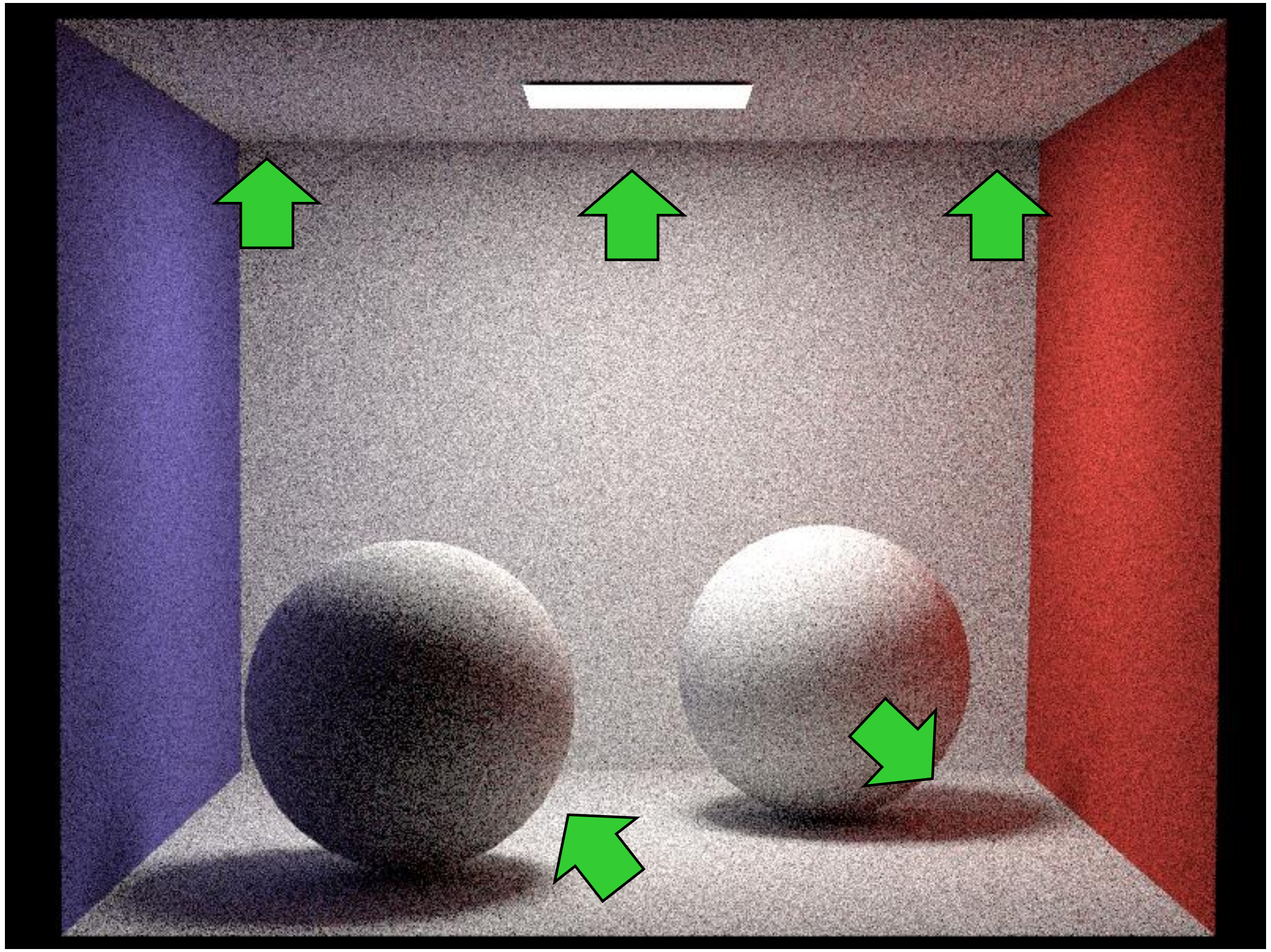
Two Bounces

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 2) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    Ray wo = BRDFsample(brdf, -ray);
    float pdf = BRDFpdf(brdf, wo);
    Color brdfValue = BRDFevaluate(brdf, -ray, wo);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, wo, depth + 1);
    return emitted + brdfValue * indirect * cosTheta(its, wo) / pdf;
}
```

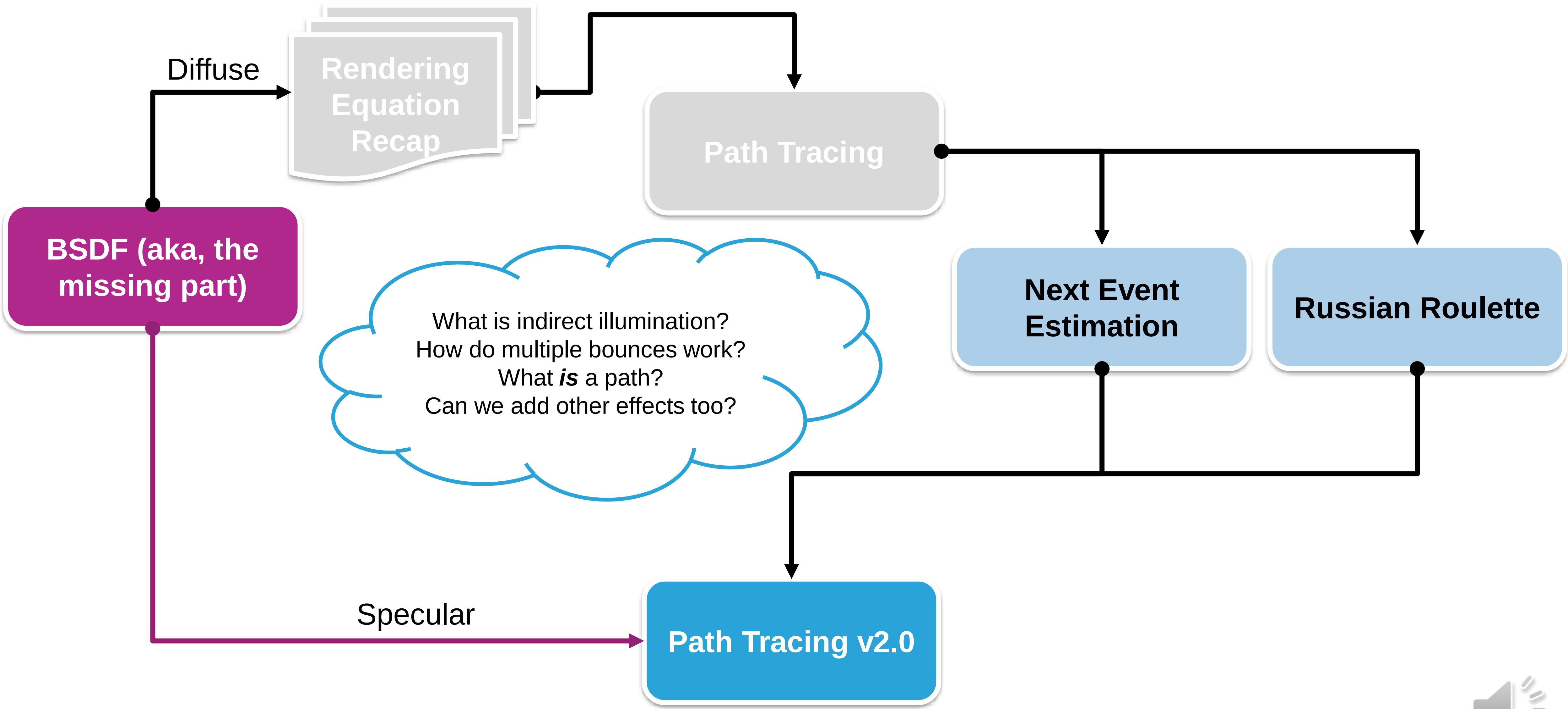


Three Bounces

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 3) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    Ray wo = BRDFsample(brdf, -ray);
    float pdf = BRDFpdf(brdf, wo);
    Color brdfValue = BRDFevaluate(brdf, -ray, wo);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, wo, depth + 1);
    return emitted + brdfValue * indirect * cosTheta(its, wo) / pdf;
}
```



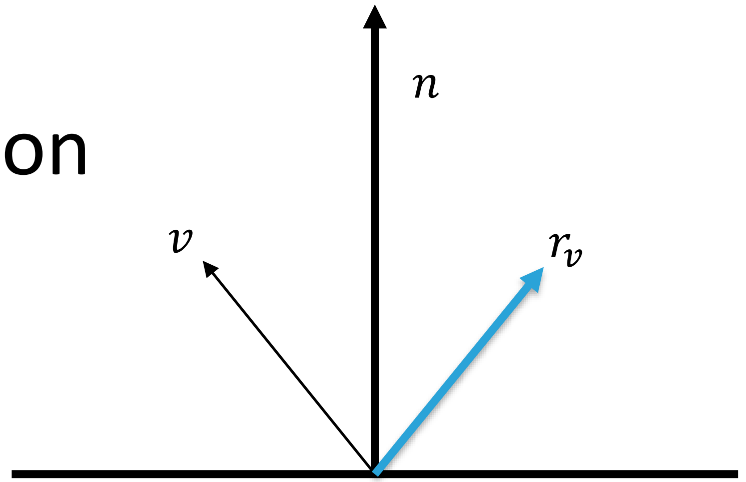
Today's Roadmap



How to Handle Specular BRDFs (Mirrors)

- For purely specular BRDFs (a perfect mirror surface), irradiance from the perfect mirror direction r_v is completely reflected to v

- Irradiance coming from any other direction does not reflect at all towards v



- $f_r(x, \omega \rightarrow v) > 0 \Leftrightarrow \omega = r_v$

- Problem: if we pick the next direction ω randomly as before, the chances of ever hitting r_v by accident are infinitely small!

The Dirac Delta Function

- Model specular reflection with the Dirac delta function
- Delta function $\delta(x)$ is defined to be 0 everywhere except at $x = 0$
- Use a shifted version $\delta_v(\omega)$ that is 0 everywhere except at $\omega = r_v$
- Per definition, $\int_\Omega \delta_v(\omega)\, d\omega = 1$ to obtain a valid PDF for sampling
- Ponder this for a moment: what value does $\delta_v(r_v)$ have?

Energy-Preserving Specular BRDF

- Full energy preservation: $\int_\Omega f_r(x, \omega \rightarrow v)\, L_i \cos_\theta(\omega)\, d\omega = L_{r_v}$
- If we integrate using $f_r(x, \omega \rightarrow v) = \delta_v(\omega)$, we get $L_{r_v} \cos_\theta(r_v)$
- We lost some light! We compensate: $f_r(x, \omega \rightarrow v) = \frac{\delta_v(\omega)}{\cos_\theta(r_v)}$
- If we consider the properties of the Dirac delta function, we can try to derive the same methods that we used before for diffuse BRDFs

Try to Implement Specular BRDF

- **sample(v)**: mirror v about n (invert v_x, v_y in *local space*) and return
- **evaluate(a, b)**: 0 if $b \neq r_a$, else return $\frac{\delta_a(r_a)}{\cos_\theta(r_a)} = \frac{\infty}{\cos_\theta(r_a)}$
  - **Problem**: How to calculate anything reasonable with ∞?
  - **Problem**: we are comparing two vectors with floats (Stability?)
- **pdf(ω)**: 0 if $\omega \neq r_v$, else: $\delta_v(r_v) = \infty$
- But, if $\omega = r_v$, **evaluate(v, ω) / pdf(ω)** $= \frac{\delta_v(\omega)}{\delta_v(\omega)\cos_\theta(r_v)} = \frac{1}{\cos_\theta(r_v)}$

How to Implement Diffuse and Specular BRDFs

- Specular BRDF: using **evaluate/pdf** without **sample** is awkward
- Let's make a change to the path tracing routine and BRDF interface
- Suggestion: let **sample** method generate ω and a multiplier for L_i
- Leave application of $\cos\theta$ and $p(\omega)$ to the BRDF (if necessary)
  - Diffuse: importance sample ω, apply $p(\omega)$, $\cos\theta$ **cancels out**
  - Specular: pick $\omega = r_v$, $p(\omega)$ **cancels out**, $\cos\theta$ **cancels out**

Revising the Specular BRDF Implementation

- **sample(**v**)**: mirror v about n (invert v_x, v_y in *local space*)
  - Return r_v as generated sample direction
  - Return multiplier for L_i as 1 (full radiance passed on)

- No other function except **sample** should be able to just *guess* r_v

- **evaluate(**a, b**)**: always return 0

- **pdf(**ω**)**: always return 0

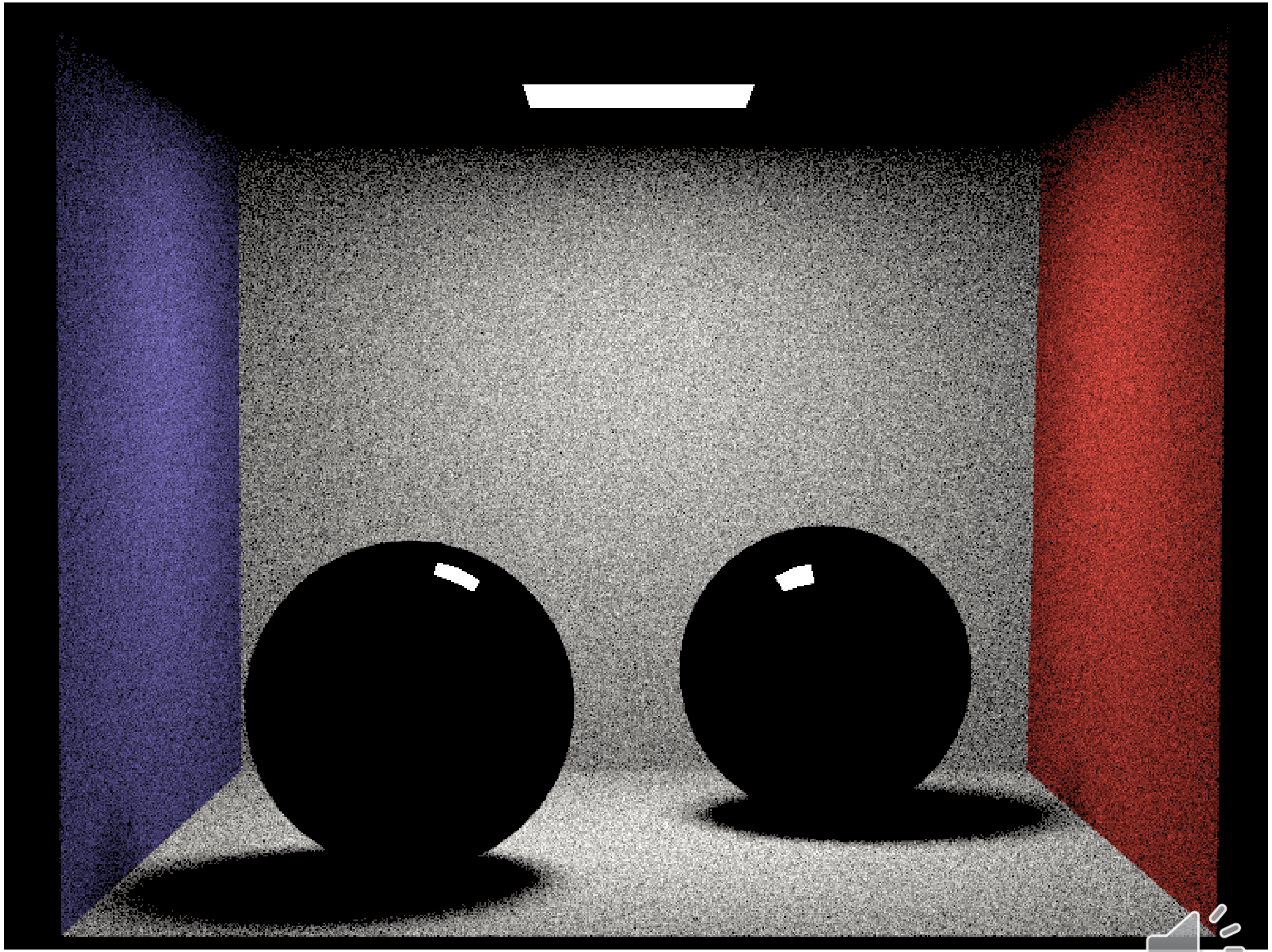
Implementing the Rendering Equation v2.0

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= max_depth) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    BRDFSample sample;

    sample = BRDFsample(brdf, -ray);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, sample.wo, depth + 1);
    return emitted + sample.value * indirect;
}
```

New, combined BRDF sample.value contains PDF and cosine factors, if necessary

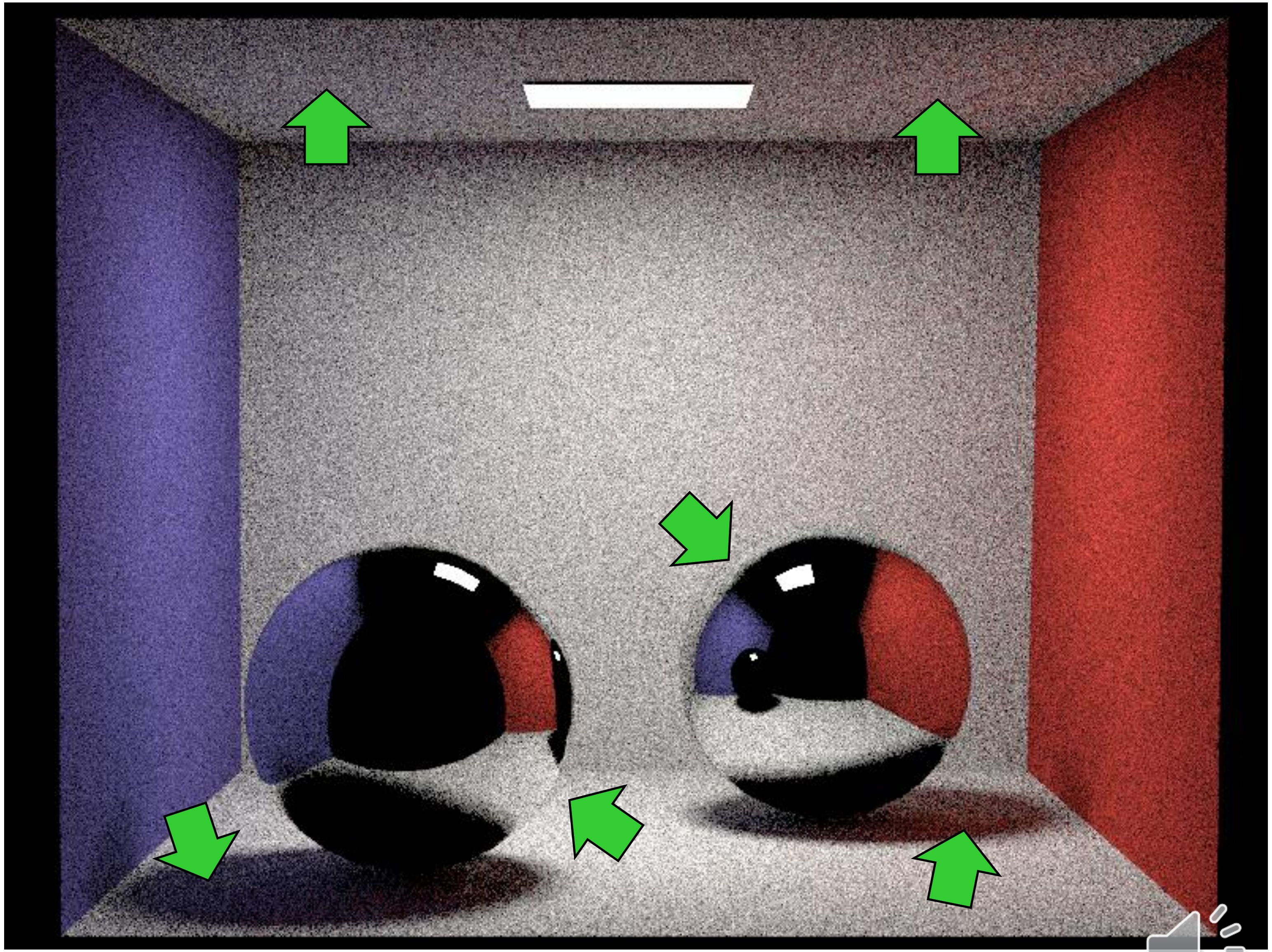
One Bounce

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 1) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    BRDFSample sample;

    sample = BRDFsample(brdf, -ray);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, sample.wo, depth + 1);
    return emitted + sample.value * indirect;
}
```



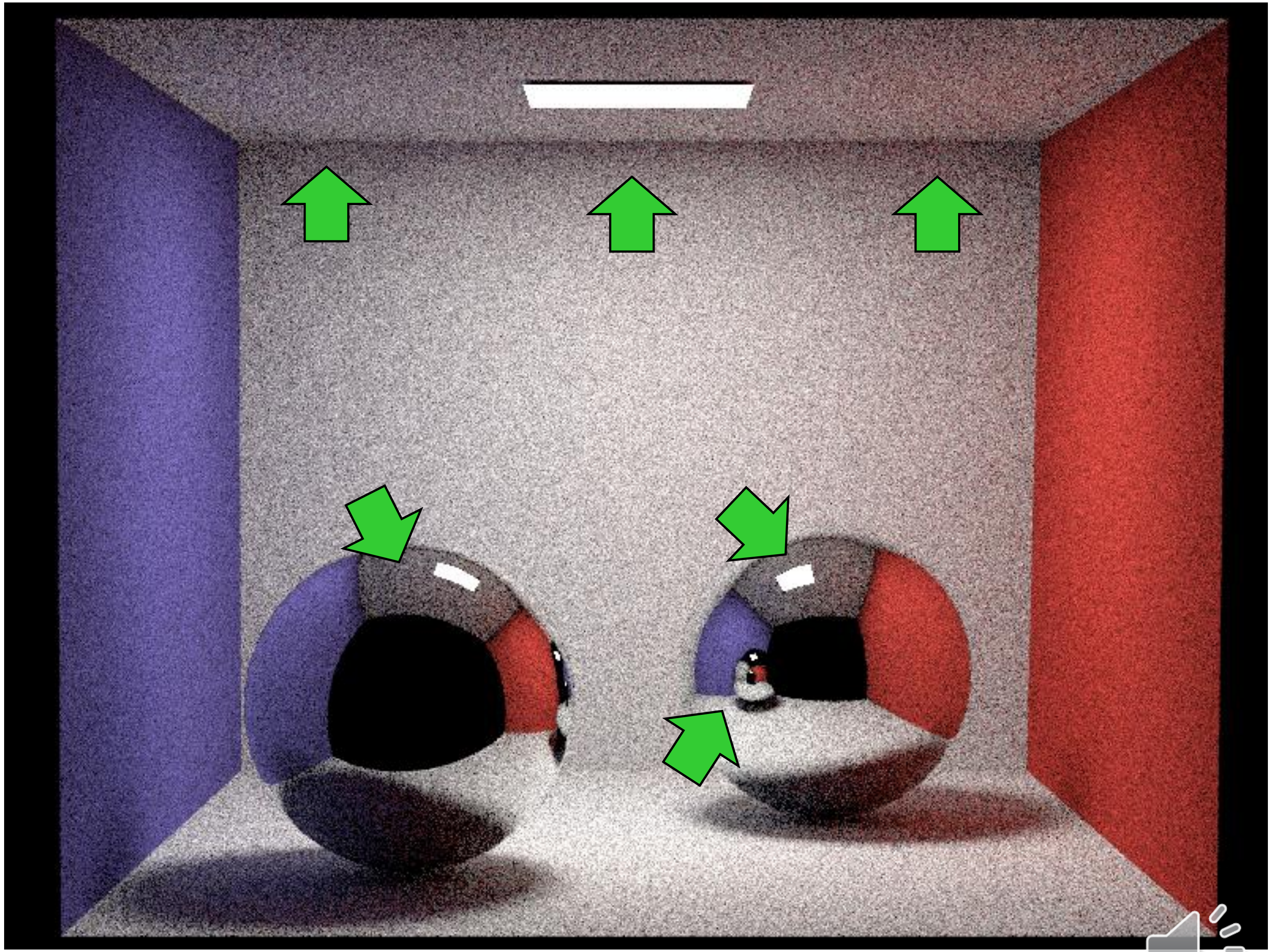
Two Bounces

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 2) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    BRDFSample sample;

    sample = BRDFsample(brdf, -ray);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, sample.wo, depth + 1);
    return emitted + sample.value * indirect;
}
```



Three Bounces

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(depth >= 3) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    BRDFSample sample;

    sample = BRDFsample(brdf, -ray);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, sample.wo, depth + 1);
    return emitted + sample.value * indirect;
}
```



How many bounces is enough?

- Remember: if we want to be unbiased, then the probability of each possible path (i.e., journey of a photon) must be non-zero
- Photons stop bouncing when they have been entirely absorbed
- Problem: no real-world material absorbs 100% of incoming light
- No matter how many bounces, the probability never goes to zero
  → you can **never** stop!

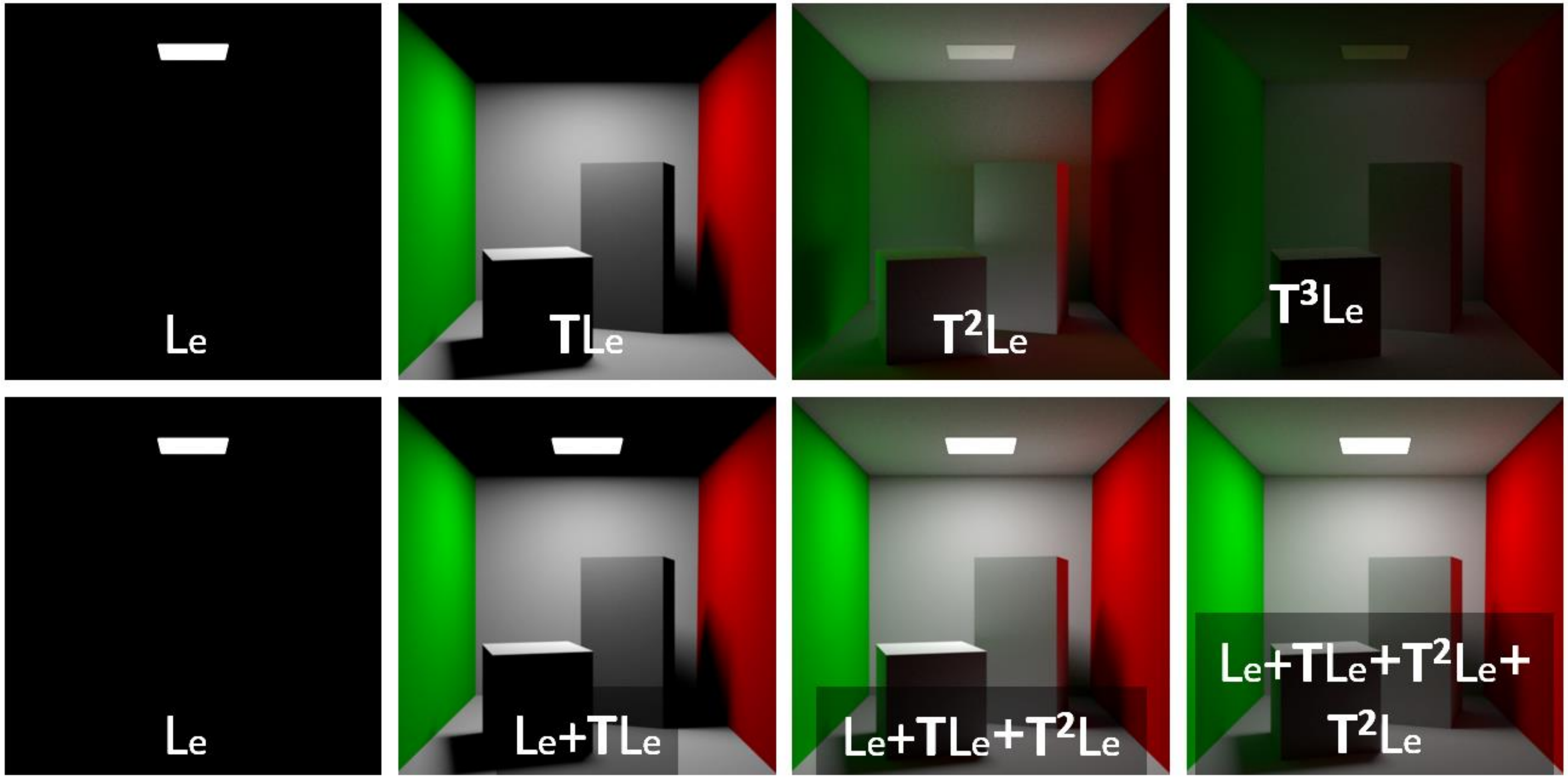
∞ Bounces

```
Li(Scene scene, Ray ray, int depth)
{
    Color emitted = 0;

    if (!findIntersection(scene, ray)) return 0;

    Intersection its = getIntersection(scene, ray);

    // Take care of emittance
    if (isLightSource(its)) emitted = getRadiance(its);

    if(false) return emitted;

    // BRDF should decide on the next ray
    // (It has to, e.g. for specular reflections)
    BRDF brdf = getBRDF(its);
    BRDFSample sample;

    sample = BRDFsample(brdf, -ray);

    // Call recursively for indirect lighting
    Color indirect = Li(scene, sample.wo, depth + 1);
    return emitted + sample.value * indirect;
}
```

- Renderer never finishes. What to do?

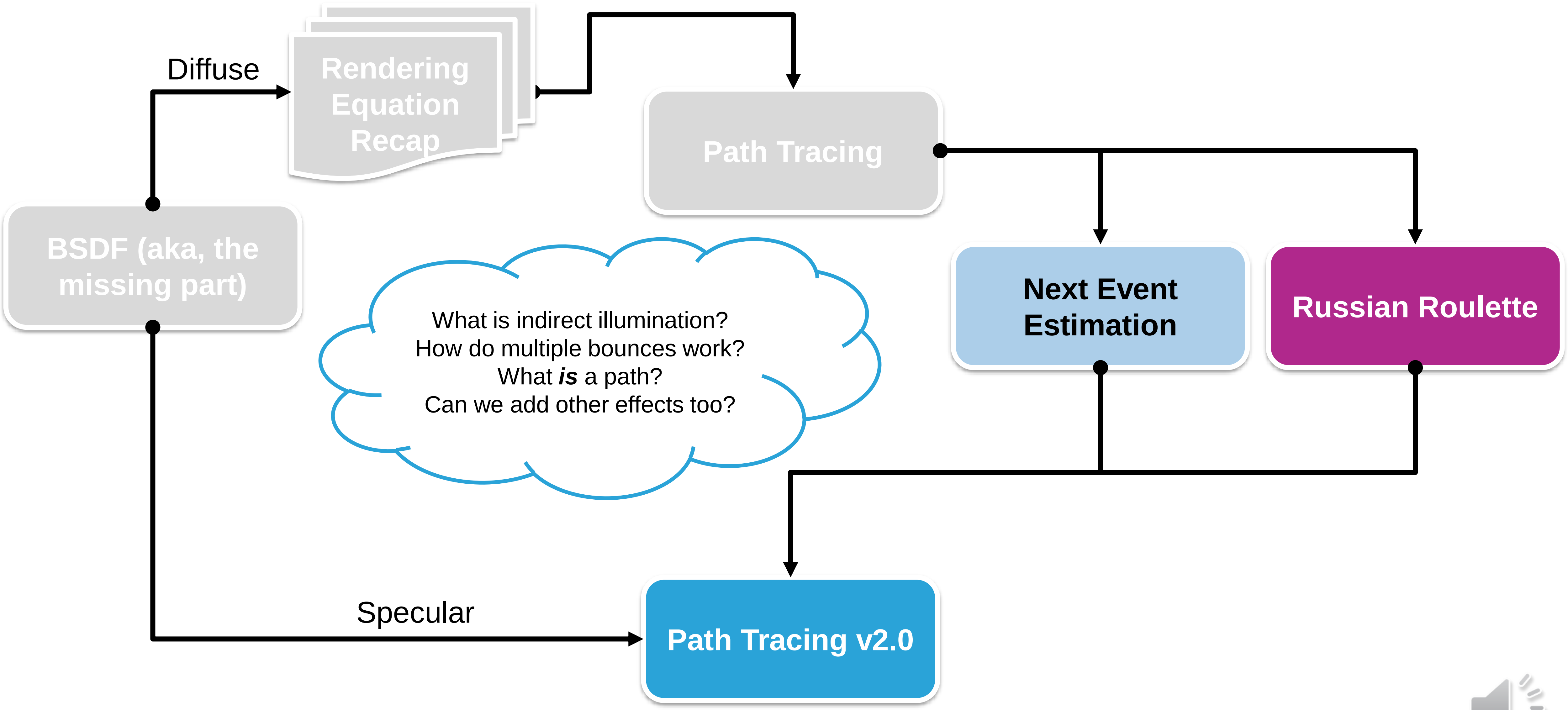
Optimizing Infinite Paths

- In practice, most contribution comes from the first few bounces



- Can we exploit this fact and make long paths possible, but unlikely?

Today's Roadmap



Russian Roulette (RR)

- Pick a $p > 0$. At each bounce, draw a random variable ξ and decide
  - $\xi < p$: keep going for another bounce
  - $\xi \geq p$: end path

- The longer a path goes on, the more likely it is to get terminated

- The probability of a ray surviving the N^{th} bounce is p^N

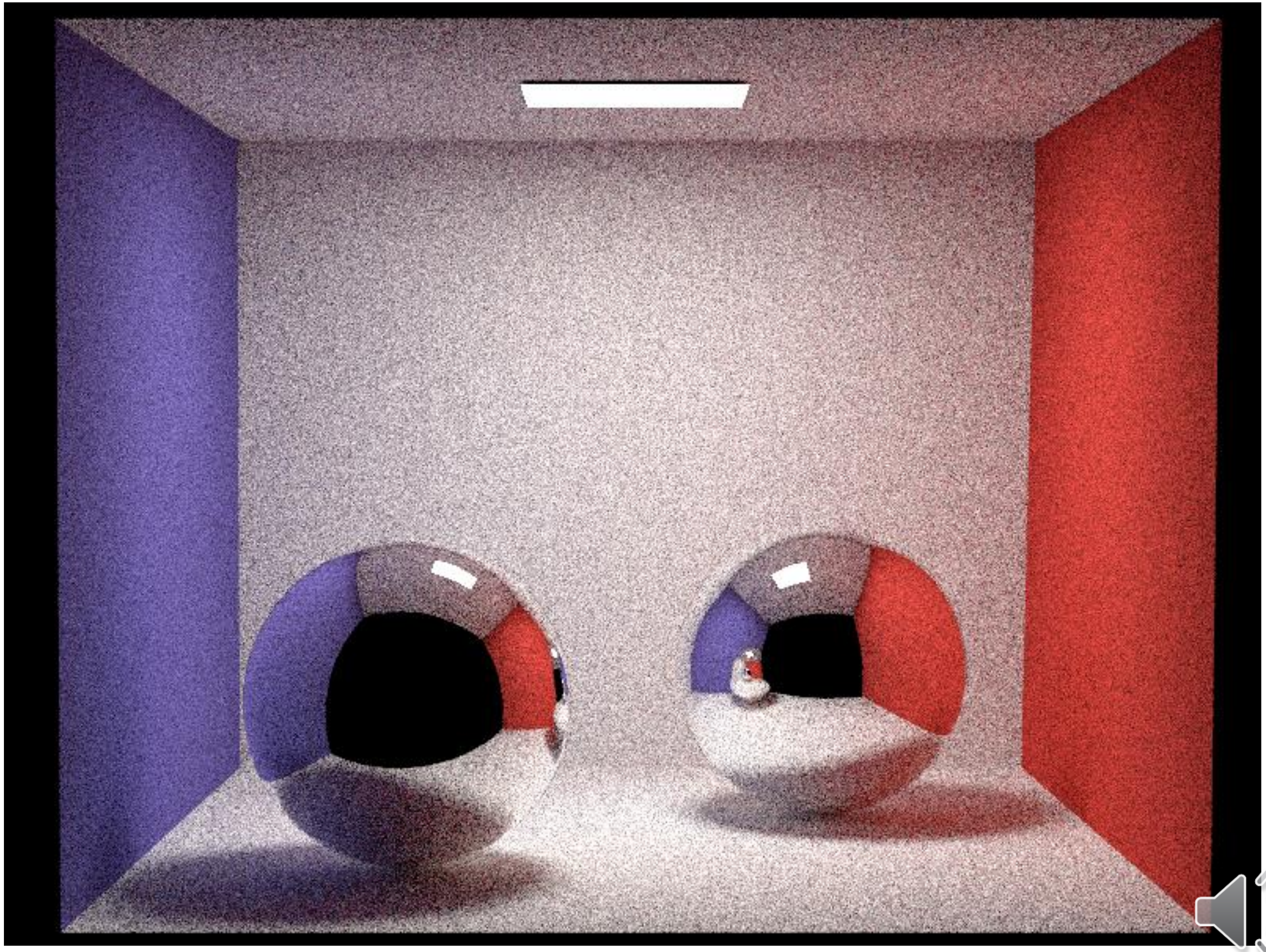
- Whenever a path continues after a bounce, compensate for its (un)-likeliness by weighting the color returned from L_i with $\frac{1}{p}$

Russian Roulette..?

- *"...but if the possibility for infinitely long paths remains, doesn't that mean that my renderer may take forever to finish?"*
- Almost certainly no
- In practice, if you choose an adequate p, you are more likely to get struck by lightning while reading this than that ever happening
- *"Ok, cool, so the lower I choose p, the better, right? Can we just take something really small?"* Well, not exactly.

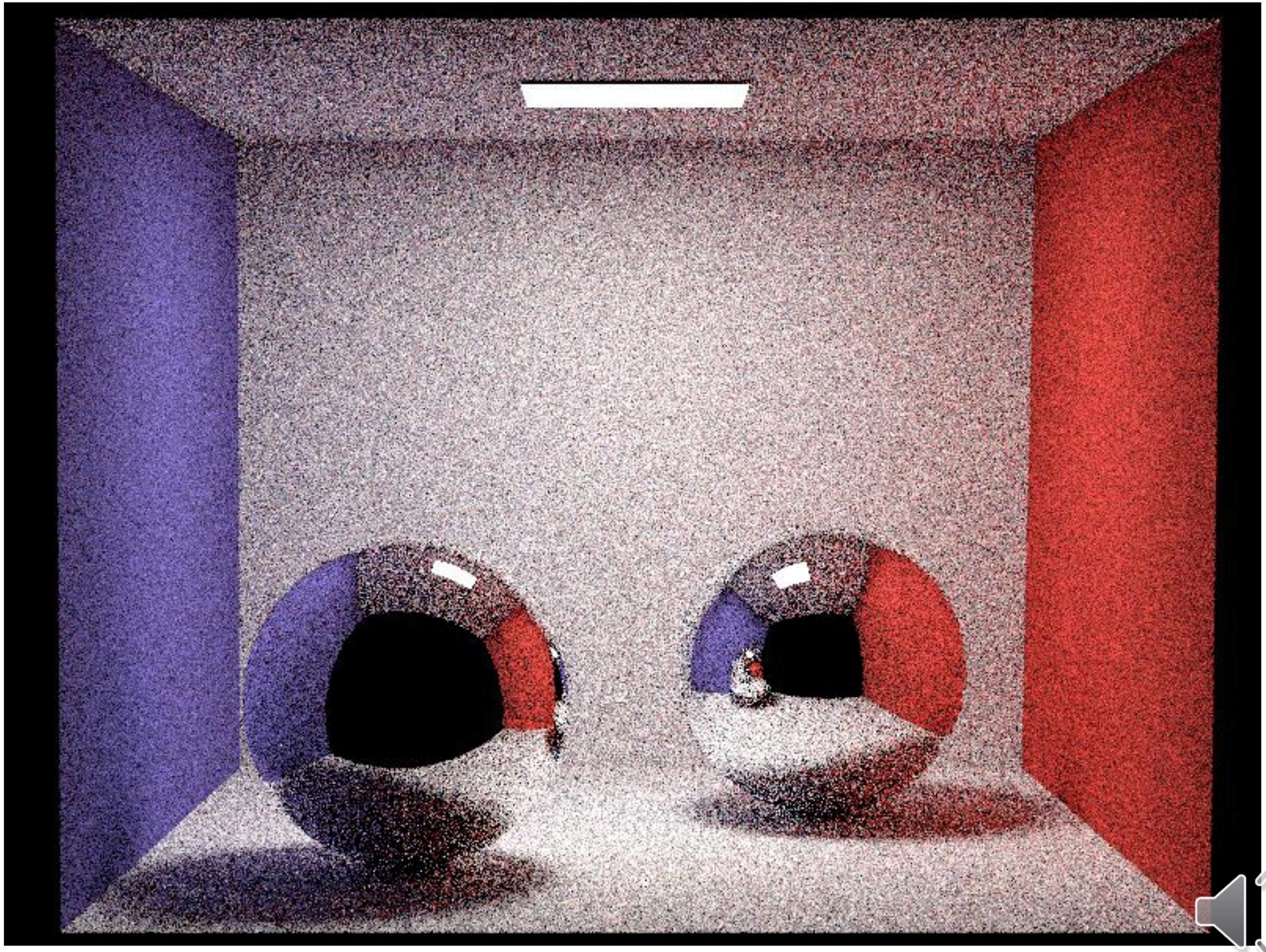
Choosing $p = 0.95$

- Low chance of stopping early
- 500 samples per pixel
- Runtime: 260s



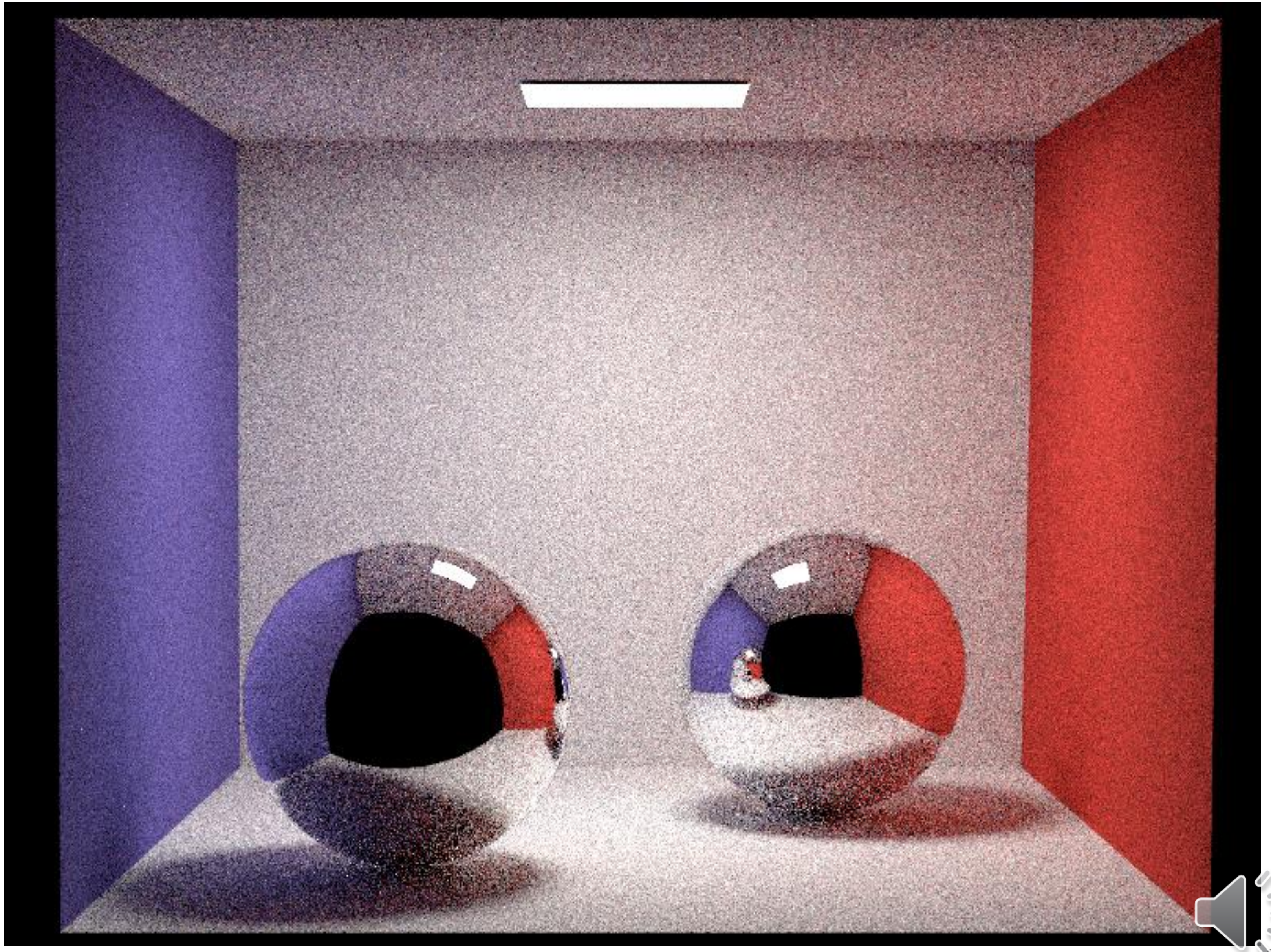
Choosing $p = 0.6$

- High chance of stopping early
- 500 samples per pixel
- Runtime: 60s
- Worse, but faster. More samples?

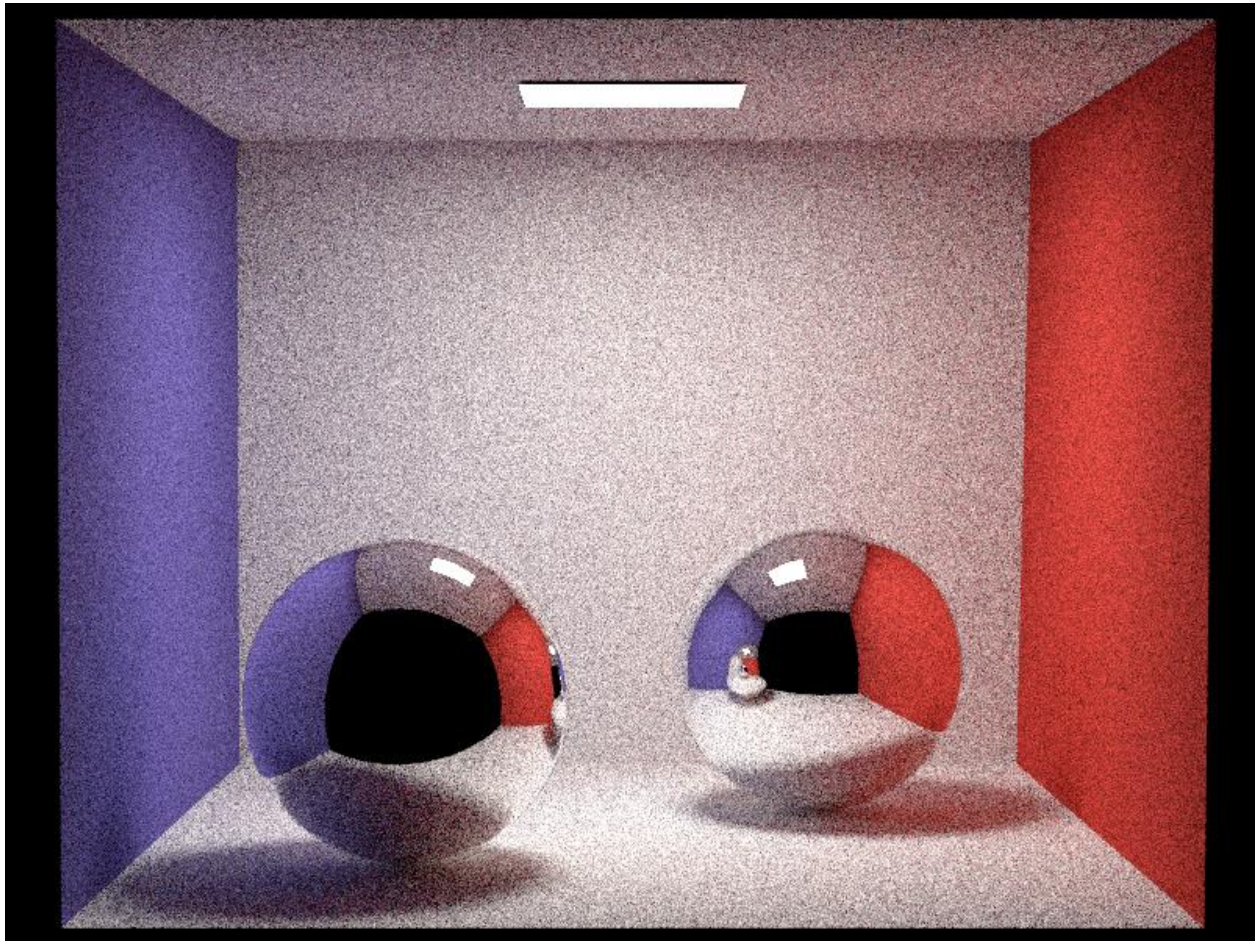


Choosing $p = 0.6$

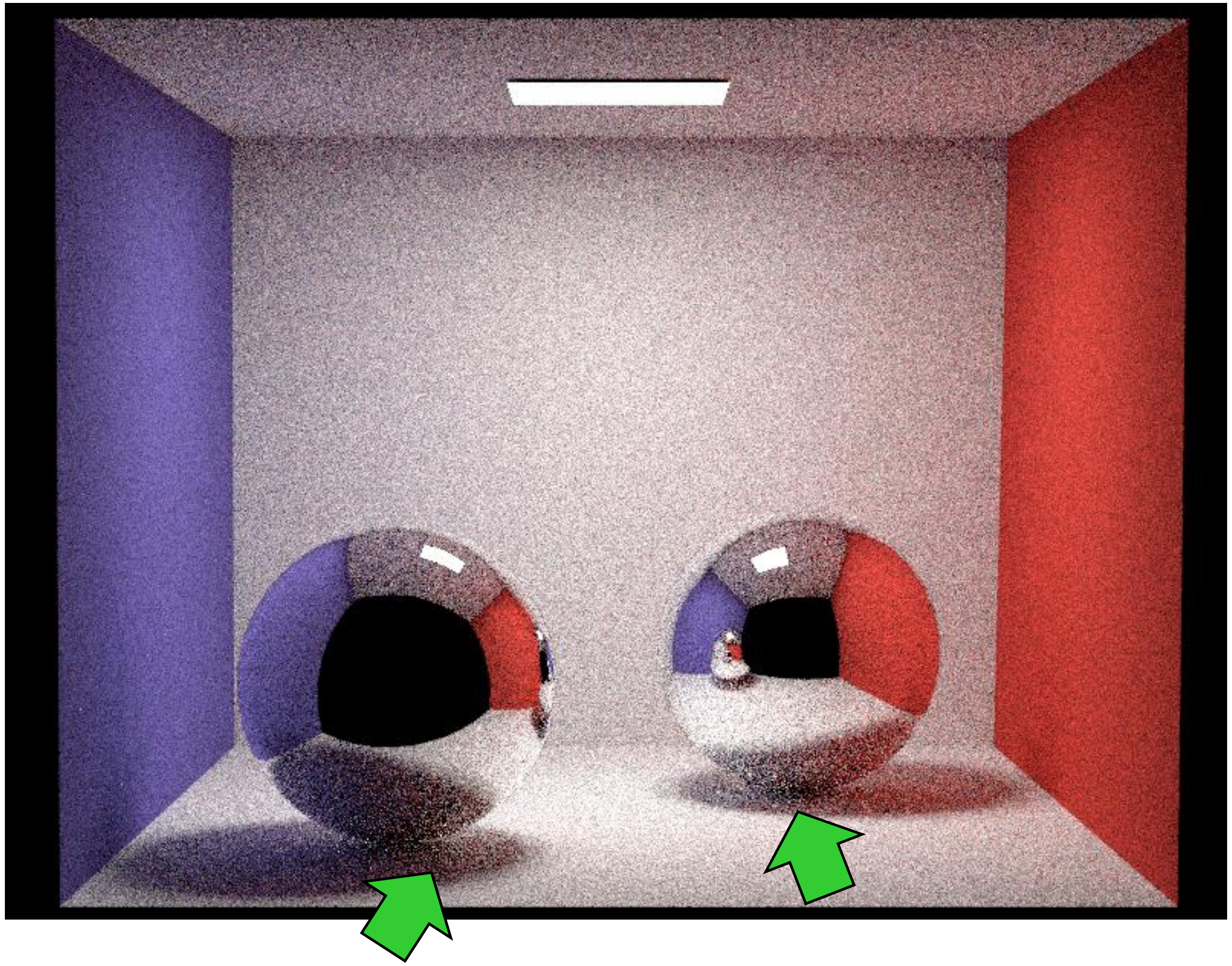
- High chance of stopping early
- **1500** samples per pixel
- Runtime: **270s**



High p vs low p



$p = 0.95$, 500 samples, 260s



$p = 0.6$, 1500 samples, 270s
Took longer but looks worse!

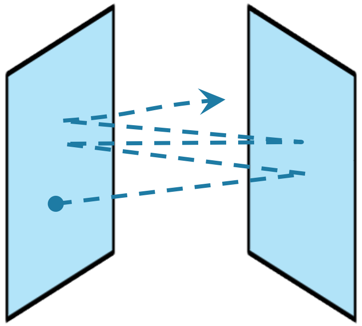
Picking the Right Russian Roulette Probability

- If $p(x)$ is low but $f(x)$ is not → high contribution of rare samples!
- Also called "fireflies"
- Hard to get rid off!

- Choose p at each bounce according to remaining color contribution
- $p_1 = 1, p_N$ at N^{th} bounce $= \max_{\text{RGB}}\left(\prod_{i=1}^{N-1}\left(\frac{f_r(x_i,\omega_i \rightarrow v_i)\cos\theta_i}{\text{pdf}(\omega_i)\, p_i}\right)\right)$

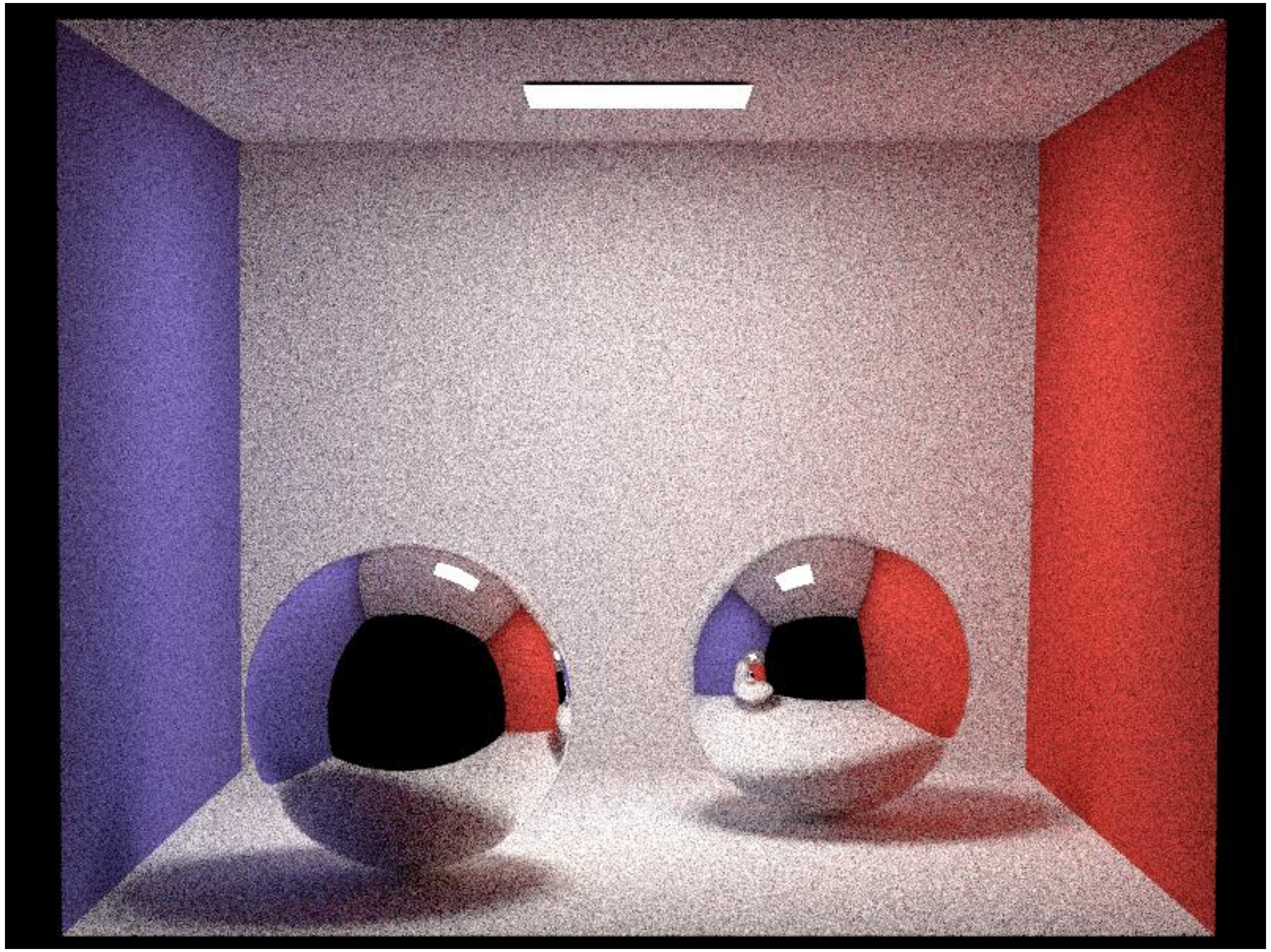
Picking the Right Russian Roulette Probability

- Some materials absorb barely any incoming light (mirrors!)
  - Imagine two mirrors opposite of each other
  - Ray may bounce between them forever
  - Bad: limit bounces to a strict maximum
  - Better: clamp RR p to a value < 1, e.g. 0.99



- Use a **minimal** depth before allowing Russian Roulette to take effect
  - Preserve a minimal path length for indirect illumination
  - Make sure to exclude guaranteed bounces from path weights

Path Tracing + Russian Roulette



- It works. But what about all that noise?

What IS a Path?

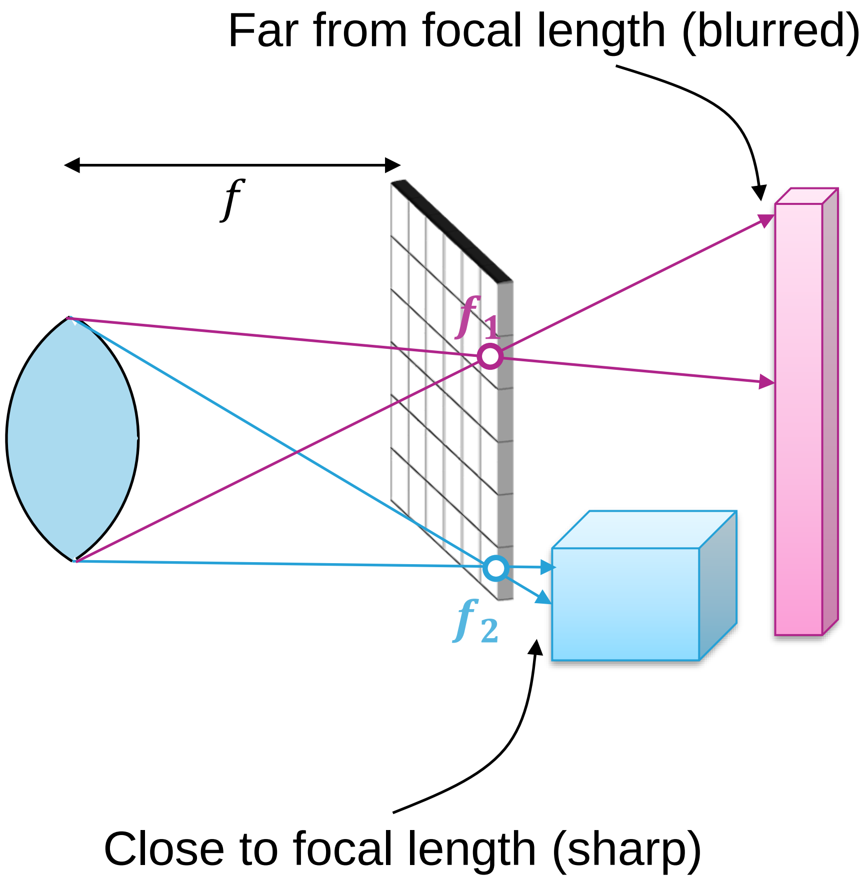
- A path is defined by the random values that you draw along it
- Path of length N can be seen as a multi-dimensional random variable, e.g.: $(\xi_1, \xi_2, \dots, \xi_{2N})^T$ (need at least θ, ϕ per bounce)
- The more bounces we make, the more dimensions we add
- Monte Carlo is fine with handling infinite-dimensional integrals
- We pay the price for additional dimensions with additional noise

Dimensions of Path Tracing

- We already know some of them
  - Random sample positions inside pixel (2)
  - Constructing a new ray after each bounce ($2N$)
  - Choosing a specific strategy for MIS (1)
  - …
- Other possible choices we have not yet considered[1]
  - Lens coordinates (for depth-of-field) (2)
  - Time (for motion blur) (1)
  - …

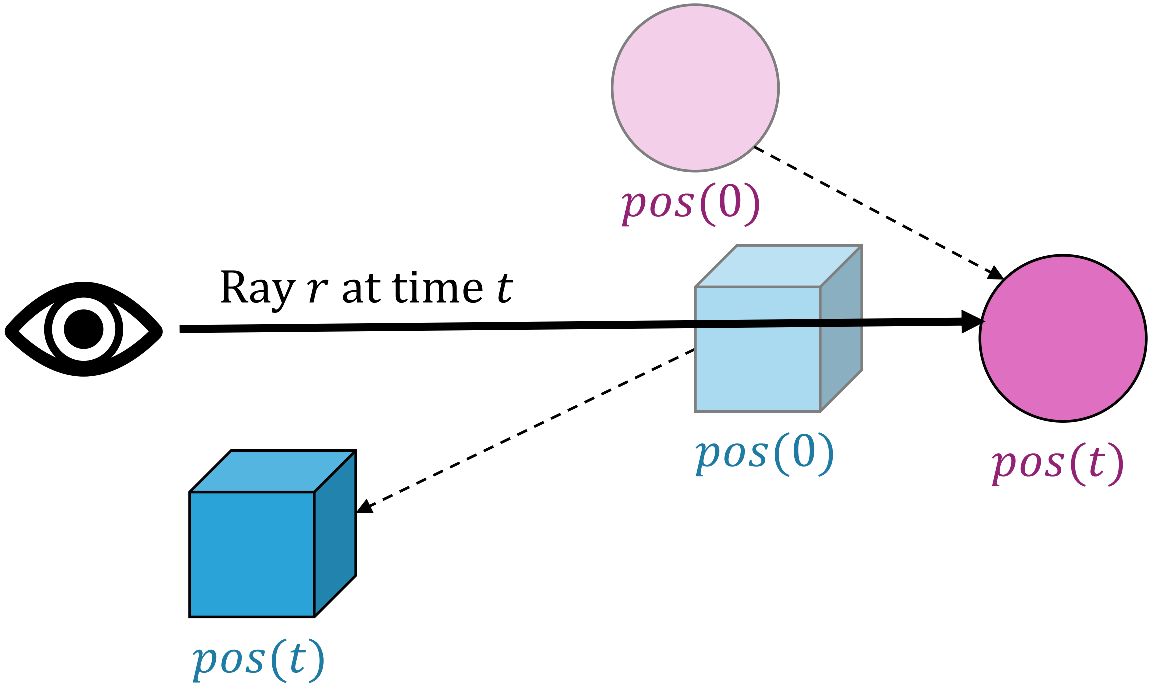
Depth-of-Field

- Simulate depth-of-field for focal length f [2]
  - Create ray r through pixel as before
  - Find focal point $\boldsymbol{f}$ along r at distance f
  - Pick random location x, y on lens (disk)
  - Actually shoot ray from x, y through $\boldsymbol{f}$



Motion Blur

- For motion blur, we make geometry a function of time t
  - Draw a random t, follow path as before
  - Check which triangles ray intersects at t
  - Acceleration structure must support parameterization with t!



Niabot, "Two animations rotating around a figure, with motion blur (left) and without", Wikipedia, "Motion Blur", horizontally flipped, CC BY-SA 3.0

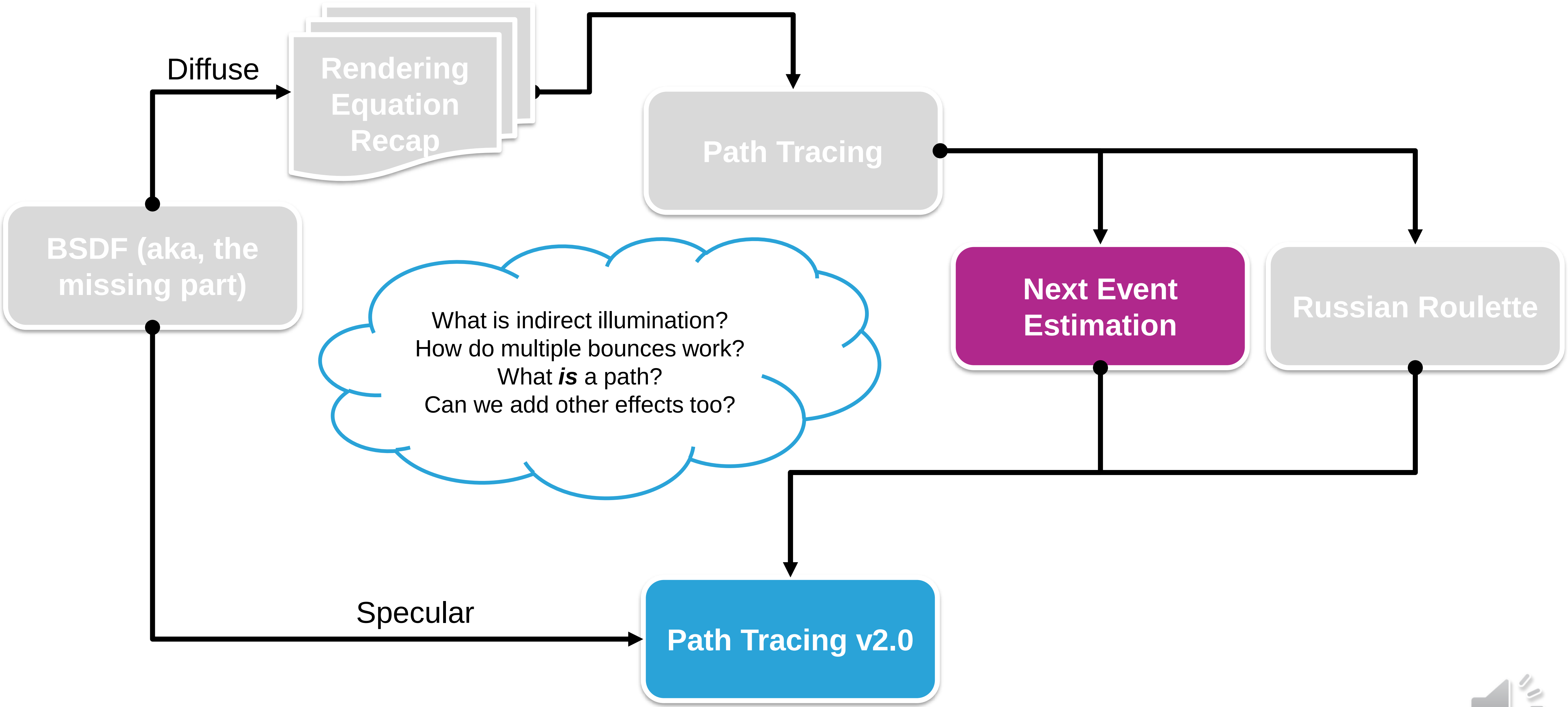
Back to Noise

- Higher-dimensional path tracing is particularly prone to noise

- How can we fix it?

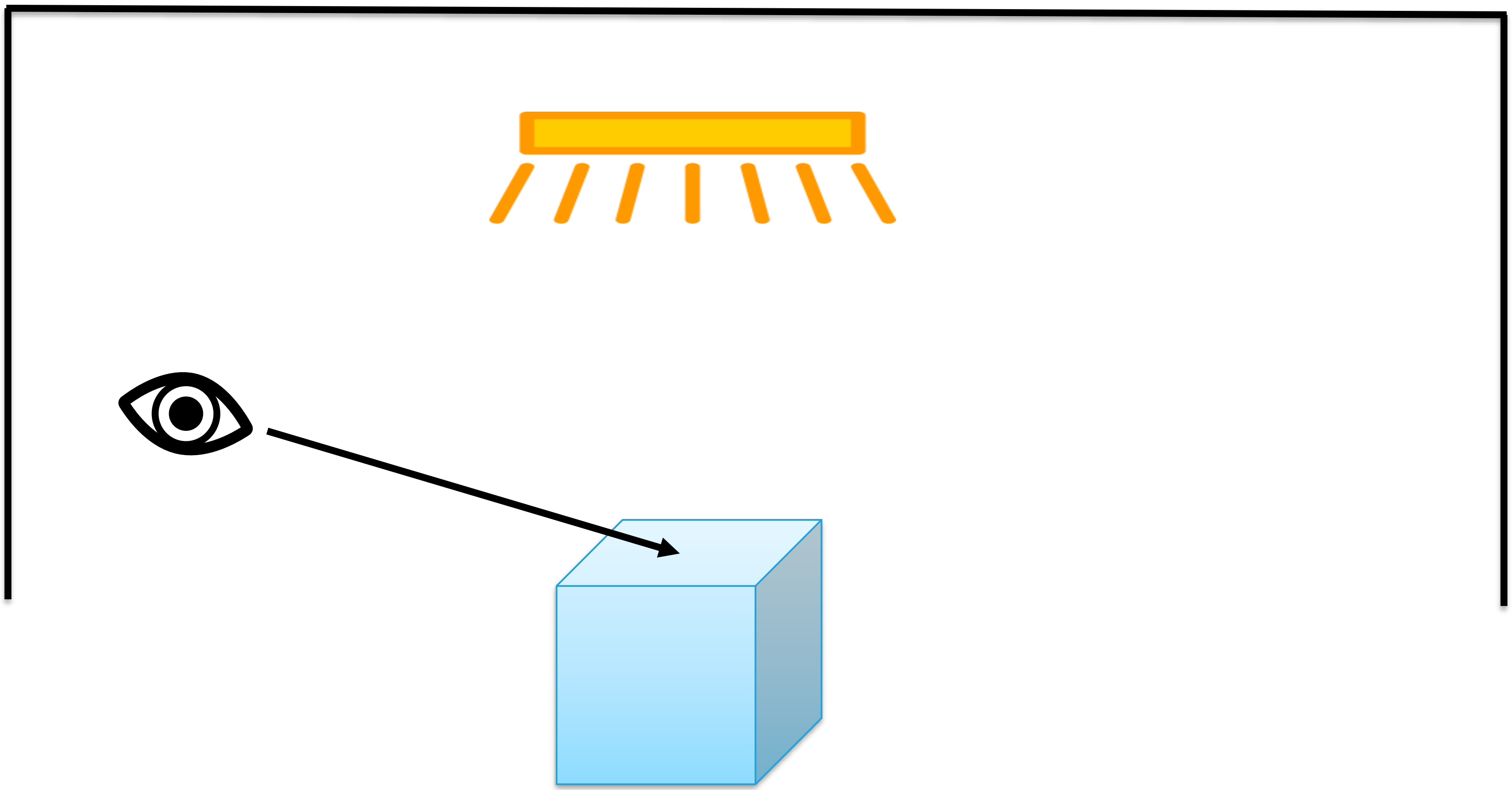
- We already saw some solutions – and they still apply
  - More samples (brute force)
  - Importance sampling whenever we can (we already do it for BRDFs)
  - Light source sampling, recursively? → **Next Event Estimation (NEE)**
  - *Building on NEE: recursive multiple importance sampling*

Today’s Roadmap

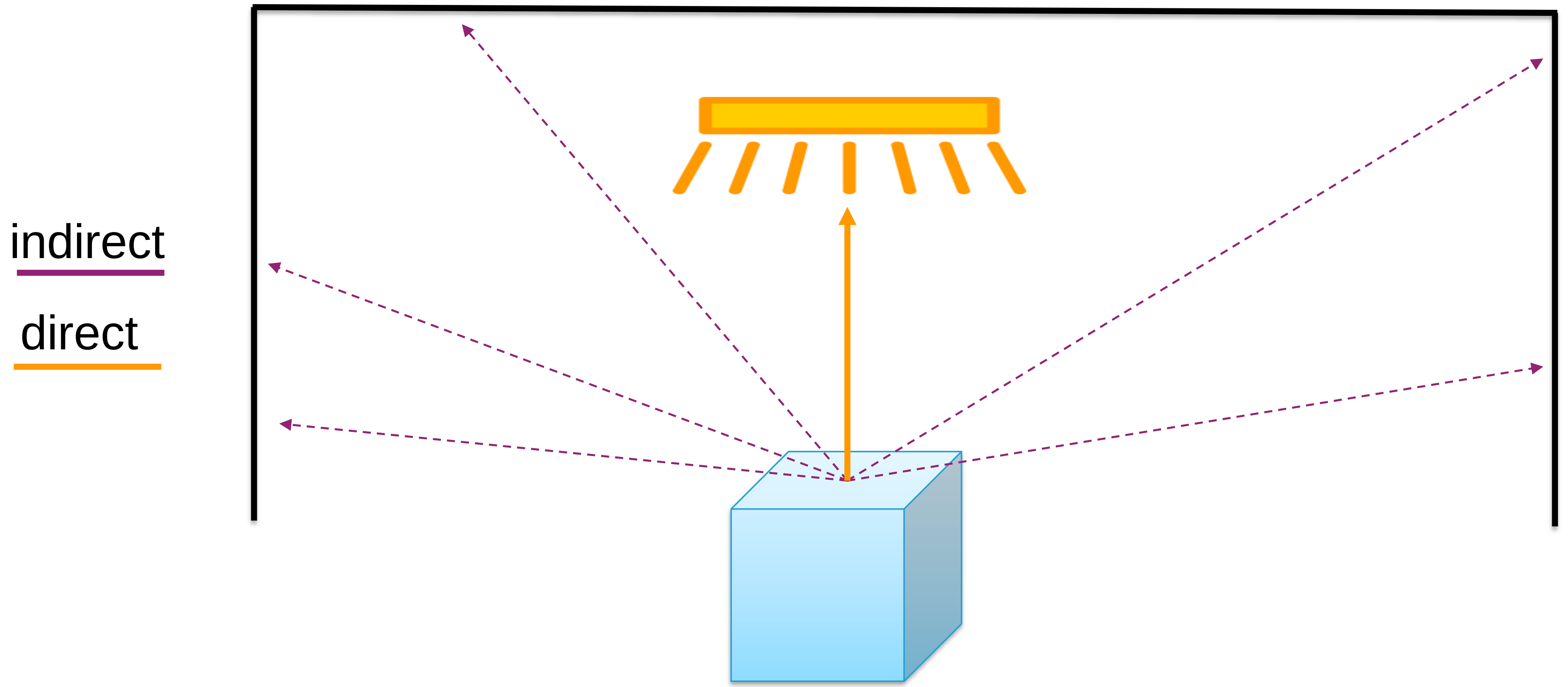


Next Event Estimation

- Builds on light source sampling. Think: where can light come from?

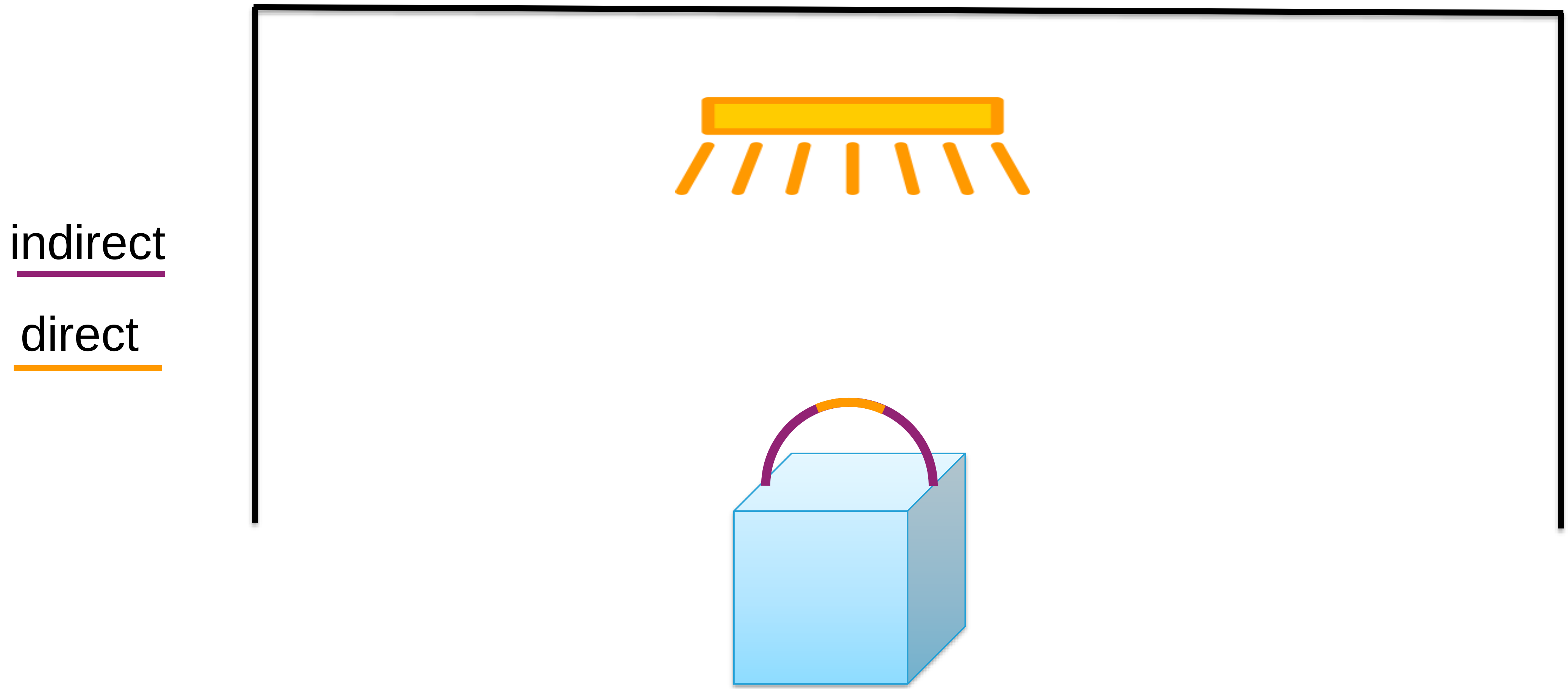


- Builds on light source sampling. Think: where can light come from?



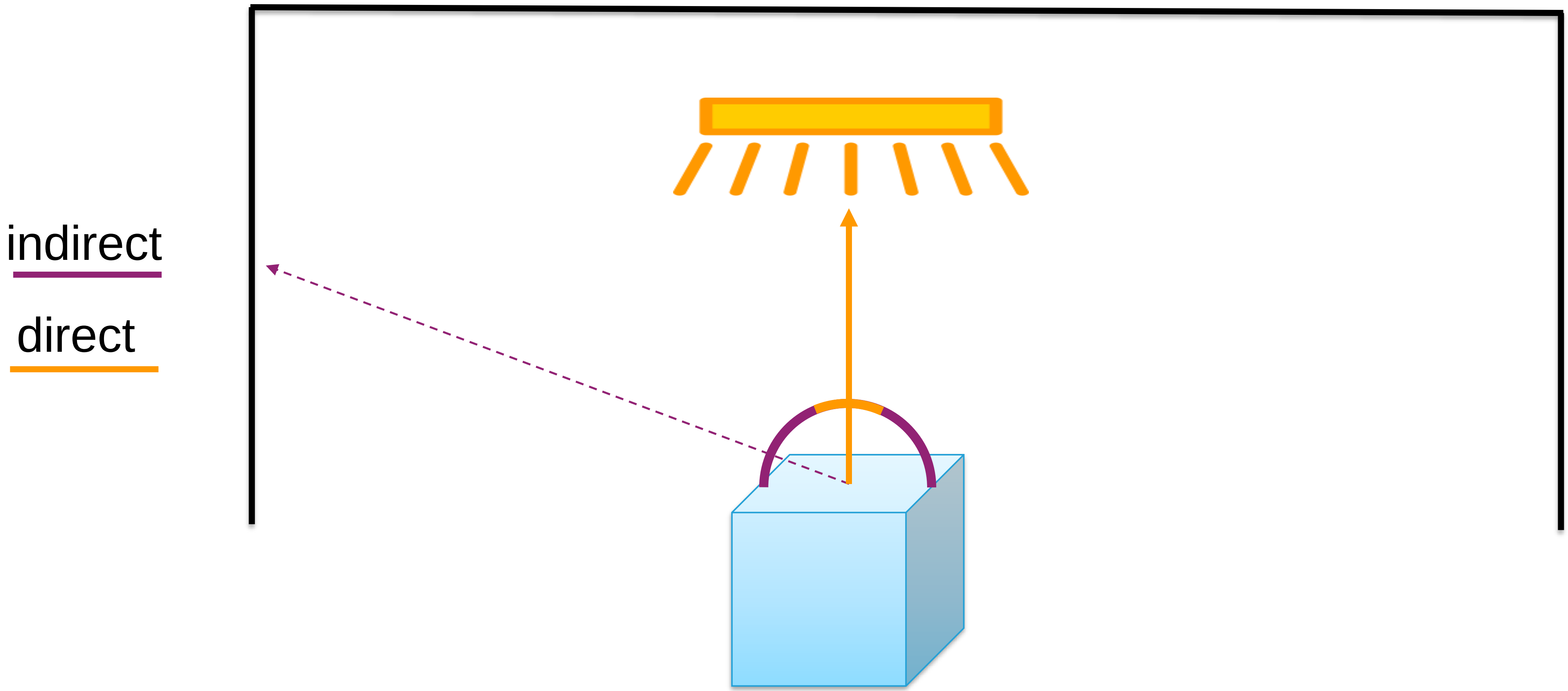
Next Event Estimation

- We can map out the full hemisphere and distinguish direct/indirect



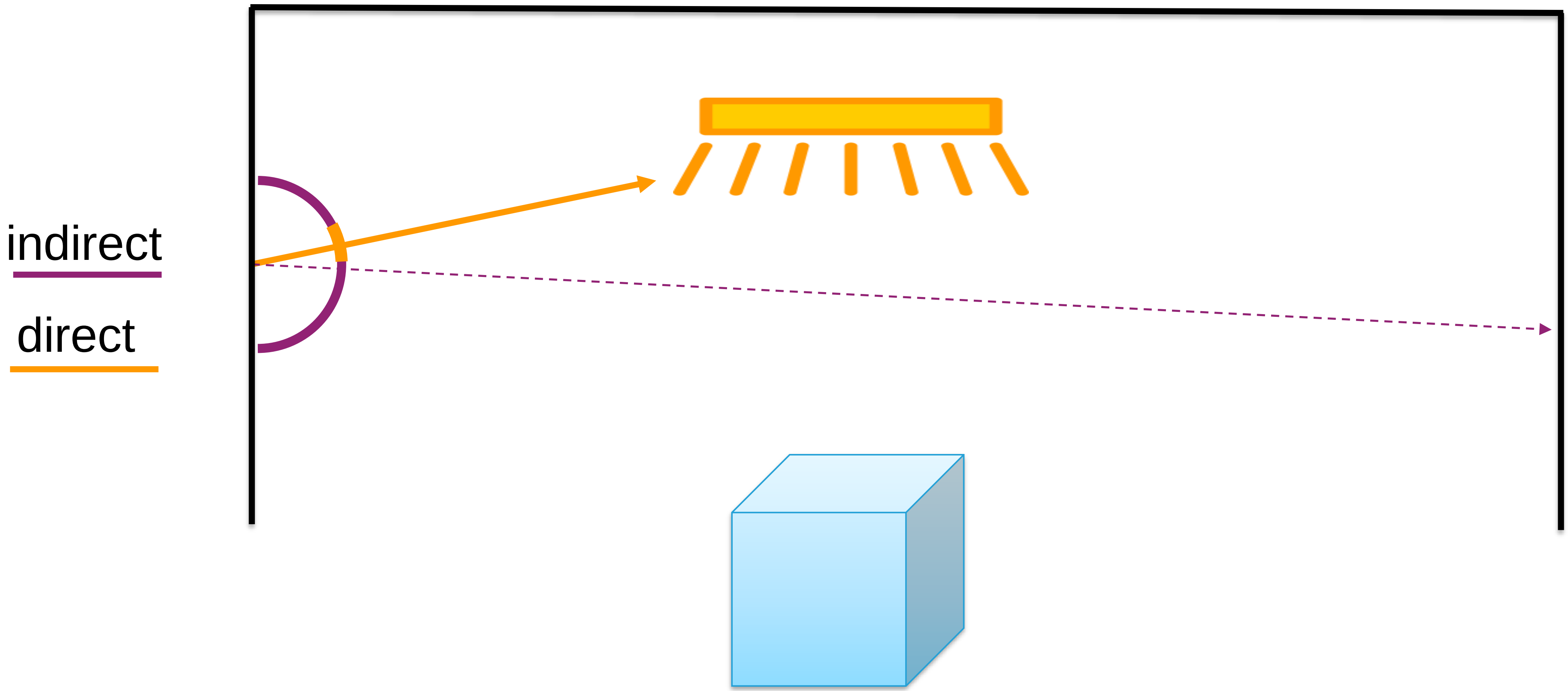
Next Event Estimation

- At each bounce, use light source sampling to get direct illumination
- Use BRDF sample to generate new direction to collect indirect light



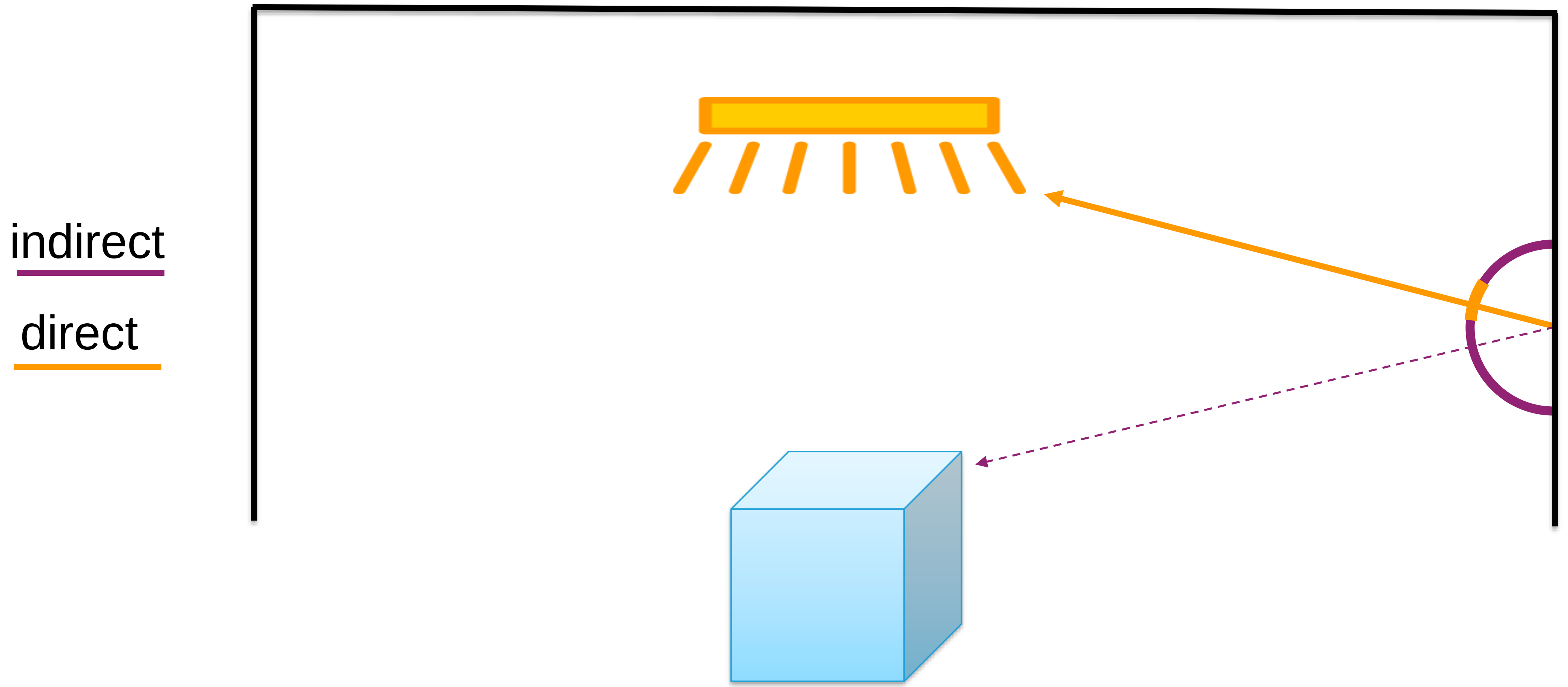
Next Event Estimation

- At each bounce, use light source sampling to get direct illumination
- Use BRDF sample to generate new direction to collect indirect light



Next Event Estimation

- At each bounce, use light source sampling to get direct illumination
- Use BRDF sample to generate new direction to collect indirect light

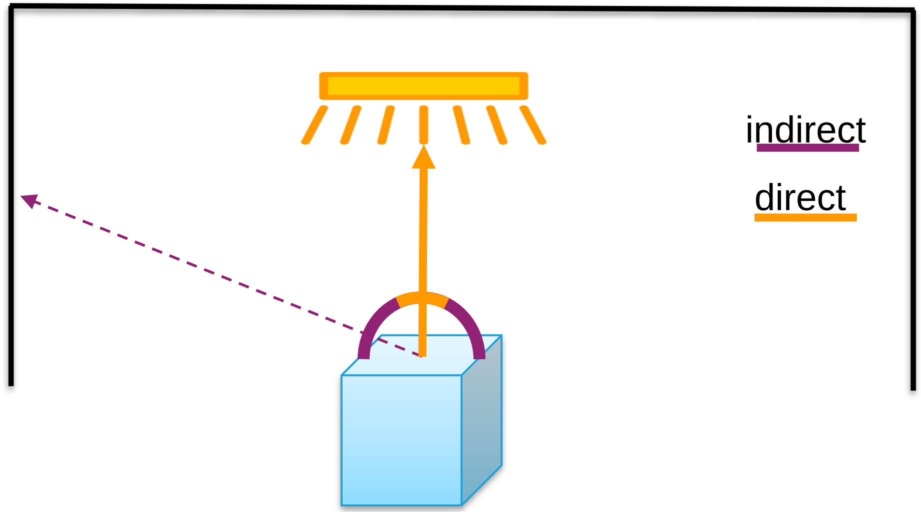


Divide and Conquer

- Light source sampling for direct light

  +

- BRDF sampling for finding indirect light

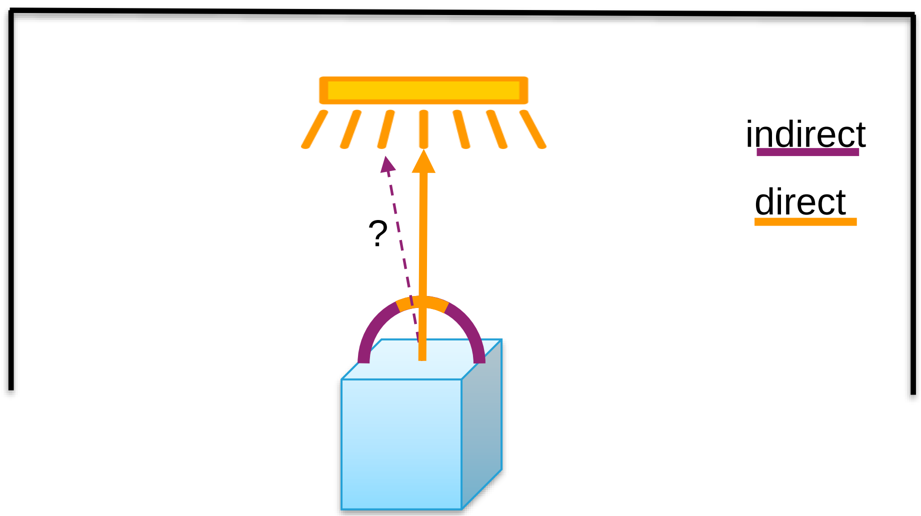


- Add them together to cover the hemisphere
  - Light source sampling to project light source onto hemisphere
  - Importance sampling of the hemisphere via BRDF to generate next direction to collect potential indirect light from next hit point

Divide and Conquer

- Problem: what happens if the indirect sample actually hits the light?

- Indirect sample accidentally direct, light is added twice in one bounce!



- We did not restrict BRDF directions (and we actually don't want to)

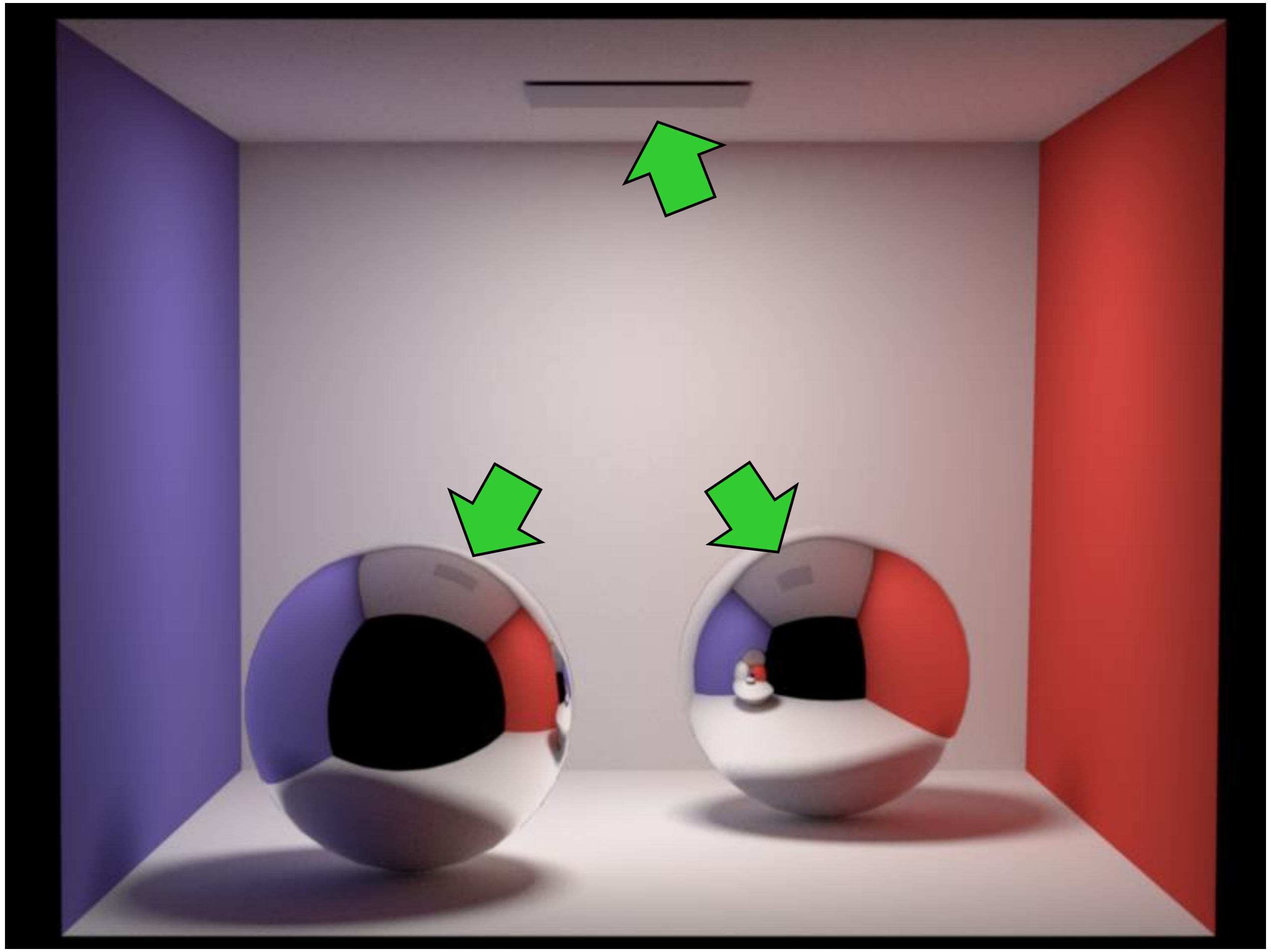
- Idea: actually ignore emittance completely! We don't need it, because what emittance did, light source sampling now does for us

First Attempt at Next Event Estimation

```
Color emitted = 0;

[...]

// DON‘T take care of emittance
// if (isLightSource(its)) emitted = getRadiance(its);

[...] // Stop at some point based on Russian Roulette probability

BRDF brdf = getBRDF(its);

// Get direct sample on a light source with light surface sampling
LightSourceSample sampleLS = sampleLightSurface(its);
// Light source direction is not generated by the BRDF, so we evaluate rendering equation the old way
// Note: sampleLS.radiance already includes light source cosTheta(y), 1/r^2, 1/dA
float direct = BRDFevaluate(brdf, -ray, sampleLS.dir) * cosTheta(its, sampleLS.dir) * sampleLS.radiance;

// BRDF should decide on the next indirect sample
BRDFSample sampleBRDF = BRDFsample(brdf, -ray);
// Call recursively for indirect lighting
Color indirect = Li(scene, sampleBRDF.wo, depth + 1);
return (emitted + direct + sampleBRDF.value * indirect) / RR_probability;
```

A First Test Run of Next Event Estimation

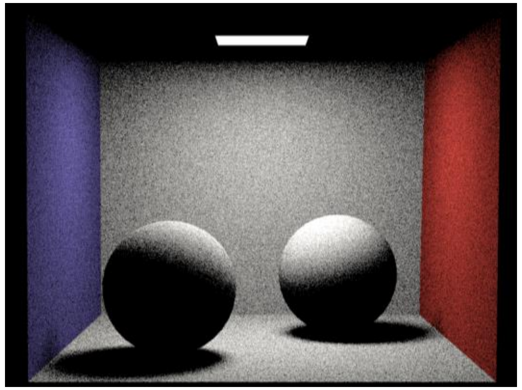
- The noise is mostly gone now!
- But some information lost:
  - Specular reflections of lights
  - Light sources themselves
  - Caustics
- It seems eliminating emittance altogether was too much…

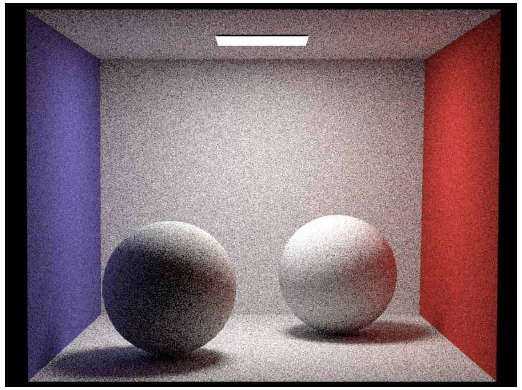


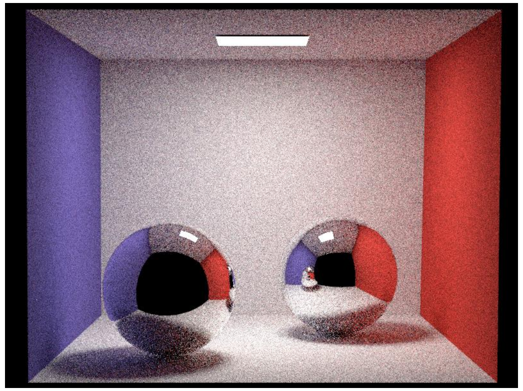
Enabling Emittance for Special Paths

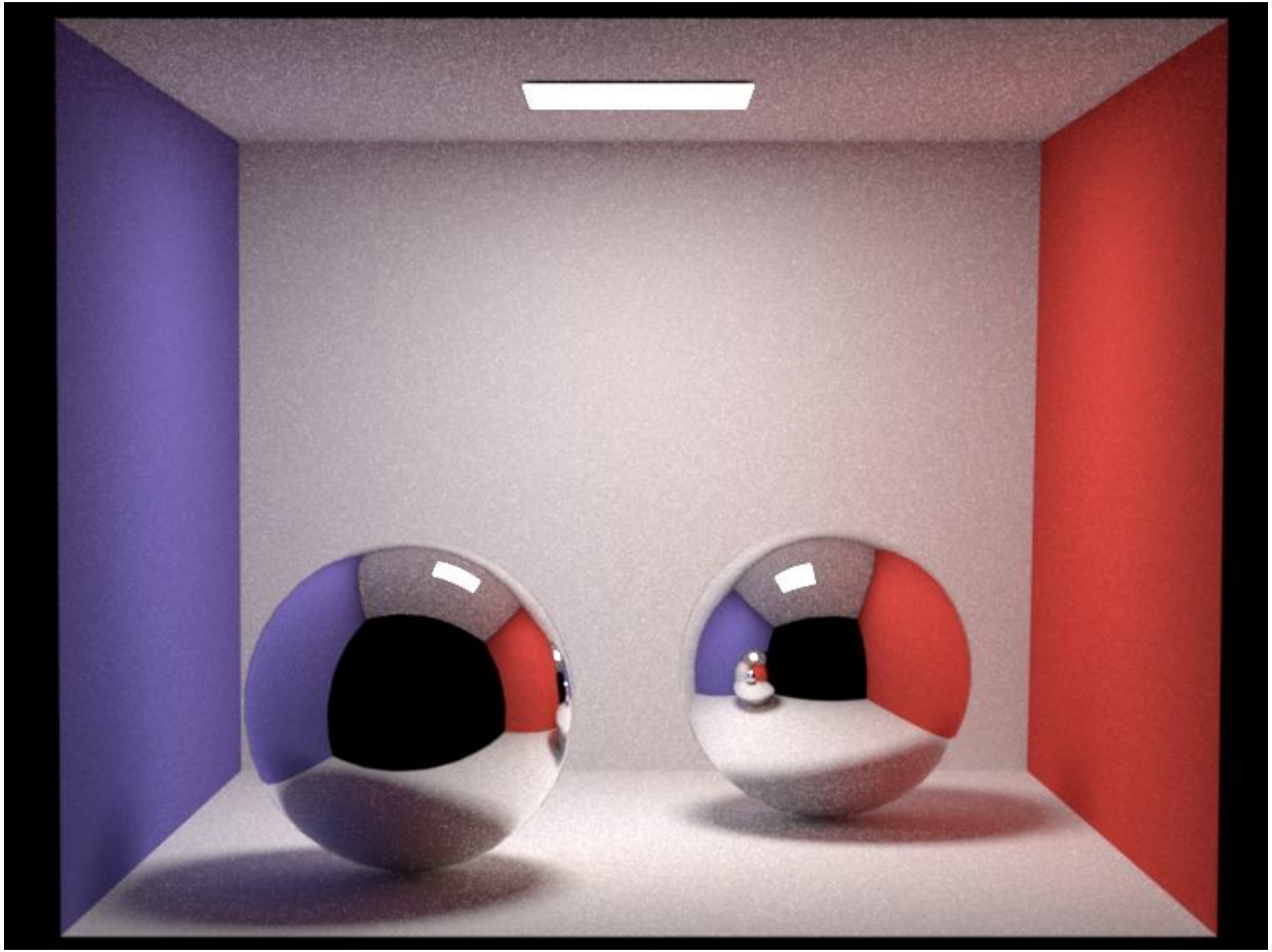
- At the first bounce, there was no previous bounce for which we computed the direct lighting (i.e., no next event estimation)
- With specular materials, we know that the BRDF allows reflection only from a single direction, thus light source sampling will fail
- Idea: actually ignore emittance **most of the time**, except if
  - The current hit point is the first hit after leaving the camera
  - The last material was fully specular (light source sampling denied)

Path Tracing + Russian Roulette + Next Event Estimation







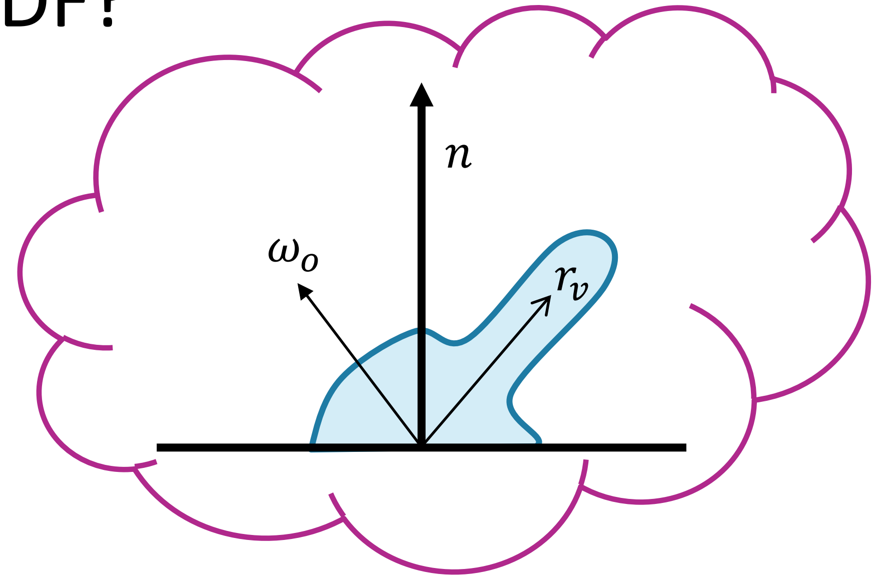


How to Handle Glossy BRDFs?

- Most objects are actually neither completely diffuse nor completely specular. We never talked about glossy BRDFs…

- Also, we only looked at *reflections* (B**R**DFs). What about other light scattering or transparency, the full B**S**DF?

- We will handle those soon…



References and Further Reading

- [1] *Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, Kevin Dale, Greg Humphreys, Matthias Zwicker, and Henrik Wann Jensen. 2008. Multidimensional adaptive sampling and reconstruction for ray tracing. ACM Trans. Graph. 27, 3 (August 2008)*

- [2] Depth-of-Field Implementation in a Path Tracer: https://medium.com/@elope139/depth-of-field-in-path-tracing-e61180417027

- [3] *Ryan Overbeck, Craig Donner, and Ravi Ramamoorthi. Adaptive Wavelet Rendering. ACM Transactions on Graphics (SIGGRAPH ASIA 09), 28(5), December 2009.*

- [4] *Johannes Hanika, Marc Droske, and Luca Fascione. 2015. Manifold Next Event Estimation. Comput. Graph. Forum 34, 4 (July 2015), 87–97.*