Rendering: Next Event Estimation

Bernhard Kerbl

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

Welcome to this new lecture, where we will discover, and apply, a method called Next Event Estimation to improve the quality of our renderings.

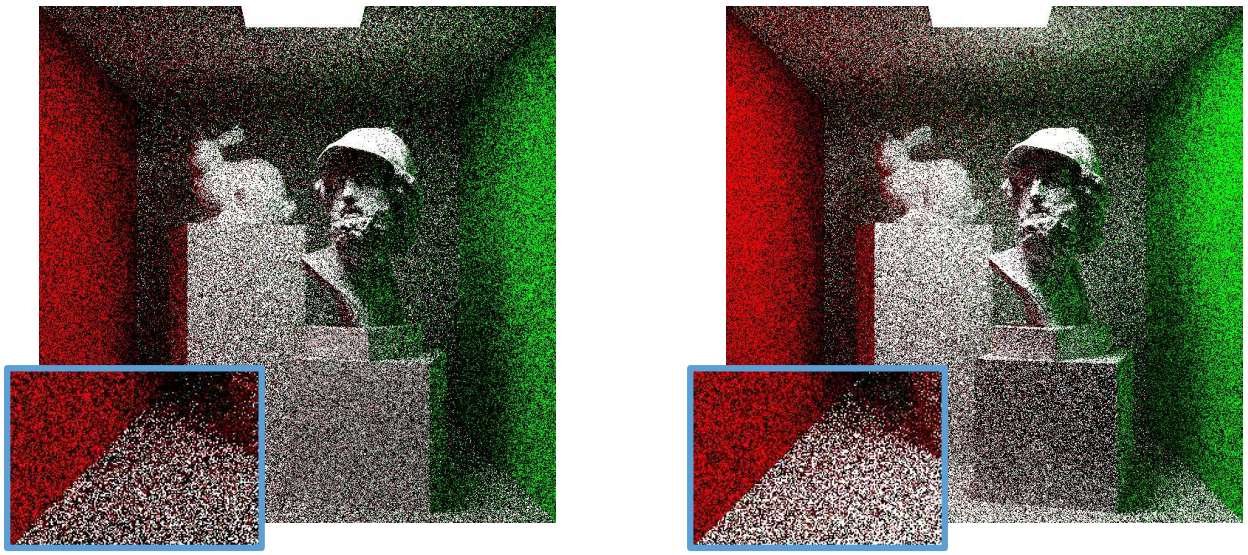
Accepting Importance Sampling as New Standard

- At this point, we should mention that we won‘t be needing uniform hemisphere sampling anymore…
- We can replace it for most purposes with BRDF sampling, that is, importance sampling the hemisphere based on the material
  - E.g., for diffuse materials, cosine-weighted hemisphere sampling
  - We will see solutions for other materials soon!
- BRDF sampling usually improves quality in most cases (how much?)

Previously, we saw how we can use importance sampling to create samples on the hemisphere in such a way, that we can cater to the needs of the BRDF. That is, we can preferably sample directions from which we can expect a lot of contribution to the reflected radiance at a surface point, depending on its material properties. We have already derived how we can do this for perfectly diffuse materials, and ended up with cosine-weighted hemisphere sampling. So from this poin on, there is actually no real need anymore to use uniform hemisphere sampling. Instead, we will use BRDF sampling as an umbrella term, for importance sampling the directions in the hemisphere that make the most sense for a given material. So far, we only have diffuse materials, but this may change in the future. BRDF sampling will always refer to a materials proper importance sampling technique of ist BRDF. We do this because we generally expect it to perform better

than naive uniform sampling. We previously saw a few examples how BRDF sampling can improve image quality. However, even when we use it, the quality of images that we construct in an acceptable amount of time, like a minute or less, can leave a lot to be desired.

Diffuse Path Tracing: Uniform vs BRDF Sampling



- Cosine-weighted hemisphere sampling “works”… can we do better?

Consider this scene for instance, where we compare BRDF sampling to uniform hemisphere sampling. The difference is there, but it quite rather subtle. That makes sense, since the difference between a uniform and a cosine-weighted sampling distribution is not that big. In contrast, today, we will be looking at a technique that can improve the quality of this path traced rendering, with all its color bleeding, multiple bounces and soft shadows, instantly and significantly.

How to Get Rid of All that Noise?

- Higher-dimensional path tracing is particularly prone to noise
- How can we reduce noise in our renderings?
- Common suggestions when looking for ways to reduce noise:
  - Use more samples (brute force, often takes too much time)
  - Use importance sampling (already applied)
  - Use today's technique, **Next Event Estimation (NEE)**
    - Based on something we saw before: light source sampling

All these multidimensional effects that path tracing can offer – multiple bounces, depth of field, area lights and so on – are impressive, but are also responsible for creating noise in our renderings.

So how can we get rid of it?

The simplest method that always applies is to just use more samples.

But, if we don't have infinite compute power, we might want to look for other ways to improve quality.

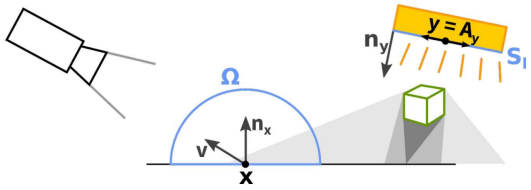
We are already importance sampling the BRDFs, and that is a good start.

But we also saw previously, that hemisphere sampling is often not the best idea. We have been looking at light source sampling before. What if we can bring it back….

But recursively?

Revisiting the Math Behind Light Source Sampling

Direct lighting (soft shadows) (usable for rendering)



light intensity at position y on the surface

visibility (new, ray tracing)

emitter cos(θ)

$$L_e^{[l]}(x) = \int_{S_l} f_r(x, y \to v) L_e^{[l]}(y) V(x, y) \cos(\theta_x) \frac{\cos(\theta_y)}{r^2} \, \mathrm{d}A_y$$

Light going in direction v

Material, modelled by the BRDF

receiver cos(θ)

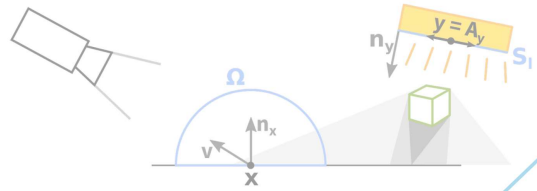
distance

Let‘s quickly revisit the concept of light source sampling, because the last time we saw it was quite a while ago. In contrast to sampling the hemisphere uniformly or according to a material‘s BRDF, with light source sampling, we pick a sample location on the surfaces of the lights in our scene, and then try to connect from a shaded surface point to that sample location. This implies that we have to know where our light sources are in the scene. In Monte Carlo integration, we then take each sample, and pretend it speaks for the full light source surface, so we project the entire light surface area onto the hemisphere and see how much solid angle it covers. The more samples we take, the more accurate this approximation becomes. Some of them will be visible, some of them will be blocked by other objects between the shaded point and the light source, and when we sum and average enough of these results, we get a good

approximation of how much light from the light source actually reaches the shaded point.

Revisiting the Math Behind Light Source Sampling

Direct lighting (soft shadows) (the same, but more explicit)



light intensity at position y on the surface

visibility (new, ray tracing)

emitter cos(θ)

$$L_e^{[l]}(x) = \int_{S_l} f_r(x, y \to v) L_e^{[l]}(y) V(x, y) \left(\frac{y - x}{|y - x|} \cdot n_x\right) \frac{\left(\frac{x-y}{|x-y|} \cdot n_y\right)}{|x - y|^2} \, \mathrm{d}A_y$$

Light going in direction v

Material, modelled by the BRDF

receiver cos(θ)

distance

We can make a few simplifications to the above formula to make it easier to transfer into code and save a few expensive trigonometric operations in the process.

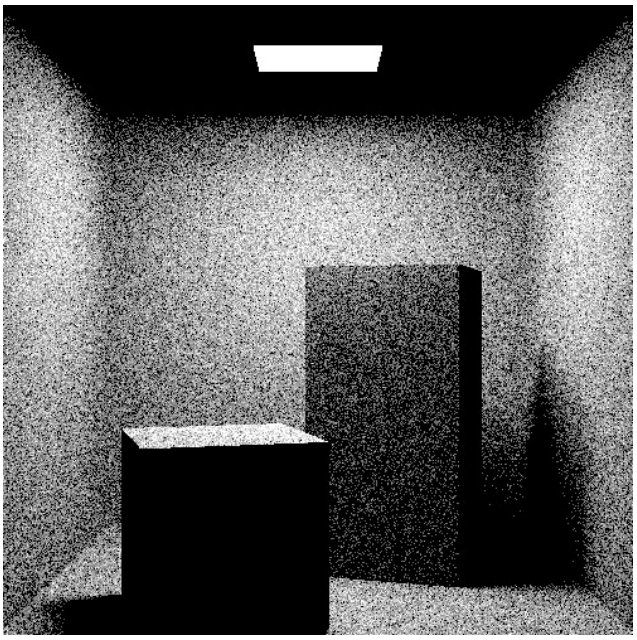
Direct Lighting with Light Source Sampling

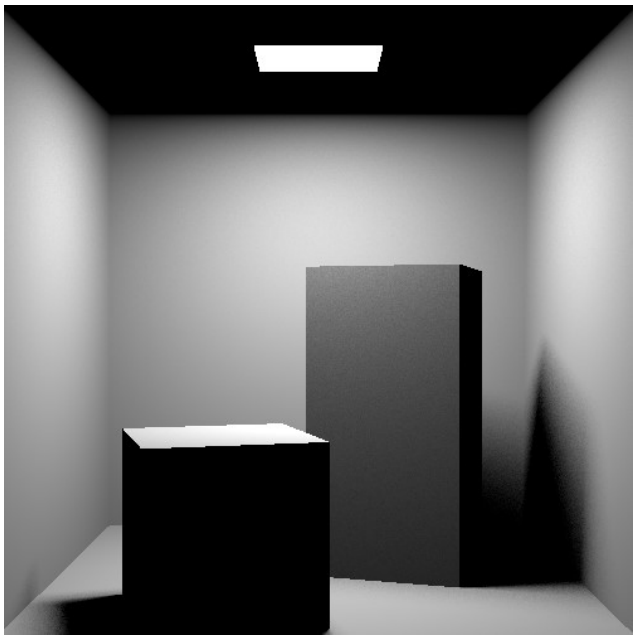
```
function Li(v_inv)
      x = scene.trace(v_inv)
      f = x.emit
      y, area = light_surface_uniform_world()
      omega = (y-x).normalised()
      r = make_ray(x, omega)
      v = 0
      if (scene.trace(r) == y)
            v = 1
      P = dot(y.normal, -omega) / dot(y-x, y-x)
      f = x.alb/pi * y.emit * v * dot(x.normal, omega) * P * area
return r
```

$$f_r(x, \omega \to v) L_e^{[l]} V(x, y) \cos\theta \frac{(-\omega \cdot n_y)}{|x - y|^2} A^{[l]}$$

A simple implementation of direct lighting might look like this. Assuming that we have uniform sampling on the surface of each light, we can first detect whether or not a sample is visible from the shaded point, scale up the radiance emitted from it by the surface area of the light source, and then project it onto the hemisphere around the shaded point.

Direct Light: BRDF Sampling vs Light Source Sampling





As we have previously seen, in many scenarios, switching from uniform or BRDF sampling to light source sampling can tremendously increase the quality of our renderings. This is because, as we can appreciate now, every sample that we invest to find some light will make an informed decision in which direction it should go. For direct lighting and with scenes that do not have very large area lights, as we see it here, this makes a lot of sense.

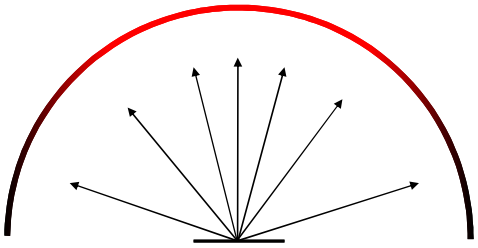
Direct Light: BRDF Sampling vs Light Source Sampling

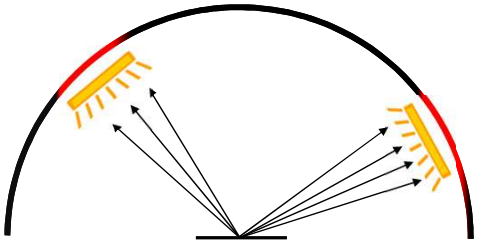
- In both cases, you want to find out from how many directions on the hemisphere a point does receive how much light
- With uniform or BRDF sampling, pick one direction for each sample, and pretend that this direction speaks for the entire hemisphere
  - But if you collect many of these estimates and average, you converge
- With light source sampling, pick a point y on the light, and pretend the evaluation (can y illuminate x?) holds for its entire surface area
  - But if you collect many of these estimates and average, you converge

Of course, in both cases we are computing the same result in the long run, so both methods will arrive at the same result given enough samples. The key difference lies in how these two methods accumulate the light that a point receives over the entire hemisphere. For uniform or BRDF sampling, we pick directions and compensate each sample's radiance so that it can provide a suitable representation of the entire hemisphere, given the underlying sampling distribution. With light source sampling, we still try to sample the hemisphere, but if there are some directions in it that cannot lead to light, right off the bat, we can skip those directions and focus sampling on directions where it matters.

Importance Sampling Direct Light: BRDF vs Light Source

- Both BRDF and Light Source Sampling perform importance sampling
- They selectively put samples at opportune places on the domain
  - BRDF: assuming uniform lighting, which directions contribute most?
  - Light source: knowing light locations, which directions may hit them?





If we pit BRDF sampling versus light source sampling, we can clearly see that both are importance sampling methods. That is, they selectively put samples at opportune places in their domain, that is, where there is a good chance to find light that affects the shaded point. We have illustrated both concepts here, and you can see in red the density of samples as they are distributed over the integration domain, the hemisphere.

Exploiting Light Source Sampling for Path Tracing?

- Can light source sampling help us with path tracing?
  - Based on projecting all known light sources onto the hemisphere
  - Every surface in the scene is a potential source of (indirect) light!
  - If we treat every surface as a potential light source, we are back to randomly sampling the full hemisphere…
- Idea: follow each ray via multiple indirect bounces, but at each bounce, compute the direct lighting from light source surfaces!
  - Detected light at each bounce is no longer dependent on coincidence
  - This is what we refer to as ***Next Event Estimation***

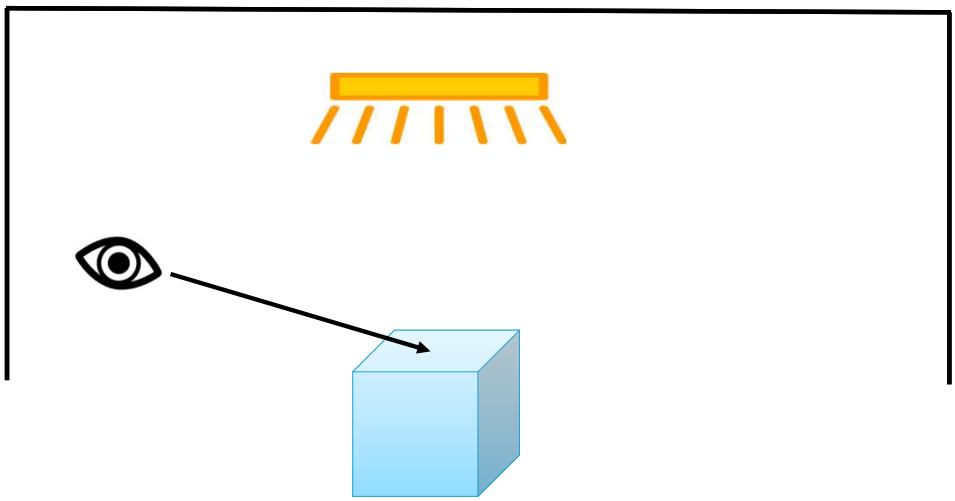
Now an important question to ask is, if light source sampling can actually really help us with path tracing. It worked great for direct lighting, but path tracing is all about indirect light. Now if we consider indirect light, things become a little more complex: with indirect light, we also consider light that is reflected off of surfaces, perhaps over multiple bounces. That means, theoretically, every surface in the scene can be an indirect light source. This means we cannot trivially benefit from light source sampling in path tracing because, if every surface point can be a light source, then we have infinitely many light sources in our scene to sample, and then we are basically back to randomly sampling the hemisphere again.

However, there is a method to make BRDF sampling and light source sampling work together nicely. The gist

of it is this: at each shaded point, we use BRDF sampling to find new locations to consider for indirect lighting. But at each point that we hit, we then perform light source sampling to collect all direct light from the known light sources in our scene. So in a way, we are separating direct and indirect light, and use the fact that indirect light is just direct light from a few bounces in the future. This concept is what we call next event estimation. Let’s illustrate this idea with visual example.

Next Event Estimation

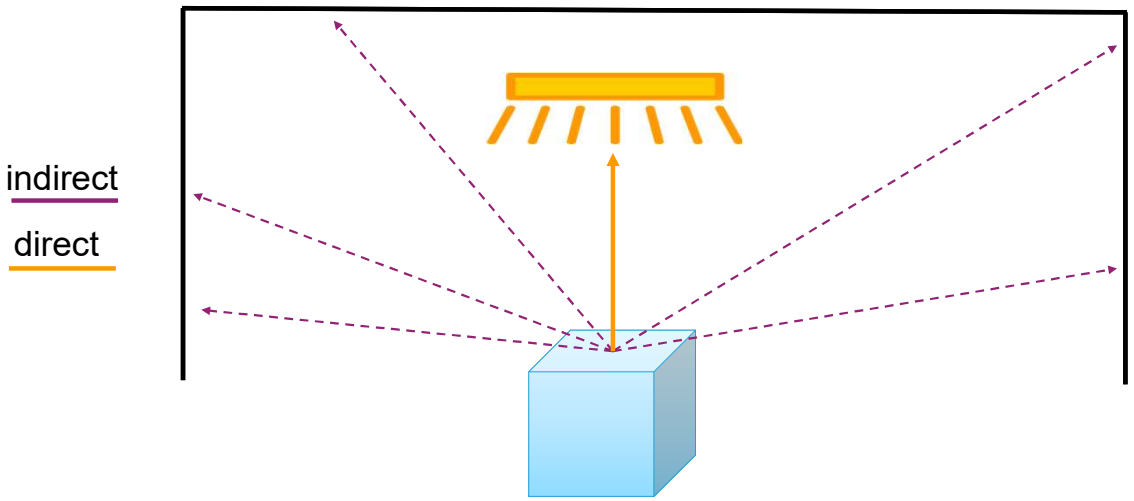
- Builds on light source sampling. Think: where can light come from?



Lets assume a simple scene with a single area light. After the first view ray has hit a point in the scene, there are two ways in which light can arrive at this current point.

Next Event Estimation

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

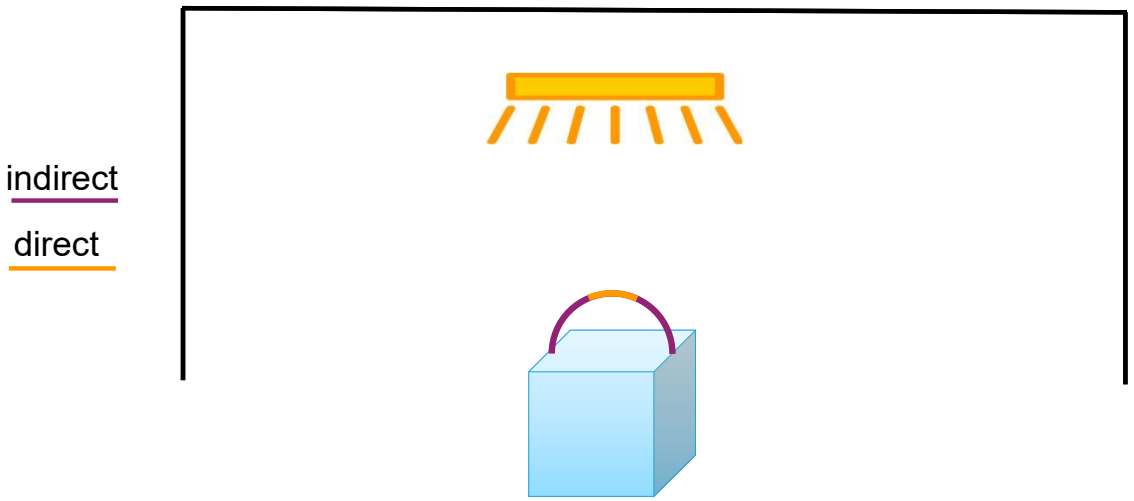


Either it comes from a point that is not a light source, in which case we might get some “indirect illumination”. Or it comes from a light source directly, which would account for the direct illumination.

Note that during simple hemisphere sampling, each of these locations would be a possible hit for the ray to land on in the next bounce.

Next Event Estimation

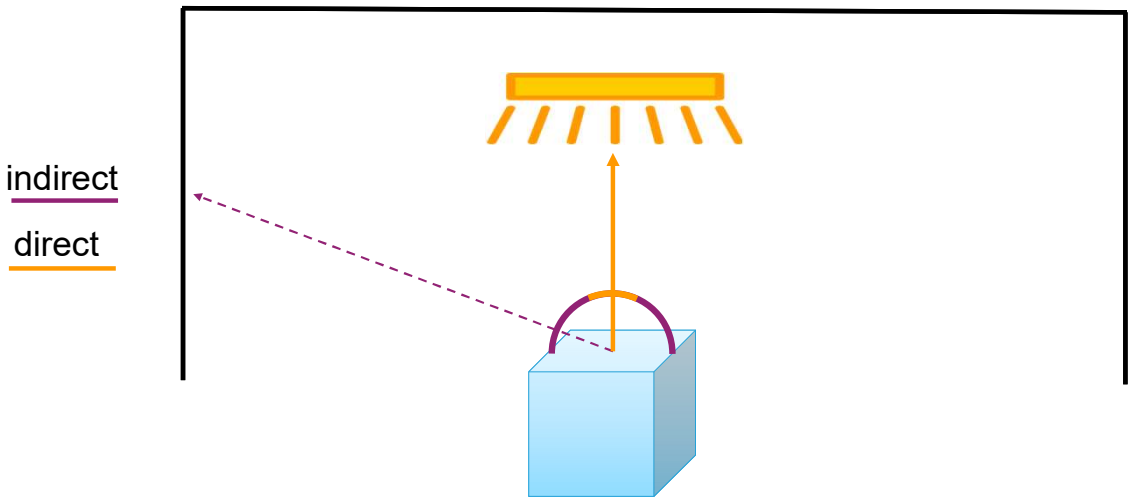
- We can map out the full hemisphere and distinguish direct/indirect



If we look at all the places that light can come from, we can map out directions on the hemisphere that will create direct or indirect light for the current hit point.

Next Event Estimation

- At each bounce, use light source sampling to get direct illumination
- Sample the BRDF to create direction for collecting indirect light



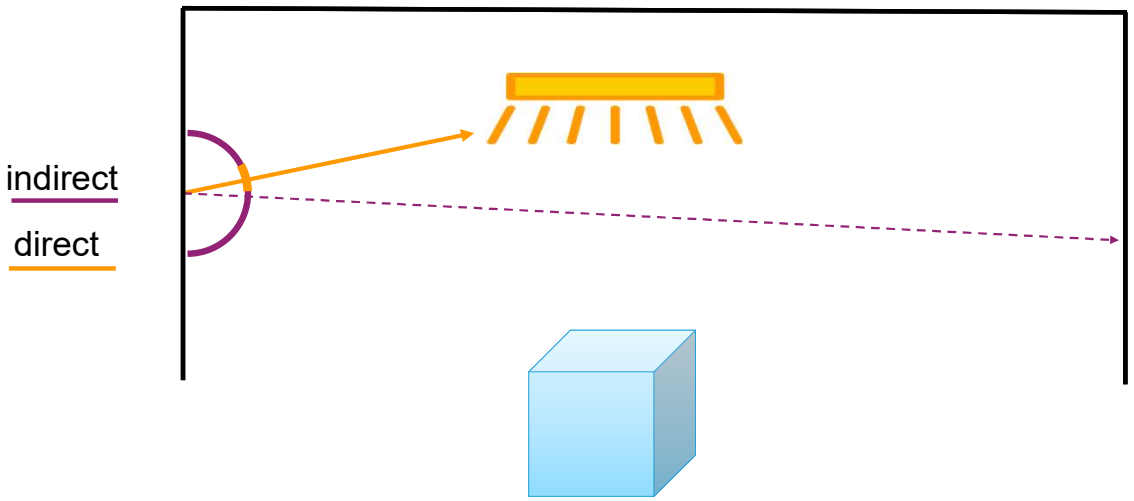
Remember that, what light source sampling essentially does if it succeeds, is projecting the area of the light source onto the hemisphere.

If we perform light source sampling, we can collect the direct illumination from the light source, thus taking care of that portion of the hemisphere.

We can then shoot an indirect sample that goes to a location where it might collect indirect illumination in the next bounce.

Next Event Estimation

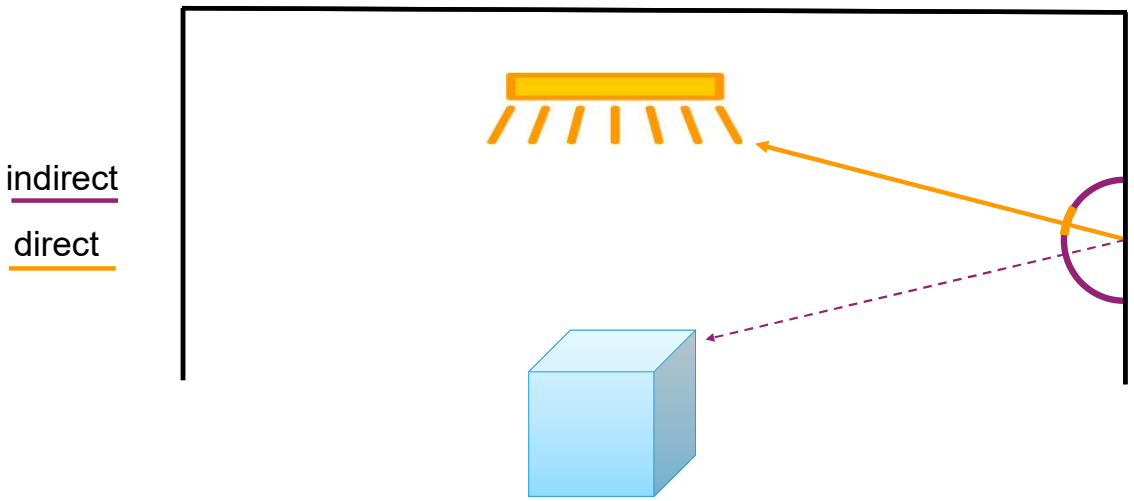
- At each bounce, use light source sampling to get direct illumination
- Sample the BRDF to create direction for collecting indirect light



If we do this recursively, we basically get next event estimation. The projection of the light source on the hemisphere is accounted for at each bounce, and indirect illumination will be accounted for by light source sampling from the following bounces.

Next Event Estimation

- At each bounce, use light source sampling to get direct illumination
- Sample the BRDF to create direction for collecting indirect light

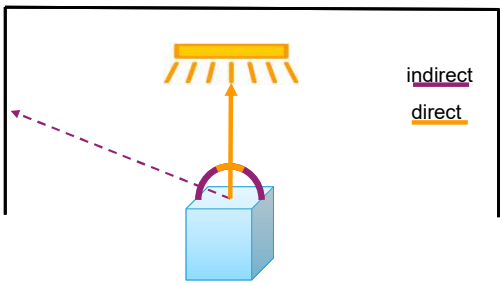


Divide and Conquer

- Light source sampling for direct light

  +

  BRDF sampling for indirect light



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

In the path tracing routine, all that we need to do is add up the results from light source sampling and from recursive hemisphere sampling.

That way, we will be projecting light sources for direct light and leave the selection of qualified directions for indirect sampling to the BRDF and its importance sampling.

First Attempt at Next Event Estimation

```
function Li(v_inv, D)
        …
        f = x.emit
        …
        // Apply Russian Roulette at some point
        …
        direct = <direct lighting with light source sampling>
        …
        indirect = <indirect light (recursive) with BRDF sampling>
        …
        f += (direct + indirect) / (rr_prob)
        return f
```


A first implementation in code could look like this. Here, we can benefit from the methods that we derived previously for computing direct and indirect light arriving at a point. Note that in this case, the variables for direct and indirect light already contain all the relevant factors like the cosine, division by the PDF, BRDF term, and so on. The direct light is the light received by a point with light source sampling. The indirect light is essentially the result of the recursive calls to Li, weighted with all the correct terms, except for the Russian roulette probability, we need to keep that out because it is part of the recursion algorithm, not the physical light computation itself. So we can simply add up direct and indirect light obtained in this way and see what this gives us.

First Attempt at Next Event Estimation

- Too bright! Better get some sunglasses to look at this…

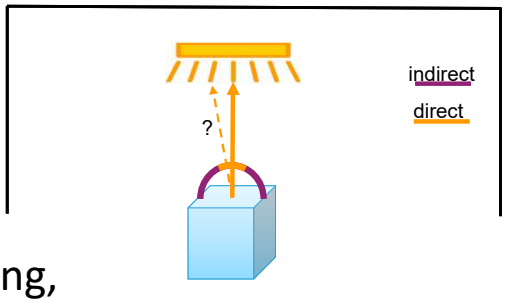
BRDF Sampling

Current attempt

So this first approach, at least, is not completely broken, but it definitely gives us images that are way brighter than they should be. We probably breaking the laws of energy conservation here, because the rendering on the left is our physically-based, unbiased reference. So what is going on here?

Revisiting our Approach

- Question: what happens if an indirect sample eventually hits a light?
- Indirect sample is accidentally direct, light is collected twice in same bounce!



- Due to the compensation of BRDF sampling, we end up with double the amount of light we should have!
- Idea: if we have double the amount of light, can we just divide by 2?

Of course, if you paid close attention to the previous lecture, something should not sit right with you. If we combine light source sampling AND hemisphere sampling and simply add them up, won't we get twice the light for the scene?

And you would be right. We assumed that our indirect samples would not be hitting the light source, but there is nothing that keeps them from doing so. And actually, we also don't want them to avoid light sources.

A light source itself might be receiving some light from elsewhere. Consider a strong spot light, shining onto a weak area light. That would be some significant indirect illumination on a light source!

So what can we do to avoid adding up double the light in the scene?

A straightforward remedy would be the following: if we have twice the amount of light that we should have, because, basically, we are now performing two different, equally valid sampling strategies simultaneously, can’t we just divide the resulting light received at each surface point by two?

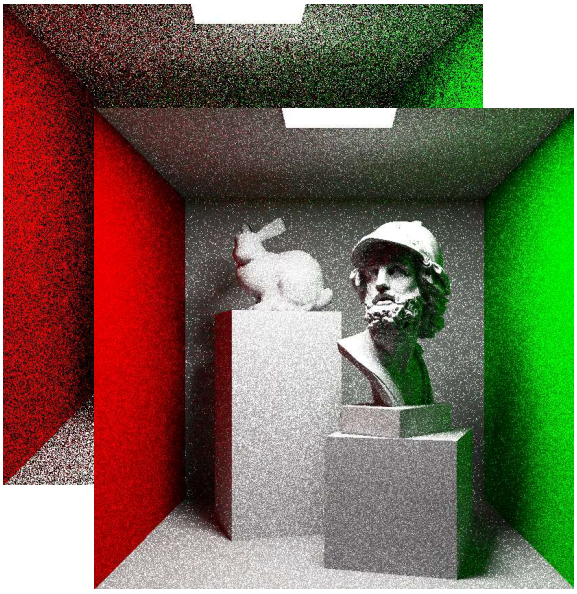
Second Attempt at Next Event Estimation

```
function Li(v_inv, D)
        …
        f = x.emit
        …
        // Apply Russian Roulette at some point
        …
        direct = <direct lighting with light source sampling>
        …
        indirect = <indirect light (recursive) with BRDF sampling>
        …
        weight = (D == 0) ? 0.5 : 1 // halve indirect light after 1st bounce
        f += weight * (direct + indirect) / (rr_prob)
        return f
```


Let's try it in another iteration of putting our solution into code. The part that we are now counting double is all the light after the first intersection. So we can simply put a condition in our computation, and weight all the light that we find recursively after the first intersection with a factor of 0.5, thereby correcting for the fact that we are combining two sampling approaches simultaneously.

Averaging BRDF Sampling and Light Source Sampling

- The noise has significantly improved!
- Mixing several importance sampling techniques and weighting them...?
- It‘s multiple importance sampling!
- There are multiple ways to do MIS, let‘s quickly revisit some of them…

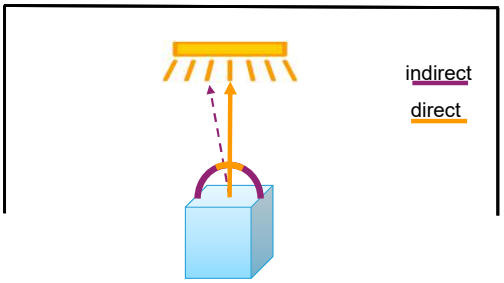


And as you can see, applying this method actually works very well to reduce the noise in our renderings. Clearly, adding in light source sampling has made our images a lot cleaner. But if you think back, what we did here should seem a little familiar to you. We mixed several importance sampling methods and then weighted them. That’s actually multiple importance sampling! So we came to this first solution almost accidentally, but let’s quickly recap what methods we have available to do proper multiple importance sampling.

Multiple Importance Sampling Techniques Revisited

- Multiple MIS weightings to choose from:
  - Averaging: $w_i(x) = \frac{1}{N}$ for N techniques
  - 1 or 0, depending on each new sample
  - Balance heuristic (**Veach 1997**)

$$w_i(x) = \frac{p_i(x)}{\sum_{k=0}^{N} pk(x)}$$



- Let's try something basic: assigning 0/1 weights to techniques
  - Assumption: light source sampling is better at **direct** light than BRDF
  - Keep BRDF sampling for indirect light, disable its direct light collection

The three relevant methods we will consider here are averaging, which we just used, then 1/0 weighting, where we give each technique a weight of 1 or 0, depending on situation or the sample itself, and lastly the balance heuristic which, as Eric Veach suggests, is usually a good choice.

So for now, let us switch from averaging to using 1/0 weights. Right now, we only used averaging to get rid of the fact that we have double the amount of light at each surface point that we should have, because we use both BRDF sampling and light source sampling to compute light. However, we can assume that most of the noise that we still see is coming from BRDF sampling, because of its uninformed randomness. But wait a minute, what if we can just disable the collection of light via BRDF sampling? Essentially, we could

disallow BRDF sampling from ever finding any direct light and leave that job completely to light source sampling, which performs it so well. We can still keep BRDF sampling as a vehicle for moving the sampling locations for light source sampling around the scene, but we will deny it the detection of emittance on objects when it hits them.

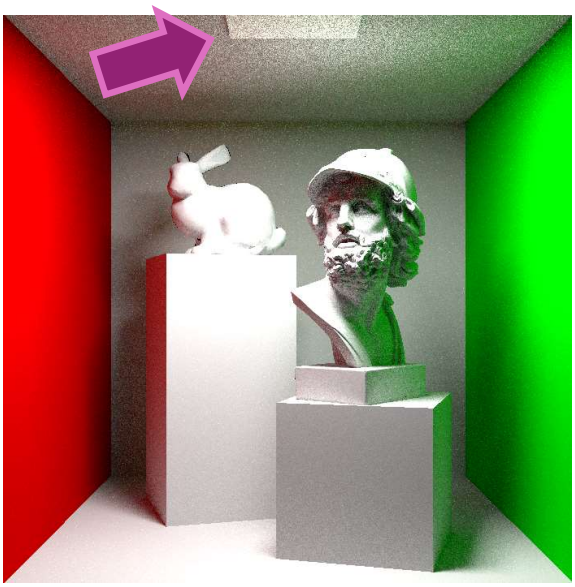
Removing Surface Emittance from Path Tracing

```
function Li(v_inv, D)
      …
      // f = x.emit
      f = 0       // <-- 0 weight, removes direct light of BRDF sample at D-1!
      …
      // Apply Russian Roulette at some point
      …
      direct = <direct lighting with light source sampling>
      …
      indirect = <indirect light (recursive) with BRDF sampling> // unchanged
      …
      f += (direct + indirect) / (rr_prob) // 1 weight
      return f
```


The change to make this work is quite straight-forward. Basically, the emittance term is always set to 0. We don't need the BRDF sampling portion of our algorithm to compute any light for us, we just need it to move the sampling locations for light source sampling around the scene recursively. Hence, we give full weight to the results that we compute with light source sampling, a very basic case of 0/1 multiple importance sampling.

Removing Surface Emittance from Path Tracing

- Looks better than averaging!
- But some information lost: light sources themselves don't seem to emit any light anymore…
- Logical, we removed emittance!
- It seems eliminating emittance altogether was too much…



By using a 0/1 weighting scheme, we have pretty much achieved what we have set out to do, that is, we have significantly reduced noise. At first sight, this seems like a great result

But actually, we have done too much: by disabling emittance completely, we have lost some effects that are vital.

Specifically, look at the light source: it doesn't seem to send out any light to the camera. But clearly, we can see its effect on everything else in the scene! Where did we go wrong?

Re-Enabling Emittance for Special Paths

- At the first bounce, there was no previous bounce for which we could compute the direct lighting with light source sampling
- I.e., we did not perform “next event estimation” at the 0^{th} hit point, the camera (or viewpoint) itself
- Simple fix: actually, ignore emittance **most of the time**, except if the current hit point is the first hit after leaving the camera / eye
- Can use recursion depth to enable or disable emittance term

We are close, we just need to make minor modifications for cases where we cant afford to lose emittance because NEE has failed:

When we first shoot our initial ray into the scene, there was no previous bounce on the pixel surface where we could have computed direct lighting with light source sampling. Basically, we did not perform next event estimation before we shot the ray into the scene.

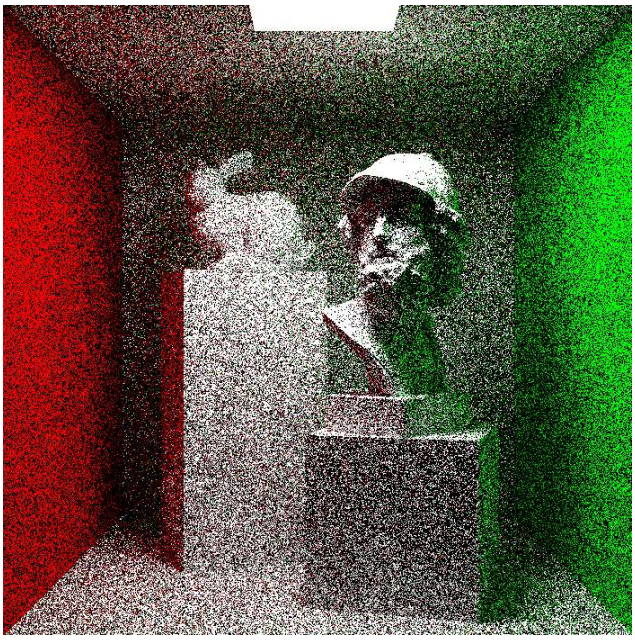
Accounting for this special case is easy enough and should relieve our issues. The simplest fix to remedy this is to allow adding emittance the first time that we hit a point in the scene. This can be seen as a crude way of light source sampling for each pixel that we render in our image.

Re-Enabling Emittance for Special Paths

```
function Li(v_inv, D)
        …
        f = 0     // 0 weight, most of the time, except at first intersection
        if (D == 0)
                f = x.emit
        …
        // Apply Russian Roulette at some point
        …
        direct = <direct lighting with light source sampling>
        …
        indirect = <indirect light (recursive) with BRDF sampling>
        …
        f += (direct + indirect) / (rr_prob)
        return f
```


That change is easily made, based on the recursion depth parameter. Now we have a slightly more complex 0/1 weighting setup, where we give a weight of 1 to BRDF sampling at the 0^{th} bounce, because light source sampling cannot run there, and 0 otherwise.

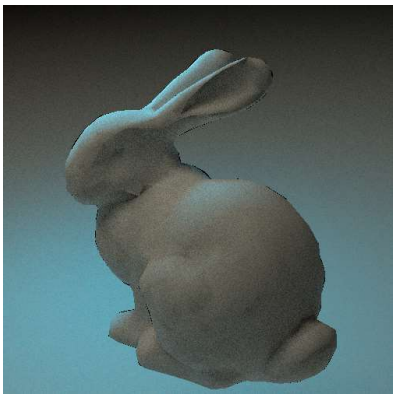
Diffuse Next Event Estimation – Unlocked!



And with this, we have already unlocked next event estimation. Believe it or not, both of these renderings were created with the same number of samples, just one with BRDF sampling and the other with next event estimation. A great improvement for your renderings, in many cases. Especially, note how we can benefit from clean images at the first few bounces. The more bounces we follow with our rays, the more noise can still creep in, but bottom line, we can now make much cleaner images in a much shorter amount of time.

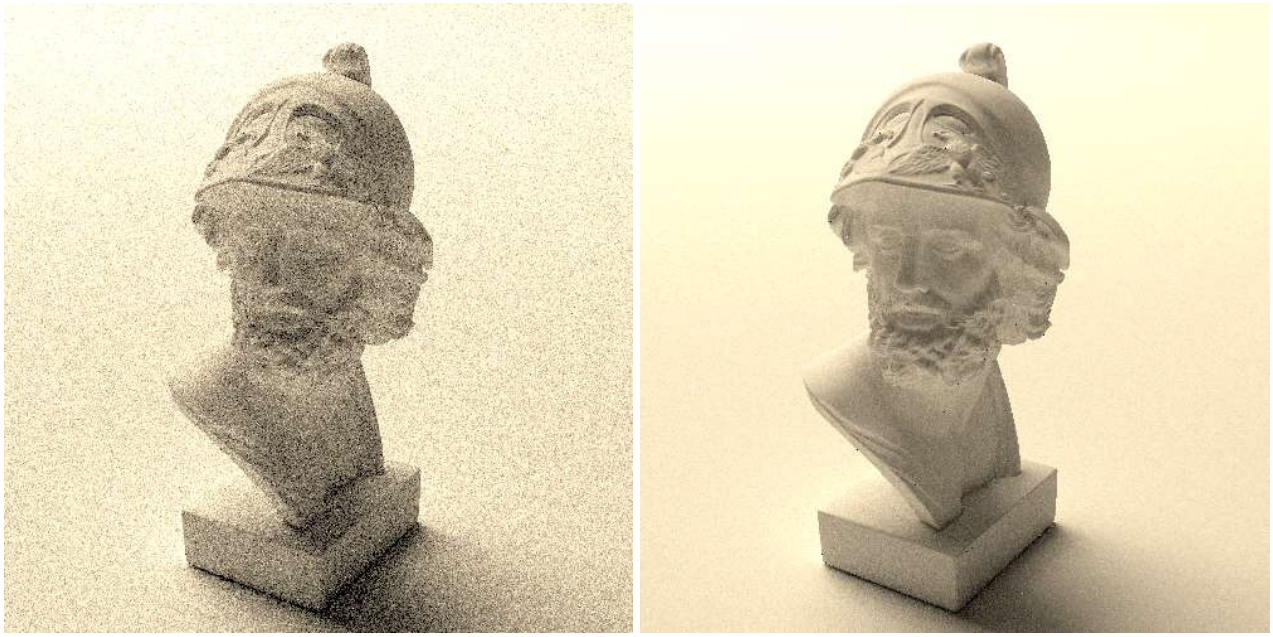
More Diffuse Next Event Estimation Results

- BRDF importance sampling vs. next event estimation
  - In many cases, significant improvement of quality
  - Same number of samples, very similar runtime (NEE slightly slower)



With this new technique, we can get significant improvements in a wide range of scenes with the same number of samples invested. It must be noted that, of course, next event estimation is somewhat slower because it basically performs an extra direct lighting method at each bounce. But the tradeoff is extremely in favor of using next event estimation over not using it. For scenes like the one depicted here, you may have to invest 10 or 20 times the number of samples with BRDF sampling to achieve the same smooth results in your output renderings.

More Diffuse Next Event Estimation Results



Here is another result of using next event estimation instead of BRDF sampling. Especially for diffuse materials, next event estimation has a huge impact on quality, because diffuse BRDF sampling is actually not extremely different from uniform hemisphere sampling.

More Diffuse Next Event Estimation Results

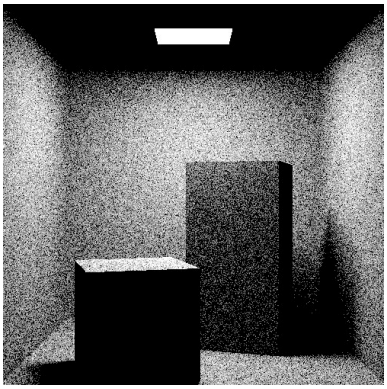
- Next event estimation is highly effective, but still no silver bullet

- Always a more challenging scene to push your renderer to its limit…

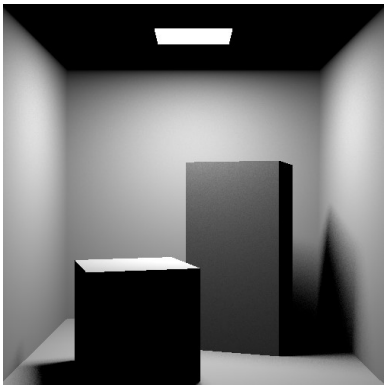
But of course, next event estimation is not a silver bullet. No matter how many improvements you add to your path tracer, there will always be another, more challenging scene that will push it to its limits. With very large scenes and a lot of complex, indirect lighting, even a path tracer with a good BVH and next event estimation may struggle to resolve images without noise in an adequate amount of time. So even though we have come a very long way, of course, even this technique eventually has its limits.

But is Using 0/1 Weights Really the Best Choice?

- Effectively, we now use light source sampling for all direct light
  - Using light source over BRDF sampling often improves direct lighting



White box, BRDF sampling



White box, light source sampling

  - But, as usual, the benefit very much depends on the input scene

But, there is one more thing we should consider about next event estimation. The way we have applied it right now is an accepted solution, and has given us a clear improvement of quality in many scenes. However, the changes we made are based on the assumption that light source sampling is strictly better than BRDF sampling. While it may seem that way when we look at common scenes, we have to test our beliefs if this is actually always true. For this purpose, let's again go back to direct lighting. Clearly, in the scene depicted here, light source sampling is significantly better at producing a noise-free image. But, as usual, the benefit of choosing one technique over the other is dependent on the input that we are processing…

Direct Light: Light Source Sampling is Not Always Better…

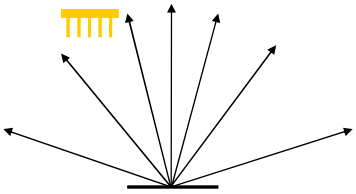
- Scene with small and large area light: BRDF/light source sampling

BRDF sampling: large grey light works well, small blue light causes a lot of noise

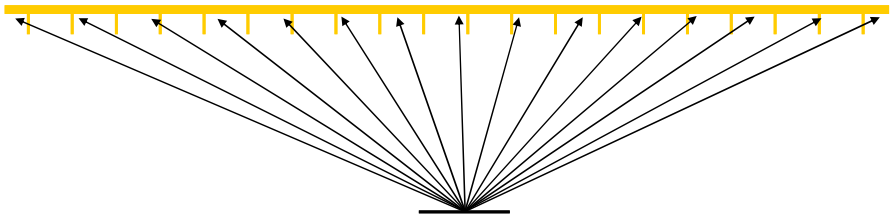
Light source sampling: small blue light works well, large grey light causes a lot of noise

Take for instance this scene here. We have a model at the center, and two prominent light sources in the scene. One is a small, blue light coming from the left side of the screen. The other is a large, grey, overhead light, for instance to simulate the open sky. We have rendered the scene here with a direct lighting integrator, using BRDF sampling on the left and light source sampling on the right. Curiously, it seems that each technique has a favorite light source and one it doesn't like so much. The BRDF sampling technique struggles with the small, blue light, while it does ok with the large overhead skylight. On the other hand, light source sampling performs well on the blue light source, but its illumination from the skylight is very noisy. What is going on here?

How Sampling Methods Can Fail (Diffuse Materials)



Importance sampling the BRDF, we may **miss** the smaller light sources



Light source sampling a large, overhead light source (sky) accumulates more samples at **flat** angles – the opposite of importance sampling!

- Light source sampling can struggle even more with other BRDFs!
- But remember, there is another MIS option: the balance heuristic

We can find out if we illustrate the two techniques side by side, with their respective failure cases. Here, we are considering their effects when we use diffuse materials. On the left, we see that BRDF sampling pointlessly distributes its samples all over the hemisphere. By doing so, it has a low chance of actually finding smaller light sources, and thus their contribution becomes patchy and noisy if we don't use a very high number of samples. On the other hand, when we have a very large overhead light source and use light source sampling to sample its surface uniformly, we find that we get distributions that are not favorable for the diffuse BRDF: its biggest contributions are made towards the apex of the hemisphere, but in this setup, we see that actually more samples are placed at the flat angles of the hemisphere. That is the exact opposite of importance sampling, and therefore we can end up increasing the variance and therefore

the noise in our renderings!

But remember, that we don’t have to pick one method or the other. We are doing this currently because we went for a 0/1 MIS weighting path, but that was a conscious choice. We know that averaging both methods is not the best idea either, but there is one last MIS technique we want to look at: the balance heuristic.

Direct Lighting with the Balance Heuristic

- Implementation in direct lighting integrator is not too complicated
  - Randomly choose technique and weight result using balance heuristic

BRDF sampling | Light source sampling | MIS, balance heuristic

And actually, while it may seem a little daunting, implementing the balance heuristic is not too hard, and the benefits are immediate. Here we see an implementation of the balance heuristic for direct lighting. For each ray, we first decide on one of the two available importance sampling techniques with a 50/50 chance and then use the winner to generate a sample. Note that if you followed along with the exercises, you probably already implemented both of them at some point. We then compute the probability of choosing this particular sample with both of the considered sampling techniques, and then use the balance heuristic formula, which we saw in a previous lecture, to compute the adequate weighting term. In essence, that is all that has to be done. Care must just be taken to compensate at the correct points for making a particular choice, but this is left to you as a brain teaser or a practical example for those who are interested.

Efficient Path Tracing (NEE) with the Balance Heuristic

- Consider both light sampling methods in every bounce
- Weight on first intersection must still be 1 for BRDF sampling
- Compute probabilities for selected sample with both techniques, use the balance heuristic to compute adequate MIS weights
- Tricky: different sampling methods are evaluated at different times
  - BRDF sample contribution is found when hitting an emitting surface
  - Contribution of light source sampling is calculated one bounce prior

Now, we can apply a very similar solution for path tracing with next event estimation as well. The difference here is, it makes little sense to choose one technique over the other. When we recursively follow the ray as it bounces through the scene, we can still always perform light source sampling for direct light. At the next indirect bounce, we can then find the emittance that a direct BRDF sample would have gotten, basically for free. So with next event estimation, it makes little sense to pick one technique over the other, instead we can simply perform them both, pretending that the indirect BRDF sample is also a direct BRDF sample.

However, there is a slightly tricky part to this in order to work out correctly. When we use a BRDF sample to compute direct light for previous bounce, we must

make sure that the proper MIS weight from the previous bounce is applied to the found emittance and nothing else. So basically, we need to remember the important properties of the previous bounce to make the appropriate computations one bounce later…

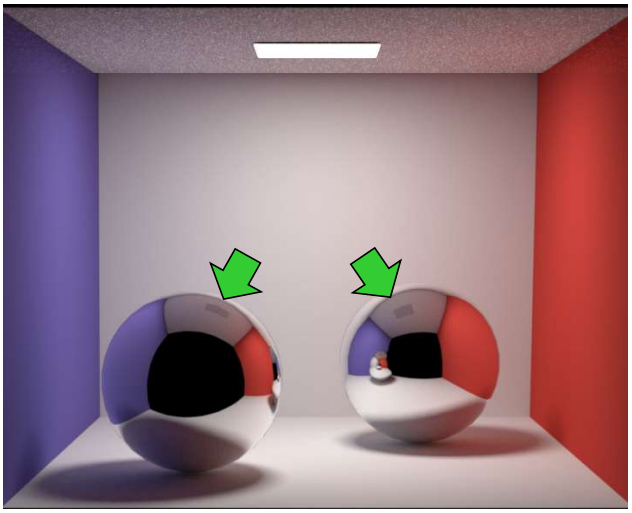
If this sounds a little confusing now, it is supposed to be a challenge if you want to go down this rabbit hole and try your hand it yourself. Using next event estimation, the way we described it before is totally fine. But if you want to go the extra mile and test your abilities on something challenging, feel free to try and figure out how to combine next event estimation with the balance heuristic. If you do, you should disable the number of guaranteed Russian roulette bounces in your path tracer, otherwise it will be difficult to see if something went wrong. Also don’t be disappointed if you can’t manage, it is a bit special. In case you desperately want to know how it could be done, you can always contact us and we will provide our solution.

Path Tracing (NEE) with 0/1 Weights vs Balance Heuristic

Once implemented, scenes like the one before, with both small and large area lights should work well even with path tracing if you use next event estimation with the balance heuristic. But the cases where you see a clear different may be more difficult to find than going from BRDF sampling to the basic next event estimation implementation.

Next Event Estimation meets Mirror Materials

- Soon, we will add some more exciting BRDFs for materials!
- Fully specular mirror materials are easy to simulate, however, they need extra care with NEE
- Naïve reflections can miss light!



- Why? Join us in the upcoming Materials lecture to find out…

One last thing that we glossed over: our current solution for next event estimation works with diffuse materials, but not so much with mirrors for instance. There, we would see results similar to the ones before: reflections of light sources can entirely miss the light!

Why would that happen? Well hopefully you will be curious enough when this discuss this, and much more, in the upcoming lecture on materials.

References

- [1] Conquering noisy images in ray tracing with next event estimation: https://developer.nvidia.com/blog/conquering-noisy-images-in-ray-tracing-with-next-event-estimation/
- [2] Dittebrandt, A., Hanika, J., & Dachsbacher, C. (2020). Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing. In Eurographics Symposium on Rendering - DL-only Track. The Eurographics Association.
- [3] Guo, J., Eisemann, M., & Eisemann, E. (2020). Next Event Estimation++: Visibility Mapping for Efficient Light Transport Simulation. Computer Graphics Forum, 39(7), 205-217.
- [4] If you didn't like our explanations of next event estimation, perhaps Jacco Bikker can do a better job: http://www.cs.uu.nl/docs/vakken/magr/2015-2016/slides/lecture%2008%20-%20variance%20reduction.pdf

And with that outlook, we hope you found today's topic interesting and perhaps are even eager to put it into practice in your path tracer and immediately crank up the quality of your renderings. As always, we leave you with a few resources in case you want to learn more and check out some more material on this topic. In any case, thanks for stopping by, and we hope to also see you in the next lecture.