Computer graphics III – Path tracing

Jaroslav Křivánek, MFF UK

Jaroslav.Krivanek@mff.cuni.cz

Tracing paths from the camera

```
renderImage()
{
  for all pixels
  {
    Color pixelColor = (0,0,0);
    for k = 1 to N
    {
      ωk := random direction through the pixel
      pixelColor += getLi(camPos, ωk)
    }
    pixelColor /= N;
    writePixel(pixelColor);
  }
}
```

Path tracing, v. 0.1 (recursive form)

```
getLi (x, ω):
    y = nearestIntersect (x, ω)
    return
        getLe(y, –ω) +                      // emitted radiance
        getLr (y, –ω)                       // reflected radiance

getLr(x, ω_inc):
    [ω_gen, pdf_gen] = genRndDirBrdfIs(ω_inc, normal(x) )
    return
        1/pdf_gen * getLi(x, ω_gen) * brdf(x, ω_inc, ω_gen)  * dot(normal(x), ω_gen)
```

Path Tracing – Loop version

```
getLi(x, wo)
{
    Spectrum throughput = (1,1,1)
    Spectrum accum  = (0,0,0)
    while(1)
    {
        hit = nearestIntersect(x, wo)
        if no intersection
                return accum + throughput * bgRadiance(x, wo)
        if isOnLightSource(hit)
                accum += thrput * getLe(hit.pos, -wo)
        [wi, pdf(wi)] := SampleDir(hit)
        Spectrum tputUpdate = 1/pdf(wi) * fr(hit.pos, wi, -wo) * dot(hit.n, wi)
        survivalProb = min(1, tputUpdate.maxComponent)
        if rand() < survivalProb // russian roulette - survive (reflect)
                thrput *= tputUpdate / survivalProb
                x  := hit.pos
                wo := wi
        else // terminate path
            break;
    }
    return accum;
}
```

Path termination – Russian roulette

```
getLi(x, wo)
{
    Spectrum throughput = (1,1,1)
    Spectrum accum   = (0,0,0)
    while(1)
    {
        hit = nearestIntersect(x, wo)
        if no intersection
            return accum + throughput * bgRadiance(x, wo)
        if isOnLightSource(hit)
            accum += thrput * getLe(hit.pos, -wo)
        [wi, pdf(wi)] := SampleDir(hit)
        Spectrum tputUpdate = 1/pdf(wi) * fr(hit.pos, wi, -wo) * dot(hit.n, wi)
        survivalProb = min(1, tputUpdate.maxComponent)
        if rand() < survivalProb // russian roulette - survive (reflect)
            thrput *= tputUpdate / survivalProb
            x  := hit.pos
            wo := wi
        else // terminate path
            break;
    }
    return accum;
}
```

Terminating paths – Russian roulette

- Continue the path with probability q
- Multiply weight (throughput) of surviving paths by 1 / q

$$Z = \begin{cases} Y / q & \text{if } \xi < q \\ 0 & \text{otherwise} \end{cases}$$

- RR is unbiased!

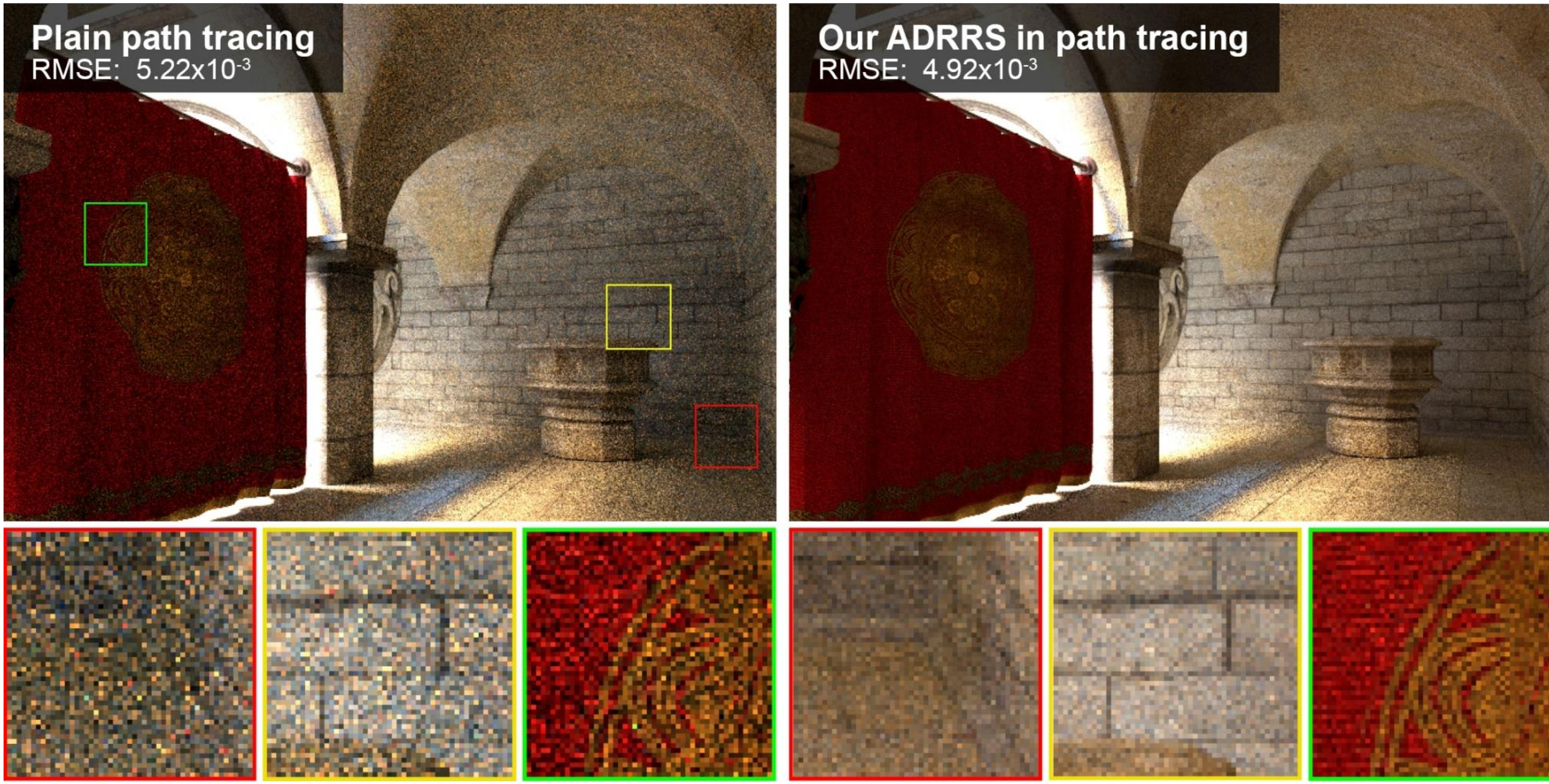
$$E[Z] = \frac{E[Y]}{q} \cdot q + 0 \cdot \frac{1}{q-1} = E[Y]$$

Survival probability – How to set it?

- It makes sense to use the surface reflectivity ρ as the survival probability
  - If the surface reflects only 30% of energy,
    we continue with the probability of 30%. That's in line with what happens in reality.
- What if we cannot calculate ρ? Then there's a convenient alternative, which in fact works even better:
  1. First sample a random direction ω_i according to $p(\omega_i)$
  2. Use the sampled ω_i it to calculate the survival probability as

$$q_{\text{survival}} = \min\left\{ 1,\ \frac{f_r(\omega_\text{i} \to \omega_\text{o})\cos\theta_\text{i}}{p(\omega_\text{i})} \right\}$$

Adjoint-drive RR and splitting



Vorba and Křivánek. **Adjoint-Driven Russian Roulette and Splitting in Light Transport Simulation**. *ACM SIGGRAPH 2016*

Direction sampling

```
getLi(x, wo)
{
    Spectrum throughput = (1,1,1)
    Spectrum accum  = (0,0,0)
    while(1)
    {
        hit = nearestIntersect(x, wo)
        if no intersection
                return accum + throughput * bgRadiance(x, wo)
        if isOnLightSource(hit)
                accum += thrput * getLe(hit.pos, -wo)
        [wi, pdf(wi)] := SampleDir(hit)
        Spectrum tputUpdate = 1/pdf(wi) * fr(hit.pos, wi, -wo) * dot(hit.n, wi)
        survivalProb = min(1, tputUpdate.maxComponent)
        if rand() < survivalProb // russian roulette – survive (reflect)
                thrput *= tputUpdate / survivalProb
                x  := hit.pos
                wo := wi
        else // terminate path
            break;
    }
    return accum;
}
```

Direction sampling

- We usually sample the direction ω_i from a pdf similar to

$$f_r(\omega_i, \omega_o) \cos \theta_i$$

  - Ideally, we would want to sample proportionally to the integrand itself

$$L_i(\omega_i) f_r(\omega_i, \omega_o) \cos \theta_i,$$

    but this is difficult, because we do not know L_i upfront. With some precomputation, it is possible to use a rough estimate of L_i for sampling [Jensen 95, Vorba et al. 2014], cf. "guiding".

BRDF importance sampling

- Let's see what happens when the pdf is **exactly proportional** to $f_r(\omega_i, \omega_o) \cos \theta_i$?

$$p(\omega_\mathrm{i}) \propto f_r(\omega_\mathrm{i} \to \omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i}$$

- Normalization (recall that a pdf must integrate to 1)

$$p(\omega_\mathrm{i}) = \frac{f_r(\omega_\mathrm{i} \to \omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i}}{\boxed{\int_{H(\mathbf{x})} f_r(\omega_\mathrm{i} \to \omega_\mathrm{o}) \cdot \cos\theta_\mathrm{i} \,\mathrm{d}\omega_\mathrm{i}}}$$

The normalization factor is nothing but the reflectance ρ

BRDF IS in a path tracer

- Throughput update for a general pdf

```
...
thrput *= fr(.) * dot(.) / ( ρ * p(wi) )
```

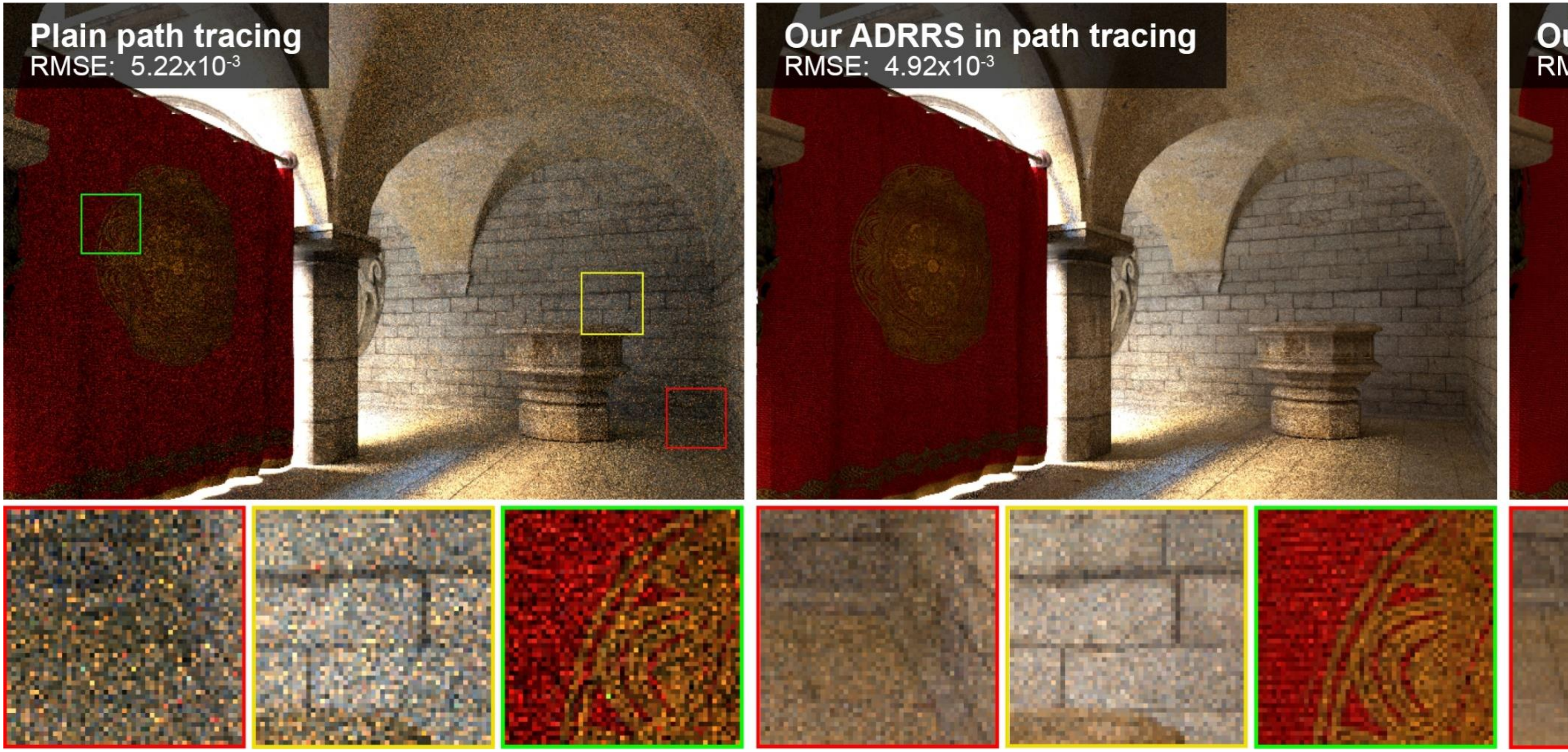
- A pdf that is exactly proportional to BRDF * cos keeps the throughput constant because the different terms cancel out!

$$p(\omega_{\mathrm{i}}) = f_r(\omega_{\mathrm{i}} \to \omega_{\mathrm{o}}) \cdot \cos\theta_i \;/\; \rho$$

```
...
thrput *= 1
```

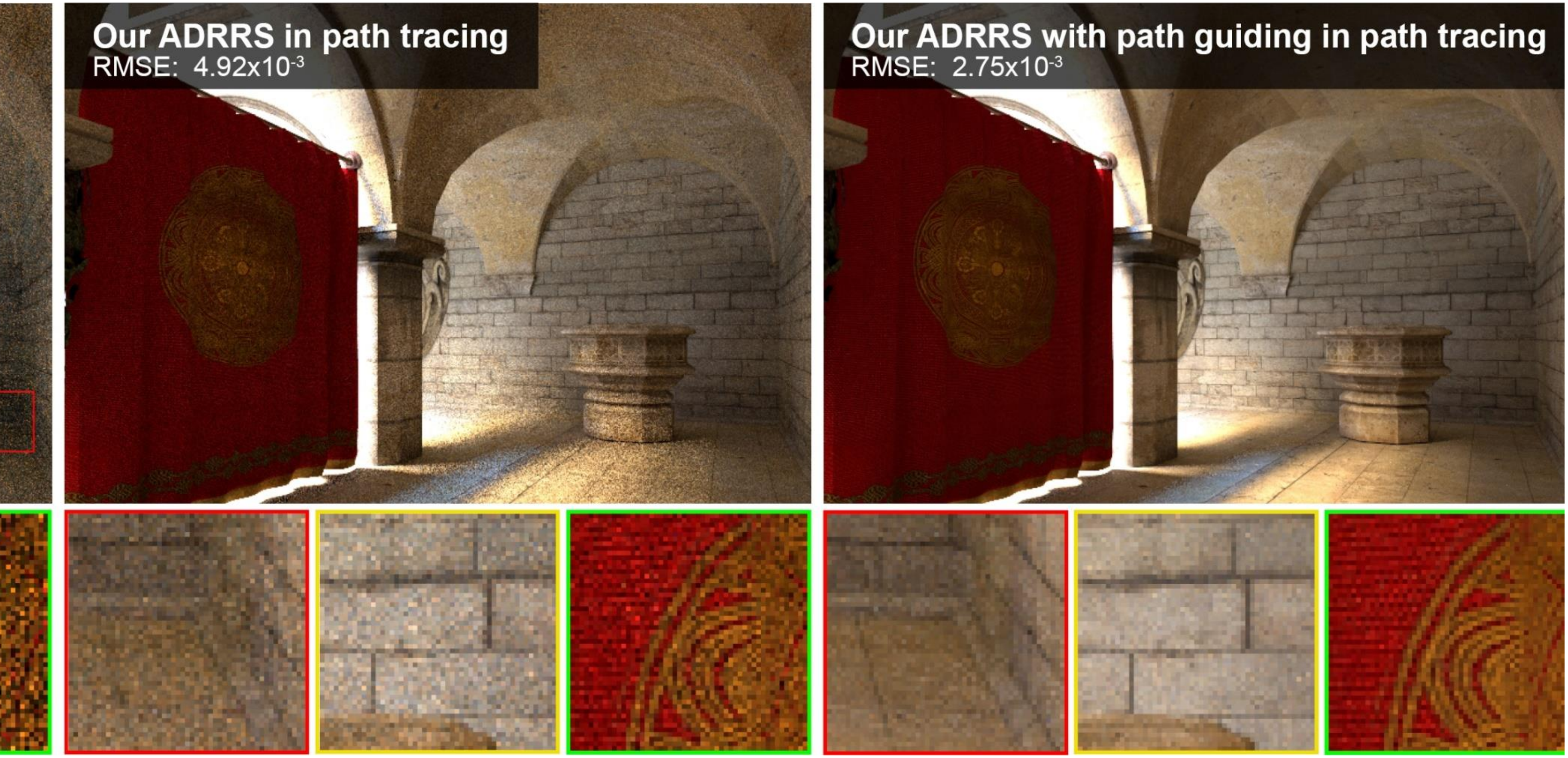
  - Physicists and nuclear engineers call this the "**analog**" simulation, because this is how real particles behave.

Path guiding



Vorba, Karlík, Šik, Ritschel, and Křivánek. **On-line Learning of Parametric Mixture Models for Light Transport Simulation**. *ACM SIGGRAPH 2014*

Path guiding



Vorba, Karlík, Šik, Ritschel, and Křivánek. **On-line Learning of Parametric Mixture Models for Light Transport Simulation**. *ACM SIGGRAPH 2014*

Direct illumination calculation in a path tracer

Direct illumination: Two strategies

- At each path vertex **x**, we are calculating **direct illumination**
  - i.e. radiance reflected from a point **x** on a surface exclusively due to the light coming directly from the sources

- Two sampling strategies
  1. **BRDF-proportional sampling**
  2. **Light source area sampling**

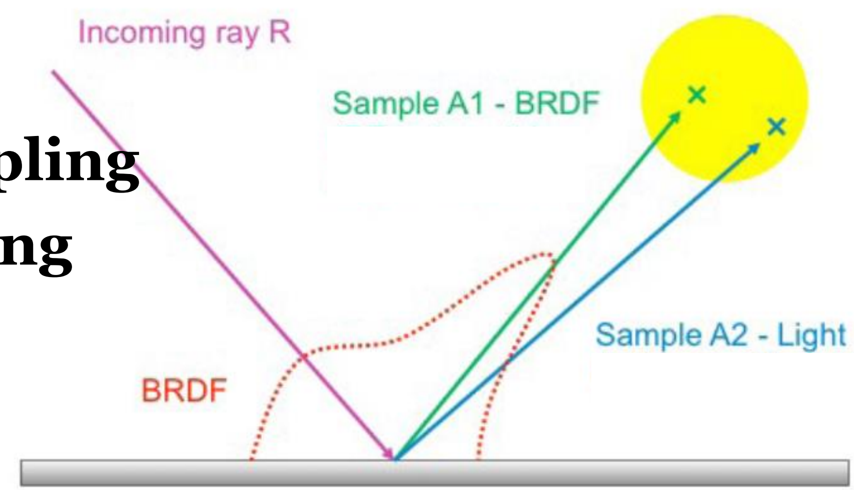


Image: Alexander Wilkie

The use of MIS in a path tracer

- For each path vertex:
  - Generate an explicit shadow ray for the techniques p_b (light source area sampling – a.k.a. "next event estimation")
  - Secondary ray for technique p_a (BRDF sampling)
    - One ray can be shared for the calculation of both **direct** and **indirect** illumination
    - But the MIS weight is – of course – applied only on the direct term (indirect illumination is added unweighted because there is no second technique to calculate it)

Dealing with multiple light sources

- Option 1:
  1. Loop over all sources and send a shadow ray to each one
- Option 2:
  1. Choose one source at random (ideally with prob proportional to light contribution)
  2. Sample illumination only on the chosen light, divide the result by the prob of picking that light
  - (Scales better with many sources but has higher variance per path)
- Beware: The probability of choosing a light influences the sampling pds and therefore also the MIS weights.

Learning the lights' contributions

Vévoda, Kondapaneni, Křivánek. **Bayesian online regression for adaptive direct illumination sampling**. *ACM SIGGRAPH 2018*

Learning the lights' contributions

Vévoda, Kondapaneni, Křivánek. **Bayesian online regression for adaptive direct illumination sampling**. *ACM SIGGRAPH 2018*