Computer graphics III

Path tracing

Jaroslav Křivánek, MFF UK

Jaroslav.Krivanek@mff.cuni.cz

Tracing paths from the (pinhole) camera

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

Tracing paths from the (pinhole) camera

- For progressive rendering, swap the loop nesting:

```
renderImage()
{
  for k = 1 to N  // rendering "passes"
  {
    for all pixels
    {
     Spectrum pixelColor = (0,0,0);
     ...
```

Path tracing, v. 0.1

estimateLin (x, ω): // radiance incident at x from direction ω

y = findNearestIntersection(x, ω)

if (no intersection)

return backgroud.getLe (y, $-\omega$) // emitted radiance from envmap

else

return getLe (y, $-\omega$) + // emitted radiance (if y is on a light)

estimateLrefl (y, $-\omega$)// reflected radiance

estimateLrefl(x, ω_{out}):

[ω_{in} , pdf] = genRandomDir(x, ω_{out}); // e.g. BRDF imp. sampling

return estimateLin(x, ω_{in}) * brdf(x, ω_{in}, ω_{out}) * dot($\mathbf{n}_x$, ω_{in}) / pdf

Path Tracing – Loop version

- Path tracing only has tail recursion
  - Can be unrolled into a loop for better efficiency
- New feature: “Russian Roulette” for unbiased path termination

```
estimateLin(x, omegaInAtX)  // radiance incident at "x" from the direction "omegaInAtX"
{                           //   ("omegaInAtX" is pointing *away* from "x")
  Spectrum throughput = (1,1,1)
  Spectrum accum  = (0,0,0)
  while(1)
  {
    hit = findNearestIntersection(x, omegaInAtX)

    if noIntersection(hit)              // ray leaves the scene - it "hits" the background
      return accum + throughput * bkgLight.getLe(x, - omegaInAtX)

    omegaOut := -omegaInAtX             // omegaOut at hit.pos
    if isOnLightSource(hit)             // ray happened to directly hit a light source
      accum += throughput * getLe(hit.pos, omegaOut)          // "pick up" emission
    // now estimate the reflected radiance
    [omegaIn, pdfIn] := generateRandomDir(hit)                // omegaIn at hit.pos
    throughput *= 1/pdfIn * brdf(hit.pos, omegaIn, omegaOut) * dot(hit.n, omegaIn)




      x := hit.pos                      // "recursion"
      omegaInAtX := omegaIn             // "recursion"


    }
    return accum;
}
```

```
estimateLin(x, omegaInAtX)  // radiance incident at "x" from the direction "omegaInAtX"
{                           //   ("omegaInAtX" is pointing *away* from "x")
  Spectrum throughput = (1,1,1)
  Spectrum accum  = (0,0,0)
  while(1)
  {
    hit = findNearestIntersection(x, omegaInAtX)

    if noIntersection(hit)           // ray leaves the scene - it "hits" the background
      return accum + throughput * bkgLight.getLe(x, - omegaInAtX)

    omegaOut := -omegaInAtX          // omegaOut at hit.pos
    if isOnLightSource(hit)          // ray happened to directly hit a light source
      accum += throughput * getLe(hit.pos, omegaOut)          // "pick up" emission
    // now estimate the reflected radiance
    [omegaIn, pdfIn] := generateRandomDir(hit)                 // omegaIn at hit.pos
    throughput *= 1/pdfIn * brdf(hit.pos, omegaIn, omegaOut) * dot(hit.n, omegaIn)

    survivalProb = min(1, throughput.maxComponent())
    if rand() < survivalProb          // Russian Roulette - survive (reflect)
      throughput /= survivalProb
      x := hit.pos                    // "recursion"
      omegaInAtX := omegaIn           // "recursion"
    else                              // terminate the path - break the while loop
      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

- It makes sense to use the surface reflectance ρ as the survival probability
  - If the surface reflects only 30% of light energy, we continue with the probability of 30%. That's how light behaves in physical reality.

Survival probability

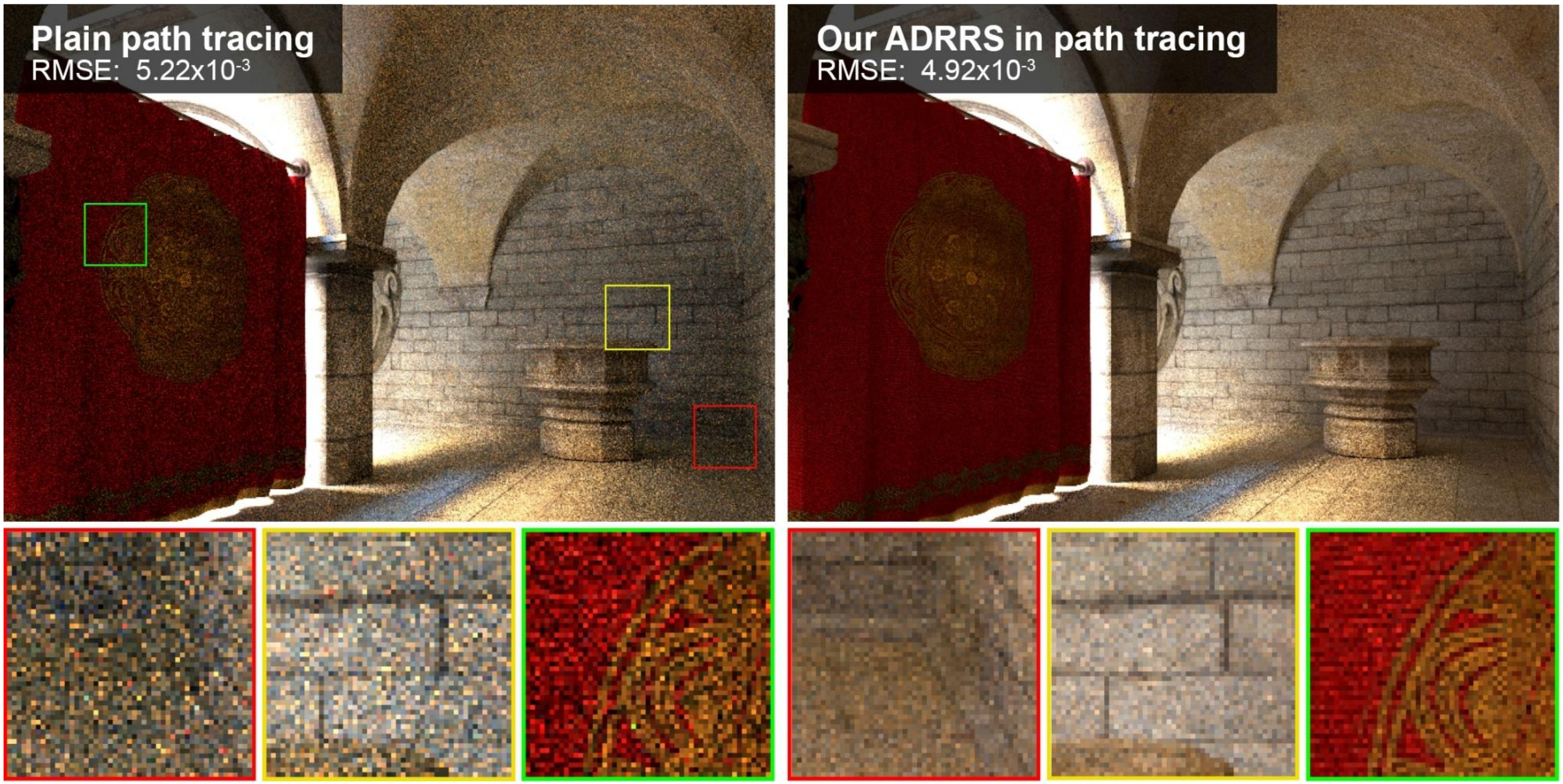
- What if we cannot calculate ρ? Then there's a convenient alternative, which in fact works even better:
  1. First sample a random direction ω_{in} according to $pdf(\omega_{in})$
  2. Update the path throughput
  3. Use the updated throughput as the survival probability

- If direction sampling $pdf(\omega_{in})$ is exactly proportional to BRDF*cos, the above strategy turns out to be exactly equivalent to setting survival probability to the surface reflectance (*prove this*).

Survival probability

- Our work: Adjoint-driven Russian Roulette & Splititng [Vorba & Křivánek 2016]
  - Weight the survival probability by the expected path contribution
    - If we enter a bright region, continue path even if throughput might be low
    - If we enter a dark region, kill the path even if throughput might be high
  - If the “survival probability” ends up > 1, split the path

Adjoint-drive RR and splitting



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

```
estimateLin(x, omegaInAtX)  // radiance incident at "x" from the direction "omegaInAtX"
{                           //  ("omegaInAtX" is pointing *away* from "x")
  Spectrum throughput = (1,1,1)
  Spectrum accum  = (0,0,0)
  while(1)
  {
    hit = findNearestIntersection(x, omegaInAtX)

    if noIntersection(hit)            // ray leaves the scene - it "hits" the background
      return accum + throughput * bkgLight.getLe(x, - omegaInAtX)

    omegaOut := -omegaInAtX           // omegaOut at hit.pos
    if isOnLightSource(hit)           // ray happened to directly hit a light source
      accum += throughput * getLe(hit.pos, omegaOut)          // "pick up" emission
    // now estimate the reflected radiance
    [omegaIn, pdfIn] := generateRandomDir(hit)                // omegaIn at hit.pos
    throughput *= 1/pdfIn * brdf(hit.pos, omegaIn, omegaOut) * dot(hit.n, omegaIn)

    survivalProb = min(1, throughput.maxComponent())
    if rand() < survivalProb          // Russian Roulette - survive (reflect)
      throughput /= survivalProb
      x := hit.pos                    // "recursion"
      omegaInAtX := omegaIn           // "recursion"
    else                              // terminate the path - break the while loop
      break;
  }
  return accum;
}
```

Direction sampling – genRandomDir()

- We usually sample the direction ω_{in} from a pdf similar to

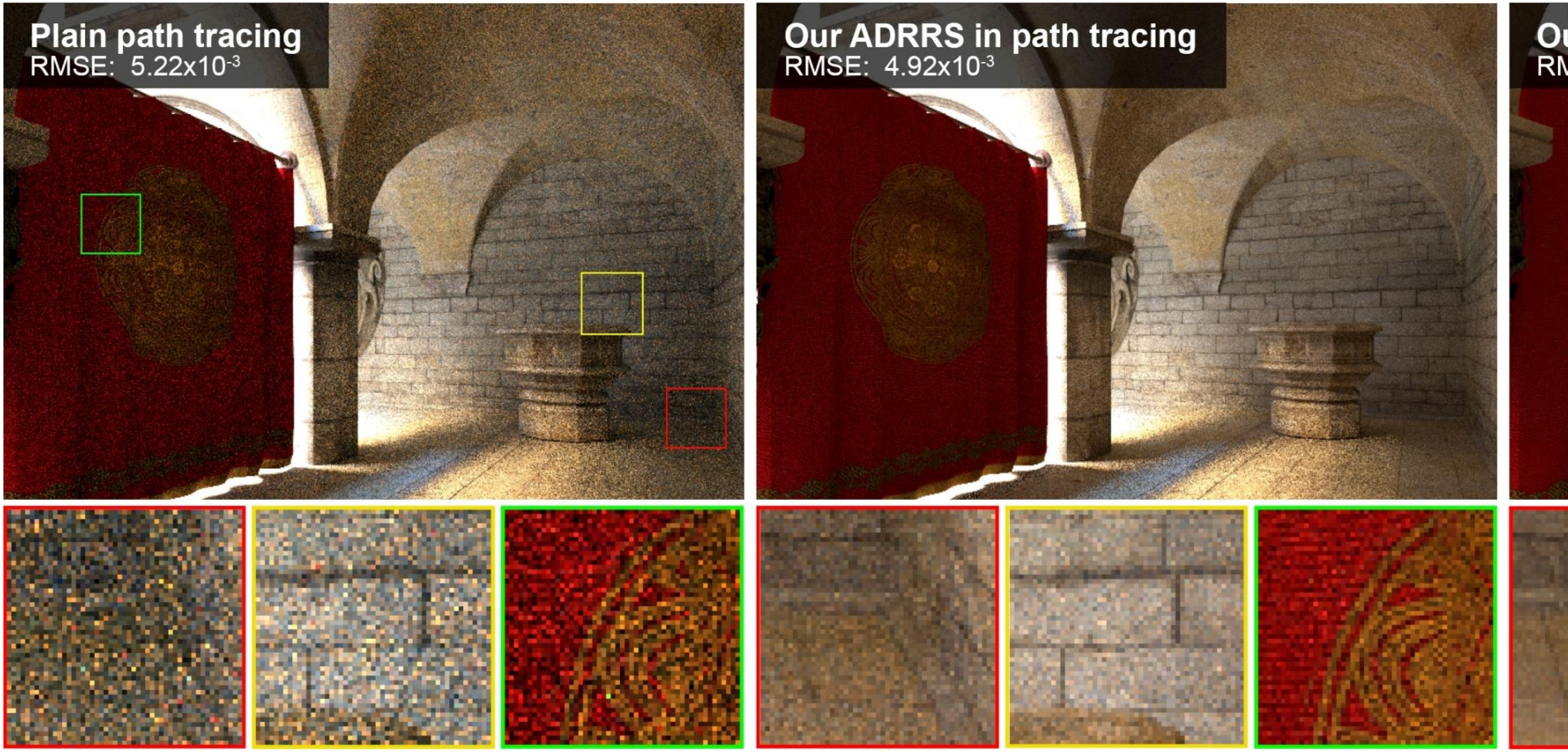
$$f_r(x, \omega_{in} \rightarrow \omega_{out}) \cos \theta_{in}$$

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

$$L_{in}(x, \omega_{in})\, f_r(x, \omega_{in} \rightarrow \omega_{out}) \cos \theta_i,$$

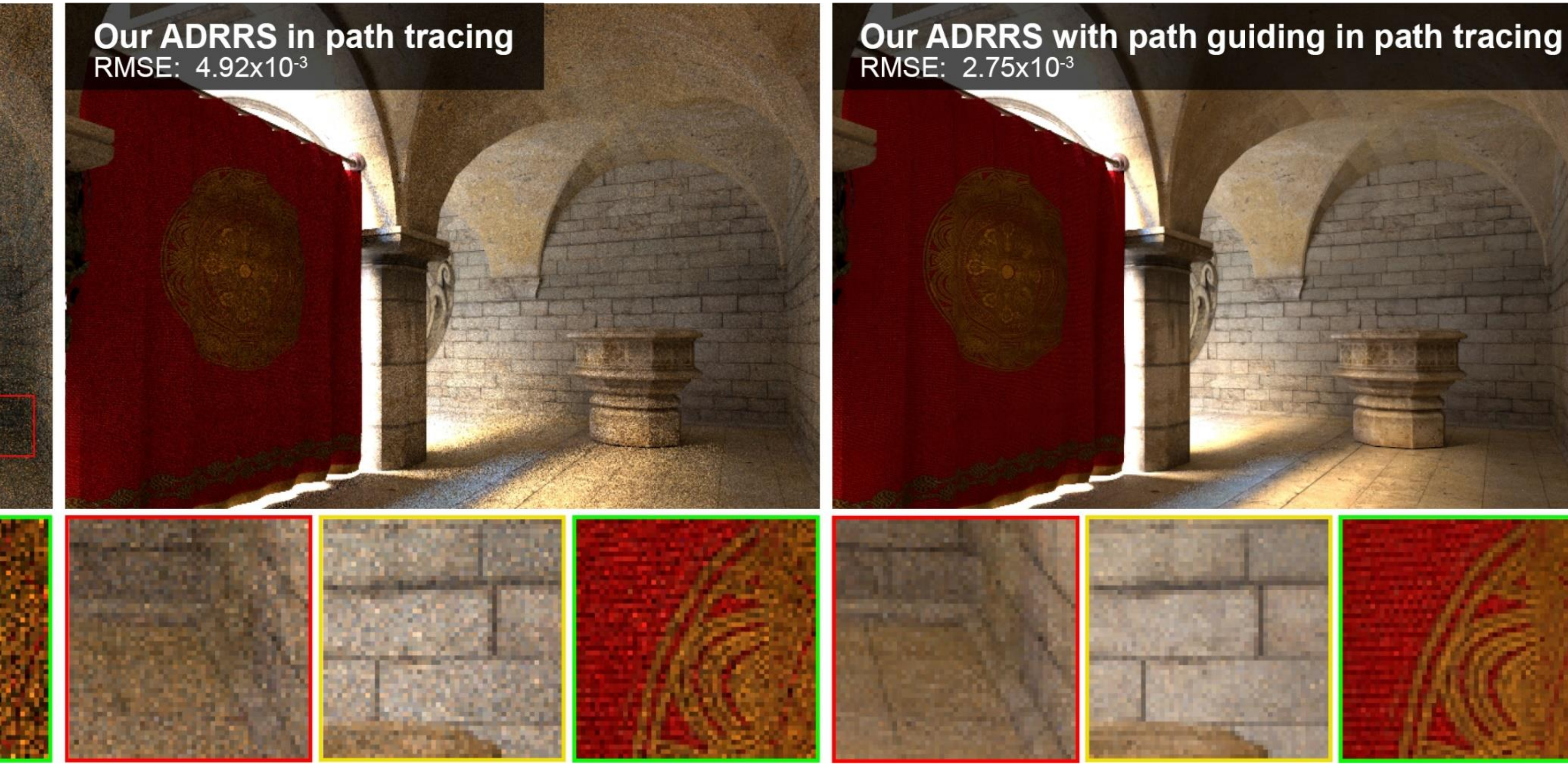
  but this is difficult, because we do not know L_{in} upfront. With some precomputation, it is possible to use a rough estimate of L_{in} for sampling [Jensen 95, Vorba et al. 2014]. This is called "path guiding".

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, we are calculating **direct illumination**
  - i.e. radiance reflected from the surface point exclusively due to the light coming *directly* from the light sources

- Two sampling strategies
  1. **Explicit light source sampling** ("next event estimation")
  2. **BRDF-proportional sampling** (already in the above code)

The use of MIS in a path tracer

- At each path vertex do both:
  - **Explicit light source sampling**
    - Generate point on light source & cast shadow ray
  - **BRDF-proportional sampling**
    - One ray can be shared for the calculation of both **direct** and **indirect** illumination
    - But the MIS weight is applied only on the direct term (indirect illumination is added unweighted because there is no alternative 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*