CS5630 Physically Based Realistic Rendering

Steve Marschner

Spring 2026

11 Path Tracing 2

Kajiya-style path tracing, version 1.1:

```
reflectedRadianceEst(x, ωr):
   ωl, pll = luminaireSample(x, n(x))
   pbl = brdfPDF(ωl)
   ωb, pbb = brdfSample(x, n(x), ωr)
   plb = luminairePDF(ωb)
   yl = traceRay(x, ωl)
   yb = traceRay(x, ωb)
   fl = brdf(x, ωl, ωr)
      * emittedRadiance(yl, –ωl)
   fb = brdf(x, ωb, ωr)
      * emittedRadiance(yb, –ωb)
   reflRad = fl / (pll + pbl) + fb / (plb + pbb)
   if random() < survivalProbability:
      reflRad += brdf(x, ωb, ωr) / pbb
         * reflectedRadianceEst(yb, –ωb)
         / survivalProbability
   return reflRad
```

Topics

Sampling collections of lights

Iterative path tracing

Managing delta components

Cameras and camera modeling

Path space integral formulation

Path tracing as a path space sampling method

Sampling collections of lights

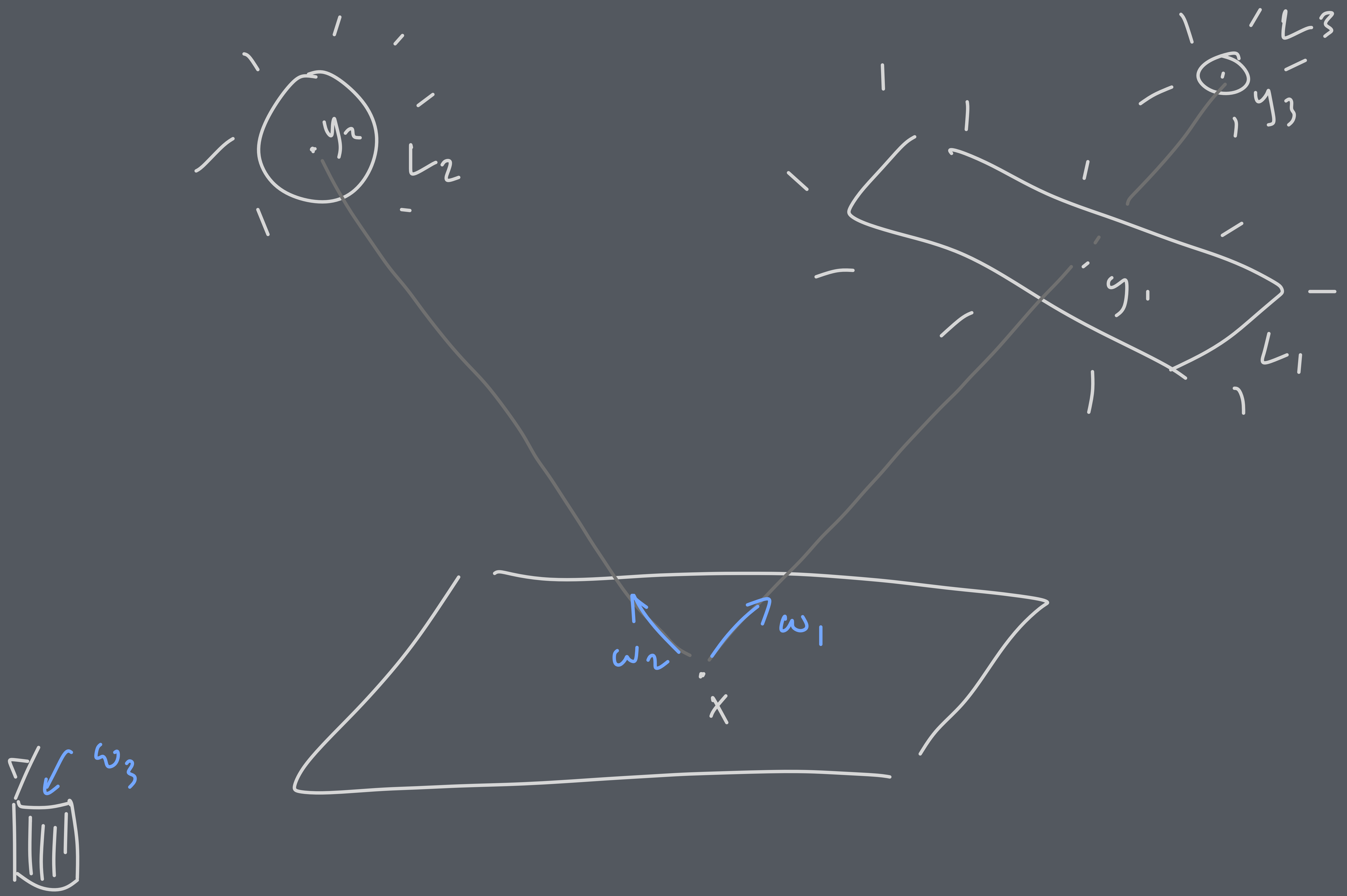
This code samples lights by solid angle

- this is simple in the integrator code but asks a lot of the light sources
- think of the case where one light is behind another…
- also wastes time re-finding the light source point to get radiance

Letting the sampling process return the radiance provides key flexibility

- first randomly choose a light source, note probability q
- next select a point on the source, note pdf p
- check visibility
  - visible: return source radiance and solid angle pdf pq/G where $G = \cos\theta/r^2$
  - occluded: return zero for the radiance

Sampling from luminaires with discarding



$$g(\omega_1) = \frac{L_1}{p_\omega(\omega_1)}$$

$$p_\omega(\omega_1) = p_1 p_A(y_1)\, r_1^2 / \cos\theta_1$$

$$g(\omega_2) = \frac{L_2}{p_\omega(\omega_2)}$$

$$p_\omega(\omega_2) = p_2 p_A(y_2)\, r_2^2 / \cos\theta_2$$

$$g(\omega_3) = 0$$

$$p(\omega_3) = ?$$

Discarding samples

Important practical solution to a frequent annoying problem

- have procedure to generate samples from a known pdf
- …but there are some cases where the procedure generates something that is not useful
- …and there’s no easy way to get the probability with which this happens

What can we do about this?

- ignore the bad sample?
  - no, because we then don’t know how to renormalize the pdfs of good samples
- try again until we do get a sample?
  - no, now we don’t know the probability of the new sample (for the same reason)
- just average the sample in as a zero?
  - yes, this works; crucially, we don’t need a probability for this failed sample

Failed samples in software

Your interface might require returning x, $f(x)$, and $p(x)$

- what about a failed sample though?

Option 1: lie

- if $f(x)$ is zero then the values of x and $p(x)$ probably don't matter
- so just return some bogus values
- example for light sampling: $p(\omega)$ will be false but it will be OK

Option 2: invent some kind of null

- allow sampling function to return "failed" and not report values
- makes the programmer feel better, maybe avoids some subtle bugs

In either case, skip moot computations for failed samples

Kajiya-style path tracing, version 1.1:

```
reflectedRadianceEst(x, ωr):
   ωl, pll = luminaireSample(x, n(x))
   pbl = brdfPDF(ωl)
   ωb, pbb = brdfSample(x, n(x), ωr)
   plb = luminairePDF(ωb)
   yl = traceRay(x, ωl)
   yb = traceRay(x, ωb)
   fl = brdf(x, ωl, ωr)
      * emittedRadiance(yl, –ωl)
   fb = brdf(x, ωb, ωr)
      * emittedRadiance(yb, –ωb)
   reflRad = fl / (pll + pbl) + fb / (plb + pbb)
   if random() < survivalProbability:
      reflRad += brdf(x, ωb, ωr) / pbb
         * reflectedRadianceEst(yb, –ωb)
         / survivalProbability
   return reflRad
```

Kajiya-style path tracing, version 1.2 (reordered on right to reveal the 3 contributions):

```
reflectedRadianceEst(x, ωr):
   Ll, ωl, pll = luminaireSample(x, n(x))
   pbl = brdfPDF(ωl)
   fr, ωb, pbb = brdfSample(x, n(x), ωr)
   plb = luminairePDF(ωb)
   yb = traceRay(x, ωb)
   fl = brdf(x, ωl, ωr) * Ll
   fb = fr * emittedRadiance(yb, –ωb)
   reflRad = fl / (pll + pbl) + fb / (plb + pbb)
   if random() < survivalProbability:
      reflRad += brdf(x, ωb, ωr) / pbb
         * reflectedRadianceEst(yb, –ωb)
         / survivalProbability
   return reflRad
```

```
reflectedRadianceEst(x, ωr):
   reflRad = 0

   if Ll, ωl, pll = luminaireSample(x, n(x)):
      pbl = brdfPDF(ωl)
      fl = brdf(x, ωl, ωr) * Ll
      reflRad += fl / (pll + pbl)

   if fr, ωb, pbb = brdfSample(x, n(x), ωr):
      plb = luminairePDF(ωb)
      yb = traceRay(x, ωb)
      fb = fr * emittedRadiance(yb, –ωb)
      reflRad += fb / (plb + pbb)

      if random() < survivalProbability:
         reflRad += brdf(x, ωb, ωr) / pbb
            * reflectedRadianceEst(yb, –ωb)
            / survivalProbability

   return reflRad
```

Iterative vs. recursive implementation

Path tracing is often presented as a recursion

- in some cases the function calls can be slow
- there is no way to guarantee a bound for the stack space consumed
- it’s almost a tail recursion so the recursive implementation is hardly necessary

Path tracing is usually implemented as an iteration instead

- loop over number of bounces
- keep track of path throughput
- accumulate radiance

Kajiya-style path tracing, version 1.2i:

```
rayRadianceEst(x, ω):
   y = traceRay(x, ω)
   return emittedRadiance(y, –ω)
      + reflectedRadianceEst(y, –ω)

reflectedRadianceEst(x, ωr):
   reflRad = 0

   if Ll, ωl, pll = luminaireSample(x, n(x)):
      pbl = brdfPDF(ωl)
      fl = brdf(x, ωl, ωr) * Ll
      reflRad += fl / (pll + pbl)

   if fr, ωb, pbb = brdfSample(x, n(x), ωr):
      plb = luminairePDF(ωb)
      yb = traceRay(x, ωb)
      fb = fr * emittedRadiance(yb, –ωb)
      reflRad += fb / (plb + pbb)

      if random() < survivalProbability:
         reflRad += brdf(x, ωb, ωr) / pbb
            * reflectedRadianceEst(yb, –ωb)
            / survivalProbability

   return reflRad
```

```
rayRadianceEst(x, ω):
   x, ωr = traceRay(x, ω), –ω
   rad = emittedRadiance(x, ωr)
   throughput = 1.0

   while true:

      if Ll, ωl, pll = luminaireSample(x, n(x)):
         pbl = brdfPDF(ωl)
         fl = brdf(x, ωl, ωr) * Ll
         rad += throughput * fl / (pll + pbl)

      if fr, ωb, pbb = brdfSample(x, n(x), ωr):
         plb = luminairePDF(ωb)
         yb = traceRay(x, ωb)
         fb = fr * emittedRadiance(yb, –ωb)
         rad += throughput * fb / (plb + pbb)

         if random() < rrProb: return rad

         throughput *= brdf(x, ωb, ωr) / pbb
            / (1 – rrProb)
         x, ωr = yb, –ωb

      else: return rad
```

Managing delta components

Delta functions

- one explanation: limit of narrow peaks
- preferred explanation: define via integral: $\int \delta(x) f(x) dx = f(0)$
- in 1D it's traditional to use $\delta(x - x_0)$ to move peak to x_0
- in more general domains better to define something like $\delta_{x_0}(x)$ where $\int \delta_{x_0}(x) f(x) dx = f(x_0)$

Expresses the notion of finite "stuff" piled at a single point

- incoming radiance due to a point light source
- BRDFs of ideal specular surfaces

Delta functions in Monte Carlo integration

One example: two lights, one point, one area

- incident radiance has a finite part and a delta part
- how do we sample this?
  - easy: half the time choose the point, half the time sample the area
- what is the pdf?
  - $p_A(y)/2$ for points on the area light and, ummm…
  - probability density is the wrong idea for the delta part … it's a finite probability
- what is the radiance?
  - we know it for the area source, but for the point source? It doesn't have radiance…
  - radiance is the wrong idea … it is intensity

Delta functions in Monte Carlo integration

One example: two lights, one point, one area

- incident radiance has a finite part and a delta part
- how do we sample this?
  - easy: half the time choose the point, half the time sample the area
- what is the pdf?
  - $p_A(y)/2$ for points on the area light and, ummm…
  - probability density is the wrong idea for the delta part … it's a finite probability
- what is the radiance?
  - we know it for the area source, but for the point source? It doesn't have radiance…
  - radiance is the wrong idea … it is intensity

Delta functions in Monte Carlo integration

Second example: surface with mix of specular and diffuse

- BRDF has a smooth part and a delta part
- how do we sample this?
  - easy: half the time choose the specular direction, half the time sample the smooth part
- what is the pdf?
  - $p(\omega)/2$ for samples from the smooth component and, ummm…
  - probability density is the wrong idea for the delta part … it's a finite probability
- what is the BRDF value?
  - we know it for the smooth component, but for the specular? It doesn't have a BRDF…
  - BRDF is the wrong idea … it is a unitless reflectance factor