Rendering and Photon Mapping

Plan of action

$$L_o(x,\omega) = L_e(x,\omega) + \int_S f_r(x, x' \to x, \omega) L_i(x, x' \to x) V(x,x') G(x,x') dA'$$

- Find a way to approximate the Li terms
- Find a way to approximate the integral
- This will produce Lo result.
- Do this only for points *x* and directions ω that contribute to the image you're making

Practical Constraints

- Our goal is often to produce the best image within a limited amount of time
- This means we can't perfectly simulate LT
- Variance Errors
  – Look like noise
- Bias (Mean) Errors
  – Physically wrong (e.g. too dark in certain places)

Noisy Estimators

- Say the *true* value is $L_{rT}(x, \omega)$
- Imagine some method that computes

$$L_{rN}(x,\omega) = L_{rT}(x,\omega) + \frac{1}{N}\sum_{i=1}^{N} rand()$$

- Limit as *N* goes to infinity is correct
- For any finite *N*, the result is noisy (has variance)

Biased Sampling Estimators

- Say the *true* value is $L_{rT}(x, \omega)$
- Imagine some method that computes

  $L_{rB}(x, \omega) = L_{rT}(x, \omega) - K(x, \omega)/M$

- Limit as *M* goes to infinity is correct
- If K is everywhere positive, then for any finite *M*, our solution is too small

Sources of Bias

- May result from assumptions about model
  – Radiosity assumes perfectly diffuse surfaces
- May result from biased sampling
  – Photon mapping emphasizes LS*DE and LDE paths

Joton

- Representation of a probabilistic photon group – a bunch of photons that we may want to sample.
- $J = (x, \omega_I, \Phi)$, where Φ is power arriving at the surface, and ω_I is the direction of incident light, x = pt on surface. Units of J = radiance.
- Photon map = record of lots of J-values.

Estimating light from a surface to the eye

- Look at J values near the relevant surface point
- Reflectance function (fR) on the surface
- Combine by summation (low budget integration) to estimate L_R

Russian Roulette

- Suppose 100 Jotons of power 1 hit a surface that reflects diffusely with reflectance k.
- Naïve sim: 100 Jotons with power k leave surface
- Clever hack: (100 k) photons with power 1 leave surface.

Why?

- Fewer jotons (‘cuz $k < 1$)
- “Weak” jotons disappear and we don’t waste computation on them
- Photon map will only store photons with power ~ 1, so all contribute equally to estimate of integral, so variance is reduced.

How does Photon Mapping work?

- Reflect jotons just like photons…but instead of a fraction of incoming power, reflect with a *probability* proportional to reflectance.
- If not reflected, it gets dropped from simulation.
- $P(\text{bounce } A) = L_R(x, \omega_O) / L_I(x, \omega_I)$
- (for diffuse surface, this is just diffuse reflectivity!)

Program Structure

- Psuedo-code for the Photon Mapping algorithm:

Forward Trace Caustic (Specular Interreflection) Paths into Caustic Photon Map (High-res)

Forward Trace Diffuse Interreflection Paths into Diffuse Photon Map (Low-res)

Balance Caustic and Diffuse Trees

Backward Trace Photons

- Illumination = Caustic + Diffuse + direct illumination

Caustic tracing

```
repeat numCaustics times
   J := random photon from random light
   absorbed = false
    do
        S = first intersection between J and scene
        r = random(0,1)
        if (r < P(diffuse)) // diff. reflection
                if "LS+" path then write J to caustics map
                absorbed := true
        else if (r < P(diffuse) + P(specular))
                J := mirror J about normal
                scale Jpower by specular color
```

```
        else if r < P(diffuse) + P(specular) + P(transmit)
                J : = refract J
                if total internal refraction then
                        absorbed = true
                scale Jpower by transmission color

        else
                absorbed := true
    while not absorbed
```

Initial joton power

- For each point from which jotons are emitted:

Starting power = totalEmitterPower/numCaustics

Dealing with color:
(brdf.emissive/brdf.emissive.sum()) * (totalEmitterPower / numCaustics)

TotalEmitterPower = Sum(tri.triangle.area() * emissive.sum())

Summed over all emitters.

Diffuse tracing

```
repeat numDiffuse times
   J : = random photon from random light
   absorbed := false
   while not absorbed
        S = first intersection between J and scene
        r := random(0, 1)
        if r < P(diffuse) // i.e., if it's diffusely reflected
                if (not "LS*D" path)
                        write J to diffuse photon map
                        scale Jpower by diffuse color
                        J := random hemisphere direction
                else if r < P(diffuse) + P(specular)
                        J := reflect J about normal
                        scale Jpower by specular color
                else if r < P(diffuse) + P(specular) + P(transmission)
                        J := refract J
                        if total internal refraction then absorbed = true
                        scale Jpower by transmission color
                else
                        absorbed := true
```

Backward Tracing

```
for each pixel(x, y)
        R := ray from eye through (x, y)
        S := get first intersection(R, Scene)
        image(x,y) :=
                direct illumination at S
                from all lights (with shadowing)
        + caustic radiance estimate
                + diffuse radiance estimate
```

Direct Illumination

Direct Illumination (x, N)

C := 0

for each light L with normal N_L, radiosity B

for count := 1 … numShadowRays

  x_L := random point on L

  $\omega_L := (x_L - x) / || xL - x ||$

  $r := || x_L - x ||$

  if visible(x, x_L)

    $C := C + \max(N \cdot \omega_L, 0) * k_d *$

      $\max(-N_L \cdot \omega_L, 0) * B(x_L) / (\pi * r^2)$

    $C := C * A_L$ / numShadowRays

return C

Radiance Estimate(x, N)

(used for both diffuse and caustic maps)

C := 0

For each photon J in photon map within radius *r* of *x*

    C := C + max($N \cdot -\omega_J$, 0) * k_d * L_J

return C / (πr^2)

Fin