## Rendering and Photon Mapping

#### Plan of action

 $L_{o}(x,\omega) = L_{e}(x,\omega) + \int f_{r}(x,x' \to x,\omega) L_{i}(x,x' \to x) V(x,x') G(x,x') dA'$ 

- · 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  $\Phi$  is power arriving at the surface, and  $\omega_I$  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<sub>R</sub>

#### **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)</li>
- "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(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

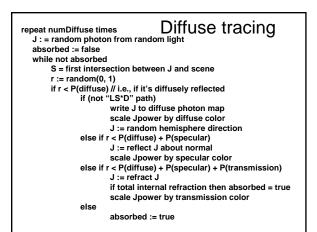
## Initial joton power

• 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.



## **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 (x, N)

C := 0

for each light L with normal N<sub>L</sub>, 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<sub>L</sub>)

C := C + max(N · \omega_L, 0) * k<sub>d</sub> *

max(-N<sub>L</sub> · \omega_L, 0) * B(x<sub>L</sub>) / (\pi * r<sup>2</sup>)

C := C * A<sub>L</sub> / 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<sub>d</sub> \*  $L_J$ return C / ( $\pi$   $r^2$ )

