Photon mapping
Course announcements

• Programming assignment 4 posted, due Friday 4/1 at 23:59.
  - How many of you have looked at/started/finished it?
  - Any questions?

• Take-home quiz 8 will be posted tonight.

• Suggest topics for this week’s reading group.

• No lecture on Thursday! Vote for when to reschedule.

• Talk by Nathan Matsuda in graphics lab meeting.

• Presentation by Yannis in imaging reading group.
Overview of today’s lecture

• Photon mapping.
Most of these slides were directly adapted from:

- Wojciech Jarosz (Dartmouth).
Today’s Menu

Difficult light paths

Photon Mapping
Specular-Diffuse-Specular Paths
Specular-Diffuse-Specular Paths

Reference  Bidirectional PT

Images courtesy of J. Křivánek
Specular-Diffuse-Specular Paths

*SDS* paths are difficult for unbiased techniques

*LSDSE* paths are difficult for unbiased techniques
Specular-Diffuse-Specular Paths

Path tracing

Path misses light source
Specular-Diffuse-Specular Paths

Path tracing with NEE

delta BSDF
(zero value)
Specular-Diffuse-Specular Paths

Bidirectional Path tracing

delta BSDF (zero value)
Specular-Diffuse-Specular Paths

Bidirectional Path tracing

points do not “meet”

What now?
Regularize delta functions (path points):
e.g., by employing kernel density estimation (blurring in space)
“Backward” Ray Tracing
(predecessor of photon mapping)
“Backward” Ray Tracing

James Arvo. In *Developments in Ray Tracing*, SIGGRAPH ‘86 Course Notes

Emit photons from light sources and store them in *illumination maps*

*Illumination map* = texture for accumulating irradiance

Note on the name of the technique: In retrospect, Arvo regretted using the term “backward” to refer to tracing light paths since many later publications use it in the opposite sense, i.e. tracing eye paths. To avoid confusion, he recommends terms such as *light tracing* and *eye tracing* as they are unambiguous.
“Backward” Ray Tracing

Preprocess:
- shoot light from light sources
- deposit photon energy in illumination maps

Irradiance: “number of photons hitting a small patch of a wall per second, divided by size of patch”
“Backward” Ray Tracing

For each shading point
- compute direct lighting
- lookup indirect lighting from illumination maps
“Backward” Ray Tracing

“Backward” Ray Tracing

✓ One of the first techniques to simulate caustics!

✗ Requires parametrizing surfaces or meshing
  - Difficult to handle complex or procedural geometry

✗ Hard to choose illumination map resolution
  - high resolution with few photons: high-frequency noise
  - low resolution with many photons: blurred illumination
Photon Mapping
Path Tracing

100 paths/pixel (5 minutes)
Photon Mapping

10 rays/pixel (5 seconds)
Path Tracing

1000 paths/pixel
Photon Mapping
Photon Mapping

A two-pass algorithm:

- Pass 1: Tracing photons from light sources, and caching them in a *photon map*

- Pass 2: Tracing from the eye and approximating indirect illumination using the photons

Similar to “backward” ray tracing, but different way of storing photons & computing density
Photon Mapping

A two-pass algorithm:

- Pass 1: Tracing photons from light sources, and caching them in a photon map

- Pass 2: Tracing from the eye and approximating indirect illumination using the photons

Similar to “backward” ray tracing, but different way of storing photons & computing density
Photon Tracing

1) Emit photons
2) Scatter photons
3) Store photons
Photon Tracing

1) Emit photons
2) Scatter photons
3) Store photons
Photon Tracing

1) Emit photons
2) Scatter photons
3) Store photons
Visualization of the Photon Map
Photon Emission

Photons carry power (flux) not radiance!

- not a physical photon
- just a fraction of the light source power
- in most practical implementations, each photon carries multiple wavelengths (e.g. RGB)
Photon Emission

Define initial:
- $x_p$: position
- $\omega_p$: direction
- $\Phi_p$: photon power

General recipe:
- Sample position on surface area of light with $p(x_p)$
- Sample direction with $p(\omega_p \mid x_p)$

$$\Phi_p = \frac{1}{M} \frac{L_e(x_p, \vec{\omega}_p) \cos \theta_p}{p(x_p) p(\vec{\omega}_p \mid x_p)}$$

# of emitted photons
Photon Emission

Interesting derivation:
- if PDFs are proportional to the emission:

\[
p(x_p) = \frac{\int_{H^2} L_e(x_p, \omega) \cos \theta \, d\omega}{\int_A \int_{H^2} L_e(x, \omega) \cos \theta \, d\omega \, dx}
\]

\[
p(\omega_p | x_p) = \frac{L_e(x_p, \omega_p) \cos \theta_p}{\int_{H^2} L_e(x_p, \omega) \cos \theta \, d\omega}
\]

- then:

\[
\Phi_p = \frac{1}{M} \frac{L_e(x_p, \omega_p) \cos \theta_p}{p(x_p) p(\omega_p | x_p)}
\]

\[
= \frac{1}{M} \frac{\int_{H^2} L_e(x_p, \omega) \cos \theta \, d\omega}{\int_A \int_{H^2} L_e(x, \omega) \cos \theta \, d\omega \, dx} = \frac{\Phi}{M}
\]

If you perfectly importance sample the emitted radiance, just take the total power and divide by # of emitted photons.
Photon Emission Examples

Isotropic point light:
- Generate uniform random direction over sphere

Spotlight:
- Generate uniform random direction within spherical cap

Diffuse area light:
- Generate uniform random position on surface
- Generate cosine-weighted direction over hemisphere
Pseudocode

```java
void generatePhotonMap()

repeat:
(l, Prob_l) = chooseRandomLight()
(x, ω, Φ) = emitPhotonFromLight(l)
tracePhoton(x, ω, Φ / Prob_l)
until we have enough photons;
divide all photon powers by number of emitted photons

void tracePhoton(x, ω, Φ)
```
void tracePhoton(x, ω, Φ)
(x', n) = nearestSurfaceHit(x, ω)
possiblyStorePhoton(x', ω, Φ)
(ω', pdf) = sampleBSDF(x', -ω)
Φ' = Φ * absDot(n, ω') * BSDF(x', -ω, ω') / pdf
tracePhoton(x', ω', Φ')

Pseudocode
Storing Photons

Store only on diffuse (or moderately glossy) surfaces
- Specular surfaces need to be handled using path tracing from the camera

Stored data: [36 bytes]

```c
struct Photon
{
    float position[3];
    float power[3];
    float direction[3];
};
```
Storing Photons

Store only on diffuse (or moderately glossy) surfaces

- Specular surfaces need to be handled using path tracing from the camera

Stored data:

```c
struct Photon
{
    float position[3];
    char power[4];       // Packed RGBE format
    char phi, theta;     // Packed direction
};
```
Scattering of Photons

Photons can be:

- absorbed or scattered (reflected or refracted)
- BSDF sampling chooses either reflection or refraction
- the power of the scattered photon is lowered to account for absorption

Problem:

- as photons bounce they carry less and less power
- ideally all stored photons would have the same power
- also, when should we terminate the recursion?

Solution: Russian roulette
void tracePhoton(x, ω, Φ)

(x', n) = nearestSurfaceHit(x, ω)

possiblyStorePhoton(x', ω, Φ)

(ω', pdf) = sampleBSDF(x', -ω)

Φ' = Φ * absDot(n, ω') * BSDF(x', -ω, ω') / pdf

tracePhoton(x', ω', Φ')
Pseudocode

```c
void tracePhoton(x, ω, Φ)
(x', n) = nearestSurfaceHit(x, ω)
possiblyStorePhoton(x', ω, Φ)
(ω', pdf) = sampleBSDF(x', -ω)
Φ' = Φ * absDot(n, ω') * BSDF(x', -ω, ω') / pdf
if survivedRussianRoulette(Φ, Φ')
tracePhoton(x', ω', Φ')
```
Photon Path Termination

Probabilistically terminate the photon walk using Russian roulette (continue with prob. $p$)

$$E[F'] = (1 - p) \cdot 0 + p \cdot \frac{E[F]}{p} = E[F]$$

Option 1: local termination probability:

$$p = \min \left(1, \frac{\Phi'}{\Phi} \right)$$
Photon Path Termination

```python
bool survivedRussianRoulette(Φ, Φ')

p = min(1, Φ'/Φ)

if rand() > p:
    // terminate
    return false

else:
    // continue with re-weighted power
    Φ'/= p

return true
```

if Φ'/Φ is smaller than 1, then Φ’ = Φ' / p = Φ
i.e., the scattered photon has the same power!
Photon Path Termination

Probabilistically terminate the photon walk using Russian roulette (continue with prob. $p$)

$$E[F'] = (1 - p) \cdot 0 + p \cdot \frac{E[F]}{p} = E[F]$$

Option 1: local termination probability:

$$p = \min \left( 1, \frac{\Phi'}{\Phi} \right)$$

Option 2: history-aware termination probability:

- try to keep each photon same power
void \texttt{tracePhoton}(x, \omega, \Phi)\\
(x', n) = \texttt{nearestSurfaceHit}(x, \omega)\\
\texttt{possiblyStorePhoton}(x', \omega, \Phi)\\
(\omega', pdf) = \texttt{sampleBSDF}(x', -\omega)\\
\Phi' = \Phi \times \text{absDot}(n, \omega') \times \text{BSDF}(x', -\omega, \omega') / pdf\\
\textbf{if} \ \text{survivedRussianRoulette}(\Phi, \Phi')\\
\text{tracePhoton}(x', \omega', \Phi', \Phi)
void tracePhoton(x, ω, Φ, Φorig)
(x', n) = nearestSurfaceHit(x, ω)
possiblyStorePhoton(x', ω, Φ)
(ω', pdf) = sampleBSDF(x', -ω)
Φ' = Φ * absDot(n, ω') * BSDF(x', -ω, ω') / pdf
if survivedRussianRoulette(Φorig, Φ')
tracePhoton(x', ω', Φ', Φorig)
Russian Roulette Example

300 photons with power 1.0 W hit a surface with reflectance 50%

Instead of reflecting 300 photons with power 0.5 W, RR will make ~150 photons continue with power 1.0 W

Very important!
Photon Mapping

A two-pass algorithm:

- **Pass 1**: Tracing of photons from light sources, and caching them in a photon map

- **Pass 2**: Tracing from the eye and approximating indirect illumination using the photons
Photon Mapping

A two-pass algorithm:

- Pass 1: Tracing of photons from light sources, and caching them in a photon map

- Pass 2: Tracing from the eye and approximating indirect illumination using the photons
Rendering

For each shading point:

- Find the k closest photons
- Approx. radiance using density of photons
Rendering

For each shading point:

- Find the k closest photons
- Approx. radiance using density of photons
The Radiance Estimate

Based on kernel density estimation
- Non-parametric way of estimating the probability density of a random variable (photon density)
The Radiance Estimate

Based on kernel density estimation

- Non-parametric way of estimating the probability density of a random variable (photon density)

\[
L_r(x, \omega) = \int_{H^2} f_r(x, \omega', \omega) L_i(x, \omega') \cos \theta' d\omega'
\]

\[
= \int_{H^2} f_r(x, \omega', \omega) \frac{d\Phi^2(x, \omega')}{\cos \theta' d\omega' dA} \cos \theta' d\omega'
\]

\[
= \int_{H^2} f_r(x, \omega', \omega) \frac{d\Phi^2(x, \omega')}{dA}
\]

\[
\approx \sum_{p=1}^{n} f_r(x, \omega_p, \omega) \frac{\Delta \Phi_p(x, \omega_p)}{\Delta A}
\]
The Radiance Estimate

Approach 1: first define area, then find photons

\[ L_r(x, \omega) \approx \sum_{p=1}^{k} f_r(x, \omega_p, \omega) \frac{\Phi_p}{A} \]
The Radiance Estimate

Approach 1: first define area, then find photons

\[ L_r(x, \vec{\omega}) \approx \sum_{p=1}^{k} f_r(x, \vec{\omega}_p, \vec{\omega}) \frac{\Phi_p}{\pi r^2} \]

# of photons within disk

Assuming a disk region of radius \( r \)
The Radiance Estimate

Approach 2: first find $k$ nearest photons, then define area

$$L_r(x, \bar{\omega}) \approx \sum_{p=1}^{k} f_r(x, \bar{\omega}_p, \bar{\omega}) \frac{\Phi_p}{A}$$
The Radiance Estimate

Approach 2: first find $k$ nearest photons, then define area

$$L_r(x, \tilde{\omega}) \approx \sum_{p=1}^{k-1} f_r(x, \tilde{\omega}_p, \tilde{\omega}) \frac{\Phi_p}{\pi r_k^2}$$

Ignore the $k$-th photon

Distance to the $k$-th photon

[García et al. 2012]
The Radiance Estimate

Using a non-constant kernel:

\[ L_r(x, \bar{\omega}) \approx \sum_{p=1}^{k-1} f_r(x, \bar{\omega}_p, \bar{\omega}) \Phi_p K_{2D}(r_p, r_k) \]
The Photon Map Data Structure

Requirements:
- Compact (we want many photons)
- Fast nearest neighbor search

KD-tree
Photon Mapping

200 photons / 50 photons in radiance estimate
Photon Mapping

100,000 photons / 50 photons in radiance estimate
Photon Mapping

500,000 photons / 500 photons in radiance estimate
Path Tracing
Photon Mapping

500,000 photons / 500 photons in radiance estimate
Photon Mapping

Radiance estimate contains error/bias
- Produces darker/brighter, blotchy, blurry appearance
- Requires *many* photons for high quality

Split up lighting computation into components:
- Direct lighting
- Caustics (caustic photon map)
- Remaining indirect illumination (global photon map)
Improving Caustics

Higher quality photon map for caustics
- Only stores LS+D paths
- Many photons shot directly at specular objects
Improving Remaining Indirect

Original approach: direct density estimation
Improving Remaining Indirect

Improved approach: using *final gather* (i.e., path trace until second non-specular surface from camera)
Improved Photon Mapping

Camera tracing

- Trace camera paths until they hit the first non-specular surface point $x$

At $x$ we sum:

- Emission

- Direct illumination: trace shadow rays to lights

- Caustics: density estimation at $x$ using only the caustic photon map

- Remaining indirect: continue path tracing until next non-specular vertex $y$, perform density estimation from global photon map at $y$
Photon Mapping

500000 photons / 500 photons in radiance estimate
Photon Mapping (Improved)

final gather + global photon map (200000) + caustic photon map (50000)
Path Tracing
Validation Tests

Test idea 1:
- store only direct photons
- visualize photon map directly
- compare to standard direct illumination
- should look identical with many photons

Test idea 2:
- create a perfectly transparent sphere (IOR = 1.0)
- store only caustic photons
- render direct illumination + caustics
- shadow should disappear
Recall: Path Integral Measurement Eq.

\[ I_j = \int_{\mathcal{P}} W_e(x_0, x_1) L_e(x_k, x_{k-1}) T(\bar{x}) \, d\bar{x} \]

path throughput

\[ T(\bar{x}) = G(x_0, x_1) \prod_{j=1}^{k-1} f(x_j, x_{j+1}, x_{j-1}) G(x_j, x_{j+1}) \]

• Monte Carlo estimator:

\[ I \approx \frac{W_e(x_0, x_1) T(x_0, \cdots, x_k) L_e(x_k, x_{k-1})}{p(x_0, \cdots, x_k)} \]

joint PDF of path vertices
Photon Mapping

\[ I \approx \frac{W_e(x_0, x_1)T(x_0, \cdots, x_k)L_e(x_k, x_{k-1})}{p(x_0, \cdots, x_k)} \]

split path contribution into two parts

\[ \Phi_p = \frac{T(x_2, \cdots, x_k)L_e(x_k, x_{k-1})}{p(x_2, \cdots, x_k)} \]

eye subpath

light subpath/photon “power”

connect with density estimation
Light Sources in the Real World

Complex shape

Covered with transparent materials

Only a small part emits light
Light Sources in CG

Simple shape

Bare light source

Entire part emits light
Why?
Scene with “Realistic” Lights

Images courtesy of T. Hachisuka
Path Tracing
Bidirectional Path Tracing

Images courtesy of T. Hachisuka
Robustness of Rendering Methods

None of these unbiased methods can handle real light sources well:

- Path Tracing
- Bidirectional Path Tracing

Photon Mapping?
Photon Mapping

Images courtesy of T. Hachisuka
Photon Mapping
Photon Mapping - Summary

Advantages
- Handles difficult paths more robustly than unbiased algorithms
- Consistent estimator
- Reuse of computation (photons)

Disadvantages
- Bias shows up in many different forms
- Requires additional data structure (KD-tree)
- No progressive rendering
- Large memory footprint
  Non-intuitive hyperparameter fine-tuning
Characteristics of Estimators

**Unbiased estimator**

- expected value equals the true value being estimated

\[ E[F] = \int f(x) \, dx \]

- variance (noise) is the only error

- averaging infinitely many estimates (each with finite number of samples) also yields the correct answer

\[ \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \langle F^k \rangle = \int f(x) \, dx \]
Characteristics of Estimators

Bias of an estimator

- difference between the expected value of the estimator and the true value being estimated

\[ \beta = E[F] - \int f(x) \, dx \]

- expected average difference

- averaging infinitely many estimates yields the correct answer plus the bias

\[ \lim_{N \to \infty} \frac{1}{N} \sum_{i=1}^{N} \langle F^k \rangle = \int f(x) \, dx + \beta \]
Characteristics of Estimators

**Consistent estimator**

- bias disappears in the limit

\[ \lim_{N \to \infty} E[F] = \int f(x) \, dx \]

Consistent estimators and *unbiased* estimators are asymptotically equivalent

- both need an infinite number of samples to reduce the error to zero
Characteristics of Estimators

**Mean Squared Error (MSE)** of an estimator
- combines variance and squared bias

\[
\text{MSE}[F] = \text{Var}[F] + \text{Bias}[F]^2
\]

**Root Mean Squared Error (RMSE)**
- has the same units as the quantity being estimated
- for unbiased estimators equal to std. deviation

\[
\text{RMSE}[F] = \sqrt{\text{MSE}[F]}
\]
Rendering Techniques

Examples of unbiased methods

- Path tracing
- Light tracing
- Bidirectional path tracing

Examples of biased/consistent methods

- (Progressive) photon mapping
- Many-light methods
Consistency of Photon Mapping

Result converges to the correct solution

Conditions for convergence:
- Infinitesimally small radius
- Infinite number of nearby photons
  - **Infinite storage requirement!**

Images courtesy of T. Hachisuka
Progressive Photon Mapping
Key Idea

Progressively shrink the density estimation kernel

Hachisuka et al. 2008, 2009, ...
- store/update statistics at each camera ray hitpoint

Knaus & Zwicker 2011
- no statistics, just render independent images with smaller and smaller radius, and average
Different kernel radii

Images courtesy of C. Knaus and M. Zwicker
Progressive Radius Reduction

Image 1, $r = 20$

Images courtesy of C. Knaus and M. Zwicker
Progressive Radius Reduction

Image 10, $r = 11.87$

Images courtesy of C. Knaus and M. Zwicker
Progressive Radius Reduction

Image 100, $r = 6.71$

Images courtesy of C. Knaus and M. Zwicker
Progressive Radius Reduction

Image 1000, $r = 3.78$

Noise $\propto \frac{1}{r^2}$

Bias $\propto r^2$

Kernel size

Images courtesy of C. Knaus and M. Zwicker
Running Average

Image 1

Images courtesy of C. Knaus and M. Zwicker
Running Average

Average of Images 1-10

Images courtesy of C. Knaus and M. Zwicker
Running Average

Average of Images 1-100

Images courtesy of C. Knaus and M. Zwicker
Running Average

Average of Images 1-1000

Kernel size

Bias of average

Bias per iteration

Noise of average

Noise per iteration

Images courtesy of C. Knaus and M. Zwicker
Individual iterations

Running average

Images courtesy of C. Knaus and M. Zwicker
Radius Reduction

Given:
- Iteration $i$
- Kernel radius $r_i$
- Parameter $\alpha \in (0, 1)$ for controlling the shrinking

The radius for the next iteration is:

$$r_{i+1}^2 = \frac{i + \alpha}{i + 1} r_i^2$$

See [Knaus & Zwicker 2011] for derivation
Algorithm

Step 1:
- Photon tracing: emit, scatter, store photons

Step 2:
- Trace camera paths
- Evaluate radiance estimate using radius $r_i$

Display running average

Compute new radius $r_{i+1}^2 = \frac{i + \alpha}{i + 1} r_i^2$ and repeat...

Trivially parallelizable by iteration
Path Tracing

Images courtesy of C. Knaus and M. Zwicker
Bidirectional Path Tracing

Images courtesy of T. Hachisuka
Metropolis Light Transport

Images courtesy of T. Hachisuka
Progressive Photon Mapping

Images courtesy of T. Hachisuka
Photon Mapping
Glass Lantern

Path tracing  Bidirectional path tracing  Metropolis light transport  Photon mapping  Progressive photon mapping

Images courtesy of T. Hachisuka
Torus in Cube (LS+D*+E)

Path Tracing  Bidirectional Path Tracing  Progressive Photon Mapping

Images courtesy of T. Hachisuka
Progressive PM - Summary

Reduces memory footprint

- Converges without requiring infinite memory

Renders progressively (user-friendly)

Data structure does not need to be as sophisticated

No need to bother using a caustic map, just use a single photon map for everything
More On Photon Mapping