VO Rendering
SS 2010

Unit 2: Rendering Theory

Sources:

Overview

- Rendering Equation
- Potential Equation
- Basic Strategies for solving the RE and the PE
- A taxonomy of rendering algorithms
- **Overall goal:** to present a manageable mathematical framework into which all common rendering algorithms fit in one way or another

Unit 2 – Part 1

Math

Rendering Equation

$$I(x,x') = g(x,x') \left[\varepsilon(x,x') + \int_S \rho(x,x',x'') \, I(x',x'') \, dx'' \right]$$

- Kajia's 1986 version of the RE as commonly found in literature
- Fredholm Integro-differential equation
- Surface (area) formalism → integrates over visible surfaces
- Completely describes the problem of image synthesis

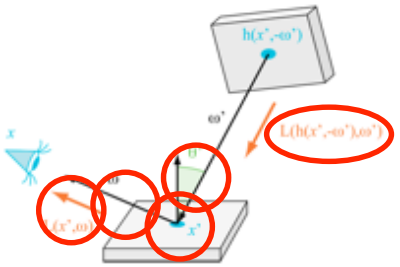
RE – Alternative Form

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega$$

- Directional formalism → integrates over entire hemisphere
- Change in notation (g ⇨ h, rho ⇨ f_r, epsilon ⇨ L_e, I ⇨ L)
- The term for the direct influence of surface emission is moved outside the integral

Alternative RE Geometry

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta' \, d\omega$$



Short Form of the RE

- Given the previous change in notation, we can introduce an integral operator T as

$$(TL)(\vec{x},\omega)=\int_{\Omega} L(h(\vec{x},-\omega'),\omega')\cdot f_r(\omega',\vec{x},\omega)\cdot\cos\theta' d\omega$$

- which enables us to write a shorthand version of the RE as

$$L = L^e + TL$$

- Analytical solutions are usually impossible

Potential Equation

- The PE describes the problem of light transport from the viewpoint of the emitter:

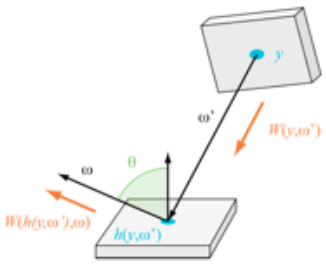
$$(T'W)(\vec{y},\omega')=\int_{\Omega} W(h(\vec{y},\omega'),\omega')\cdot f_r(\omega',h(\vec{y},\omega'),\omega)\cdot\cos\theta d\omega$$

- T' is the adjoint operator of T from the RE
- The PE can also be written in short form as

$$W = W^e + T'W$$

Potential Equation Geometry

$$(T'W)(\vec{y},\omega')=\int_{\Omega} W(h(\vec{y},\omega'),\omega')\cdot f_r(\omega',h(\vec{y},\omega'),\omega)\cdot\cos\theta d\omega$$



RE & PE Symmetry

- The rendering equation sees the problem of light transport from the viewpoint of the receiver
- The potential equation is the adjoint problem; it models the situation from the viewpoint of the emitter
- Both equations are of similar type and have to be approached in similar ways
- The PE is introduced for reasons of symmetry and to explain certain methods

RE & PE Difference

- The key difference between the two equations is what is being computed during their evaluation:
- **Rendering Equation**
  - Individual radiance values are computed for each viewing ray
  - Immediately useful for rendering
- **Potential Equation**
  - Computes the „radiance state“ for entire scenes
  - Results have to be stored and evaluated later

Contractivity

- For all physically plausible environments, the integral operators T and T' are *contractive*, which means that
- Repeated applications of T or T' yield successively smaller results because all realistic surface reflectances are < 1!
- Scenes with highly specular surfaces are less contractive than diffuse environments, i.e. iterative solutions take longer to converge

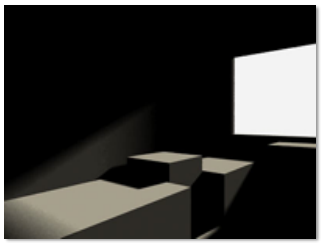
Solution Technique Classification

$$L = L^e + TL$$

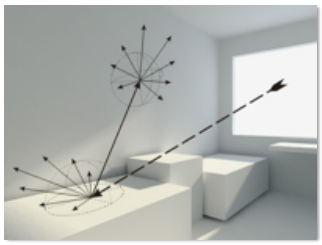
The two sides of the equation are *coupled*

- Local illumination models
  - Coupling is ignored – no recursion
- Recursive ray-tracing
  - Certain „easy“ types of coupling are followed (specular & transmission)
- Global illumination methods
  - Full treatment of RE coupling

Practical Example



Direct illumination alone



Direct and indirect illumination

Global Illumination Solution Strategies

- **Inversion**
  - Not used in practice
- **Expansion**
  - Almost exclusively used by stochastic techniques (ray tracers are an exception)
- **Iteration**
  - Both stochastic and deterministic (i.e. finite element) approaches exist

Example

- Simple equation

$$x = 0.1x + 1.8$$

- Ignoring the coupling

$$x = 0.1x + 1.8 \approx 1.8$$

- Approximation

$$x = 0.1x * 1.8 + 1.9 = 1.98$$

Inversion

- Groups terms that contain the unknown on the same side of the equation
- Then a formal inversion operation is applied

$$L = L^e + TL$$
$$(1 - T)L = L^e$$
$$L = (1 - T)^{-1} L^e$$

- Example equation $x = 0.1x + 1.8$

$$-x = 0.9^{-1} * 1.8 = 2$$

Inversion in Practice

- T is infinite dimensional and cannot be inverted in closed form
- Problem can be approximated by a finite element approach, which eventually yields a system of linear equations, which then have to be inverted
- No longer used due to *cubic time complexity* and *numerical instability*
- Not dependent on contractivity of T!

Gathering Expansion

- Recursive substitution of L:

$$L = L^e + TL$$
$$L = L^e + T(L^e + TL)$$
$$L = L^e + TL^e + T^2 L$$

- If repeated *n* times, a Neumann series results:

$$L = \sum_{i=0}^{n} T^i L^e + T^{n+1} L$$

- Example equation $x = 0.1x + 1.8$

$$x_i = 1.8 + 0.1 * 1.8 + 0.1^2 * 1.8 + ... + 0.1^{i+1} * x$$
$$x_0 = 1.8 \rightarrow x_1 = 1.98 \rightarrow x_2 = 1.998 \quad ...$$

Gathering Expansion

- If T is a contraction (and in rendering it is) then

$$\lim_{n \to \infty} T^{n+1} L = 0$$

- which leads us to

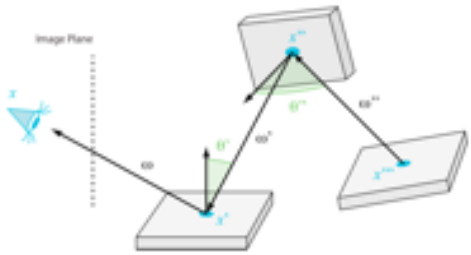
$$L = \sum_{i=0}^{\infty} T^i L^e$$

- as a solution for the rendering problem.

How is this useful?

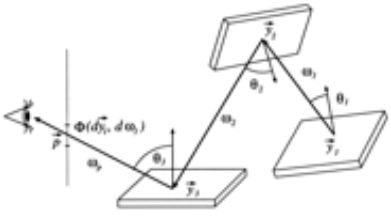
- We replaced an intractable equation with an infinite series of integrals with successively higher dimensionality...
- Did we gain anything through this?
- Obviously, otherwise we would not have bothered! ;-)
- The series of integrals corresponds to the levels in a recursive gathering algorithm!

Gathering Expansion Geometry



- The recursive substitution corresponds to recursion levels during a ray-casting process which originates from the eye

Shooting Expansion



$$W = \sum_{i=0}^{\infty} T'^i W^e$$

- The recursive substitution corresponds to recursion levels during a ray shooting process which originates at the emitter

Expansion Steps

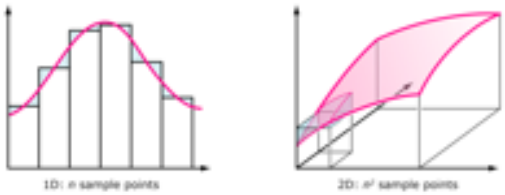
- At each recursion level, we have to integrate over the entire hemisphere for each sample point
- This quadrature has to be performed numerically for all but trivial scenes
- The integral is high-dimensional since it includes the recursion from there onwards!

Numerical Quadrature

- A finite number of samples is taken from the integration domain
- The integrand is evaluated for these samples
- The numerical result of the integral is computed as a weighted sum of these result values

$$I = \int_V f(z)dz \approx \sum_{i=1}^{N} f(z_i) \cdot w(z_i)$$

Classical Numerical Integration



- Brick rule, Simpson's rule - simple and effective for low-dimensional integrands
- Effort needed for given accuracy *rises exponentially* with dimension of the integrand!
- Monte Carlo integration is the only viable method of performing this quadrature in practice

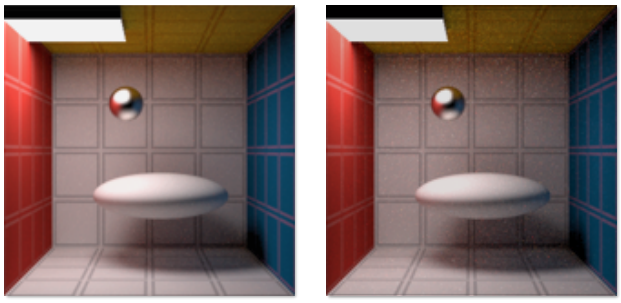
Monte Carlo Quadrature

- Converts the calculation of an integral to an equivalent expected value problem

$$E[f(z)] \approx \hat{f} = \frac{1}{N}\sum_{i=1}^{N} f(z_i)$$

- Random sampling of the integrand used as a basis for determining the result
- Number of samples needed for a given dimension of the integrand is **not** dependent on the dimension!

Rays per Pixel



Reference | Increasing samples / pixel

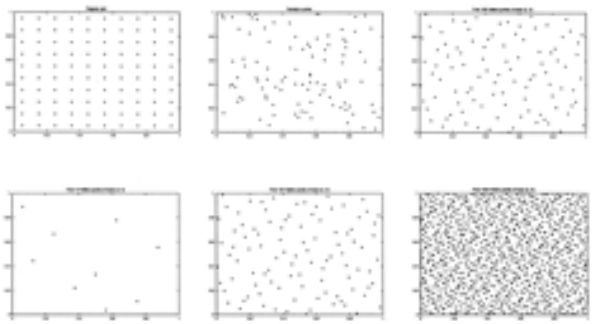
Choosing Sampling Points for MC

- Two strategies are possible:
  - Importance sampling tries to find the best sample points by trying to guess the correct distribution
  - Stratification aims at using samples which cover the integrand very evenly
- Discrepancy is the measure of sampling quality for sequences of sample points
  - Regular grids have very high discrepancy!

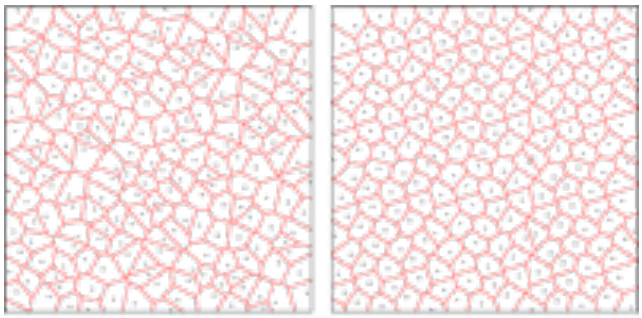
Quasi Monte Carlo

- True random numbers have a too high discrepancy for good behaviour in MC integration algorithms
- Deterministic low-discrepancy sequences are used instead:
  - Halton sequences for arbitrary numbers of points
  - Hammersley sequences if the number of needed points is known in advance
  - TMS nets for 2D integrands

Quasi Monte Carlo Example

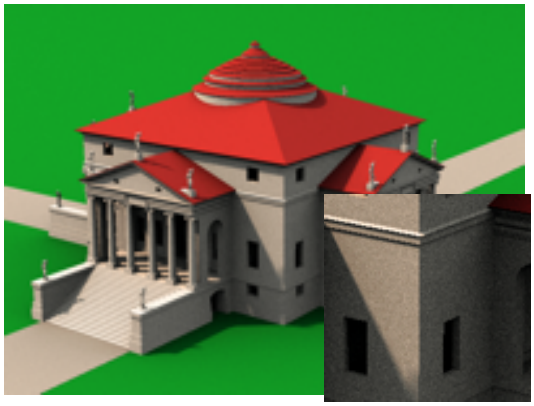


Halton vs. Hammersley

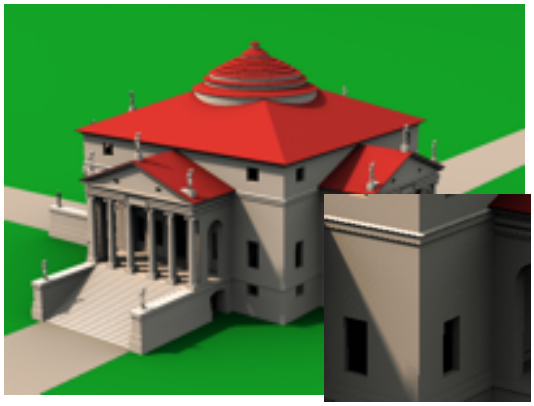


(a) 196 Halton points (b) 196 Hammersley points

Random Values



Halton Sequence



QMC Issues

- QMC sequences provide substantial performance and quality gains for rendering applications
- However, QMC generators are **not** drop-in replacements for normal random generators like `rand()`
- For example, one must not use values from the same sequence twice during one recursive descent into a scene if Halton sequences are used

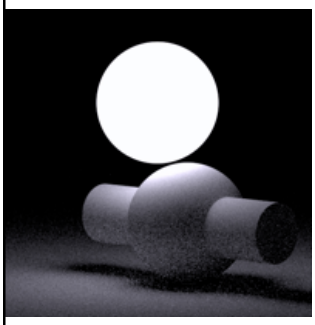
Expansion Disadvantages

- Paths have to be independent, so no coherency between them can be exploited
- It requires the evaluation of very high dimensional integrals
  - Either the walks are truncated (when they reach a threshold), which introduces a bias
  - or they are stopped randomly at some level, which reduces sampling of higher recursion levels (scenes with mirrors!)

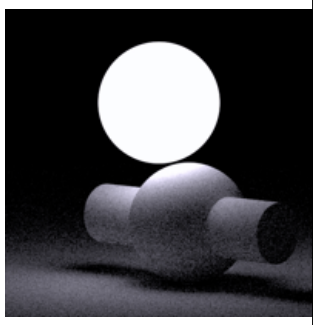
Path Termination

- **Fractional propagation / attenuation**
  - The absorbed energy is subtracted at every step, and the path is terminated once it carries less energy than a certain threshold
  - Biased, but more intuitive
- **Russian Roulette**
  - Random termination of entire ray according to propagation probability
  - No bias, less intutitive

Russian Roulette vs. Fractional Termination



RR, 1000 samples



FT, 100 samples

Expansion Advantages

- No temporary representations of the complete radiance function are required for gathering expansion (storage space & accuracy issues!)
- For shooting expansions the storage techniques can be comparatively flexible
- Algorithms can work on the original geometry without tesselations
- Walks are independent and can be parallelised

Iteration

- A solution of the RE is a fixed point of

$$L_n = L^e + TL_{n-1}$$

- If T is contractive, this will converge from any initial distribution L_0
- Finite element techniques (which introduce a discretisation error) have to be used
- Example equation: $x = 0.1x + 1.8$

$$x_n = 0.1x_{n-1} + 1.8$$

$$x_0 = 1.8 \qquad x_1 = 1.98 \qquad x_2 = 1.998$$

Iteration Disadvantages

- Requires object tesselation and finite element representation
  - Geometric accuracy and coherence is lost
  - Substantial storage requirements even for moderately complex scenes
- Accuracy of high frequency shadows, reflections and caustics is problematical
- A solution is computed even for parts of the scene which are invisible

Iteration Advantages

- Coherence can be exploited well
- Approximating functions L_n are viewpoint-independent: potential advantage for animations
- Provides implicit smoothing through discretisation, i.e. more visually pleasing images than noisy expansion
- More robust for highly reflective environments

Unit 2 – Part 2

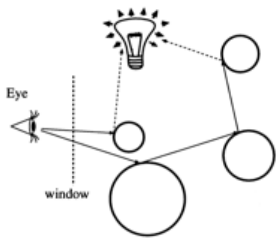
Gathering Expansion Algorithms in Comparison

Random Walk Algorithms

- Algorithms can be categorized according to their basic strategy:
  - Gathering type RW
  - Shooting type RW
  - Bi-directional algorithms
  - Global methods

Gathering Type Random Walk

- Starts at the eye
- Gathers the emission of the visited points
- Differences in `Trace()` function determine actual algorithm type



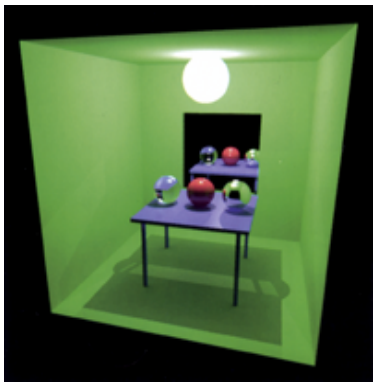
Gathering Type Random Walk

```
for each pixel do
  colour = 0
  for i = 0 to N do
    ray = random ray through pixel
    samplecolour = c·Trace(ray)
    colour += samplecolour / N
  endfor
endfor
```

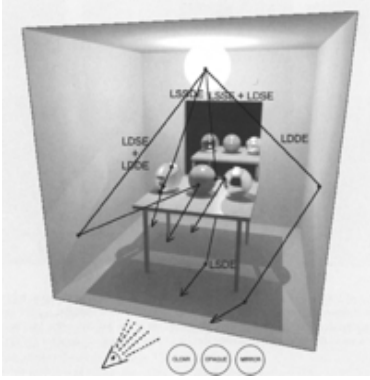
Heckbert's Taxonomy

- Heckbert's taxonomy provides further information
- Used to categorize rendering algorithms
  - E is the eye
  - L is the lightsource
  - D is a non-ideal reflection or refraction
  - S is an ideal reflection or refraction
  - * is the sign of iteration
  - [ ] represents optionality
  - | means selection

Global Illumination Scene



Global Illumination Scene

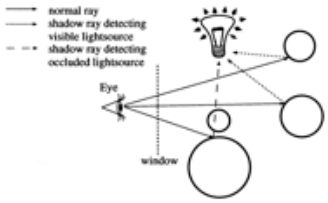


Gathering Type Algorithms

- Ray casting – LDE
  - ***Ray casting*** is the act of intersecting a single ray with a scene
- Ray tracing – L[D]S*E
  - ***Ray tracing*** is a photorealistic rendering algorithm
  - Raytracers – as well as more sophisticated renderers – use raycasters!
- Distribution raytracing – L[D|S]*E
- Path tracing – L[D|S]*E

Raycasting – LDE

- Sometimes referred to as First Hit Raytracing or Nonrecursive Raytracing



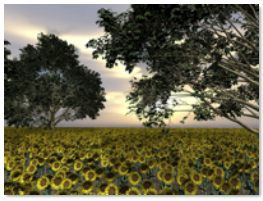
- Possible to implement as real time renderer
- Potentially more efficient than OpenGL for highly complex scenes (>10M polygons)

Raycasting

```
Trace(ray)
  hit = FirstIntersect(ray)
  if no intersection
     return backgroundColour
  else return
     emission(@hit,-ray.dir) +
     directLighting(@hit,-ray.dir)
```

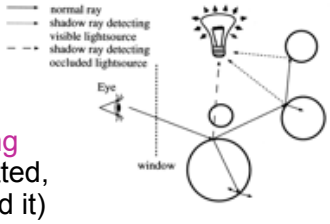
Realtime RT

- Based on ray casting
- Certain limited types of recursion possible
- GI also possible
- Current state of the art: 1 billion polygons 640x480
- www.openrt.de



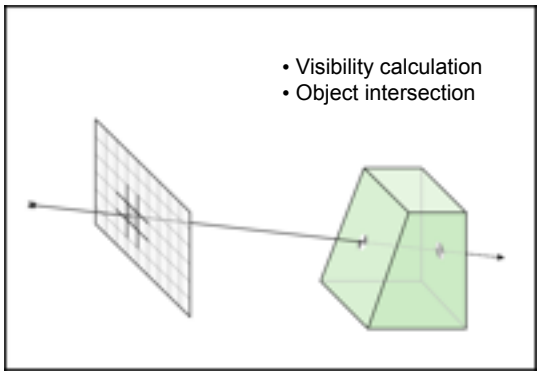
Raytracing – L[D]S*E

- "Classical" raytracing as discussed in CG1

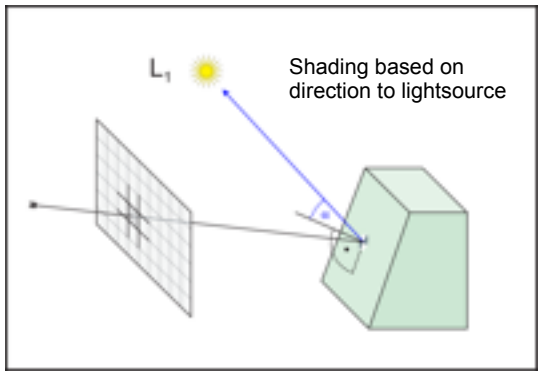


- Also known as Whitted Raytracing (after Turner Whitted, who first published it)
- This is a *hybrid* algorithm:
  - Recursion is evaluate for perfect mirrors
  - Coupling is ignored otherwise

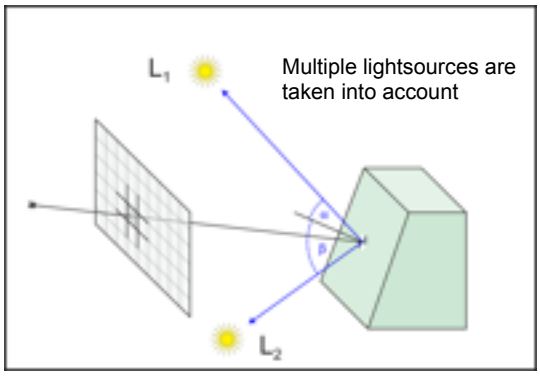
Raytracing – Step 1



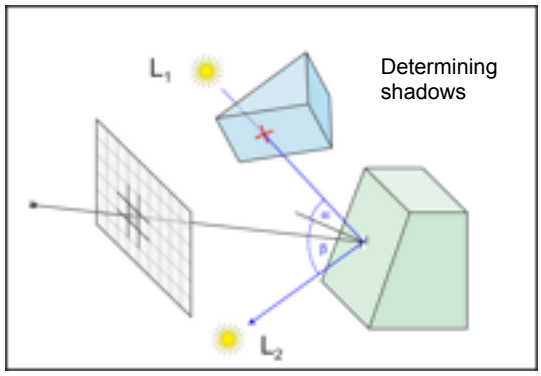
Raytracing – Step 2



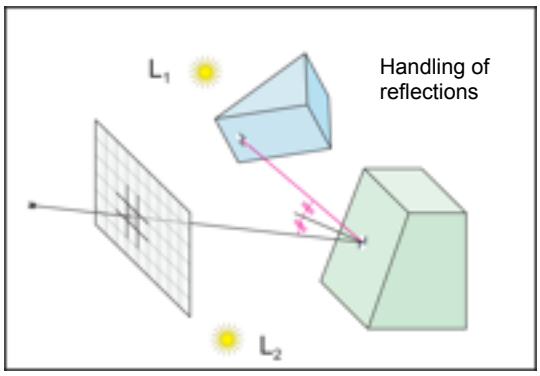
Raytracing – Step 3



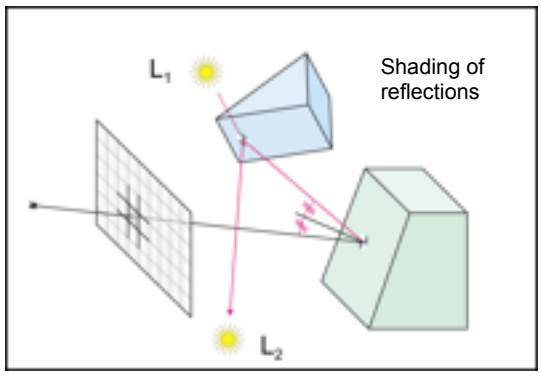
Raytracing – Step 4



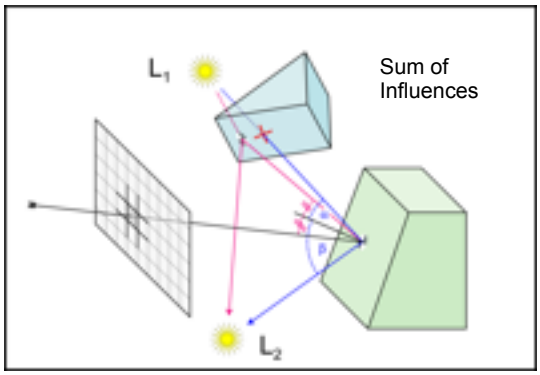
Raytracing – Step 5



Raytracing – Step 6



Raytracing – Step 7



Raytracing

```
Trace(ray)
  hit = FirstIntersect(ray)
  if no intersection
     return backgroundColour
  colour =   emission(@hit,-ray.dir)
           + directLighting(@hit,-ray.dir)
  if kr > 0 then
     colour += kr * Trace(reflectedRay)
  if kt > 0 then
     colour += kt * Trace(transmittedRay)
  return colour
```

Last Unit

Theory

- Rendering Equation

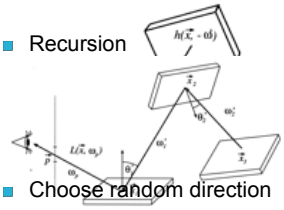
$$(TL)(\vec{x},\omega)=\int_{\Omega} L(h(\vec{x},-\omega'$$

- Solution Strategies
  - Inversion
  - Expansion
- Monte Carlo Sampling
  - Evaluate Integrand with finit number of samples

$$I=\int_V f(z)dz \approx \sum_{i=1}^{N} f(z_i)\cdot w(z_i)$$

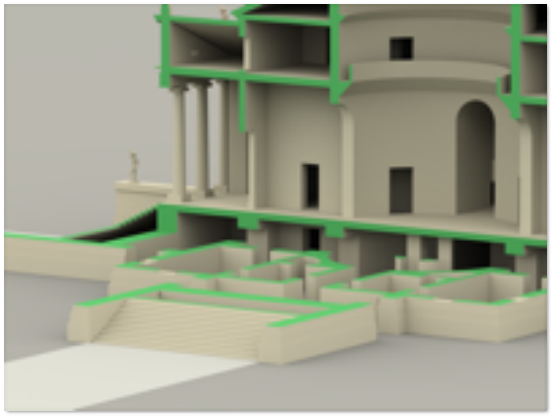
Practice

- Light propagation in scene
- Recursion



- Choose random direction
- Follow more than one ray per pixel
- Weight ray according to propagation probability

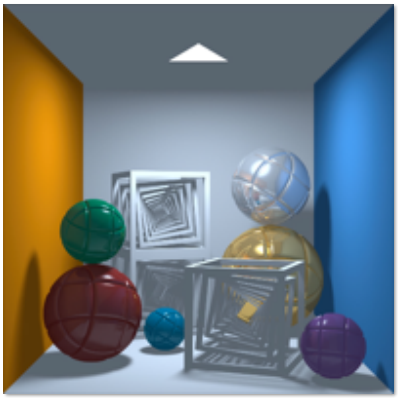
Global Illumination Example



Global Illumination

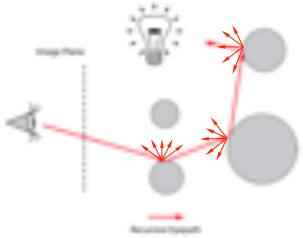
Raytraced Scene

Raytraced Scene with Ambient Term



Distribution Raytracing

- L[D|S]*E
- Also known as distributed raytracing
- Impractical due to fan out of N at every level
- Very slow convergence



Distribution Raytracing

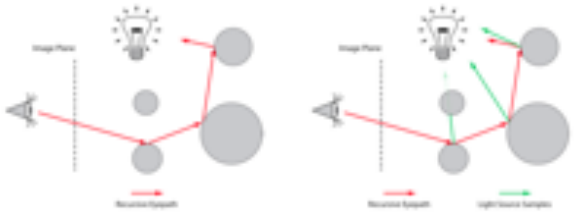
```
Trace(ray)
  hit = FirstIntersect(ray)
  if no intersection return backgroundColour
  colour =   emission(@hit,-ray.dir)
           + directLighting(@hit,-ray.dir)
  for sample = 1 to N do
     p = BRDFSampling(-raydir,normal,newRay)
     if p > 0 then colour += Trace(newRay)
        * weight(newRay.dir,normal,-ray.dir)
        / (p * N)
  endfor
  return colour
```

Example

Distribution Raytracing

- **Pro:**
  - Simple!
- Process eventually converges to true solution
- **Con**:
  - Very bad convergence characteristics
- In its original form the algorithm has been completely superseded by more modern approaches

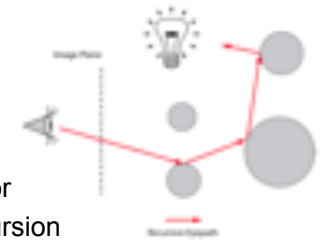
Path Tracing – L[D|S]*E



- A single path is traced through the scene
- Versions without (left) and with (right) intermediate light source evaluation exist

Simple Path Tracing

- FAST!
- Simple to code, since all you ever do is to follow a path until you either
  - Hit a light source or
  - Exceed some recursion threshold



- Drawback: does not work for small light sources

Path Tracing

```
Trace(ray)
  hit = FirstIntersect(ray)
  if no intersection return backgroundColour
  colour =    emission(@hit,-ray.dir)
           + directLighting(@hit,-ray.dir)
  p = BRDFSampling(-ray.dir,normal,newRay)
  if p > 0 then return colour
  colour += Trace(newRay)
     * weight(newRay.dir,normal,-ray.dir) / p
  return colour
```

Problem: Hitting the Light Source

Reference | Simple Path Tracer (150 samples / pixel)

Improved Path Tracing

- Include sampling of the light sources
  - „Multiple importance sampling“
- Key problem:
  - Correct weighting of the two samples

- Solved in 1995 by Veach and Guibas
  - No real follow-up work yet

Estimating Incident Light

- **Hemispherical Integration (RE v2)**
  - Done by simple path tracer
  - No $1/r^2$ sample weighting
  - No partitioning of integrand
- **Direct Lightsource Sampling (RE v1)**
  - Partitions integrand into direct and indirect illumination
  - $1/r^2$ sample weighting
  - Potentially much more efficient than HI in some cases

Rendering Equation

- Area Formalism

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')\, I(x',x'')\, dx''\right]$$

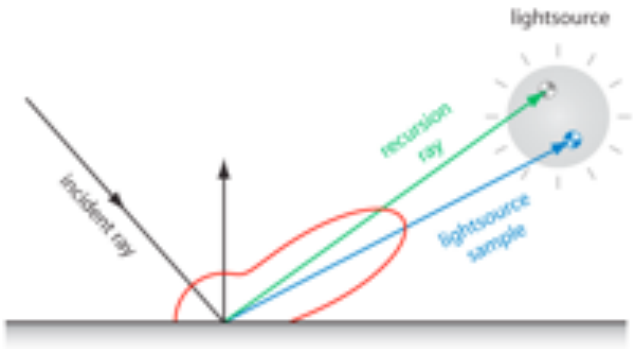
- Hemispherical formalism

$$L(\vec{x},\omega) = L^e(\vec{x},\omega) + \int_\Omega L(h(\vec{x},-\omega'),\omega') \cdot f_r(\omega',\vec{x},\omega) \cdot \cos\theta'\, d\omega$$

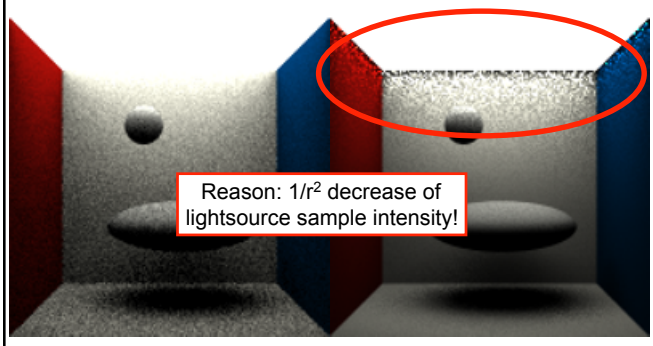
Path Tracing: Sample Weighting

- At each surface intersection, two possibilities to continue the ray exist:
  - According to the BRDF
  - Through sampling of the lightsources
- Both techniques have their merits depending on the circumstances
- BIG problem: knowing which one to choose requires knowledge of the solution

Multiple Importance Sampling

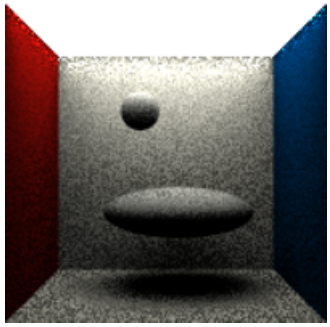


BRDF vs. Lightsource Sampling



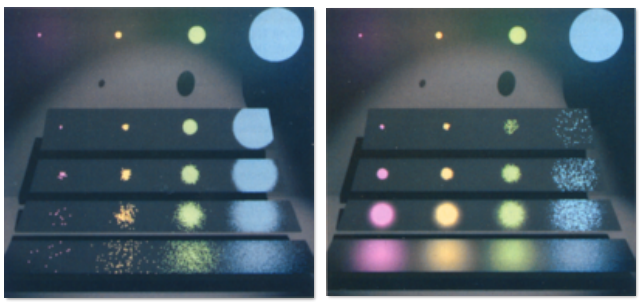
BRDF lightsource

Simple Averaging



- Retains the worst properties of both :-)

Sample Weighting: Example

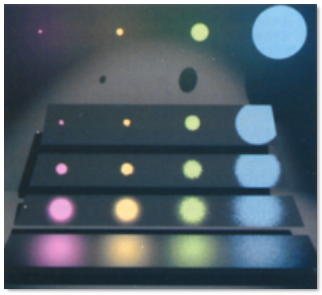


BRDF Lights

- Using both samples without scaling yields wrong results

Sample Weighting: Solutions

- No simple solution is possible
- Averaging is sub-optimal
- Programs use a heuristic to determine the type of sample to prefer
- Heuristic is based on the relative sample probabilities



The Balance Heuristic

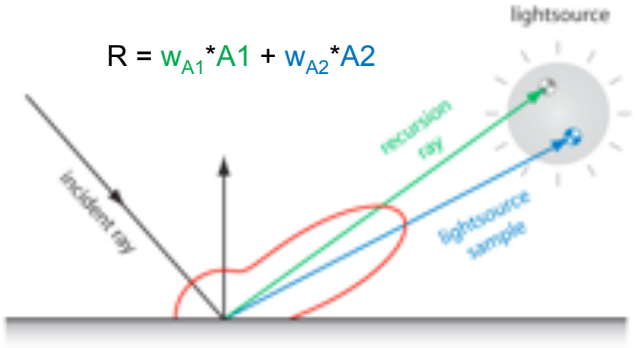
- This is only used when both rays hit the same light source - all other cases are simply added!
- Key idea: weigh each sample according to its relative probability:

$$w_i = \frac{p_i(x)}{\sum_j p_j(x)} \qquad R = w_A \cdot A + w_B \cdot B$$

- Weights sum to one - no energy is counted twice!

Multiple Importance Sampling

$R = w_{A1}*A1 + w_{A2}*A2$



Relative Sample Probabilities

- Each ray (BRDF A1 and LS A2) has two p_j:
  - **p_1 - the BRDF probability**
    ("how probable is it that this ray gets created as a BRDF sample")
  - **p_2 - the light source sample probability**
    ("how probable is it that this ray gets created as a light source sample")
- Formulas for each of the two can be found in literature
- Each ray uses its „own" p for its w, e.g. p_1 for the BRDF ray:

$$w_{A1} = \frac{p_{1A1}}{p_{1A1} + p_{2A2}}$$

Sample p1 Formulae

- Lambertian surface

$$p_1(\omega) = \frac{\cos\theta}{\pi}$$

- Phong-type specular lobe

$$p_1(\omega) = \frac{n+1}{2\pi}\cos^n\varphi$$

- Mirror

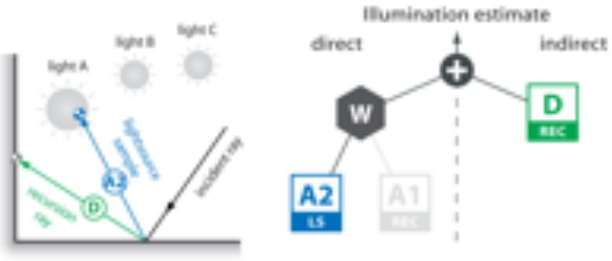
$$p_1(\omega) = \delta(\omega - \omega_r)$$

Area Light Source Probability p2

- Dependent on:
  - absolute area S_e
  - squared distance between sample and surface point
  - cosine of angle

$$p_2(\omega) = \frac{1}{S_e}\frac{|x-y|^2}{\cos\theta}$$

Scenario 1

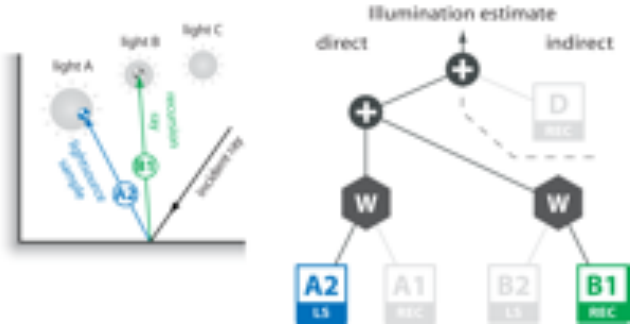


Green = evaluation according to RE v. 1 (solid angle sampling)
Blue = evaluation according to RE v. 2 (lightsource sampling)

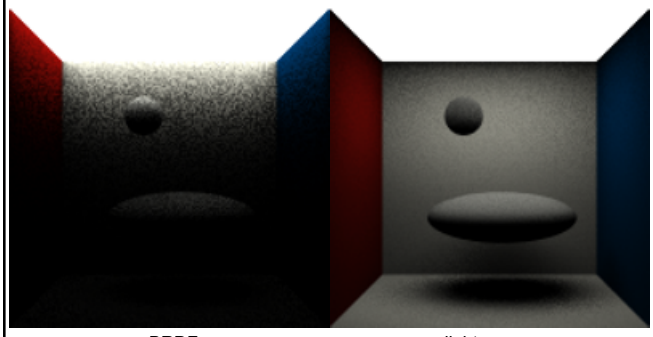
Scenario 2

Green = evaluation according to RE v. 1 (solid angle sampling)
Blue = evaluation according to RE v. 2 (lightsource sampling)

Scenario 3

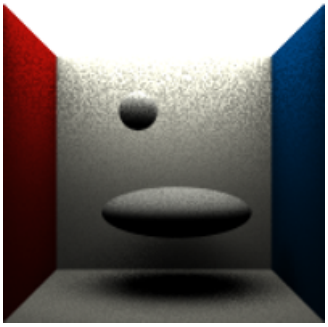


Weighted BRDF vs. Light Source



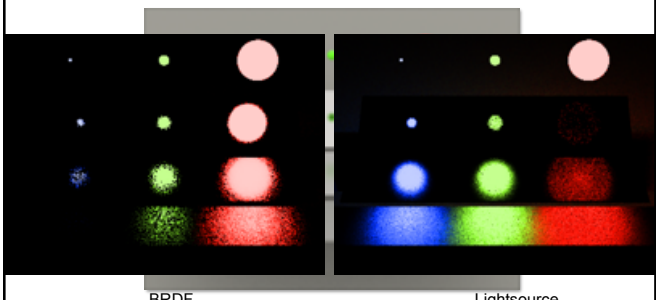
BRDF lightsource

Weighted Addition



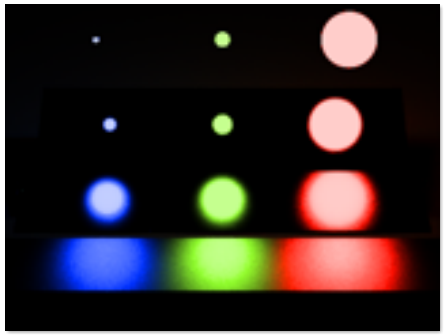
- Not perfect, but a big improvement

Veach & Guibas Replica



BRDF Lightsource

Veach & Guibas Replica - Final

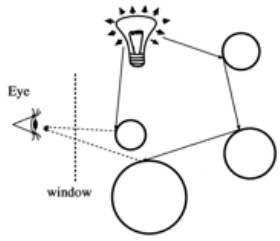


Path Tracing

- **Pro:**
  - Simple!
  - Converges to true solution
  - Better convergence than Distribution RT
- **Con:**
  - Fairly bad convergence characteristics for arbitrary scenes, especially if the lightsources are small

Shooting Type Random Walk

- Starts at the lightsources
- Spreads the emission to the visited points, which are then projected into image space



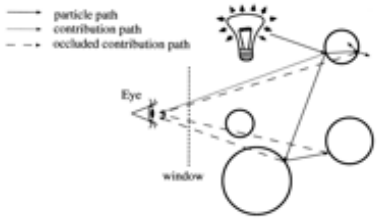
- Differences in **Shoot()** function determine actual algorithm type

Shooting Type Random Walk

```
clearImage
for i = 0 to N do
 ray = random ray from light
   with selection probability p_e
 power =   L_e * cos(phi)
         / (P_e * N)
 Shoot(ray,power)
endfor
```

Forward Raytracing

- LS*DE
- Also known as photon tracing
- Inverse of raytracing
- Unuseable convergence speed
- Limited to RT-type images

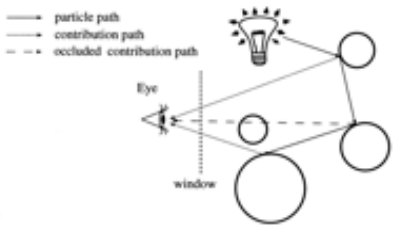


Forward Raytracing

```
Shoot(ray,power)
 hit = FirstIntersect(ray)
 if no intersection then return
 if hit is visible from pixel p then
  colour[p] += emission(eyedirection)
    + power * brdf(@hit,ray.dir,eyedir) * c
 endif
 if kr>0 then Shoot(reflectedRay,kr*power)
 if kt>0 then Shoot(transmittedRay,kt*power)
 return
```

Light Tracing

- L[D|S]*E
- Could also be called forward path tracing
- Inverse of path tracing
- High variance
- Converges to true solution
- Unuseable in practice



Light Tracing

```
Shoot(ray,pow)
 hit = FirstIntersect(ray)
 if no intersection then return
 if hit is visible from pixel p then
  colour[p] += emission(eyedirection)
    + pow * brdf(@hit,ray.dir,eyedir) * c
 endif
 p = BRDFSampling(-ray.dir,normal,newRay.dir)
 if p = 0 then return
 newPow = pow*w(-ray.dir,normal,newRay.dir)/p
 Shoot(newRay,newPow)
 return
```

Gathering vs. Shooting Algorithms

- Dual Algorithms, solve same problem
- Which performs better?
- Depends on several factors
  - Image size vs. scene size
  - Surface types
  - Light sources

Bidirectional Random Walk Algorithms

- Attempt to overcome the difficulties of gathering and shooting by combining them
- Two algorithms exist:
  - Bidirectional path tracing
  - Metropolis light transport

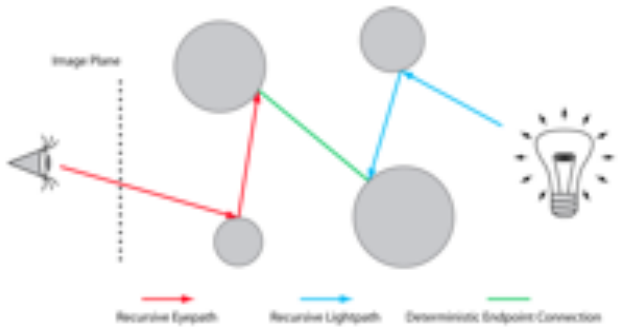
Bidirectional Path Tracing

- Similar to Path or Light Tracing, except:
  - Two paths are randomly cast, one from the eye, and one from one of the lightsources
  - To convert a shooting type walk to a gathering type walk the radiance has to be multiplied by

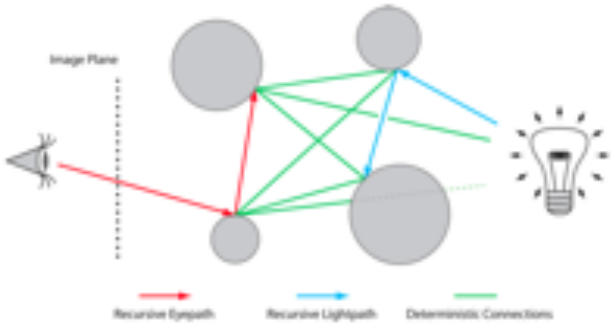
$$\frac{\cos\theta'_k \cdot \cos\theta_{n-k+1}}{r_k^2}$$

- In one version, all mutual inter-connections are evaluated, in the other just the last one
- Both paths are eventually stopped when below a certain importance threshold, or russian roulette is applied

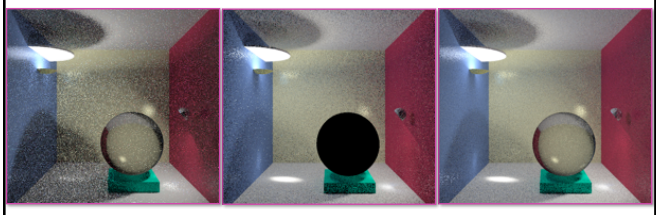
Bi-directional Path Tracing Concept



Full Bi-directional Path Tracing



Comparison

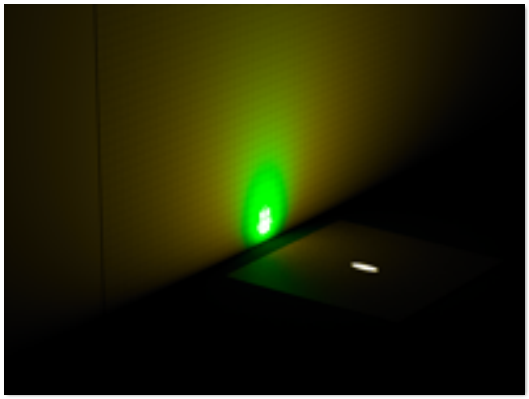


Stochastic ray tracing	Particle tracing	Bidirectional path tracing
(9 samples per pixel)	(9 samples per pixel)	(4 samples per pixel)

Same computation time

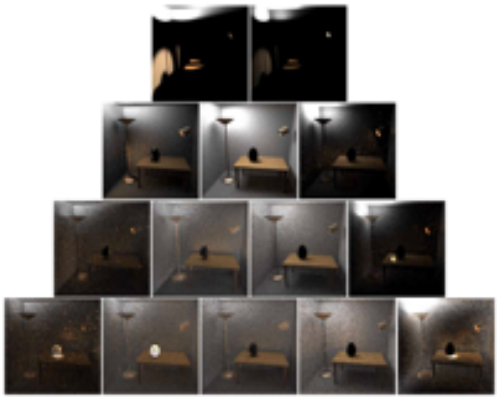
Typical BPT Scene

Another BPT Scene

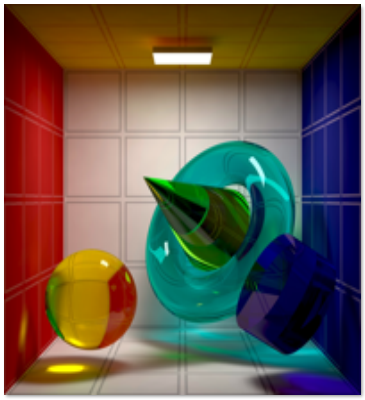


BPT vs. PT

BPT Sample Weighting



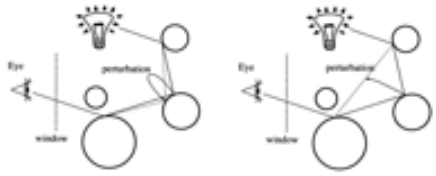
Less Usefull for Caustics...



Metropolis Light Transport

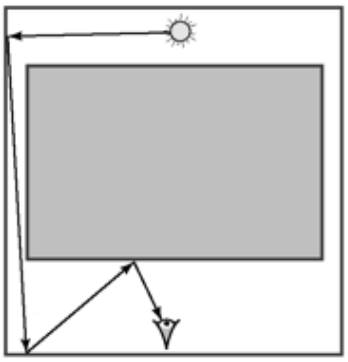
- Bi-directional tracing until a useful path is found (costly & rare)
- One then attempts to change the found path „a little bit“ in order to (hopefully) gain more useful paths
- Problem: doing this without breaking the stochastic simulation
- Metropolis sampling

Changing a Path

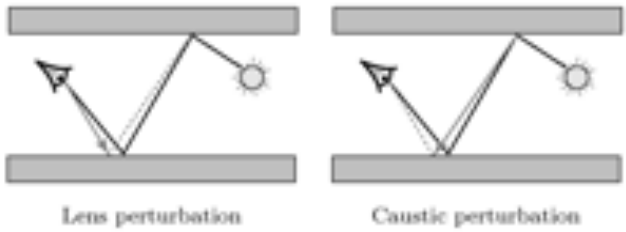


- **Bidirectional Mutations**: Changes in path length – vertices are added or deleted
- **Perturbations**: Directions are slightly altered at certain „neuralgic“ points in a path

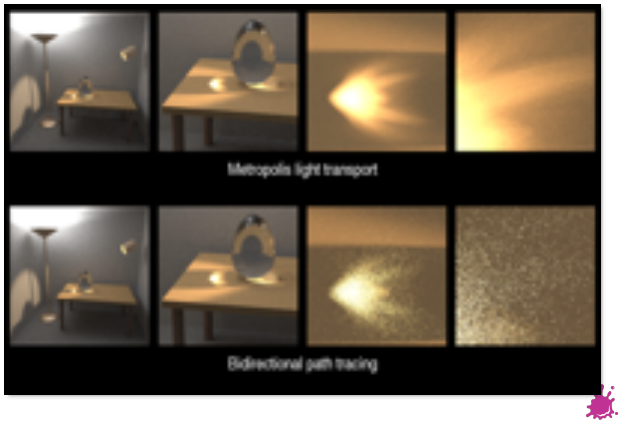
Path Mutations



Perturbations



Metropolis Example #1



Metropolis Example #2 a

Metropolis Example #2 b

Metropolis: Useability

- **Pro:**
  - Handles „difficult“ situations well
- **Con:**
  - Startup bias makes it less than optimal for „normal“ scenes
  - Very difficult to implement

Unit 2 – Part 3

Iteration and Storage-Based Shooting Algorithms in Comparison

Iteration

- A solution of the RE is a fixed point of

$$L_n = L^e + TL_{n-1}$$

- If T is contractive, this will converge from any initial distribution L_0
- In order to store the needed intermediate approximating functions L_n, finite element techniques (which introduce a discretisation error) have to be used

Iteration Disadvantages

- Requires object tesselation and finite element representation
  - Geometric accuracy and coherency is lost
  - Substantial storage requirements even for moderately complex scenes
- Accuracy of high frequency shadows, reflections and caustics is problematical
- A solution is computed even for parts of the scene which are invisible

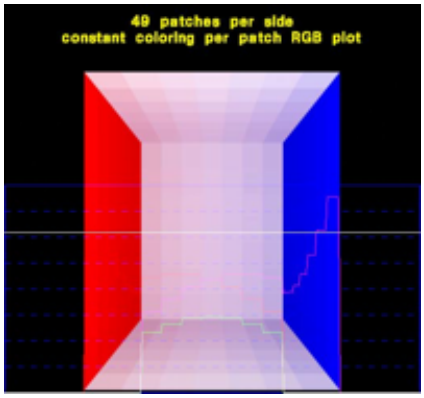
Iteration Advantages

- Coherency can be better exploited
- Approximating functions L_n are viewpoint-independent: potential advantage for animations
- Provides implicit smoothing through discretization, i.e. more visually pleasing images than noisy expansion
- More robust for highly reflective environments

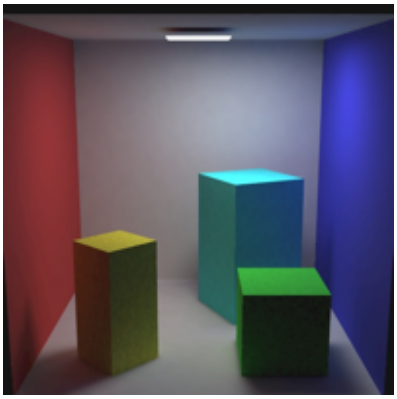
Types of Storage-Based Methods

- Classical Radiosity - Iteration
  - Deterministic
  - Form-factor based
  - Discretization of scene into patches
- Stochastic Approaches - Expansion (!)
  - Global lines
  - Photon tracing

Cornell Box #1



Cornell Box #2



Radiosity Factory (1988)

The Rendering Equation

- **Radiosity B:** rate at which energy leaves a surface (energy per unit area per unit time)
- Full Rendering Equation:

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')\, I(x',x'')\, dx''\right]$$

- Diffuse interaction only:

$$B = E + k_d \cdot \int_\Omega I(x)\, dx$$

The Radiosity Equation

- Diffuse Interaction, version 2 (radiosity integral equation):

$$B(x) = E(x) + \rho(x)\int_S B(x')G(x,x')dA'$$

- The same after discretisation:

$$B_i = E_i + k_{di} \cdot \sum_{j=1}^{n} F_{ij} \cdot B_j$$

$$F_{ij} = \left(\frac{1}{A_i}\int_{Si}\int_{Sj} \frac{\cos(\Theta_{xy},N_x)\cos(-\Theta_{xy},N_y)}{\pi r_{xy}^2} dA_y dA_x\right) V(x,y)$$

The Radiosity Problem

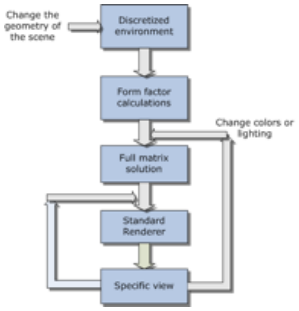
$$B_i - k_{di} \cdot \sum_{j=1, j \neq i}^{n} F_{ij} \cdot B_j = E_j$$

$$\begin{pmatrix} 1 & -k_{d1}F_{12} & \cdots & -k_{d1}F_{1n} \\ -k_{d2}F_{21} & 1 & \cdots & -k_{d2}F_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ -k_{dn}F_{n1} & -k_{dn}F_{n2} & \cdots & 1 \end{pmatrix} \cdot \begin{pmatrix} B_1 \\ B_2 \\ \vdots \\ B_n \end{pmatrix} = \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_n \end{pmatrix}$$

Components of the Problem

- Discretisation of input geometry
- Computation of the form factors for every pair of patches
- Numerical solution of the radiosity system
- Display of the solution

- Problems
  - Scene discretisation
  - Form factor computation



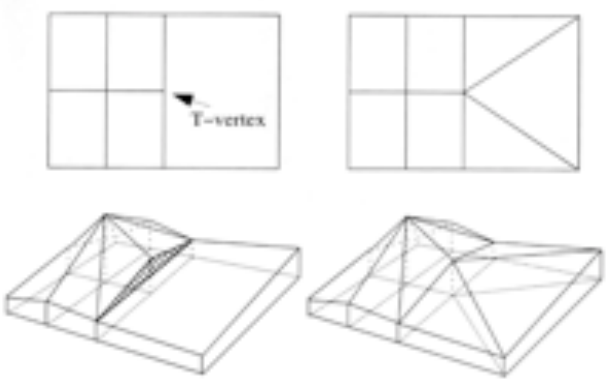
Illumination Representation

- Illumination information has to be stored on patches
- Deterministic radiosity algorithms have at least $O(n^2)$ complexity
- More patches are expensive, so a better representation on fewer patches can improve the method

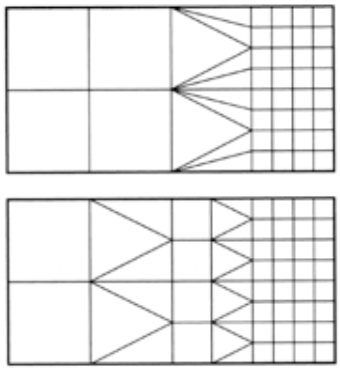
Mesh Topology

- Scene has to be meshed into a set of well-formed polygons
  - *A priori*: Meshing before radiosity solution is invoked
  - *A posteriori*: Mesh is refined as the solution progresses
- No odd (oblique) shapes
- No T-vertices
- Mesh boundaries in appropriate places are desirable

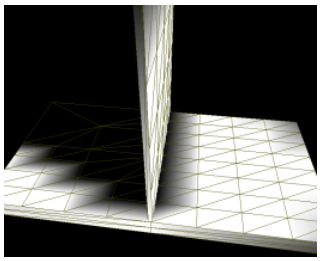
T-Vertices



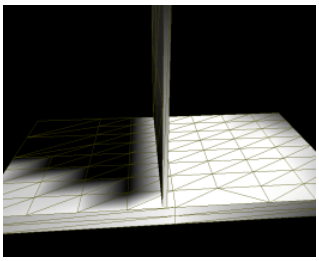
Mesh Transitions



Meshing artefacts

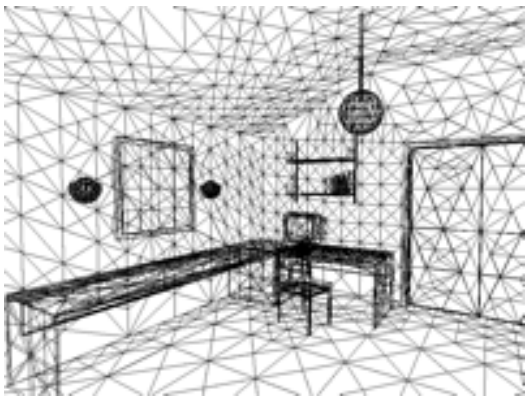


Shadow leak

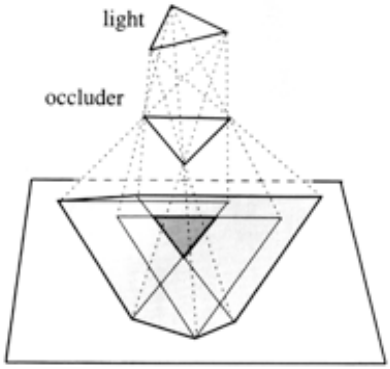


Light leak

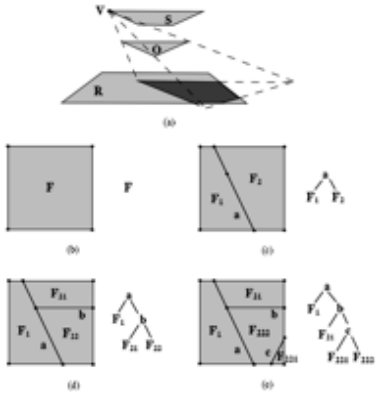
Adaptive Meshing



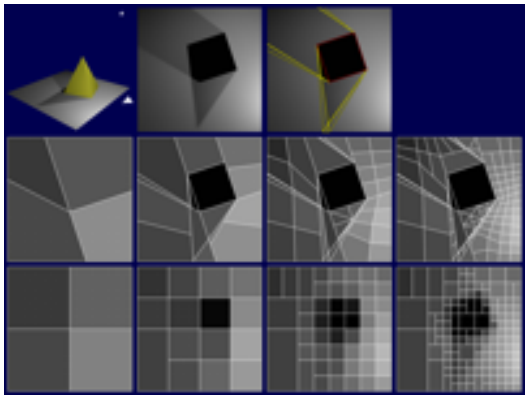
Discontinuity Meshing



Incremental Mesh Construction



Dincontinuity Meshing vs. Reg. Subdevision



Discomesh Example

Standard solution

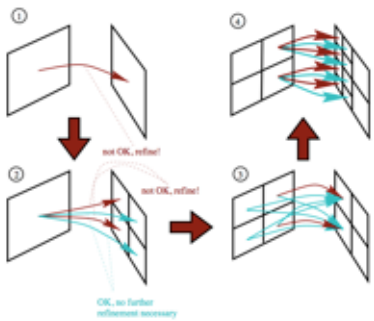
Discontinuity meshing radiosity

Hierarchical Radiosity

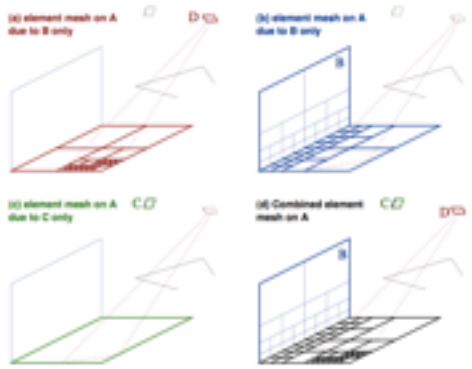
- Raising the number of patches
  - increases computation time significantly
  - increases the accuracy of the solution not nearly as much
- Solution: only compute those interactions that matter
- Problem: adapt radiosity solver
- HR looks directly on the form factors

Build Hierarchical Subdivision

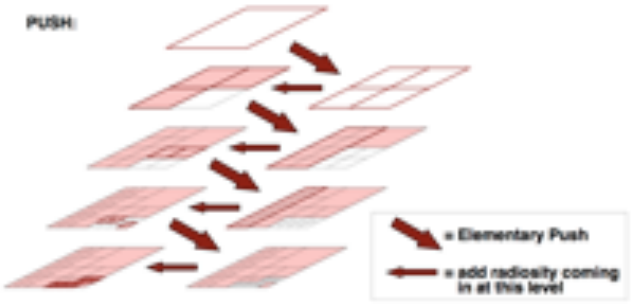
- Start with n patches (initial mesh)
- Recursively apply
  - Approximate form factors with quick estimate
  - Subdivide if FF fall below a threshold



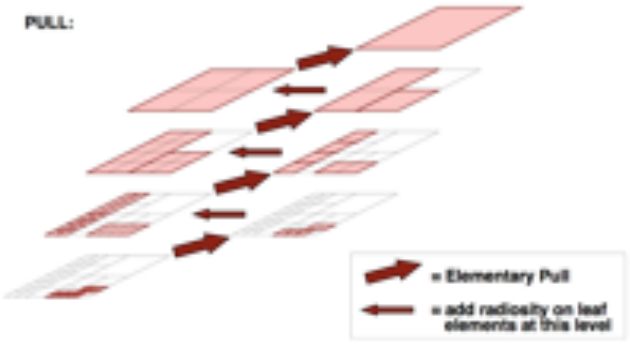
Hierarchical Radiosity



Hierarchical Radiosity Push



Hierarchical Radiosity Pull

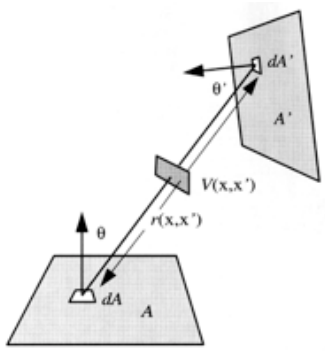


Disco Meshing vs. Hierarchical Radiosity

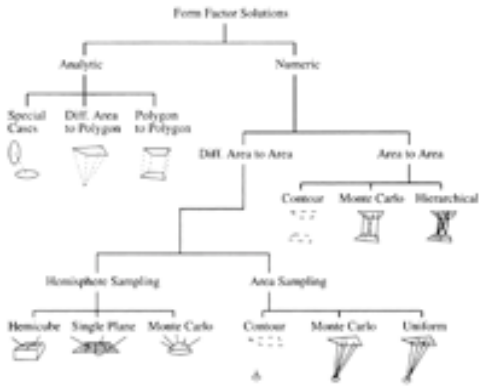
Form Factors F_{IJ}

- A geometric property
- Encode the energy transfer between patches
- For a scene with n patches, this amounts to a matrix with n x n elements
- Calculation time-intensive, but has to be performed only once per geometric setup
- Independent of illumination

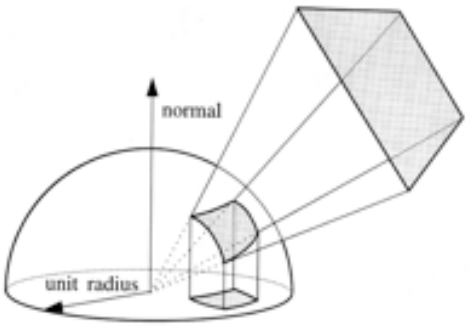
Form Factor Geometry



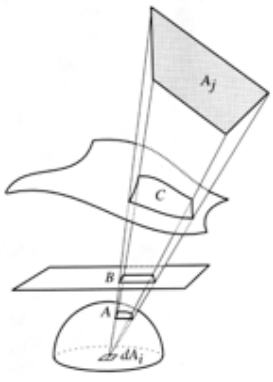
Form Factor Calculations



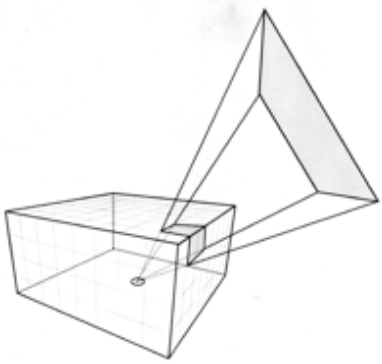
Nusselts Analogon



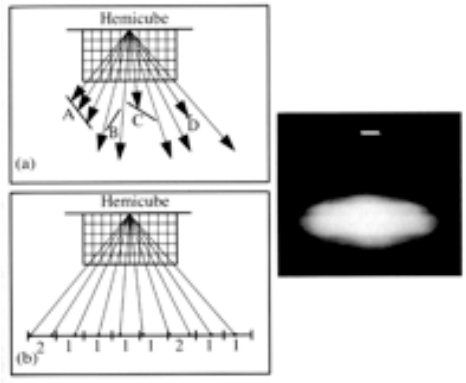
Form Factor Relationship



Hemicube



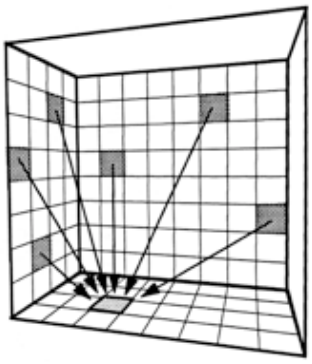
Hemicube Problem: Aliasing



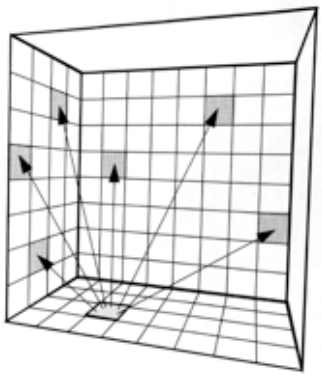
Solving the Radiosity Matrix

- Various iterative methods exist, e.g.:
  - Jacobi iteration
  - Gauss-Seidel iteration
  - Southwell relaxation
- These correspond to different light propagation strategies:
  - shooting vs.
  - gathering

Gathering Step



Shooting Step



Rendering the Solution

- The mesh can directly be used for walkthroughs or ray tracing
- The radiosity information can be used in a multi-pass renderer
- The mesh has to be interpolated for display purposes (potential problems with disco meshing and other extreme tessellations)

Progressive Refinement

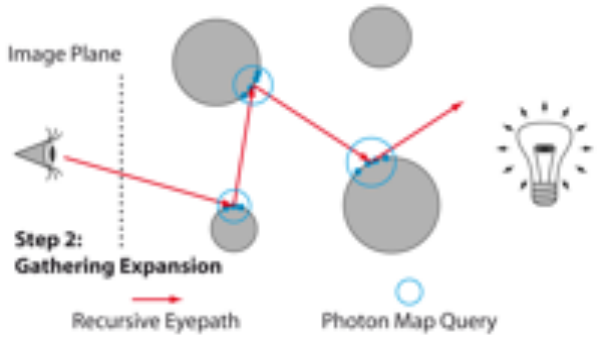
Limitations & Scope

- Only polygonal scenes
- Only diffuse materials
- Extensions for mirrors possible
- Newer, stochastic methods are state of research
- Form factor Radiosity is state of the art in commercial products

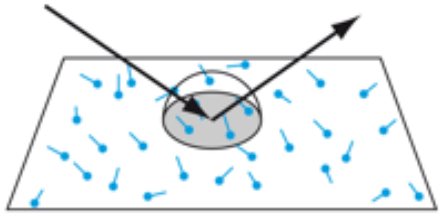
Photon Density Estimation

- Physically plausible simulation of light transport
- Photon paths are traced through the scene and their interaction with surfaces is recorded
- Different storage structures:
  - Photon maps
  - Lightmaps
- Normally used in a multipass renderer

Photon Tracing



Illumination Reconstruction

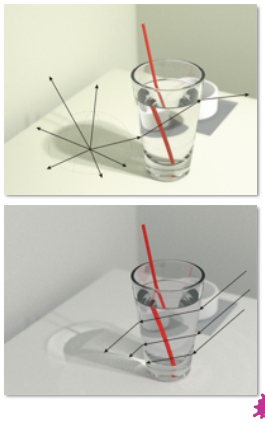


Photon Maps

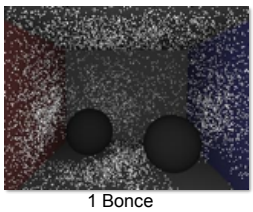
- Highly efficient for caustics
- Always used as a stage in two- or multipass renderers
- On absorption, photons are stored in kD-Trees and used for various purposes:
  - Caustic Photon Map
  - Global Photon Map
  - Shadow Photon Map

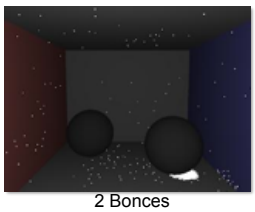
Caustic Photon Map

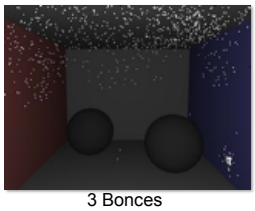
- Many photons are emitted towards specular objects
- Stored upon intersection with diffuse surfaces
- Visualized directly by using nearest n photons for illumination reconstruction

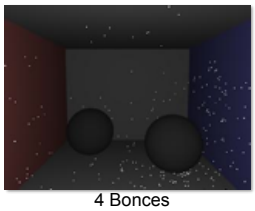


Visualisation of a Caustic Map


1 Bonce

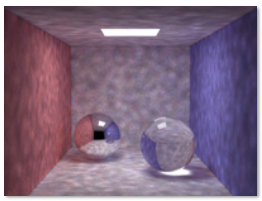

2 Bonces

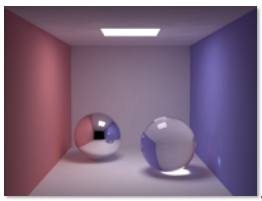

3 Bonces


4 Bonces

Global Photon Map

- Photons are emitted towards all objects in a scene
- Used as a rough approximation of light transport in a scene
- Not visualized directly



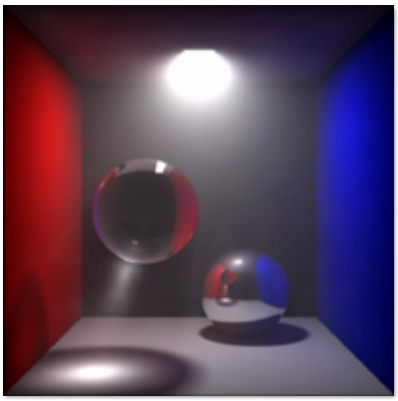


Shadow Photon Map

- Rays with origin at lightsource are traced through entire scene
- First intersection is recorded as „light“, subsequent hits as „shadow“
- Used for improvement of raytracing pass

Photon Map Example #1

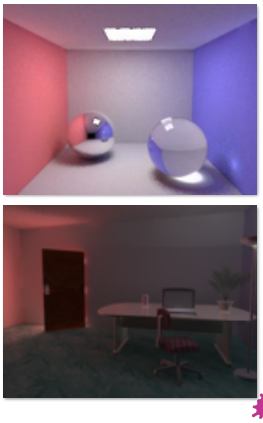
Photon Map Example #2



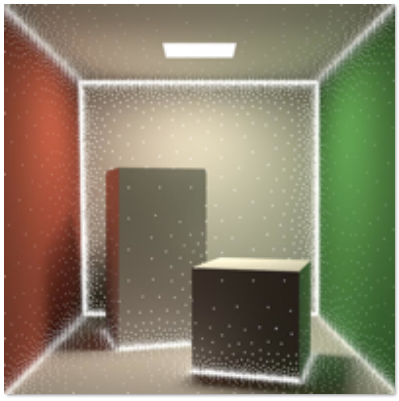
Photon Map Example #3

Photon Maps - Disadvantages

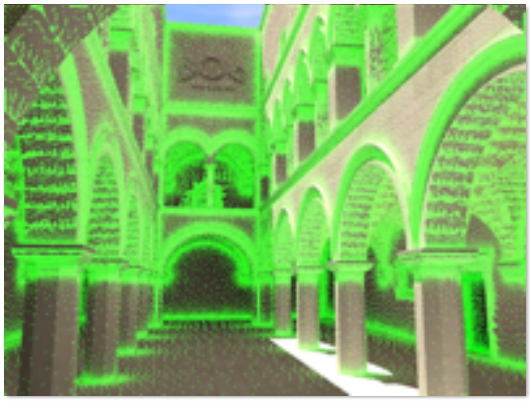
- Slow
- Memory consumption
- Illumination reconstruction depends on distance
- Biased



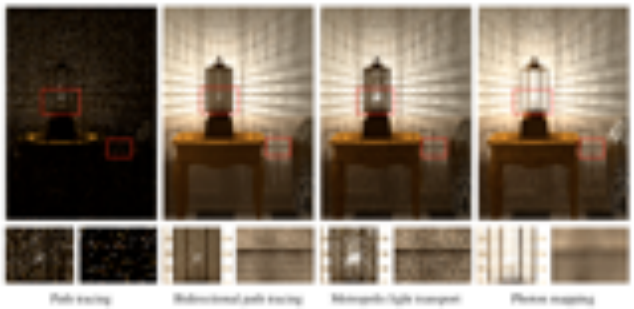
Irradiance Cache #1



Irradiance Cache #2



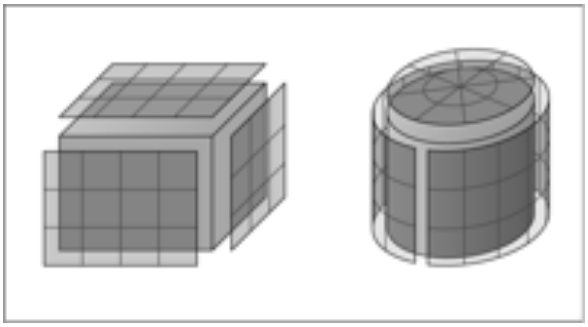
Comparison



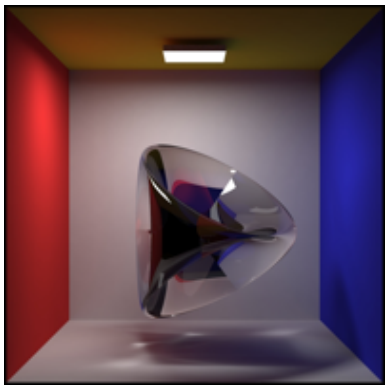
Lightmaps

- 2D “light textures” on objects
- Each texture element averages the energy of all photon hits it receives
- Higher order representations possible
- Area of all texels has to be known and has to be computed as a preprocessing step
- Interpolation over texels after tracing pass

Lightmaps



Photon Tracing: Caustics



Photon Tracing vs. Complexity

- Memory consumption: texels on all primitives are wasteful
- Preprocessing: area computation for large numbers of texels takes too long
- Execution time: far too many photons have to be cast for a stable estimate
- Impossible to attach to implicit objects, e.g. L-systems

Complex Scenes: Points of Failure

- Memory consumption explodes
- Execution time is unacceptable
  - Set-up times are too high
  - Convergence is too slow for Monte Carlo methods
- Ray-based methods converge too slowly if number of primitives is large

Overview

- Rendering equation
- BRDF
  - Empirical vs. Physically based
  - Torrance-Sparrow surface model
  - Ward reflection model
- Beyond BRDFs

Last Unit(s)

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')\, I(x',x'')\, dx''\right]$$

- Fredholm Integro-differential equation
- Completely describes light transport in a scene
- Usually impossible to solve analytically (infinite cascade)
- Most solution strategies take only certain aspects into account

Solving the RE: Scanline Rendering

$$I(x,x') = g(x,x')\left[\varepsilon(x,x') + \int_S \rho(x,x',x'')\, I(x',x'')\, dx''\right]$$

- **Geometry**: partially (polygonisation!)
- **Emission**: yes (if implemented)
- **Integral**: no
- **Surfaces**: rudimentary
- **Recursion**: no

OpenGL Scanline Rendering

Solving the RE: Raytracing

$$I(x,x') = g(x,x') \left[\varepsilon(x,x') + \int_S \rho(x,x',x'') \, I(x',x'') \, dx'' \right]$$

- **Geometry**: yes, accurately
- **Emission**: yes
- **Integral**: no
- **Surfaces**: partially / rudimentary
- **Recursion**: partially (for mirrors and transparent surfaces)

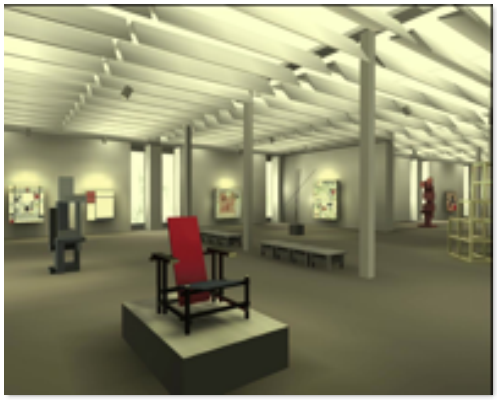
Raytraced Scene

Solving the RE: Radiosity

$$I(x,x') = g(x,x') \left[\varepsilon(x,x') + \int_S \rho(x,x',x'') \, I(x',x'') \, dx'' \right]$$

- **Geometry**: yes (sort of – polygonisation!)
- **Emission**: yes
- **Integral**: partially (only diffuse surfaces)
- **Surfaces**: same as integral
- **Recursion**: yes

Radiosity



Solving the RE: Photon Tracing

$$I(x,x') = g(x,x') \left[\varepsilon(x,x') + \int_S \rho(x,x',x'') \, I(x',x'') \, dx'' \right]$$

- **Geometry**: yes
- **Emission**: yes
- **Integral**: partially (from lightsource – yes, direct viewing – ?)
- **Surfaces**: same as integral
- **Recursion**: yes

Photon Tracing

Solving the RE: Path Tracing

$$I(x,x') = g(x,x') \left[\varepsilon(x,x') + \int_S \rho(x,x',x'') \, I(x',x'') \, dx'' \right]$$

- **Geometry**: yes
- **Emission**: yes
- **Integral**: yes
- **Surfaces**: yes
- **Recursion**: yes

Path Tracing (Metropolis)

Rendering VO Unit

The End

Thank you for your attention!