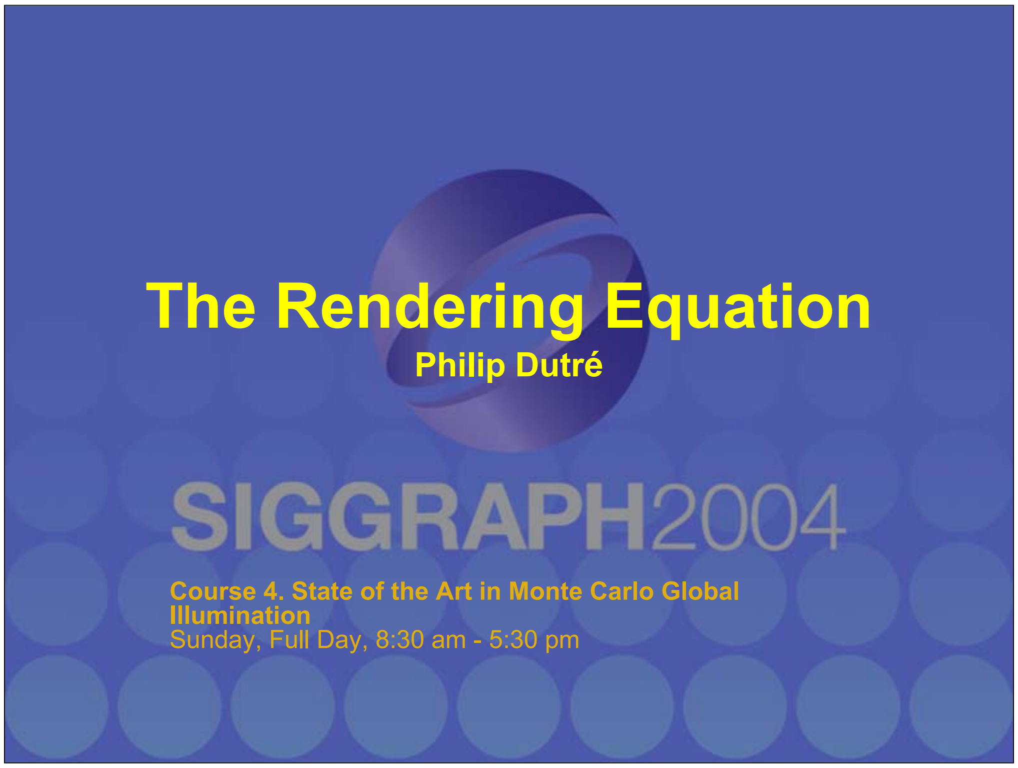


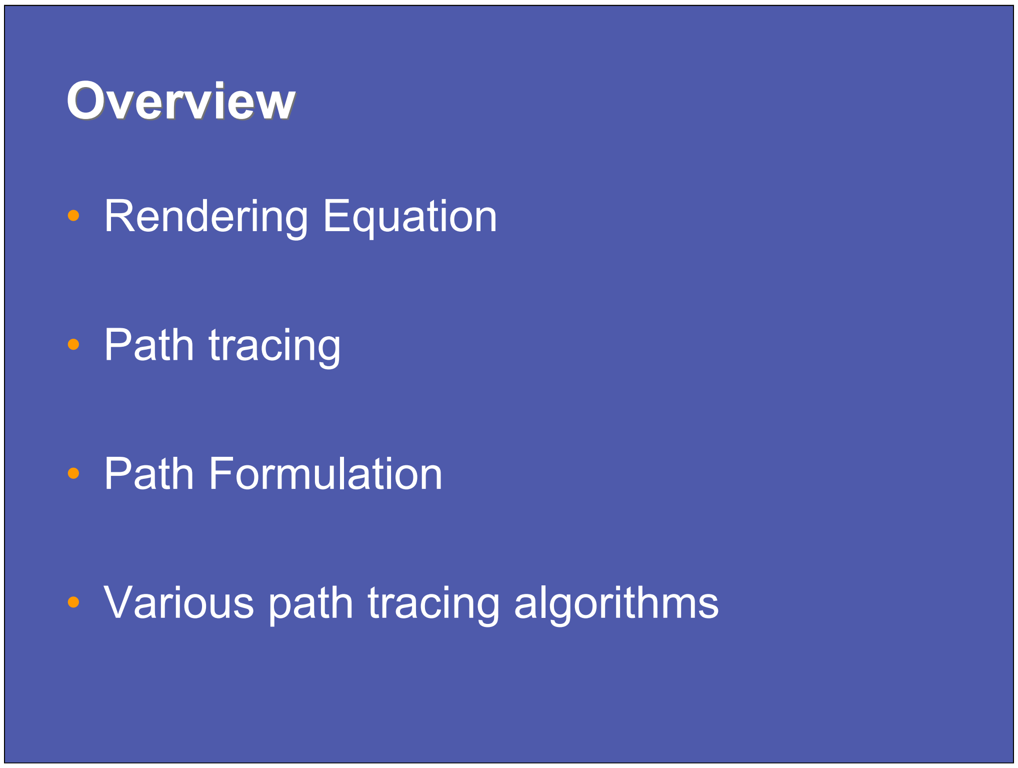
The Rendering Equation

Philip Dutré

SIGGRAPH2004

Course 4. State of the Art in Monte Carlo Global Illumination

Sunday, Full Day, 8:30 am - 5:30 pm

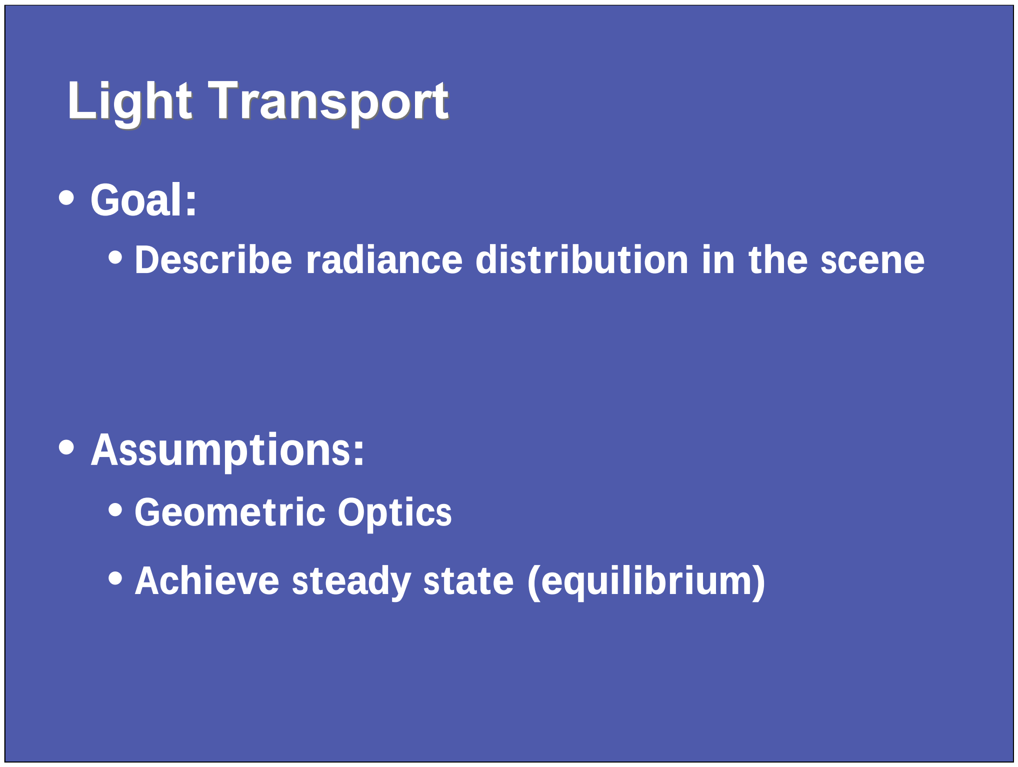


Overview

- Rendering Equation
- Path tracing
- Path Formulation
- Various path tracing algorithms

This part of the course will cover in detail the rendering equation and how to reason about it.

We will start by repeating a few concepts seen before, namely the definition of the BRDF.

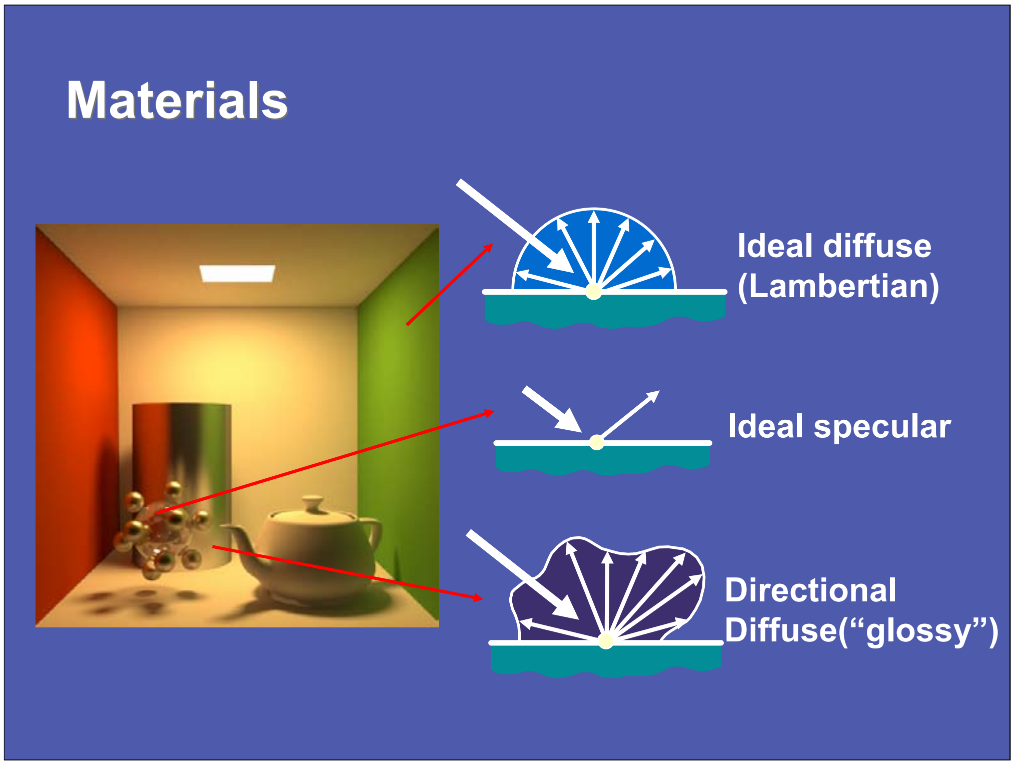


Light Transport

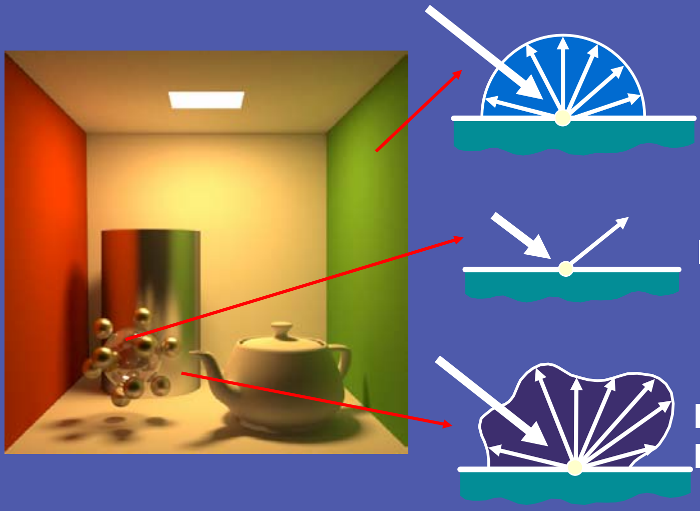
- Goal:
  - Describe radiance distribution in the scene
- Assumptions:
  - Geometric Optics
  - Achieve steady state (equilibrium)

Rendering Equation (RE)

- RE describes energy transport in a scene
- Input:
  - **light sources**
  - **geometry of surfaces**
  - **reflectance characteristics of surfaces**
- Output: value of radiance at all surface points and in all directions

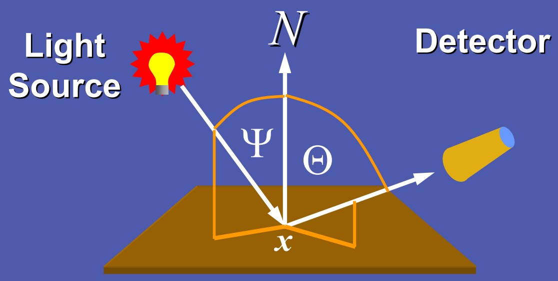


Materials


Ideal diffuse
(Lambertian)
Ideal specular
Directional
Diffuse("glossy")

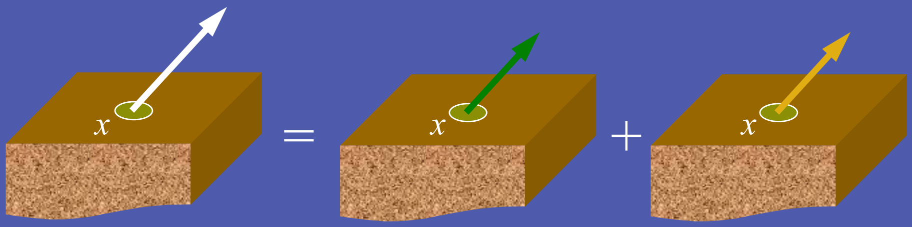
BRDF

- Bidirectional Reflectance Distribution Function

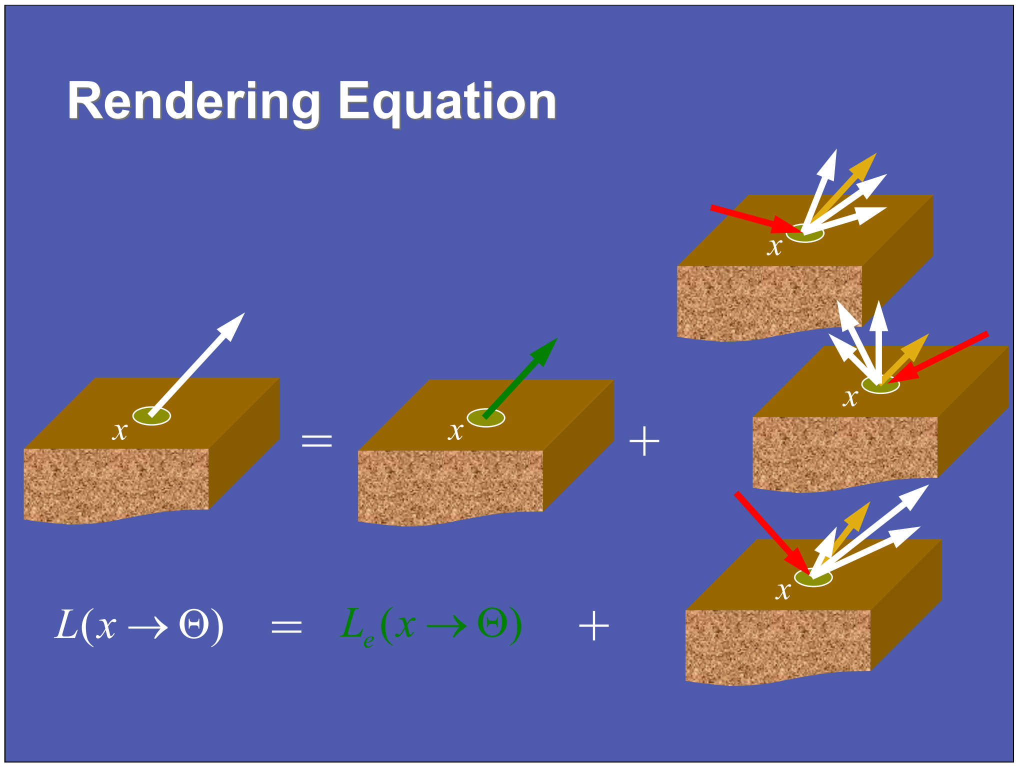


$$f_r(x, \Psi \rightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)} = \frac{dL(x \rightarrow \Theta)}{L(x \leftarrow \Psi)\cos(N_x, \Psi)\, d\omega_\Psi}$$

Rendering Equation



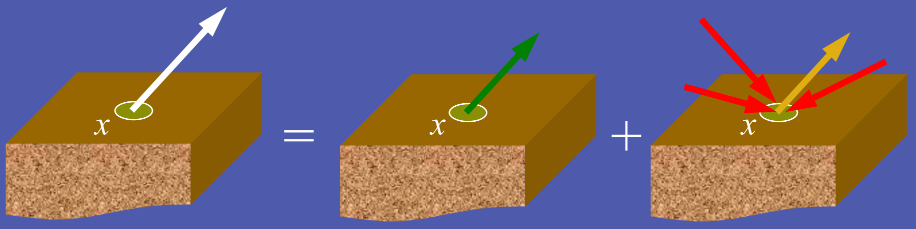
$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + L_r(x \rightarrow \Theta)$$



Rendering Equation

$$L(x \to \Theta) = L_e(x \to \Theta) +$$

Rendering Equation



$$L(x \rightarrow \Theta) \quad = \quad L_e(x \rightarrow \Theta) \quad + \int_{hemisphere} L(x \leftarrow \Psi) \cdots$$

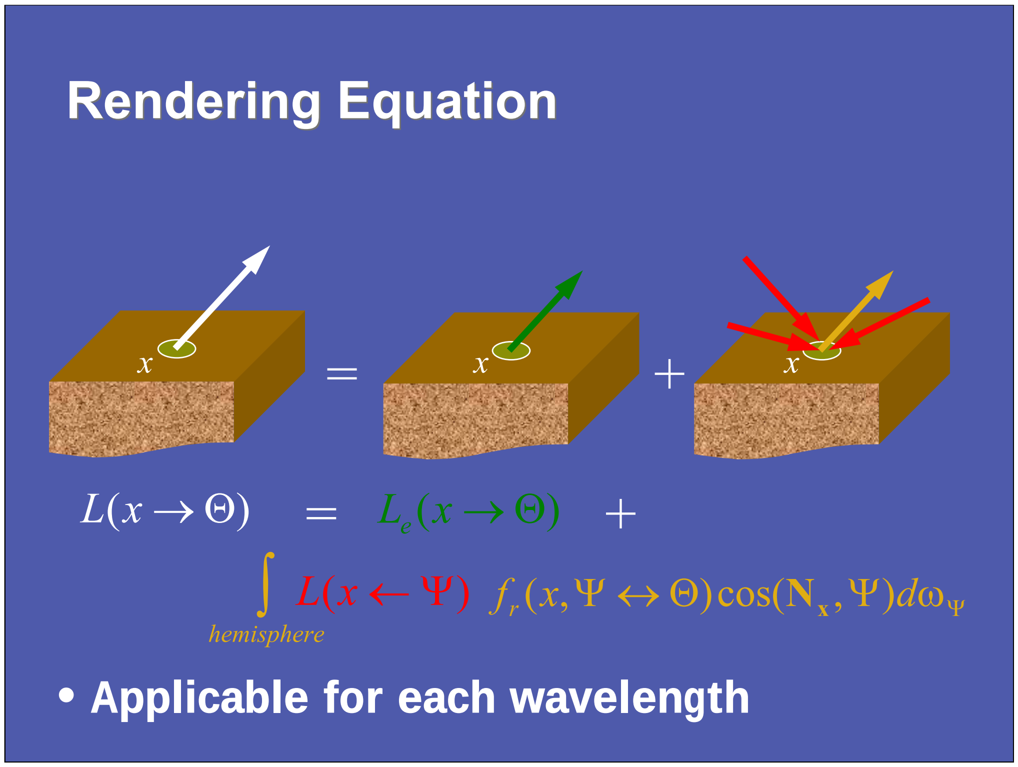
Rendering Equation

$$f_r(x, \Psi \leftrightarrow \Theta) = \frac{dL(x \rightarrow \Theta)}{dE(x \leftarrow \Psi)}$$

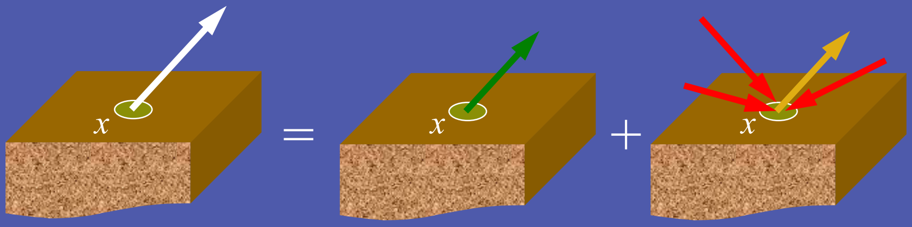
$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) dE(x \leftarrow \Psi)$$

$$dL(x \rightarrow \Theta) = f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$

$$L_r(x \rightarrow \Theta) = \int_{hemisphere} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) \cos(N_x, \Psi) d\omega_\Psi$$



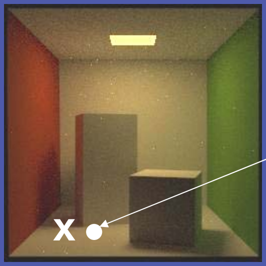
Rendering Equation



$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L(x \leftarrow \Psi)\, f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N_x}, \Psi)\, d\omega_\Psi$$

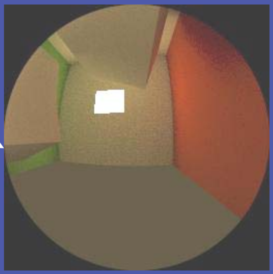
- Applicable for each wavelength

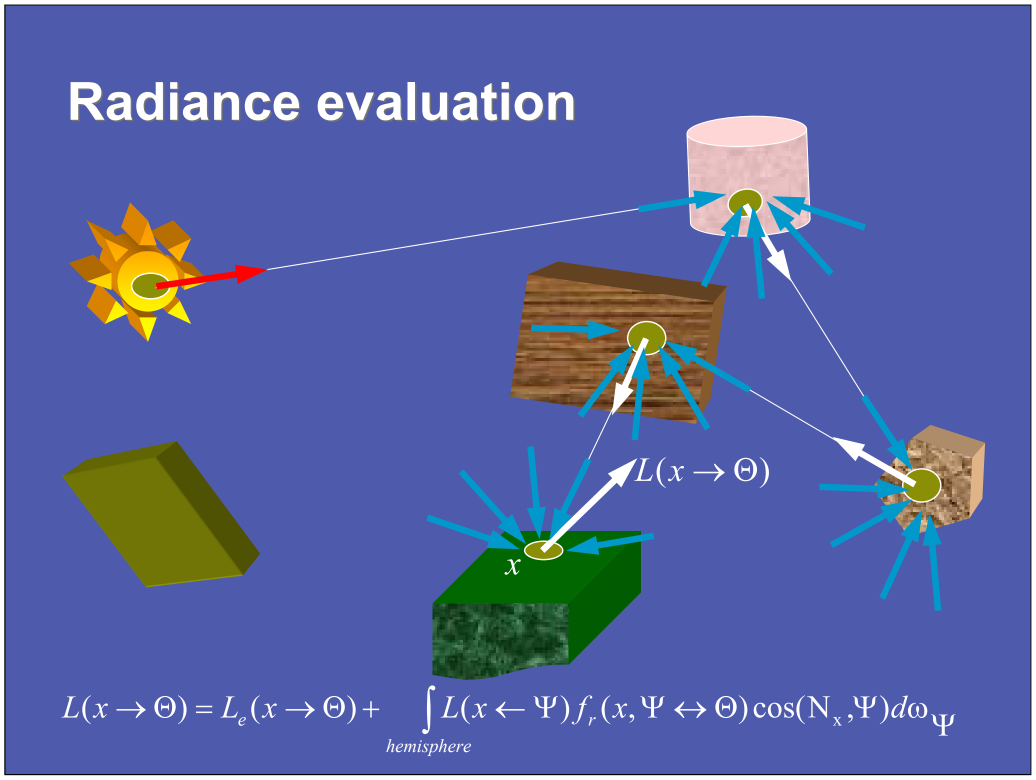
Rendering Equation



$$\underline{L(x \rightarrow \Theta)} = L_e(x \rightarrow \Theta) + \int_{hemisphere} \underline{L(x \leftarrow \Psi)} f_r(x, \Psi \leftrightarrow \Theta) \cos(\mathbf{N_x}, \Psi) d\omega_\Psi$$

incoming radiance





Radiance evaluation

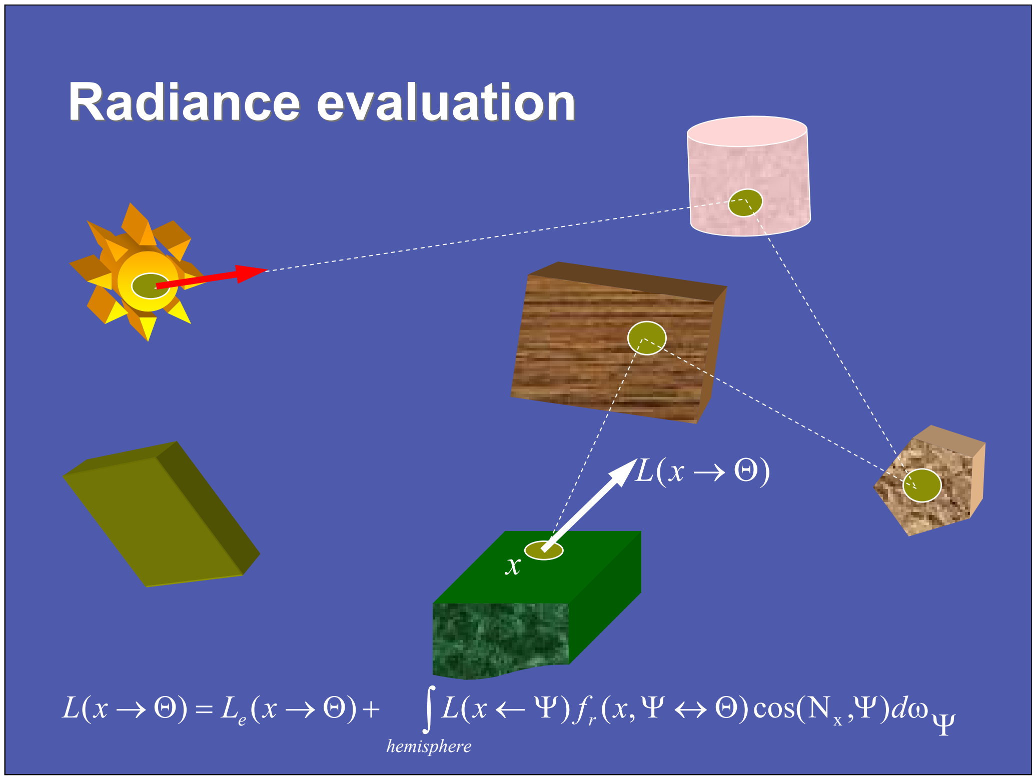
$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(N_x, \Psi) d\omega_\Psi$$

This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.



Radiance evaluation

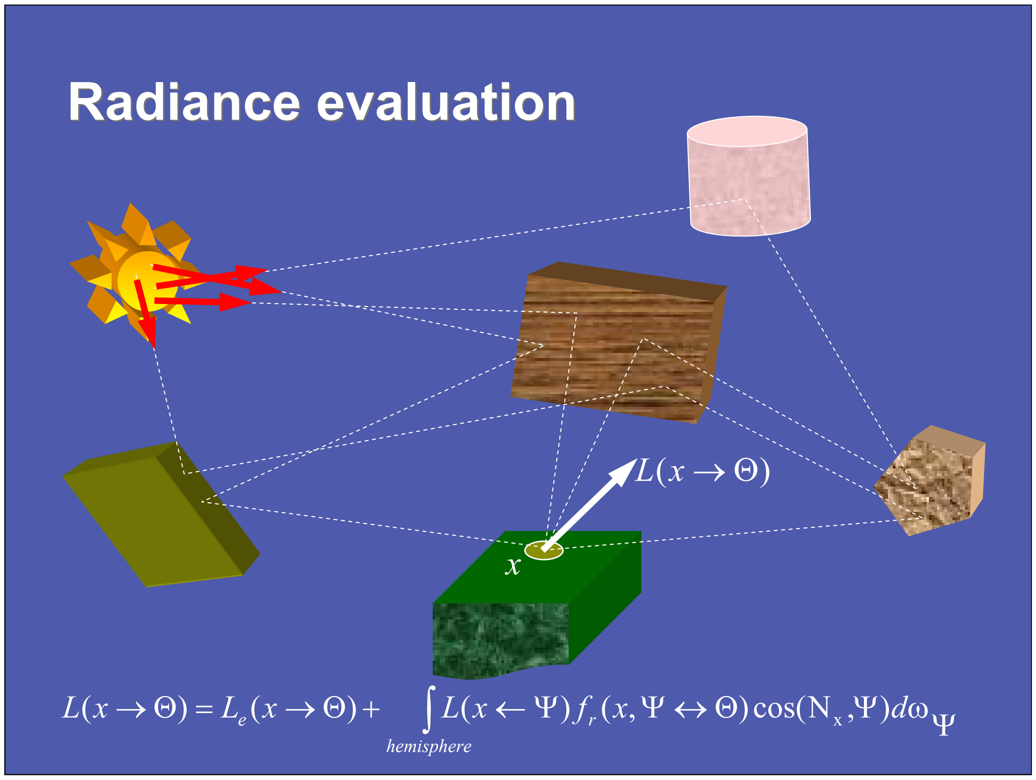
$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(N_x, \Psi) d\omega_\Psi$$

This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.



Radiance evaluation

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) \cos(N_x, \Psi) d\omega_\Psi$$

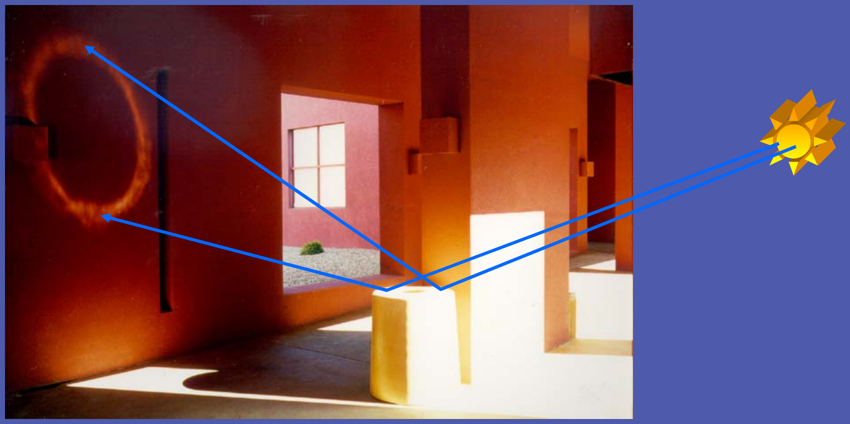
This is an illustration of the recursive nature of the rendering equation.

All radiance values incident at surface point x are themselves outgoing radiance values. We have to trace back the path they arrive from. This means tracing a ray from x in the direction of the incoming radiance. This results in some surface point, and the incoming radiance along the original direction now simply equals the outgoing radiance at this new point.

The problem is then stated recursively, since this new radiance value is also described exactly by the rendering equation.

This process will continue until the paths are traced back to the light source. Then we can pick up the self-emitted radiance, take into account all possible cosine factors and possible BRDF values along the path, perform the necessary integration at each surface point, to finally arrive at the original radiance value we are interested in.

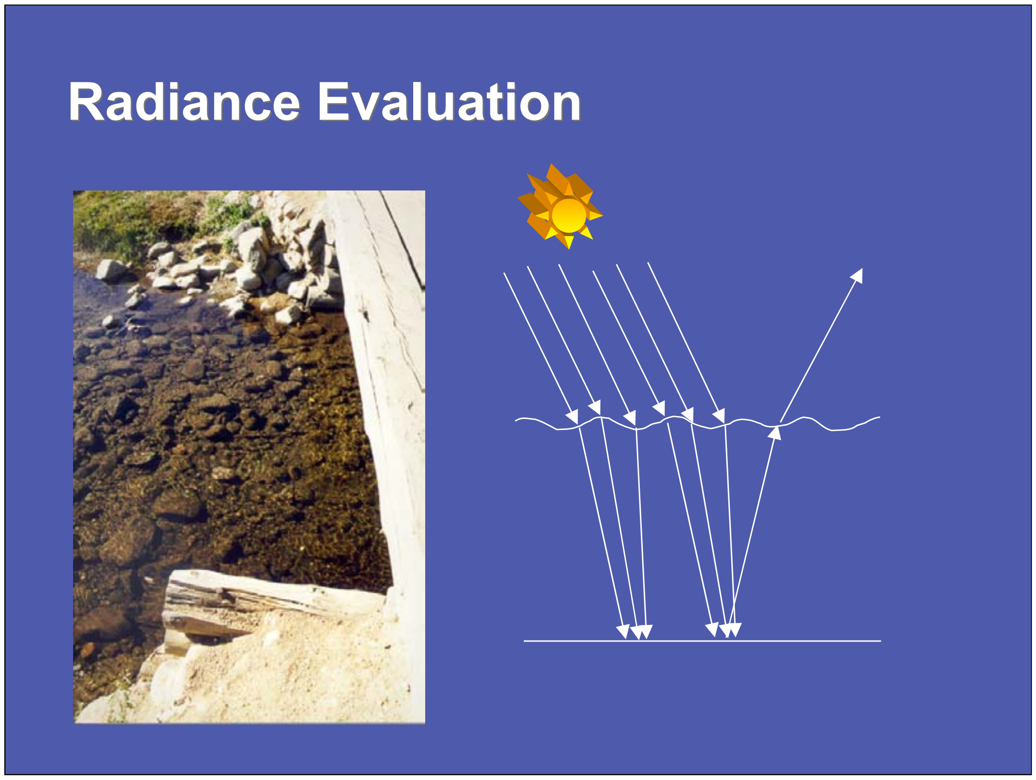
Radiance Evaluation



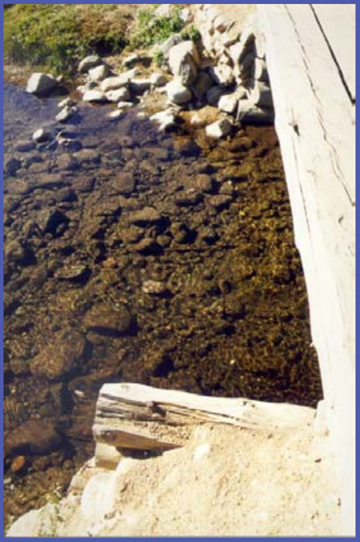
Reconstructing all possible paths between the light sources and the point for which one wants to compute a radiance value is the core business of all global illumination algorithms.

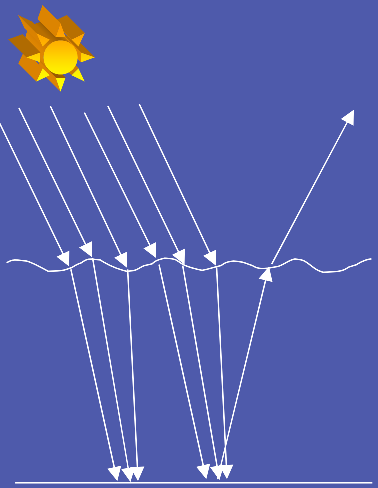
This photograph was taken on a sunny day in New Mexico. It is shown here just to illustrate some of the unexpected light paths one might have to reconstruct when computing global illumination solutions.

The circular figure on the left wall is the reflection of the lid on the trash can. The corresponding light paths (traced from the sun), hit the lid, then hit the wall, and finally end up in our eye. For a virtual scene, these same light paths need to be followed to reconstruct the reflection.



Radiance Evaluation





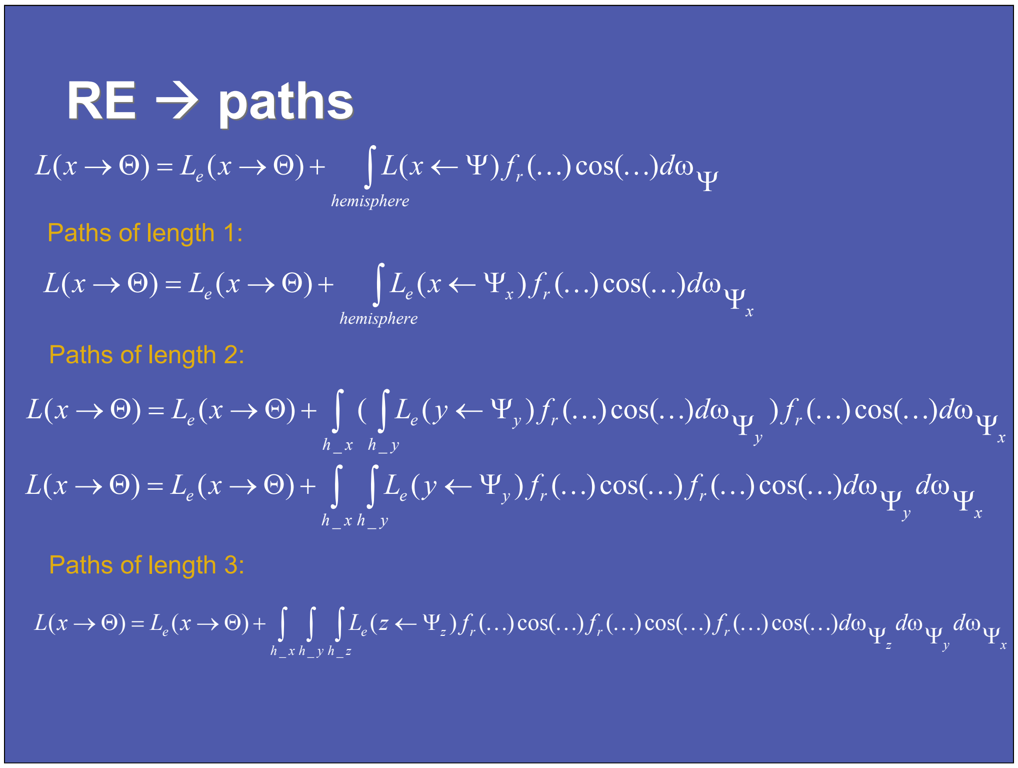
This photograph shows a similar effect.

We see shimmering waves on the bottom of the river (a similar effect is noticable in swimming pools). Light rays from the sun hit the transparent and wavy surface of the water, then are reflected on the bottom of the river, are refracted again by the water, the they hit our eye.

The complex pattern of light rays hitting the bottom, together with the changing nature of the surface of the water, causes these shimmering waves.

This effect is known as a caustic: light rays are reflected or refracted in different patterns and form images of the light source: the circular figure in the previous photograph, or the shimmering waves in this one.

(www.renderpark.be)



RE → paths

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L(x \leftarrow \Psi) f_r(\ldots) \cos(\ldots) d\omega_{\Psi}$$

Paths of length 1:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{hemisphere} L_e(x \leftarrow \Psi_x) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_x}$$

Paths of length 2:

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \left(\int_{h\_y} L_e(y \leftarrow \Psi_y) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_y} \right) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_x}$$

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \int_{h\_y} L_e(y \leftarrow \Psi_y) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_y} \, d\omega_{\Psi_x}$$

Paths of length 3:

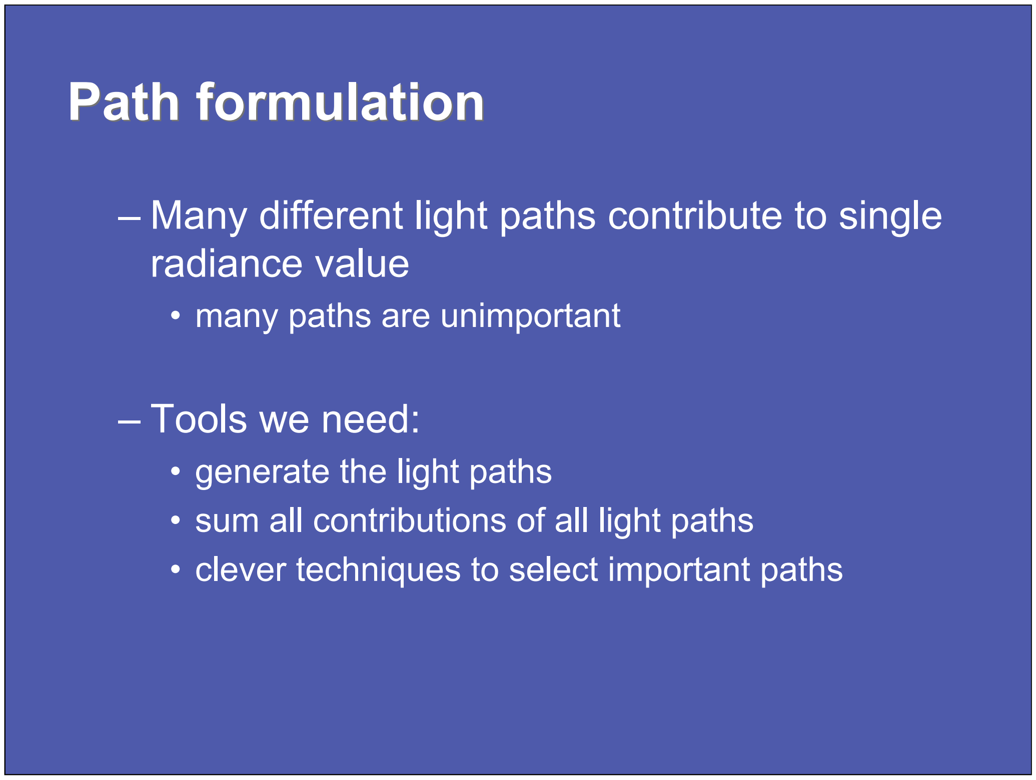
$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{h\_x} \int_{h\_y} \int_{h\_z} L_e(z \leftarrow \Psi_z) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) f_r(\ldots) \cos(\ldots) d\omega_{\Psi_z} \, d\omega_{\Psi_y} \, d\omega_{\Psi_x}$$

Path formulation

$$L(x \to \Theta) = \int_{paths} L_e(y \to \Psi)\, T((y,\Psi) \Rightarrow (x,\Theta))\, d\mu_{path}$$

$$T((y,\Psi) \Rightarrow (x,\Theta))$$

- Transfer function
  - All possible paths
  - … of any length
  - … any material reflections
  - … no light transport neglected



Path formulation

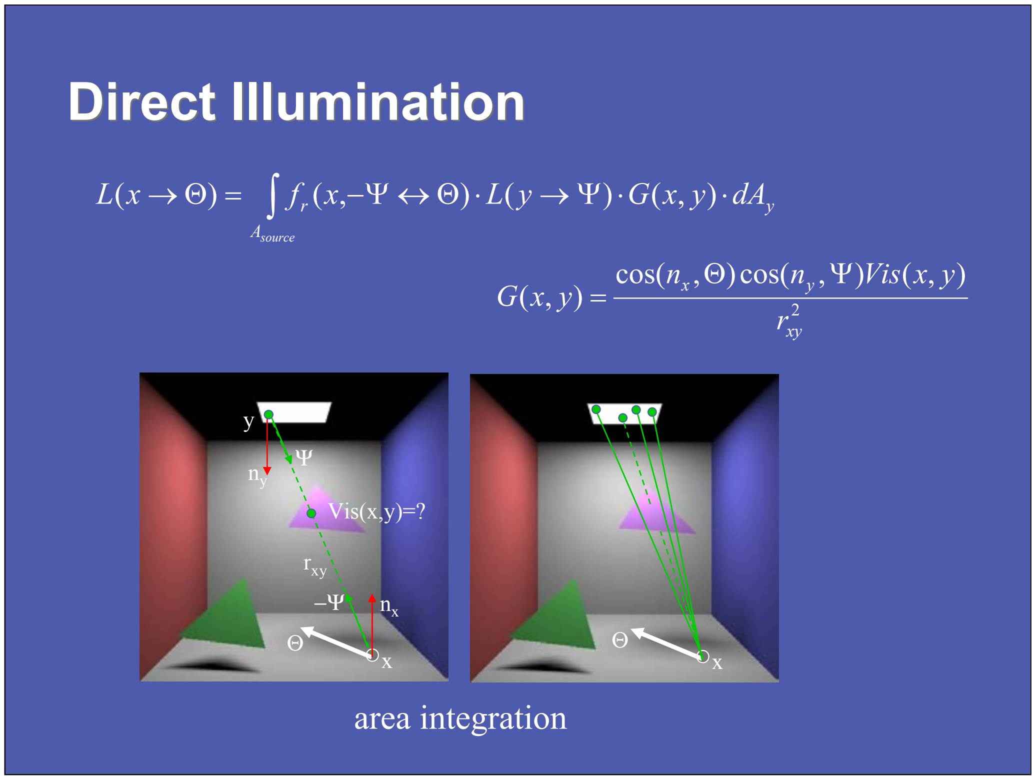
- Many different light paths contribute to single radiance value
  - many paths are unimportant
- Tools we need:
  - generate the light paths
  - sum all contributions of all light paths
  - clever techniques to select important paths

So, many different light paths, all originating at the light sources, will contribute to the value of the radiance at a single surface point.

Many of these light paths will be unimportant. Imagine a light switched on on the 1st floor of a building. You can imagine that some photons will travel all the way up to the 4th floor, but it is very unlikely that this will contribute significantly to the illumination on the 4th floor. However, we cannot exclude these light paths from consideration, since it might happen that the contribution is significant after all.

So, one of the mechanisms that a good global illumination algorithm needs is how to select the important paths from amongst many different possibilities, or at least how to try to put more computational effort into the ones that are likely to give the best contributions.

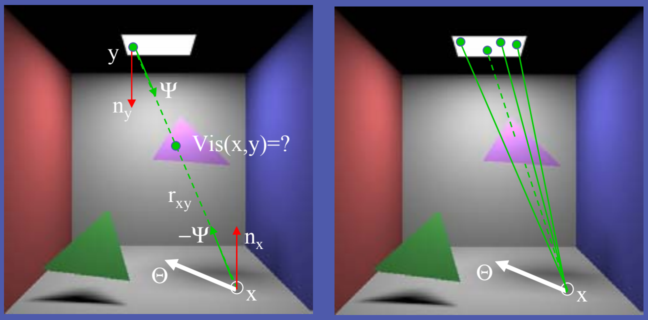
This is of course a chicken and egg problem. If we would know what the importance of each path was, we would have solved the global illumination problem. So the best we can do is to make clever guesses.



Direct Illumination

$$L(x \rightarrow \Theta) = \int_{A_{source}} f_r(x, -\Psi \leftrightarrow \Theta) \cdot L(y \rightarrow \Psi) \cdot G(x,y) \cdot dA_y$$

$$G(x,y) = \frac{\cos(n_x, \Theta)\cos(n_y, \Psi) Vis(x,y)}{r_{xy}^2}$$

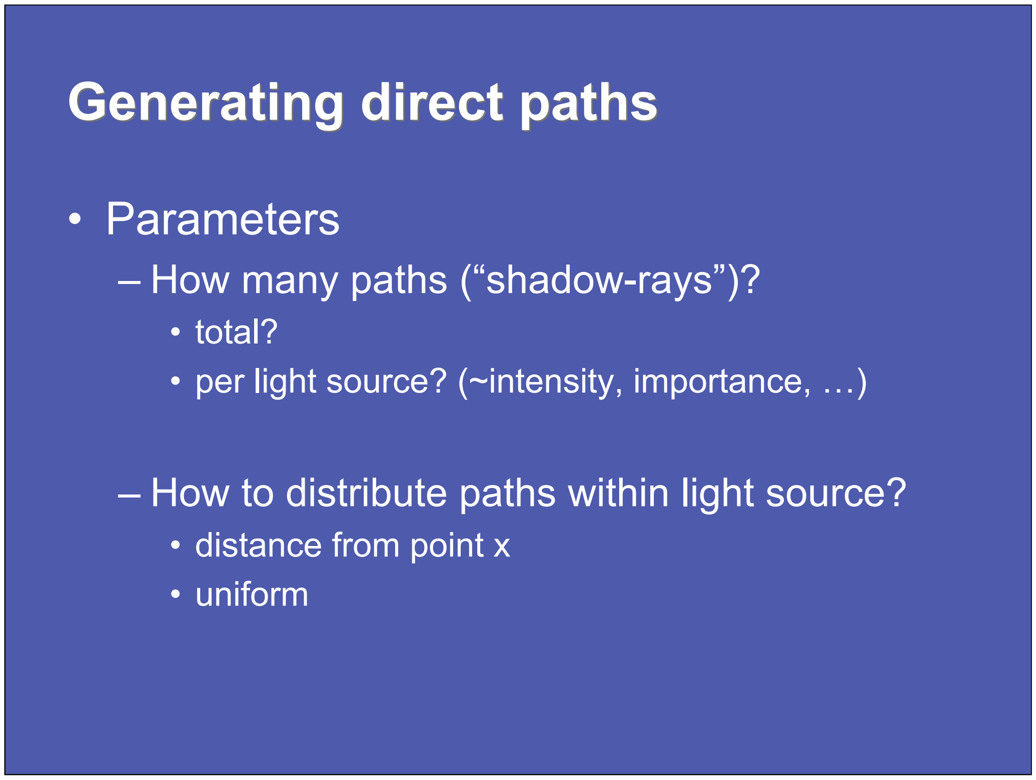


area integration

One can do better by reformulating the rendering equation for direct illumination. Instead of integrating over a hemisphere, we will integrate over the surface area of the light source. This is valid, since we are only interested in the contribution due to the light source.

To transform the hemispherical coordinates to area coordinates over the hemisphere, we need to transform a differential solid angle to a differential surface. This introduces an extra cosine term and an inverse distance squared factor.

Additionally, the visibility factor, which was hidden in the hemispherical formulation since we ‘traced’ the ray to the closest intersection point, now needs to be mentioned explicitly.



Generating direct paths

- Parameters
  - How many paths (“shadow-rays”)?
    - total?
    - per light source? (~intensity, importance, …)
  - How to distribute paths within light source?
    - distance from point x
    - uniform

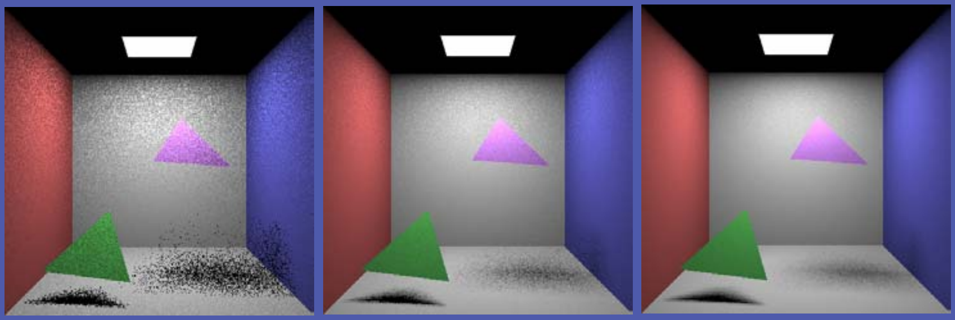
To compute the direct illumination using Monte Carlo integration, the following parameters can now be chosen:

- How many paths will be generated total for each radiance value to be computed? More paths result in a more accurate estimator, but the computational cost increases.

- How many of these paths will be send to each light source? It is intuitively obvious that one wants to send more paths to bright light sources, closer light sources, visible light sources.

- How to distribute the paths within each light source? When dealing with large light sources, points closer to the point to be shaded are more important than farther- away points.

Generating direct paths

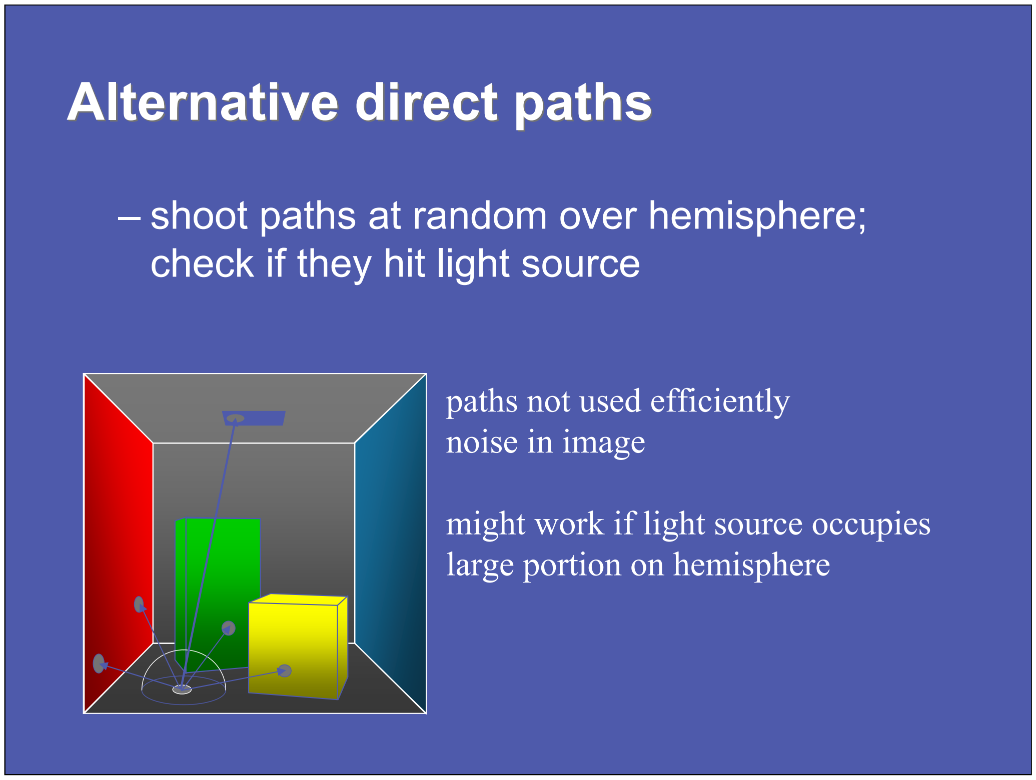


1 path / source   9 paths / source   36 paths / source

Here are a few examples of the results when generating a different number of paths per light source.

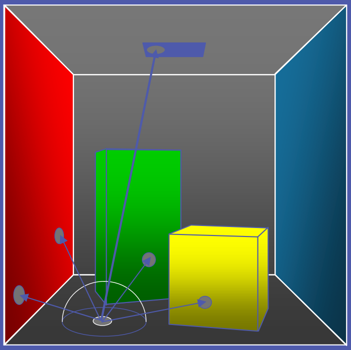
This simple scene has only one light source, and respectively 1, 9 and 36 paths are generated. The radiance values are computed more accurately in the latter case, and thus visible noise is less objectionable.

Although the first image is unbiased, its stochastic error is much higher compared to the last picture.



Alternative direct paths

– shoot paths at random over hemisphere; check if they hit light source



paths not used efficiently
noise in image

might work if light source occupies large portion on hemisphere

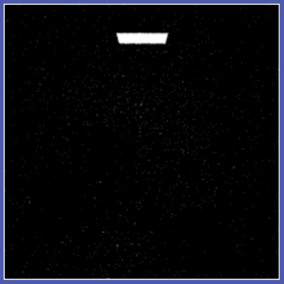
The algorithm in which the area of the light source is sampled is the most widely used way of computing direct illumination.

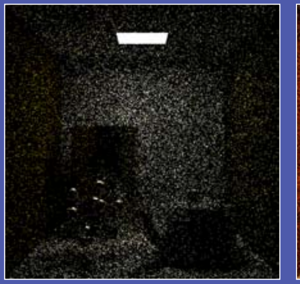
However, many more ways are possible, all based on a Monte Carlo evaluation of the rendering equation.

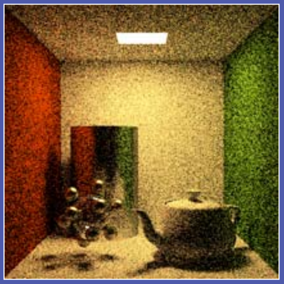
This slide shows an algorithm we have shown before: directions are sampled over the hemisphere, and they are traced to see whether they hit the light source and contribute to the radiance value we are interested in.

In this approach, many samples are wasted since their contribution is 0.

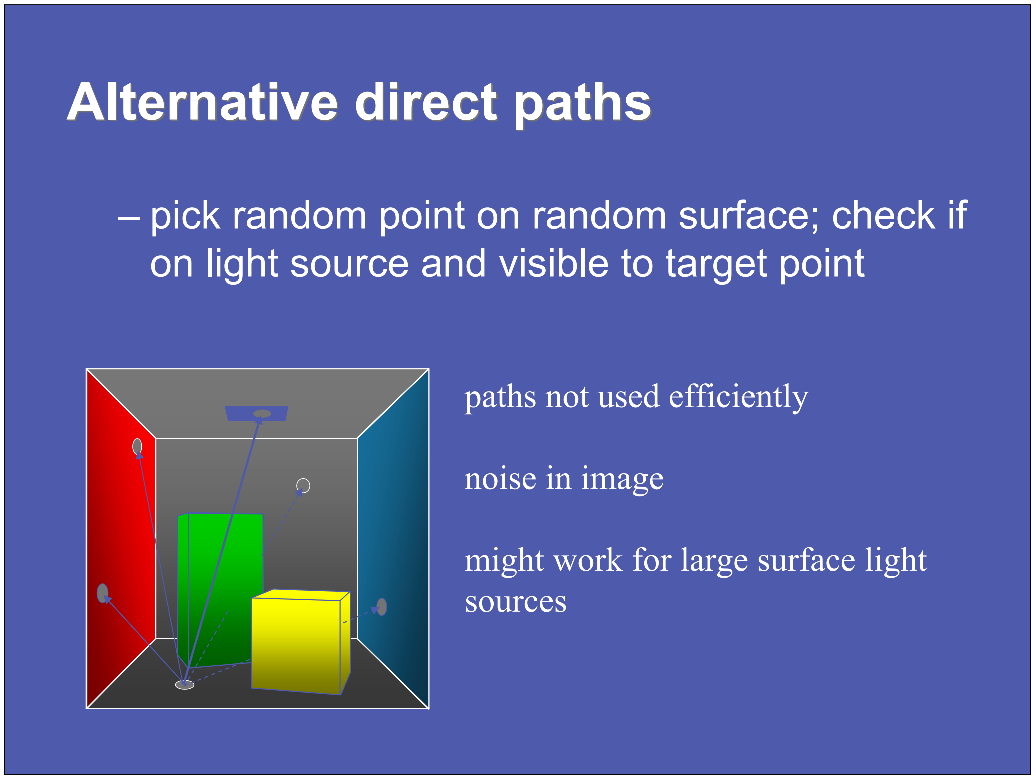
Alternative direct paths


1 paths / point


16 paths / point

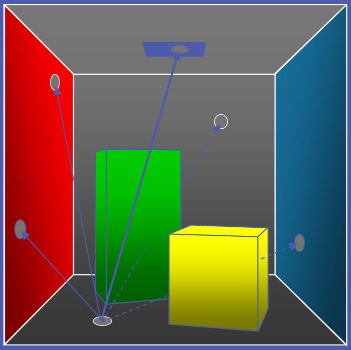

256 paths / point

These images show the result of hemispherical sampling. As can be expected, many pixels are black when using only 1 sample, since we will only have a non-black pixel if the generated direction points to the light source.



Alternative direct paths

– pick random point on random surface; check if on light source and visible to target point



paths not used efficiently

noise in image

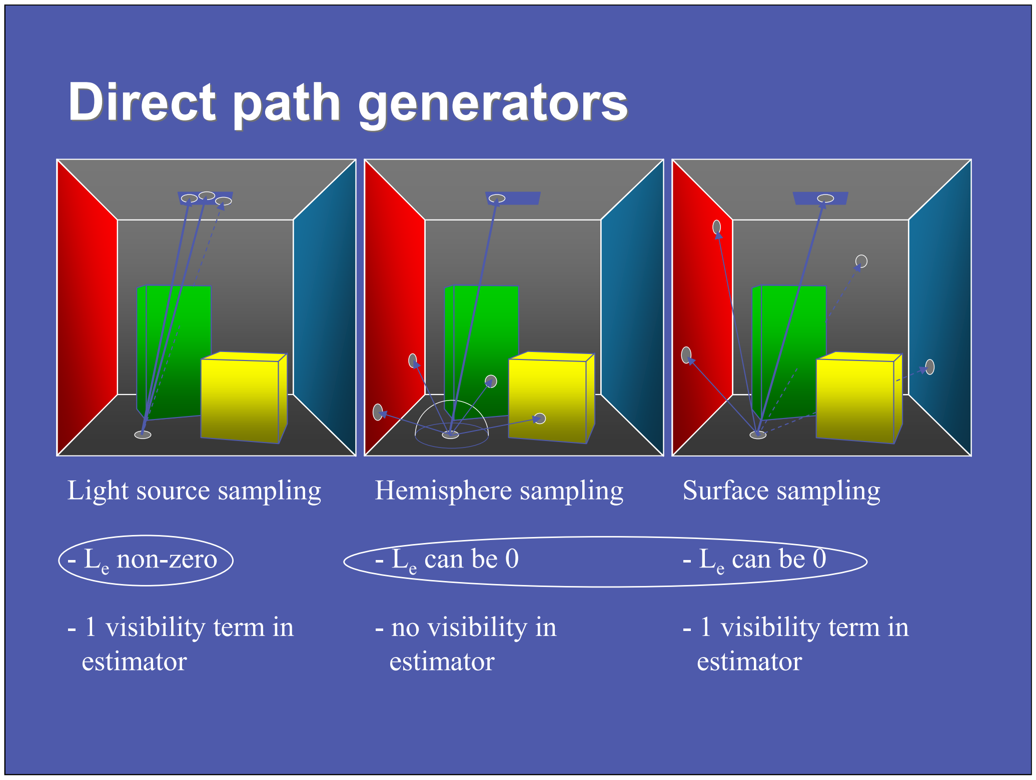
might work for large surface light sources

This is another algorithm for direct illumination:

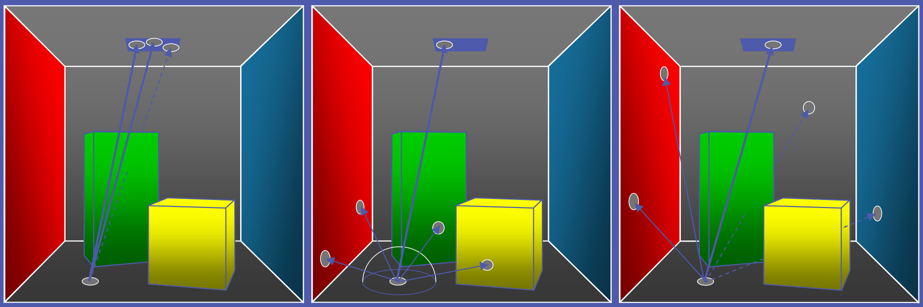
We can write the rendering equation for as an integral over ALL surfaces in the scene, not just the light sources. Of course, the direct illumination contribution of most of these surfaces will be 0.

A Monte Carlo procedure will then sample a random surface point. For each of these surface points, we need to evaluate the self-emitted radiance (only different from 0 when a light source), the visibility between the sampled point and the target point, the geometry factor, and the BRDF.

Since both the self-emitted radiance and the visibility term might produce a 0 value in many cases, many of the samples will be wasted.



Direct path generators



Light source sampling	Hemisphere sampling	Surface sampling
- L_e non-zero	- L_e can be 0	- L_e can be 0
- 1 visibility term in estimator	- no visibility in estimator	- 1 visibility term in estimator

Here we see the 3 different approaches next to each other.

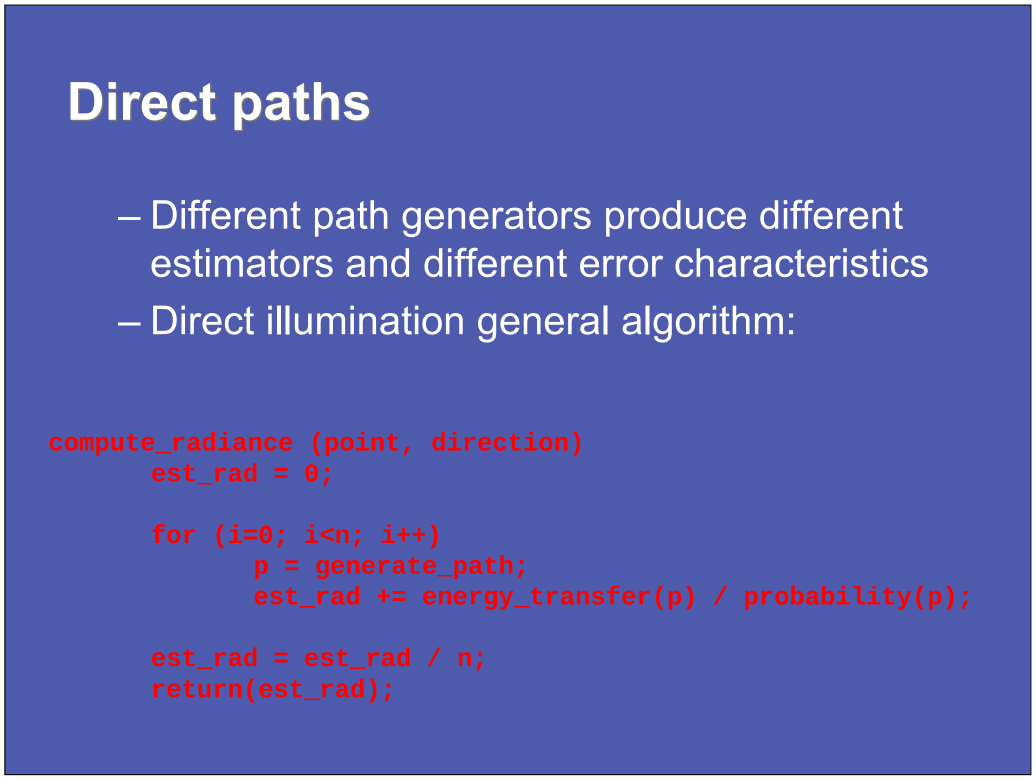
The noise resulting from each of these algorithms has different causes.

When sampling the area of the light source, most of the noise will come from failed visibility tests, and a little noise from a varying geometry factor.

When sampling the hemisphere, most noise comes from the self emitted radiance being 0 on the visible point, but the visibility itself does not cause noise. However, each sample is more costly to evaluate, since the visibility is now folded into the ray tracing procedure.

When sampling all surfaces in the scene, noise comes failed visibility checks AND self emitted radiance being 0. So this is obviously the worst case for computing direct illumination.

Although all these algorithms produce unbiased images when using enough samples, the efficiency of the algorithms is obviously different.



Direct paths

– Different path generators produce different estimators and different error characteristics

– Direct illumination general algorithm:

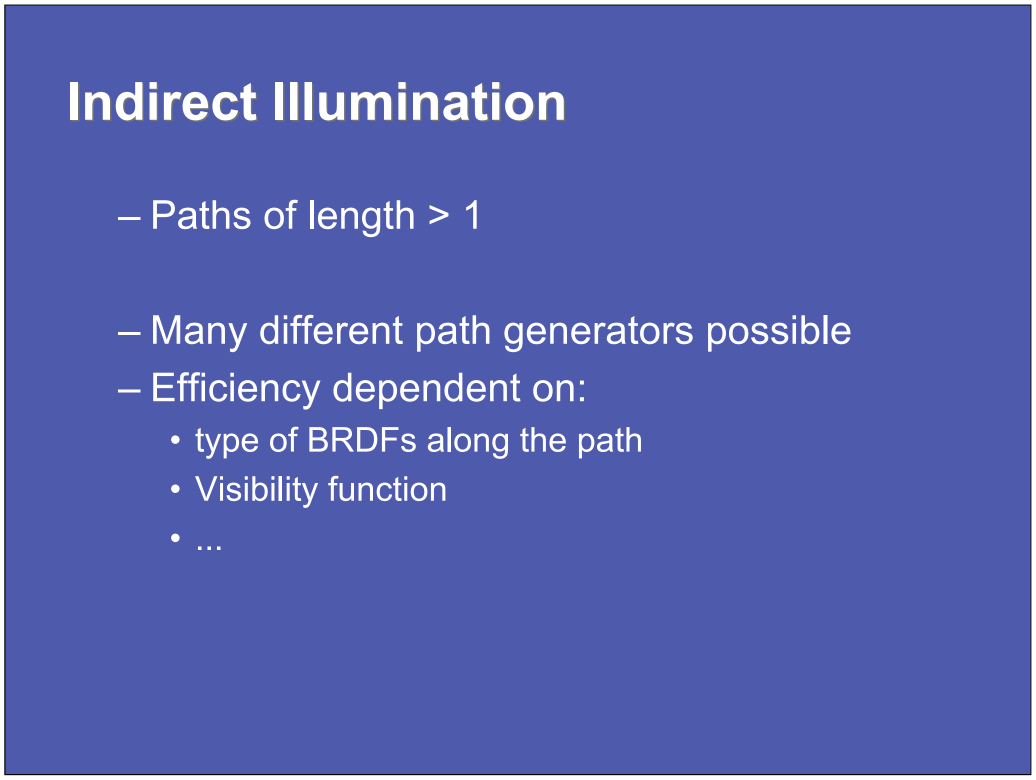
```
compute_radiance (point, direction)
        est_rad = 0;

        for (i=0; i<n; i++)
                p = generate_path;
                est_rad += energy_transfer(p) / probability(p);

        est_rad = est_rad / n;
        return(est_rad);
```

A general MC algorithm for computing direct illumination then generates a number of paths, evaluates for each path the necessary energy transfer along the path (radiance * BRDF * geometry), and computes the weighted average.

The differences in different algorithms lie in how efficient the paths are w.r.t. energy transfer.



Indirect Illumination

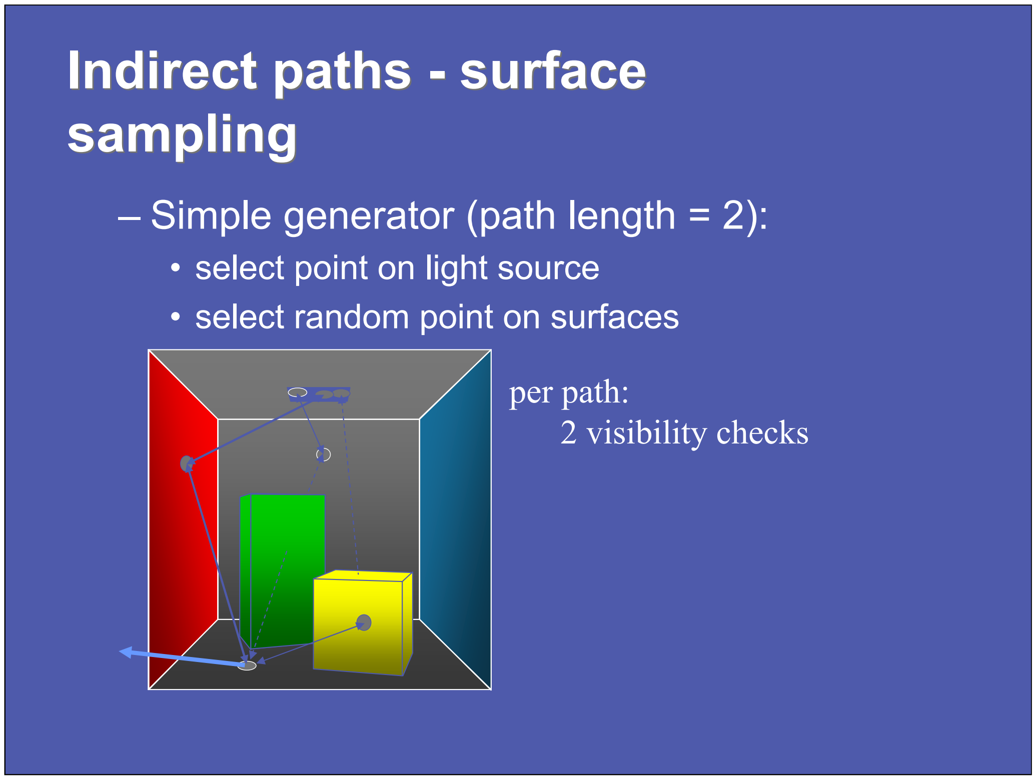
– Paths of length > 1

– Many different path generators possible
– Efficiency dependent on:
  • type of BRDFs along the path
  • Visibility function
  • ...

What about indirect illumination?

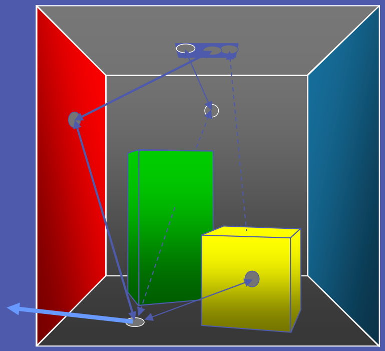
The principle remains exactly the same: we want to generate paths between a light source and a target point. The only difference is that the path will be of length greater than 1.

Again, the efficiency of the algorithm will depend on how clever the most useful paths can be generated.



Indirect paths - surface sampling

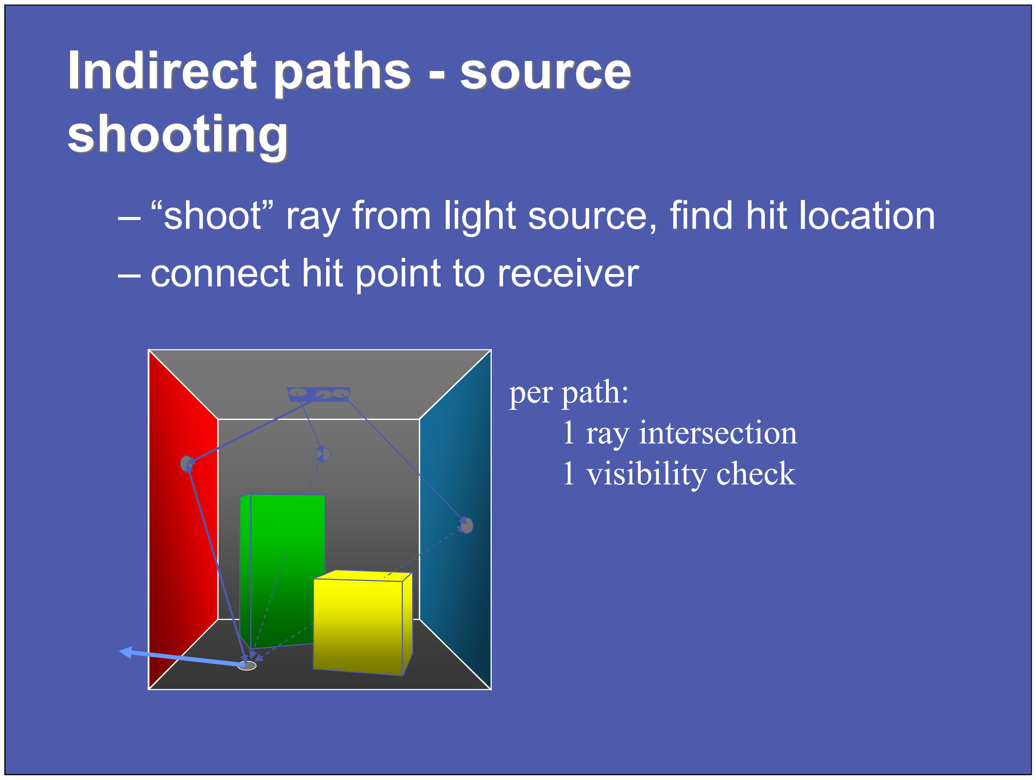
- Simple generator (path length = 2):
  - select point on light source
  - select random point on surfaces



per path:
2 visibility checks

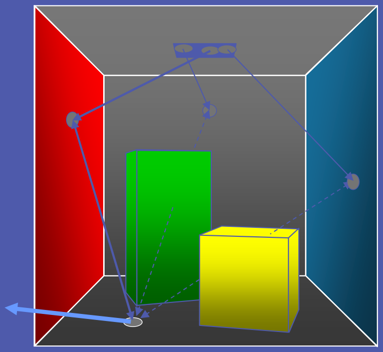
An added complexity is that we now have to deal with recursive evaluations. Although we show in these slides only the final paths between the light source and the target point, in an actual algorithm these paths will be generated recursively.

A simple algorithm involves samples all surface points in the scene. To generate paths of length 2, one can generate a random point on the surfaces, and a random point on a light source (direct illumination for the intermediate point). The necessary energy transfer is computed along the paths, and a weighted average using the correct pdf’s is computed.



Indirect paths - source shooting

- "shoot" ray from light source, find hit location
- connect hit point to receiver



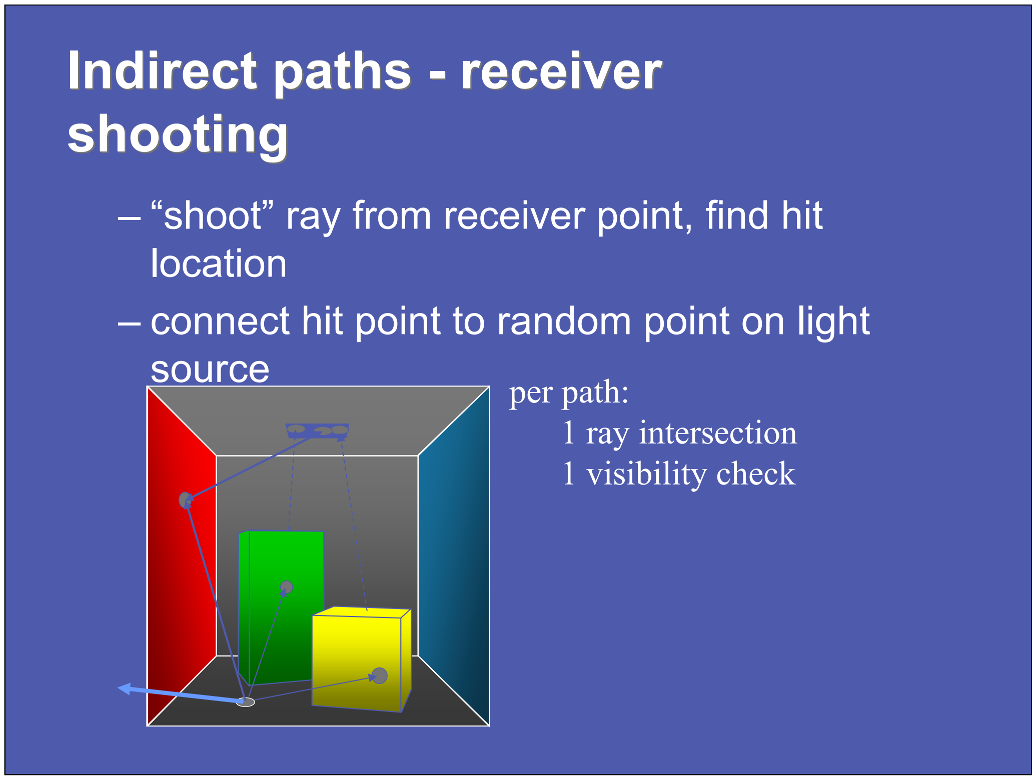
per path:

1 ray intersection

1 visibility check

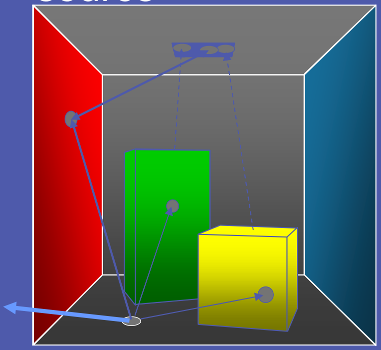
This algorithm might generate the intermediate point in a slightly different way: a random direction is sampled over the hemisphere around a random point on the light source, this ray is traced in the environment, and the closest intersection point found.

Then this visible point is connected to the target point.



Indirect paths - receiver shooting

- "shoot" ray from receiver point, find hit location
- connect hit point to random point on light source

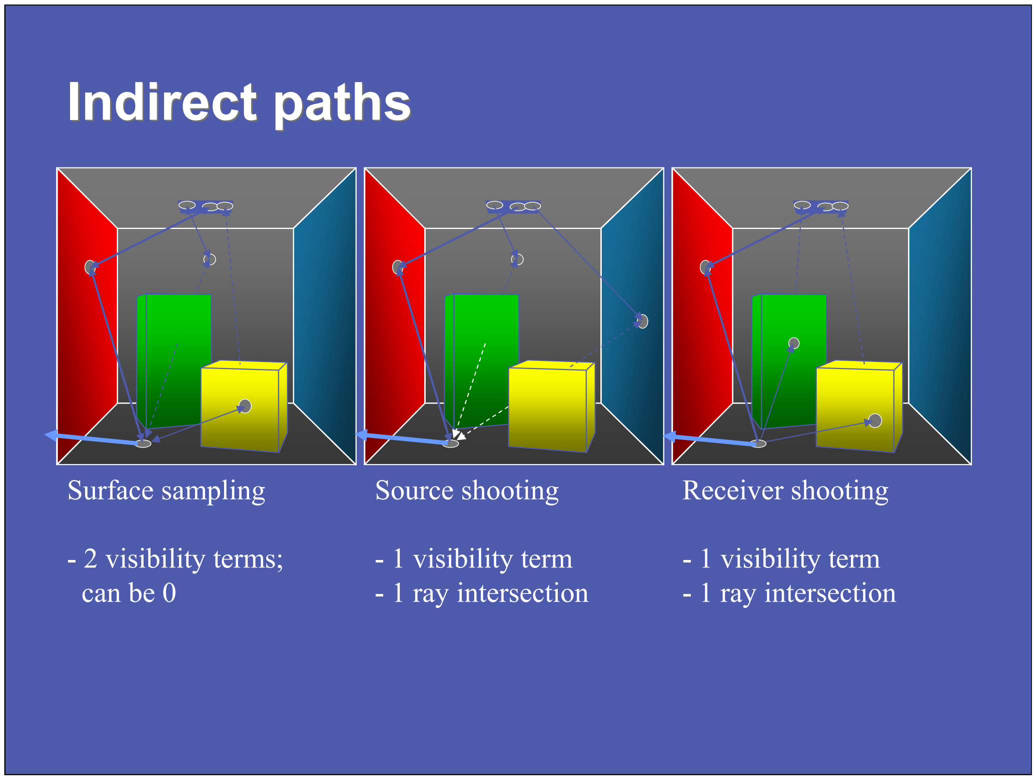


per path:
1 ray intersection
1 visibility check

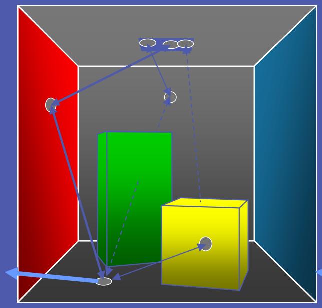
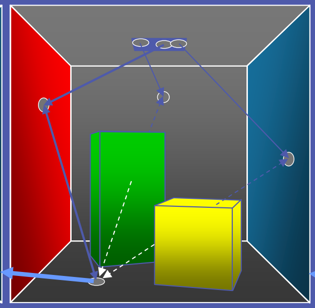
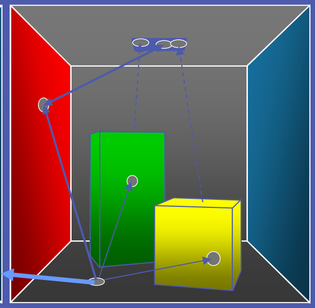
Another algorithm might generate the intermediate point in a slightly different way: a random direction is sampled over the hemisphere around the target point, this ray is traced in the environment, and the closest intersection point found.

Then this visible point is connected to a random surface point generated on the light source.

This is the usual way of generating indirect paths in stochastic ray tracing.



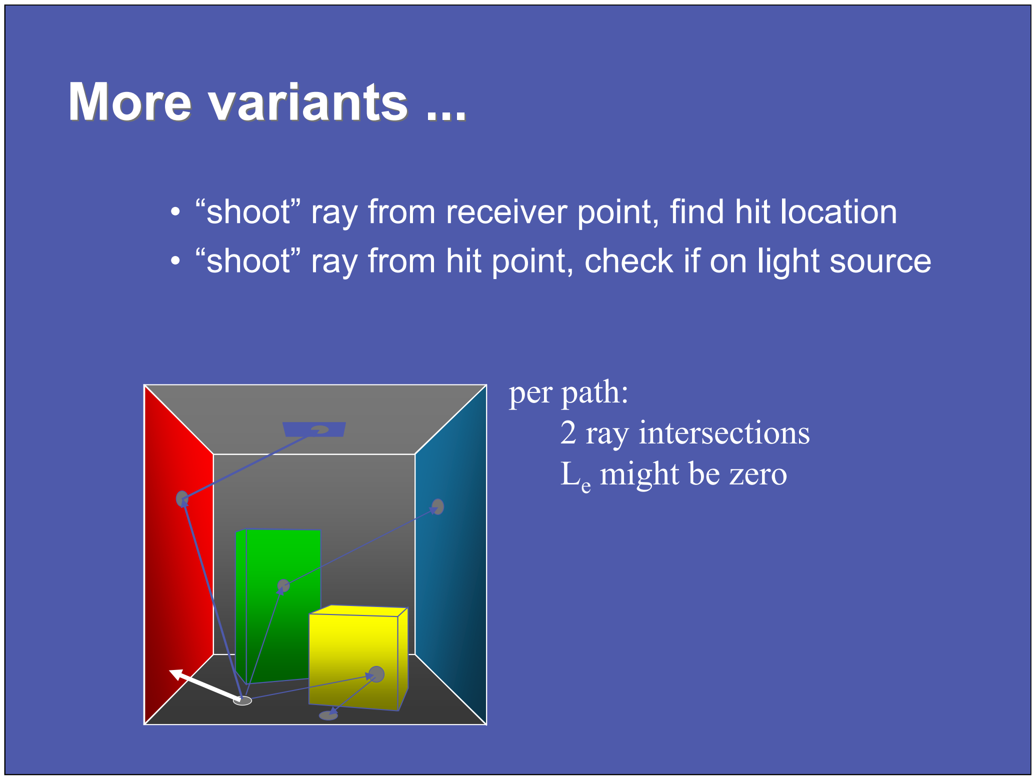
Indirect paths

Surface sampling	Source shooting	Receiver shooting
- 2 visibility terms; can be 0	- 1 visibility term - 1 ray intersection	- 1 visibility term - 1 ray intersection

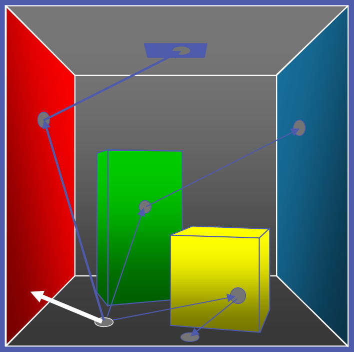
Here are all the different approaches compared.

All three of these algorithms will produce an unbiased image when generating enough samples, but the efficiency will be very different.



More variants ...

- “shoot” ray from receiver point, find hit location
- “shoot” ray from hit point, check if on light source



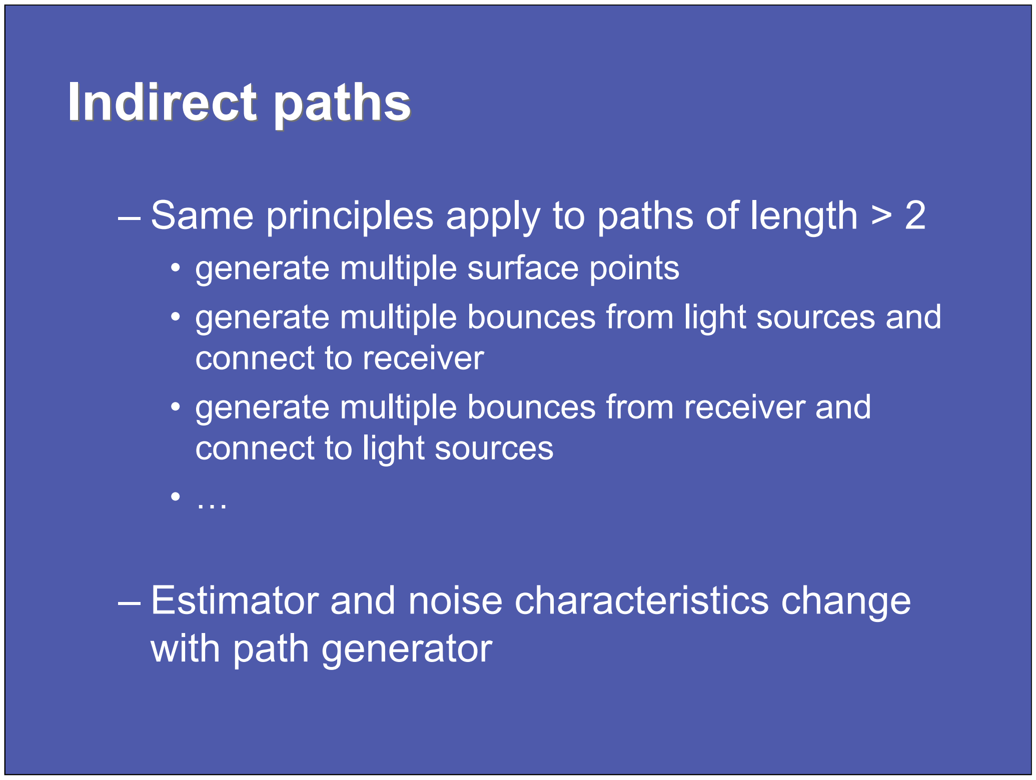
per path:

2 ray intersections

L_e might be zero

Even more variants can be thought of, as shown on this slide.

This is just to illustrate the general principle, that any path generator will do, as long as the correct energy transfer and correct probabilities for all the paths are computed.



Indirect paths

- Same principles apply to paths of length > 2
  - generate multiple surface points
  - generate multiple bounces from light sources and connect to receiver
  - generate multiple bounces from receiver and connect to light sources
  - …
- Estimator and noise characteristics change with path generator

For paths of length greater than 2, one can also come up with a lot of different path generators.

Usually these are implemented recursively.

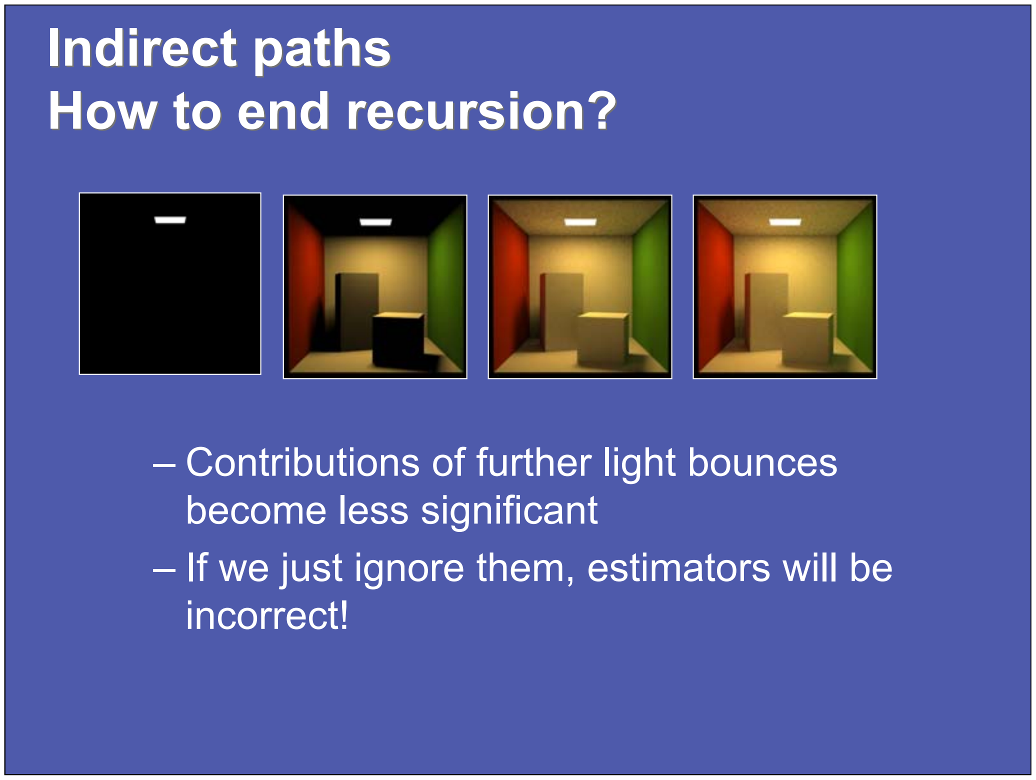
Indirect paths

– General algorithm:

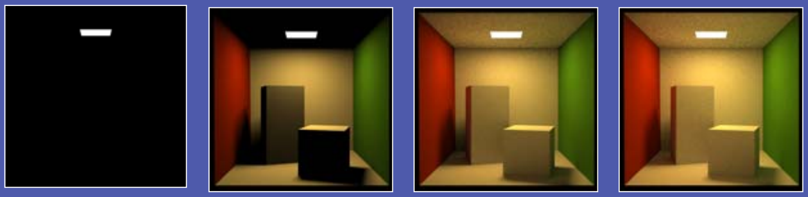
```
compute_radiance (point, direction)
      est_rad = 0;

      for (i=0; i<n; i++)
            p = generate_indirect_path;
            est_rad += energy_transfer(p) / probability(p);

      est_rad = est_rad / n;
      return(est_rad);
```



Indirect paths
How to end recursion?

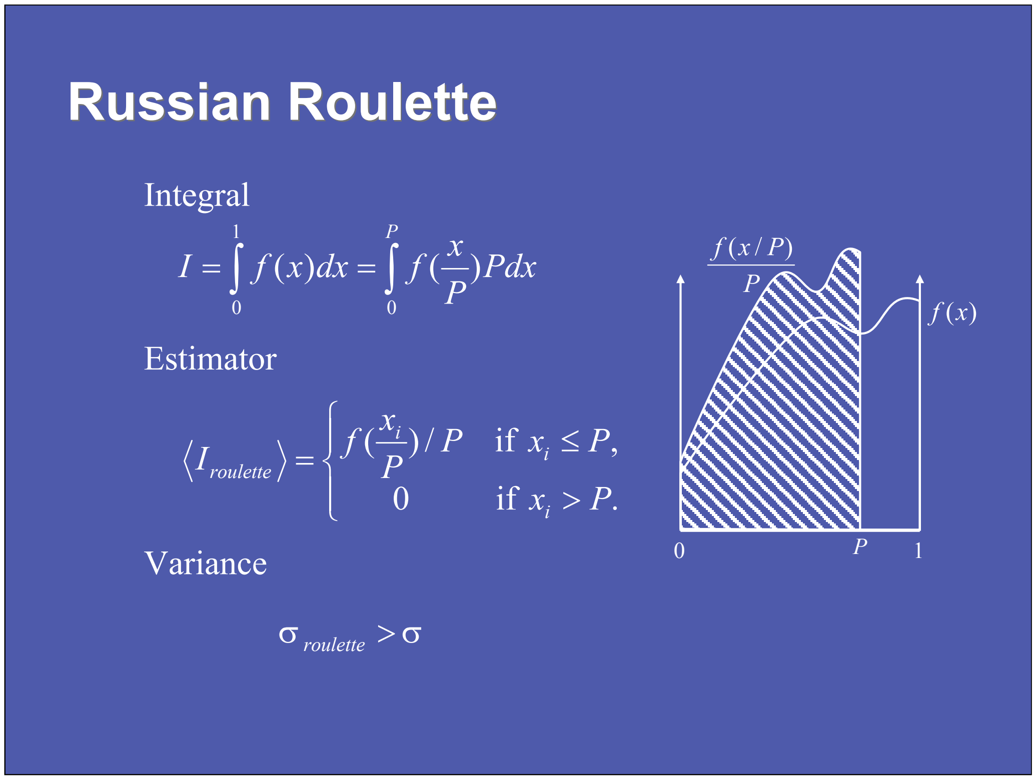


- Contributions of further light bounces become less significant
- If we just ignore them, estimators will be incorrect!

An important issue when writing a recursive path generator is how to stop the recursion.

Our goal is still to produce unbiased images, that is, images which will be correct if enough samples are being generated.

As such, we cannot ignore deeper recursions, although we would like to spend less time on them, since the light transport along these longer paths is will probably be less significant.



Russian Roulette

Integral

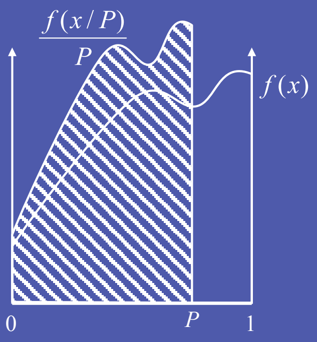
$$I = \int_0^1 f(x)dx = \int_0^P f(\frac{x}{P})Pdx$$

Estimator

$$\langle I_{roulette} \rangle = \begin{cases} f(\frac{x_i}{P})/P & \text{if } x_i \le P, \\ 0 & \text{if } x_i > P. \end{cases}$$

Variance

$$\sigma_{roulette} > \sigma$$

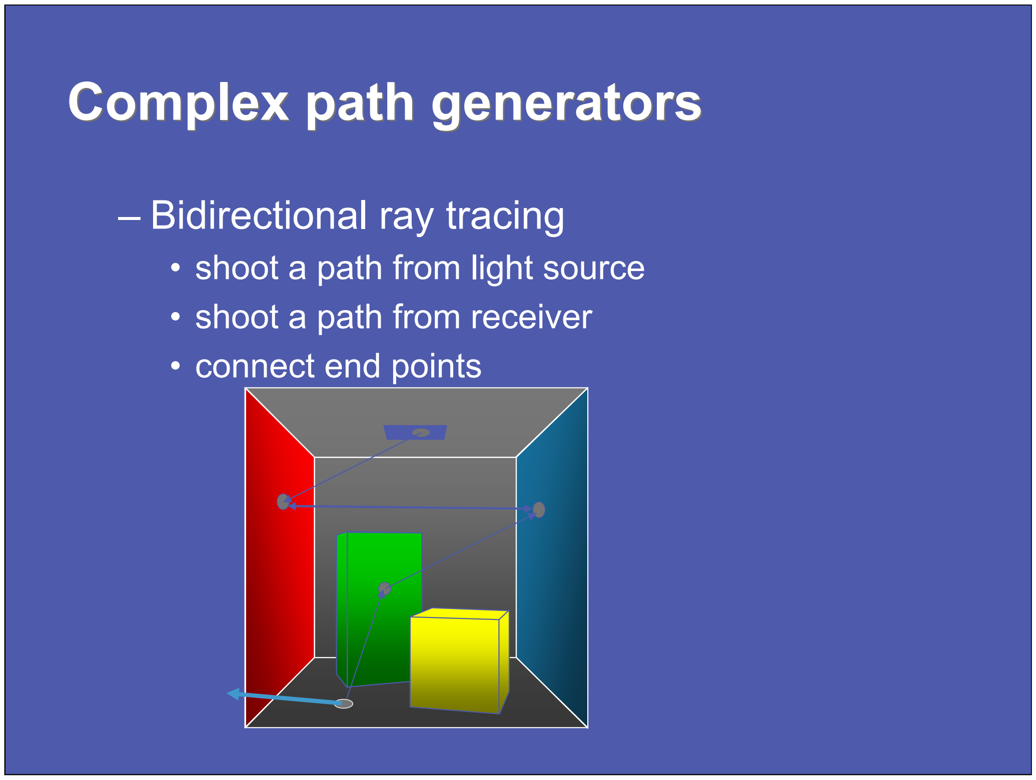


Russian Roulette is a technique that can be used to stop the recursion.

Mathematically, it means that we will cosnider part of integration domain to have a function value of 0. If a sample is generated in this part of the domain, it is ‘absorbed’. Of course, this means that the samples which are not absorbed will need to get a greater weight, since they have to compensate for the fact that we still want an unbiased estimator for the original integral.

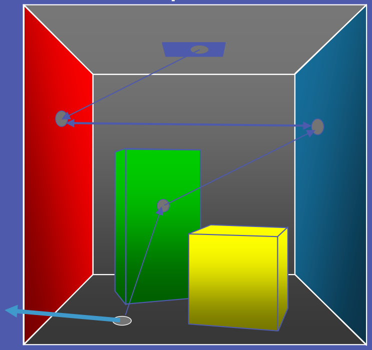
Russian Roulette

- In practice: pick some ‘absorption probability’ α
  - probability $1-\alpha$ that ray will bounce
  - estimated radiance becomes $L/(1-\alpha)$
- E.g. $\alpha = 0.9$
  - only 1 chance in 10 that ray is reflected
  - estimated radiance of that ray is multiplied by 10
- Intuition
  - instead of shooting 10 rays, we shoot only 1, but count the contribution of this one 10 times



Complex path generators

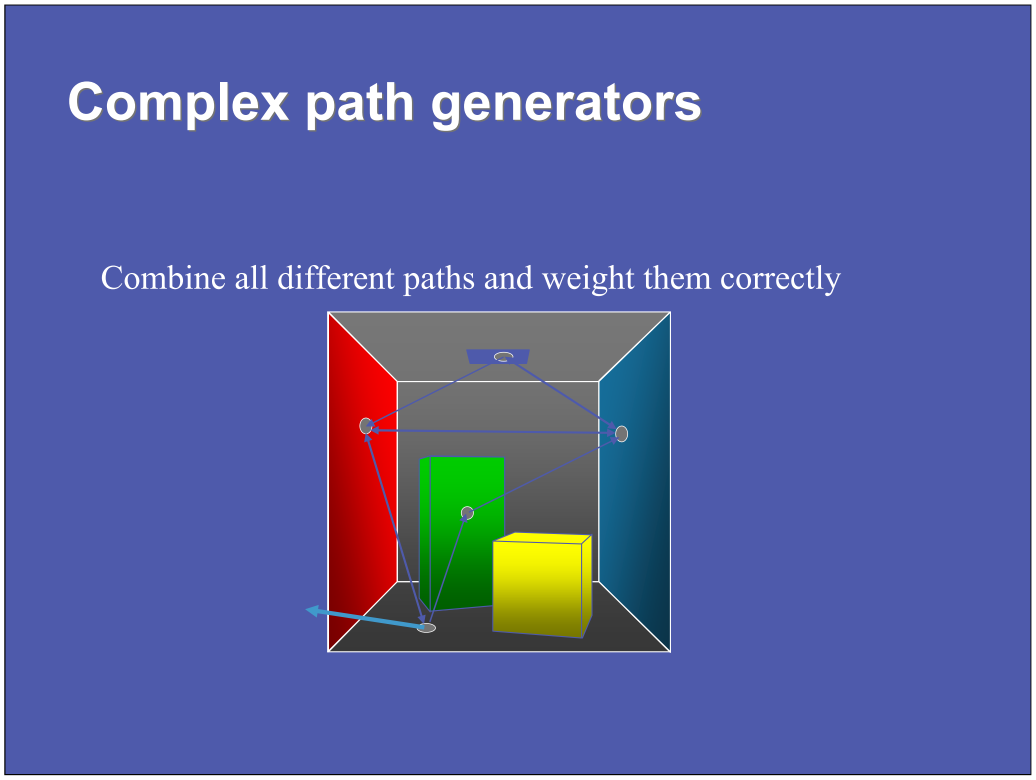
- Bidirectional ray tracing
  - shoot a path from light source
  - shoot a path from receiver
  - connect end points



More complex path generators are also possible.

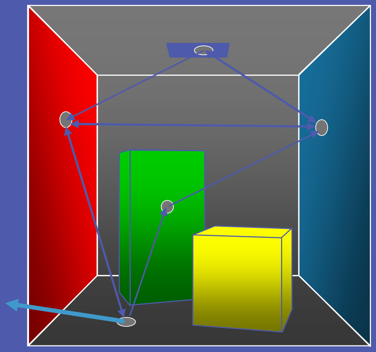
Bidirectional ray tracing is an algorithm that generates paths with variable length, both from the light source and the eye, and connects the end points.

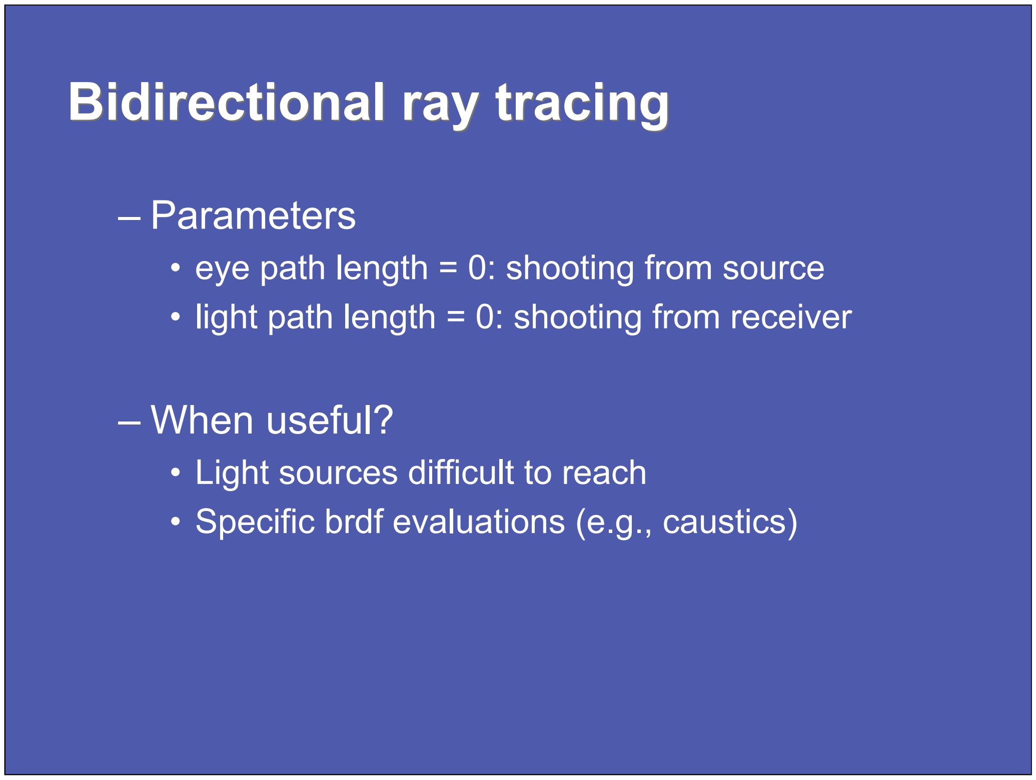
Again, this is path generator, and results in an unbiased images if all relevant pdf's are taken into account.



Complex path generators

Combine all different paths and weight them correctly

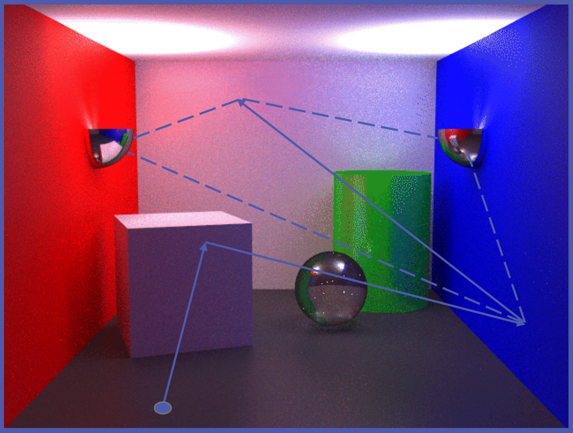




Bidirectional ray tracing

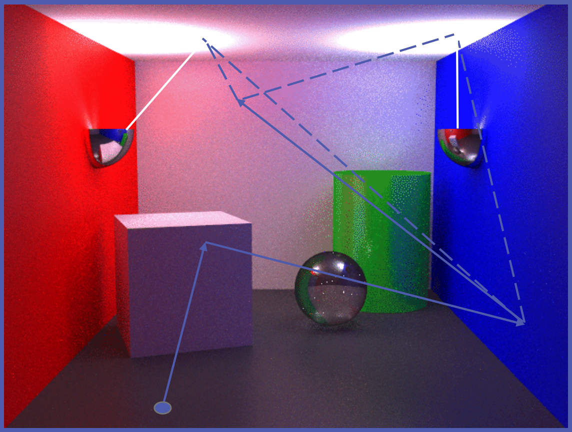
- Parameters
  - eye path length = 0: shooting from source
  - light path length = 0: shooting from receiver
- When useful?
  - Light sources difficult to reach
  - Specific brdf evaluations (e.g., caustics)

Bidirectional ray tracing

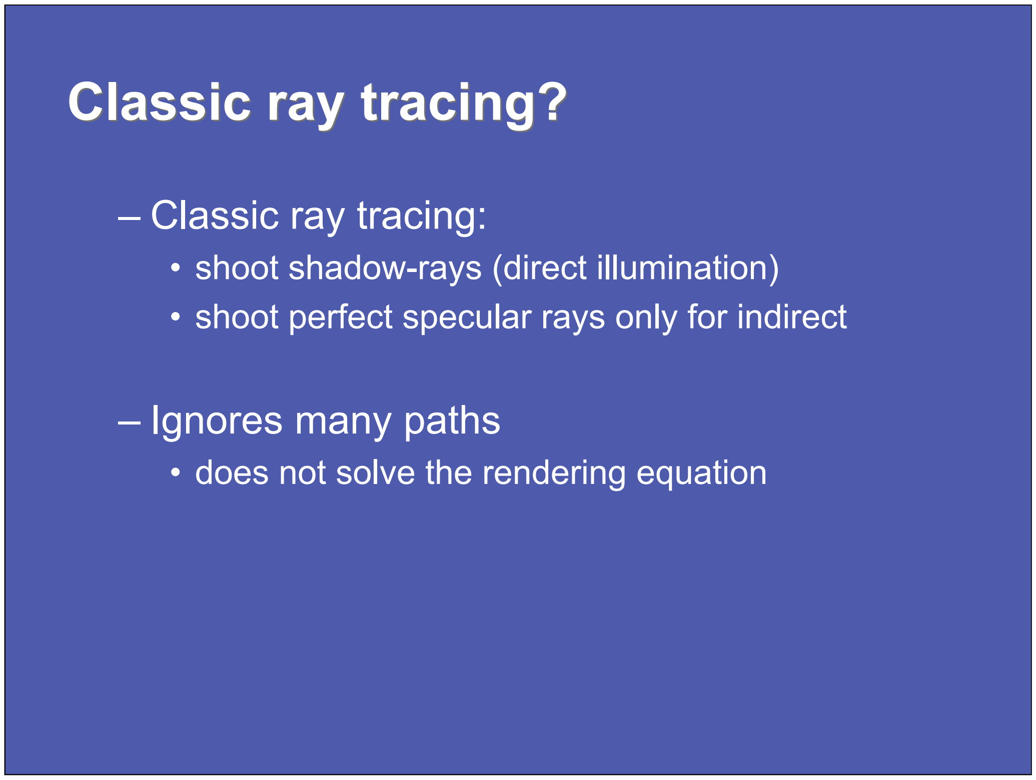


(E. Lafortune, 1996)

Bidirectional ray tracing



(E. Lafortune, 1996)

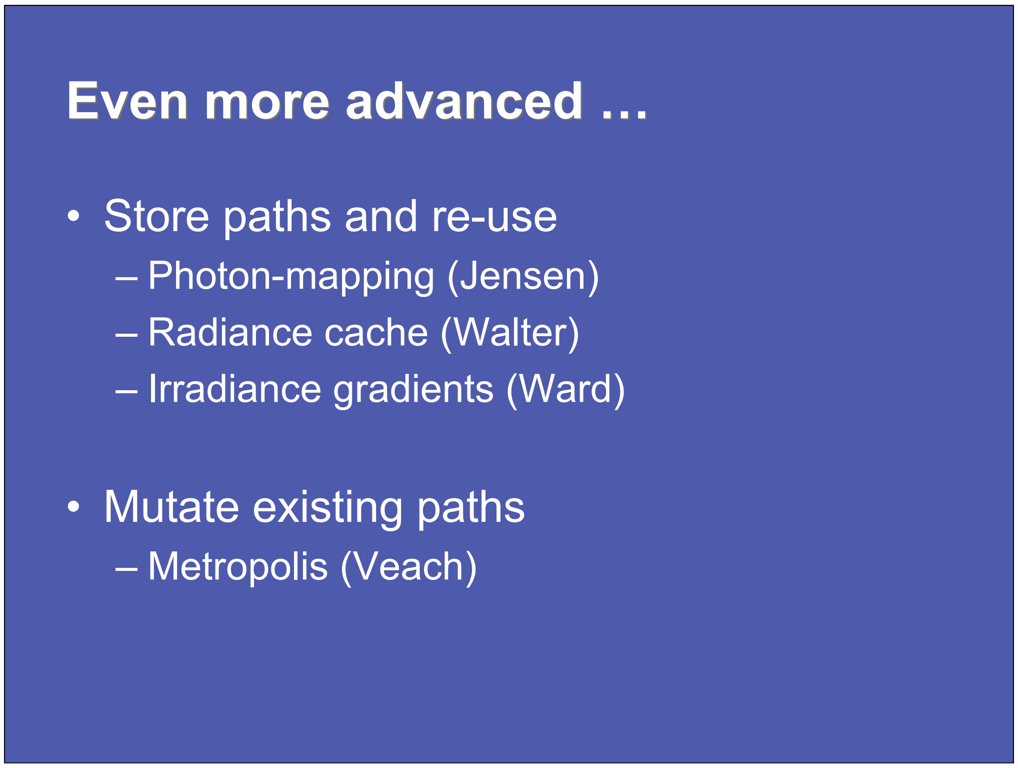


Classic ray tracing?

- Classic ray tracing:
  - shoot shadow-rays (direct illumination)
  - shoot perfect specular rays only for indirect
- Ignores many paths
  - does not solve the rendering equation

How does classic ray tracing compare to the physically correct path genertors described so far?

Classic ray tracing only generates a subset of all possible paths: shadow rays, and the perfect specular and refractive paths. As such, classic ray tracing ignores many of the other paths along which energy is transported from the light sources to the receiving surfaces.



Even more advanced …

- Store paths and re-use
  - Photon-mapping (Jensen)
  - Radiance cache (Walter)
  - Irradiance gradients (Ward)
- Mutate existing paths
  - Metropolis (Veach)

General global illumination algorithm

- Design path generators
- Path generators determine efficiency of global illumination algorithm
- Future:
  - Different path generators for different areas of the image
  - Adaptively
  - Re-use of paths as much as possible

The Rendering Equation and Path Tracing

This chapter gives various formulations of the rendering equation, and outlines several strategies for computing radiance values in a scene.

8.1 Formulations of the rendering equation

The global illumination problem is in essence a transport problem. Energy is emitted by light sources and transported through the scene by means of reflections (and refractions) at surfaces. One is interested in the energy equilibrium of the illumination in the environment.

The transport equation that describes global illumination transport is called the rendering equation. It is the integral equation formulation of the definition of the BRDF, and adds the self-emittance of surface points at light sources as an initialization function. The self-emitted energy of light sources is necessary to provide the environment with some starting energy. The radiance leaving some point x, in direction Θ, can be expressed as an integral over all hemispherical directions incident on the point x (figure 8.1):

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

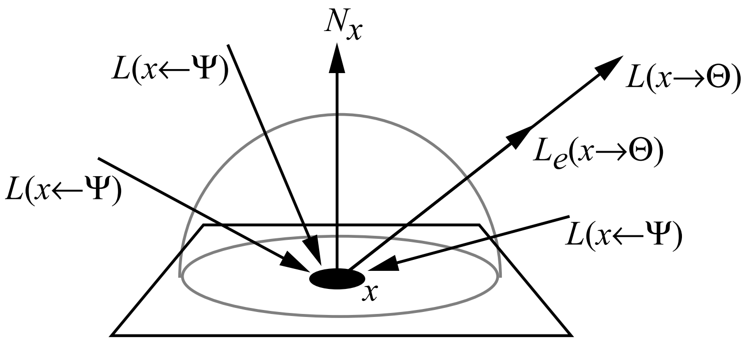


Figure 8.1: Rendering equation

One can transform the rendering equation from an integral over the hemisphere to an integral over all surfaces in the scene. Also, radiance remains unchanged along straight paths, so exitant radiance can be transformed to incident radiance and vice-versa, thus obtaining new versions of the rendering equation. By combining both options with a hemispheric or surface integration, four different formulations of the rendering equation are obtained. All these formulations are mathematically equivalent.

Exitant radiance, integration over the hemisphere

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) L(y \rightarrow -\Psi) \cos(N_x, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

When designing an algorithm based on this formulation, integration over the hemisphere is needed, and as part of the function evaluation for each point in the integration domain, a ray has to be cast and the nearest intersection point located.

Exitant radiance, integration over surfaces

$$L(x \rightarrow \Theta) = L_e(x \rightarrow \Theta) + \int_{A} f_r(x, \Psi \leftrightarrow \Theta) L(y \rightarrow \overrightarrow{yx}) V(x, y) G(x, y) dA_y$$

with

$$G(x, y) = \frac{cos(N_x, \Psi) cos(N_y, \Psi)}{r_{xy}^2}$$

Algorithms based on this formulation need to evaluate the visibility $V(x, y)$ between two points x and y, which is a different operation than casting a ray from x in a direction Θ.

Incident radiance, integration over the hemisphere

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_{\Omega_y} f_r(y, \Psi \leftrightarrow -\Theta) L(y \leftarrow \Psi) \cos(N_y, \Psi) d\omega_\Psi$$

with

$$y = r(x, \Theta)$$

Incident radiance, integration over surfaces

$$L(x \leftarrow \Theta) = L_e(x \leftarrow \Theta) + \int_A f_r(y, \Psi \leftrightarrow \overrightarrow{yz}) L(y \leftarrow \overrightarrow{yz}) V(y, z) G(y, z) dA_z$$

with

$$y = r(x, \Theta)$$

8.2 Importance function

In order to compute the average radiance value over the area of a pixel, one needs to know the radiant flux over that pixel (and associated solid angle incident w.r.t. the aperture of the camera). Radiant flux is expressed by integrating the radiance distribution over all possible surface points and directions. Let $S = A_p \times \Omega_p$ denote all surface points A_p and directions Ω_p visible through the pixel. The flux $\Phi(S)$ is written as:

$$\Phi(S) = \int_{A_p} \int_{\Omega_p} L(x \rightarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

When designing algorithms, it is often useful to express the flux as an integral over all possible points and directions in the scene. This can be achieved by introducing the initial importance function $W_e(x \leftarrow \Theta)$:

$$\Phi(S) = \int_A \int_\Omega L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x$$

$W_e(x \leftarrow \Theta)$ is appropriately defined by:

$$W_e(x \leftarrow \Theta) = \begin{cases} 1 & \text{if } (x,\Theta) \in S \\ 0 & \text{if } (x,\Theta) \notin S \end{cases}$$

The average radiance value is then given by:

$$L_{average} = \frac{\int_A \int_\Omega L(x \rightarrow \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}{\int_A \int_\Omega W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x}$$

We now want to develop the notion of importance further, by considering the possible influence of some energy value at each pair (x, Θ) on the value $\Phi(S)$. Or: if a single radiance value $L(x \rightarrow \Theta)$ is placed at (x, Θ), and if there are no other sources of illumination present, how large would the resulting value of $\Phi(S)$ be? This influence value attributed to $L(x \rightarrow \Theta)$ is called the importance of (x, Θ) w.r.t. S, is written as $W(x \leftarrow \Theta)$, and depends only on the geometry and reflective properties of the objects in the scene.

The equation expressing $W(x \leftarrow \Theta)$ can be derived by taking into account two mechanisms in which $L(x \rightarrow \Theta)$ can contribute to $\Phi(S)$:

Self-contribution If $(x, \Theta) \in S$, then $L(x \rightarrow \Theta)$ fully contributes to $\Phi(S)$. This is called the self-importance of the set S, and corresponds to the above definition of $W_e(x \leftarrow \Theta)$.

Indirect contributions It is possible that some part of $L(x \rightarrow \Theta)$ contributes to $\Phi(S)$ through one or more reflections at several surfaces. The radiance $L(x \rightarrow \Theta)$ travels along a straight path and reaches a surface point $r(x, \Theta)$. Energy is reflected at this surface point according to the BRDF. Thus, there is a hemisphere of directions at $r(x, \Theta)$, each emitting a differential radiance value as a result of the reflection of the radiance $L(r(x, \Theta) \leftarrow -\Theta)$. By integrating the importance values for all these new directions, we have a new term for $W(x \leftarrow \Theta)$.

Both terms combined produces the following equation:

$$W(x \leftarrow \Theta) = W_e(x \leftarrow \Theta) + \int_{\Omega_z} f_r(z, \Psi \leftrightarrow -\Theta) W(z \leftarrow \Psi) \cos(N_r(x, \Theta), \Psi) d\omega_\Psi$$

with

$$z = r(x, \Theta)$$

Mathematically, this equation is identical to the transport equation of incident radiance, and thus, the notion *incidence* can be attributed to importance. The source function $W_e = 1$ if x is visible through the pixel and Θ is a direction pointing through the pixel to the aperture of the virtual camera.

To enhance the analogy with radiance as a transport quantity, exitant importance can be defined as:

$$W(x \to \Theta) = W((r, \Theta) \leftarrow -\Theta)$$

and also:

$$W(x \to \Theta) = W_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \leftrightarrow \Theta) W(x \leftarrow \Psi) cos(N_x, \Psi) d\omega_\Psi$$

An expression for the flux of through every pixel, based on the importance function, can now be written. Only the importance of the light sources needs to be considered when computing the flux:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \to \Theta) W(x \leftarrow \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

It is also possible to write $\Phi(S)$ in the following form:

$$\Phi(S) = \int_A \int_{\Omega_x} L_e(x \leftarrow \Theta) W(x \to \Theta) cos(N_x, \Theta) d\omega_\Theta dA_x$$

and also:

$$\begin{aligned} \Phi(S) &= \int_A \int_{\Omega_x} L(x \to \Theta) W_e(x \leftarrow \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \\ \Phi(S) &= \int_A \int_{\Omega_x} L(x \leftarrow \Theta) W_e(x \to \Theta) \cos(N_x, \Theta) d\omega_\Theta dA_x \end{aligned}$$

There are two approaches to solve the global illumination problem: The first approach starts from the pixel, and the radiance values are computed by solving one of the transport equations describing radiance. A second approach computes the flux starting from the light sources, and computes for each light source the corresponding importance value. If one looks at various algorithms in some more detail:

- Stochastic ray tracing propagates importance, the surface area visible through each pixel being the source of importance. In a typical implementation, the importance is never explicitly computes, but is implicitly done by tracing rays through the scene and picking up illumination values from the light sources.

- Light tracing is the dual algorithm of ray tracing. It propagates radiance from the light sources, and computes the flux values at the surfaces visible through each pixel.

- Bidirectional ray tracing propagates both transport quantities at the same time, and in an advanced form, computes a weighted average of all possible inner products at all possible interactions.

8.3 Path formulation

The above description of global illumination transport algorithms is based on the notion of radiance and importance. One can also express global transport by considering path-space, and computing a transport measure over each individual path. Path-space encompasses all possible paths of any length. Integrating a transport measure in path-space then involves generating the correct paths (e.g. random paths can be generated using an appropriate Monte Carlo sampling procedure), and evaluating the throughput of energy over each generated path. This view was developed by Spanier and Gelbard and introduced into rendering by Veach.

$$\Phi(S) = \int_{\Omega^*} f(\overline{x}) d\mu(\overline{x})$$

in which Ω^* is the path-space, $\overline{x}$ is a path of any length and $d\mu(\overline{x})$ is a measure in path space . $f(\overline{x})$ describes the throughput of energy and is a succession of $G(x, y)$, $V(x, y)$ and BRDF evaluations, together with a L_e and W_e evaluation at the beginning and end of the path.

An advantage of the path formulation is that paths are now considered to be the sample points for any integration procedure. Algorithms such as Metropolis light transport or bidirectional ray tracing are often better described using the path formulation.

8.4 Simple stochastic ray tracing

In any pixel-driven rendering algorithm we need to use the rendering equation to evaluate the appropriate radiance values. The most simple algorithm to compute this radiance value is to apply a basic and straightforward MC integration scheme to the standard form of the rendering equation:

$$\begin{aligned} L(x \to \Theta) &= L_e(x \to \Theta) + L_r(x \to \Theta) \\ &= L_e(x \to \Theta) + \int_{\Omega_x} L(x \leftarrow \Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi \end{aligned}$$

The integral is evaluated using MC integration, by generating N random directions Ψ_i over the hemisphere Ω_x, according to some pdf $p(\Psi)$. The estimator for $L_r(x \to \Theta)$ is given by:

$$\langle L_r(x \to \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L(x \leftarrow \Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

$L(x \leftarrow \Psi_i)$, the incident radiance at x, is unknown. It is now necessary to trace the ray leaving x in direction Ψ_i through the scene to find the closest intersection point $r(x, \Psi)$. Here, another radiance evaluation is needed. The result is a recursive procedure to evaluate $L(x \leftarrow \Psi_i)$, and as a consequence, a path, or a tree of paths if $N > 1$, is generated in the scene.

These radiance evaluations will only yield a non-zero value, if the path hits a surface for which L_e has a value different from 0. In other words, in order to compute a contribution to the illumination of a pixel, the recursive path needs to reach at least one of the light sources in the scene. If the light sources are small, the resulting image will therefore mostly be black. This is expected, because the algorithm generates paths, starting at a point visible through a pixel, and slowly working towards the light sources in a very uncoordinated manner.

8.5 Russian Roulette

The recursive path generator described above needs a stopping condition to prevent the paths being of infinite length. We want to cut off the generation of paths, but at the same time, we have to be very careful about not introducing any bias into the image generations process. Russian Roulette addresses the problem of keeping the lengths of the paths manageable, but at the same time leaves room for exploring all possible paths of any length. Thus, an unbiased image can still be produced.

The idea of Russian Roulette can best be explained by a simple example: suppose one wants to compute a value V. The computation of V might be computationally very expensive, so we introduce a random variable r, which is uniformly distributed over the interval $[0, 1]$. If r is larger than some threshold value $\alpha \in [0, 1]$, we proceed with computing V. However, if $r \leq \alpha$, we do not compute V, and assume $V = 0$. Thus, we have a random experiment, with an expected value of $(1 - \alpha)V$. By dividing this expected value by $(1 - \alpha)$, an unbiased estimator for V is maintained.

If V requires recursive evaluations, one can use this mechanism to stop the recursion. α is called the absorption probability. If α is small, the recursion will continue many times, and the final computed value will be more accurate. If α is large, the recursion will stop sooner, and the estimator will have a higher variance. In the context of our path tracing algorithm, this means that either accurate paths of a long length are generated, or very short paths which provide a less accurate estimate.

In principle any value for α can be picked, thus controlling the recursive depth and execution time of the algorithm. $1 - \alpha$ is often set to be equal to the hemispherical reflectance of the material of the surface. Thus, dark surfaces will absorb the path more easily, while lighter surfaces have a higher chance of reflecting the path.

8.6 Indirect Illumination

In most path tracing algorithms, direct illumination is explicitly computed separately from all other forms of illumination (see previous chapter on direct illumination). This section outlines some strategies for computing the indirect illumination in a scene. Computing the indirect illumination is usually a harder problem, since one does not know where most important contributions are located. Indirect illumination consists of the light reaching a target point x after at least one reflection at an intermediate surface between the light sources and x.

8.6.1 Hemisphere sampling

The rendering equation can be split in a direct and indirect illumination term. The indirect illumination (i.e. not including any direct contributions from light sources to the point x) contribution to $L(x \to \Theta)$ is written as:

$$L_{indirect}(x \to \Theta) = \int_{\Omega_x} L_r(r(x, \Psi) \to -\Psi) f_r(x, \Theta \leftrightarrow \Psi) \cos(\Psi, N_x) d\omega_\Psi$$

The integrand contains the reflected terms L_r from other points in the scene, which are themselves composed of a direct and indirect illumination part. In a closed environment, $L_r(r(x, \Psi) \to -\Psi)$ usually has a non-zero value for all (x, Ψ) pairs. As a consequence, the entire hemisphere around x needs to be considered as the integration domain.

The most general MC procedure to evaluate indirect illumination, is to use any hemispherical pdf $p(\Psi)$, and generating N random directions Ψ_i. This produces the following estimator:

$$\langle L_{indirect}(x \to \Theta) \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{L_r(r(x, \Psi_i) \to -\Psi_i) f_r(x, \Theta \leftrightarrow \Psi_i) \cos(\Psi_i, N_x)}{p(\Psi_i)}$$

In order to evaluate this estimator, for each generated direction Ψ_i, the BRDF and the cosine term are to be evaluated, a ray from x in the direction of Ψ_i needs to be traced, and the reflected radiance $L_r(r(x, \Psi_i) \to -\Psi_i)$ at the closest intersection point $r(x, \Psi_i)$ has to be evaluated. This last evaluation shows the recursive nature of indirect illumination, since this reflected radiance at $r(x, \Psi_i)$ can be split again in a direct and indirect contribution.

The simplest choice for $p(\Psi)$ is $p(\Psi) = 1/2\pi$, such that directions are sampled proportional to solid angle. Noise in the resulting picture will be caused by variations in the BRDF and cosine evaluations, and variations in the reflected radiance L_r at the distant points.

The recursive evaluation can again be stopped using Russian Roulette, in the same way as was done for simple stochastic ray tracing. Generally, the local hemispherical reflectance is used as an appropriate absorption probability. This choice can be explained intuitively: One only wants to spend work (i.e. tracing rays and evaluating $L_{indirect}(x)$) proportional to the amount of energy present in different parts of the scene.

8.6.2 Importance sampling

Uniform sampling over the hemisphere does not use any knowledge about the integrand in the indirect illumination integral. However, this is necessary to reduce noise in the final image, and thus, some form of importance sampling is needed. Hemispherical pdf's proportional (or approximately proportional) to any of the following factors can be constructed:

Cosine sampling

Sampling directions proportional to the cosine lobe around the normal N_x prevents directions to be sampled near the horizon of the hemisphere where $\cos(\Psi, N_x)$ yields a very low value, and thus possibly insignificant contributions to the computed radiance value.

BRDF sampling

BRDF sampling is a good noise-reducing technique when a glossy or highly specular BRDFs is present. It diminishes the probability that directions are sampled where the BRDF has a low value or zero value. Only for a few selected BRDF models, however, is it possible to sample exactly proportional to the BRDF. Even better would be trying to sample proportional to the product of the BRDF and the cosine term. Analytically, this is even more difficult to do, except in a few rare cases where the BRDF model has been chosen carefully.

Incident radiance field sampling

A last technique that can be used to reduce variance when computing the indirect illumination is to sample a direction Ψ according to the incident radiance values $L_r(x \leftarrow \Psi)$. Since this incident radiance is generally unknown, an adaptive technique needs to be used, where an approximation of $L_r(x \leftarrow \Psi)$ is constructed during the execution of the rendering algorithm.

8.6.3 Overview

It is now possible to build a full global illumination renderer using stochastic path tracing. The efficiency, accuracy and overall performance of the complete algorithm will be determined by the choice of all of the following parameters. As is usual in MC evaluations, the more samples or rays are generated, the less noisy the final image will be.

Number of viewing rays per pixel The amount of viewing rays through each pixel is responsible for effects such as aliasing at visible boundaries of objects or shadows.

Direct Illumination:

- The total number of shadow rays generated at each surface point x;
- The selection of a single light source for each shadow ray;
- The distribution of the shadow ray over the area of the selected light source.

Indirect Illumination (hemisphere sampling):

- Number of indirect illumination rays;
- Exact distribution of these rays over the hemisphere (uniform, cosine, ...);
- Absorption probabilities for Russian Roulette.

The better one makes use of importance sampling, the better the final image and the less noise there will be. An interesting question is, given a maximum amount of rays one can use per pixel, how should these rays best be distributed to reach the highest possible accuracy for the full global illumination solution? This is still an open problem. There are generally accepted 'default' choices, but there are no hard and fast choices. It generally is accepted that branching out equally at all levels of the tree is less efficient. For indirect illumination, a branching factor of 1 is often used after the first level. Many implementations even limit the indirect rays to one per surface point, and compensate by generating more viewing rays.