Particle Tracing Methods in Photorealistic Image Synthesis

Rázsó István Márk
rezso@inf.bme.hu

Department of Control Engineering and Information Technology
Technical University of Budapest
Budapest / Hungary

Abstract

There are different methods to represent global illumination in image generation algorithms. One solution is to use radiosity algorithms, another one is to use particle tracing. This document gives an overview on different rendering algorithms using particle tracing.

Keywords: Rendering, Particle Tracing, Kernel Estimator, Importance Sampling

Introduction

In rendering we are interested in generating pictures of an artifically described environment. The goal is to generate images which look like pictures taken with a real camera. This requires to calculate the power reaching the camera from a given direction, i.e. through a given pixel. The rendering equation describes the reflected radiance from a surface point x:

$$L_r(x,\Psi_r) = \int_{all\Psi_i} f_r(x,\Psi_r,\Psi_i) L_i(x,\Psi_i)\cos\theta_i d\omega_i = \int_{all\Psi_i} f_r(x,\Psi_r,\Psi_i)\frac{d^2\Phi_i(x,\Psi_i)}{dAd\omega_i} d\omega_i$$

where Ψ_r and Ψ_i are the direction of the reflected incoming radiance. f_r is the BRDF (Bidirectional Reflectance Distribution Function) at x, L_i is the incoming radiance and Φ_i is the incoming flux. Different methods have been used to evaluate this equation. To describe the incoming radiance at a surface point, global illumination methods have been used, which can be divided to radiosity methods and methods using particle tracing.

Particle tracing is a method to simulate the behaviour of large number of particles. In a general algorithm there is a time unit which is used as a step interval in the simulation. In every step each particle is examined. Particles can be born, move, collide, change their properties and die in the environment. Image generation algorithms use a simpler particle tracing method. These methods utilise the particle behaviour of light, where each light particle has a small quantity of power. To simulate the light particles (their interaction with the environment), we trace particles from event to event. Light particles do not collide with each other, so each particle can be simulated separately. Usually we assume that during the flight of the particle from one surface to another no event can change its directional properties. The media between the surfaces is vacuum or affecting only the colour, but not the directional properties of the particle, so the effect can be calculated at the endpoint. Figure 1 shows the two different types of particle tracing methods in two dimensions.

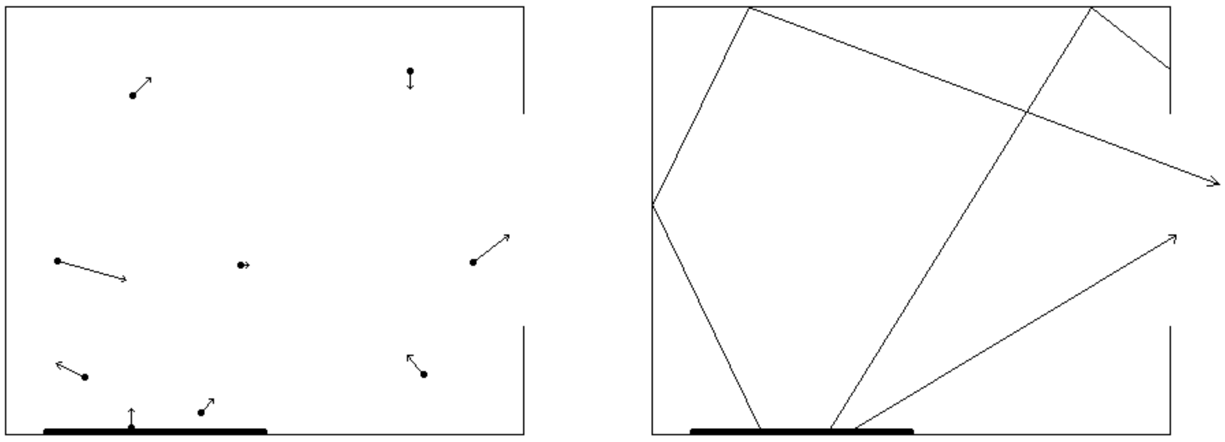


Figure 1: Left: a general particle tracing scene with several particles at the same time. Right: particle paths, which can be traced separately. Two particle left the scene, one has been absorbed.

Advantages of particle tracing methods over radiosity algorithms:

• Low memory consumption: While most of the radiosity algorithms requires meshing which consumes a huge amount of memory especially while rendering large scenes. This algorithm uses very small amount of memory, only for the currently traced particle.

• Easily handles local complexity: With radiosity algorithms where the model has a very high global complexity (large number of surfaces), but a limited local complexity (small subsets of surfaces are mutually visible), partitioning can be used to decompose the model into separately resolvable subsets. But if any subset has a high local complexity, then partitioning may not reduce the number of problems to a solvable size. This problem can occur in large closed buildings, like a railway station, or a university hall; it doesn't occur with particle tracing.

• Ideal specular effects: For specular surfaces many radiosity algorithms can only use a virtual image method. With this method only a number of ideal and planar surfaces can be handled practically.

• Parallelism. This algorithm can be easily parallelised, because large number of photons needs to be traced, and they have no connection with each other, and they don't effect the scene.

Particle Tracing Algorithm

How to trace a particle

To simulate a particle we have to generate particles in the scene, then we have to trace these particles through reflections and refractions, until they are absorbed or leave the scene.

We can total the power emitted by the lightsources (Φ). Then we generate n particles, each carrying power Φ / n. For each lightsource l_i with power Φ_i, we trace $N_i = n\Phi_i / \Phi$ rays according to the emission characteristics of the lightsource, as in [1].

Another method: for each particle we only store the attenuation. The attenuation describes the decrease of the power of the particle from its birth on. When we need the photon's power, we divide the power of the light source which emitted the photon by the total number of photons emitted by the same lightsource, and multiply this value by the attenuation of the particle, as in [2]. This method allows us to increase the number of photons emitted by each light source, but this can lead to bias in the solution.

After we have found the lightsource for the particle, a starting point and a direction on the lightsource, we trace the particle through the scene. At each intersection (a photon hits a surface) a photon can be absorbed, refracted or reflected. If it is not absorbed, we store the properties of the photon, and generate a new direction and colour according to the physical properties of the material. Figure 2 shows a scene with different particle paths and surfaces with particle hits.

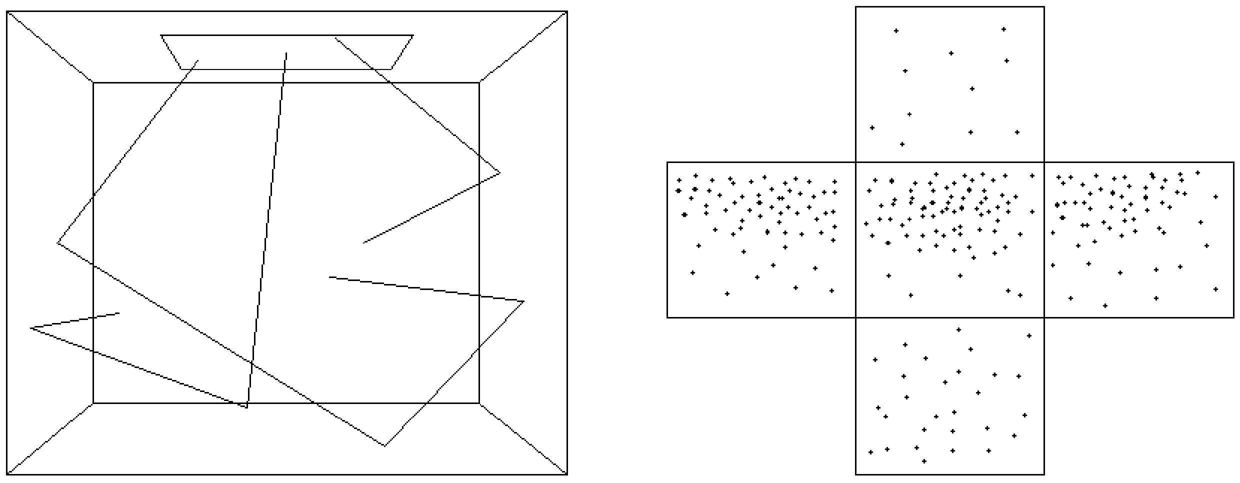


Figure 2: Left: A few particle paths in a scene. Right: Particle hits on surfaces

Particle properties

Different algorithms need different photon properties, but each algorithm requires to store the power of the particle. A particle can have a fixed wavelength power, say red or green or blue [1], or this power can represent the full spectra [4]. For many algorithms it is important to store the incoming direction of the particle. Some kind of position information on the surface hit is also needed. We can store the identifier of the surface and local coordinates on it, or we can store position in the scene, and a surface normal vector [4].

How to use the generated particle data

One type of algorithms use particle data to generate different meshes, used during the display phase. These algorithms can use the particle distribution information to estimate irradiance at a surface point. This irradiance can be used in a raytracer. The other type of algorithm directly use particle data during rendering.

The algorithms also differ in their goals. Some of the mesh generator algorithms can render realtime walkthroughs in an environment, and are used in commercial applications, but they are less physically correct. The other algorithm has a big computation time, but generates more realistic pictures.

Mesh generator algorithms

In these type of algorithms we use the previously generated hit points on the surfaces with an associated incoming power. For these algorithms we must store the power of the particle, the surface which the particle hit and a local coordinate on the same surface.

We are interested in the diffuse radiance of a surface. A Lambertian surface has a constant BRDF (R/π) for all incoming/outgoing directions, where R is the reflectance. This implies, that a Lambertian surface will have a constant surface radiance for all incoming/outgoing direction pairs. On a Lambertian surface at (u,v) coordinates, we have a surface reflectance $R(u,v)$ and an irradiance $H(u,v)$. The radiant exitance $M(u,v)$ of the surface at (u,v) is $R(u,v)\ H(u,v)$. The radiance is $L(u,v)=M(u,v)/\pi$, or it can be expressed like $L(u,v)=R(u,v)H(u,v)/\pi$. This equation implies that we can store the irradiance and the reflectance of the surface independently, and later reconstruct the radiance.

At this point, we have to generate surface meshes from the irradiance presented by the photon hits. This is a density estimation problem, where we have non-uniform random samples. To estimate this photon density, there are different classes of density estimation algorithms have been used:

- Histogram methods: The surface is subdivided into buckets in which the number of photons and/or their accumulated energy is stored.
- Nearest neighbour methods: The density at a point is estimated by dividing the power of the nearest N neighbours (usually it is a fixed number) by the area of a region (centered at the point) containing these photons.
- Kernel estimators: The density is estimated as spatially spread energy distributions.

Figure 3 illustrates these three different types of estimation techniques.

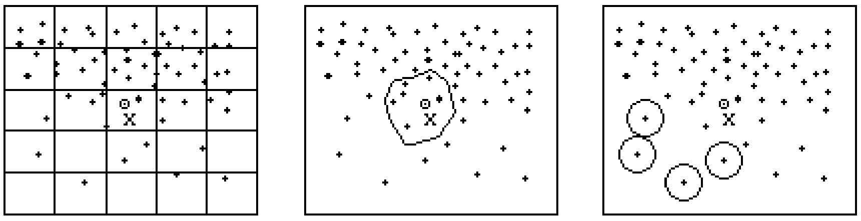


Figure 3: Different methods to estimate the irradiation at X from the distribution of the photon hits. On the left: Histogram technique, in the middle: the nearest neighbour method, on the right: there are few photons with their kernels, shown from above.

Kernel estimators

We have a list of hit points for each surface, and we have to estimate the irradiance on the surface. The irradiance is represented by these hits, where a finite amount of energy strikes an infinitely small area.

In one dimension the density function of the irradiance can be represented by taking n samples (x_i):

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \phi_n \delta(x - x_i)$$

where ϕ_n is the power of the nth photon. We know that the irradiance is a smooth function, so to place a spike at every photon hit would not be a good idea. Instead we use smooth kernel functions and replace the delta functions with $k_1(x - x_i)$, where k_1 has a unit volume. A kernel function spreads the energy of the photon over its surrounding area. These kernels should be centered at the origin, have a non-zero region, and be “lump” shaped. Figure 4 shows an example for a two dimensional kernel function.

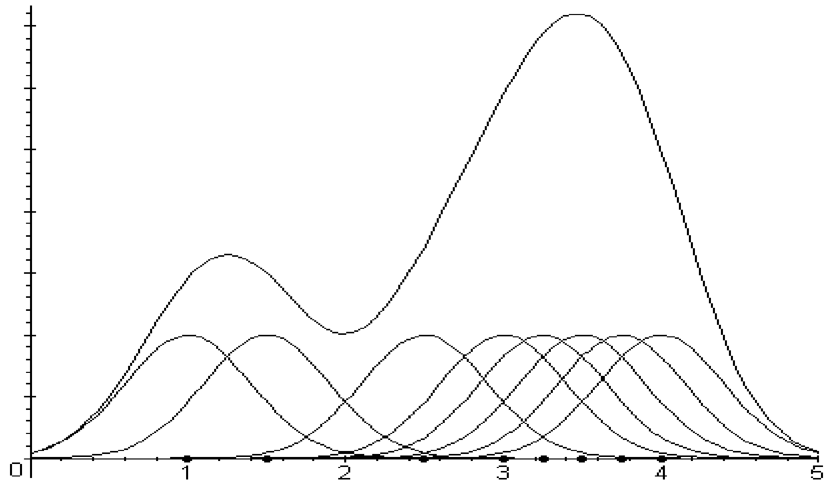


Figure 4: Example of two dimensional kernel functions. On the diagram each kernel has the same volume

In two dimension, the irradiance function can be estimated:

$$H_i(x) = \frac{1}{n}\sum_{j=1}^{n_i} \phi_n \delta(x - x_j)$$

where x_j is the position of the jth point. If we replace the delta functions with n_i kernel functions k_j:

$$H_i(x) = \frac{1}{n}\sum_{j=1}^{n_i} \phi_n k_j(x - x_j)$$

The kernel functions have the conflicting requirements of being narrow enough to capture detail, and wide enough to eliminate the randomness caused by the non-uniform sampling presented by the hit points. We can use a scaling parameter h to widen or narrow the filter. We have to preserve the volume of the kernels, so we increase the height:

$$H_i(x) = \frac{1}{h^2 n}\sum_{j=1}^{n_i} \phi_n k_j\left(\frac{x - x_j}{h}\right)$$

Examples for kernel functions can be found in [1], [2].

To generate meshes from the estimated irradiance, we can sample $H(u,v)$ at a finite set of locations and use some polinomial elements to interpolate between these values. We can use these polinoms during rendering, or we can generate meshes for the surfaces describing the irradiance.

An importance sampling technique using the particle distribution

To evaluate the rendering equation, Monte Carlo integration techniques can be used. In these methods we sample the integrand at random directions. Random sampling is very inefficient, because we have a very small chance to pick a direction where significant amount of light can come from. It would be a good idea to sample the integrand from directions contributing most of the outgoing radiance, so another algorithm, importance sampling can be used. The basic idea behind importance sampling is to place more samples where the integrand function is larger. The sampling can be described with:

$$\int_V f(z)dz = \int_V \frac{f(z)}{p(z)} p(z)dz = E[\frac{f(z)}{p(z)}] \approx \frac{1}{M}\sum_{i=1}^{M}\frac{f(z_i)}{p(z_i)} \pm \frac{\sigma}{\sqrt{M}},$$

where $p(z)$ is a probability density in V, the z_i points are selected according to this probability density, and σ is the variance. The variance can be minimized if $p(z)$ is proportional to the integrand $f(z)$.

A path tracer works in the following way: we generate a starting direction from the camera. We examine whether a surface is hit or the particle leaves the scene. If a surface has been hit, we generate a new direction. In case of importance sampling this new direction depends on some kind of knowledge about important directions. We continue this process until a lightsource has been reached or the photon leaves the scene. At each reflection point we have to scale the calculated radiance according to the probability distribution of the importances.

If we examine the rendering eqation:

$$L_r(x,\Psi_r) = \int_{all \Psi_i} f_r(x,\Psi_r,\Psi_i) L_i(x,\Psi_i)\cos\theta_i d\omega_i,$$

we can see that an impotance sampling algorithm can use the physical characteristic of the surface (its BRDF) to generate the important directions. With the outgoing direction and the BRDF we can select the incoming directions which carries most of the radiance. This algorithm does not need to have knowledge about the global illumination at the surface, which is also part of the rendering eqation.

We can use the particle data to drive an importance sampler in a path tracer algorithm. For this algorithm the important properties of a particle are its power, its position in the scene and its incoming direction.

How to generate important directions

We are interested in generating directions which contribute most of the outgoing radiance at x. We estimate the incoming radiance using the generated particle distribution in the scene. This importance sampling technique is presented in [4]. At a surface point x we construct a hemisphere or a sphere (if the surface is diffusely transparent) which will describe the important sampling directions.

We build this importance map by the following algorithm:

- Initialise the (hemi)sphere to zero in each region
- Find the N nearest neighbour of the point currently hit. This N is a fixed number during the algorithm.

•Assume that each neighbour has been reflected at x. For each neighbour, calculate the product of the incoming power of the particle and the BRDF. The BRDF is evaluated in the direction of the sampling ray, and the direction of the incoming particle.

•Add this value to the region which contains the particle’s incoming direction.

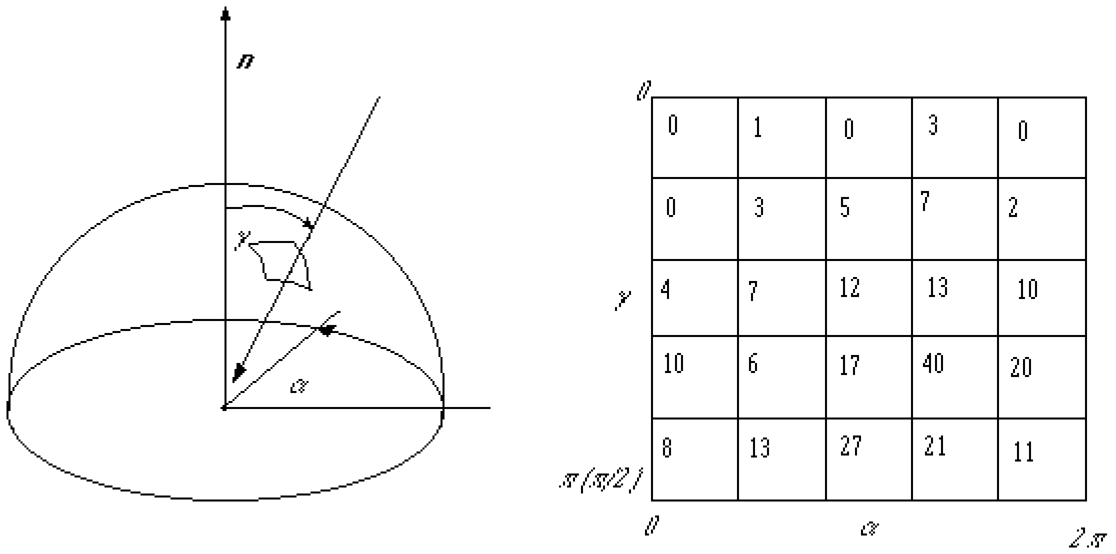
•When every particle’s contribution is added to the sphere, distribute a small portion of the total importance in the sphere among each region. This operation is important, because only this way can the algorithm generate sampling rays in directions where no particle came from.

•Generate a new sampling direction.

•Scale the returned radiance. Because importance sampling is used, we have to scale the radiance returned by the sampling ray. The scaling factor is:

$$s = \frac{\text{Total importance in the sphere}}{(\text{Importance of the region}) * (\text{Number of regions})}$$

Figure 5 shows a hemisphere, and an importance map. To decide how many regions should an importance map have, we have to run simulations, and measure the quality of the images and the computation times.



0	1	0	3	0
0	3	5	7	2
4	7	12	13	10
10	6	17	40	20
8	13	27	21	11

Figure 5: Left: a hemisphere with an incoming photon direction. Right: an importance map

To generate an outgoing sampling direction, we have two possibilities. The first method is based on mip-mapping. We demonstrate this algorithm in a case where we have just four areas with importances I1, I2, I3 and I4. This transformation decomposes a rectangle of size A x B to four subrectangles having an area proportional to the importances of the original rectangles. Then we generate a uniformly distributed random point in the transformed rectangle. We find the original rectangle for this point and scale it’s coordinates to represent a point in the original rectangle. Figure 6 illustrates this method.

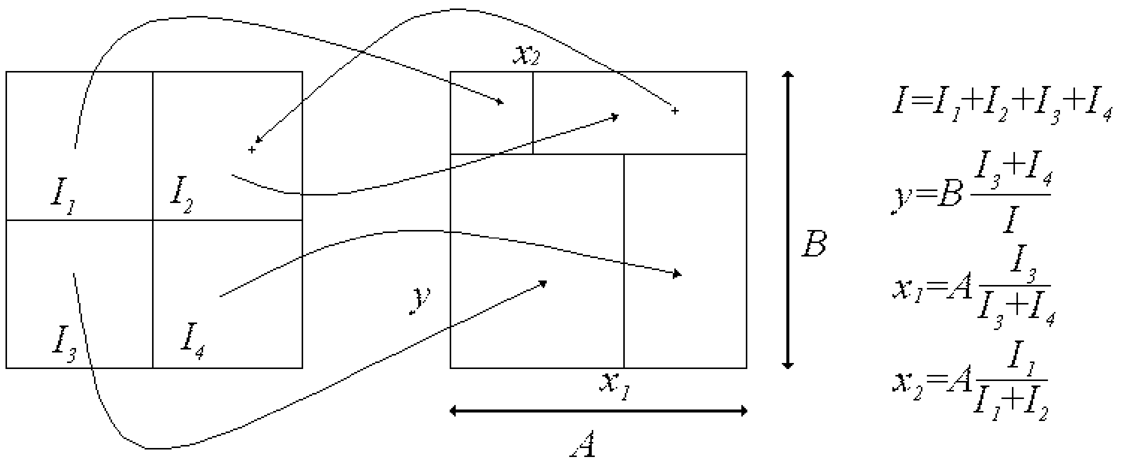


Figure 6: The mip-map based generation of a sampling direction

The second method is a kind of a linear inversion technique. This method takes advantage of the fact that the inportance map is discretized, thus it can be considered as a one-dimensional, row continous array. The algorithm selects an element of this array proportional to its importance. We sum all the importance weights of the areas and generate a random number in the range of $[0, I]$, where I is the sum. Then we scan the rows of the importance map and sum up the importance values. When this sum becomes greater than the generated random number, we select the last region whose weight was added to the sum. Then we can generate a random point in the selected region. Figure 7 shows how this algorithm works.

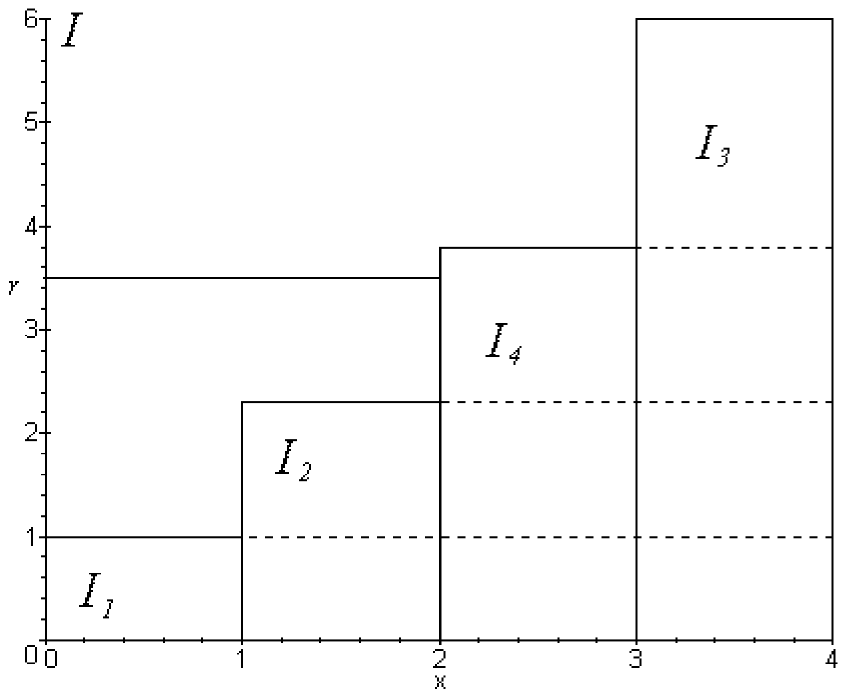
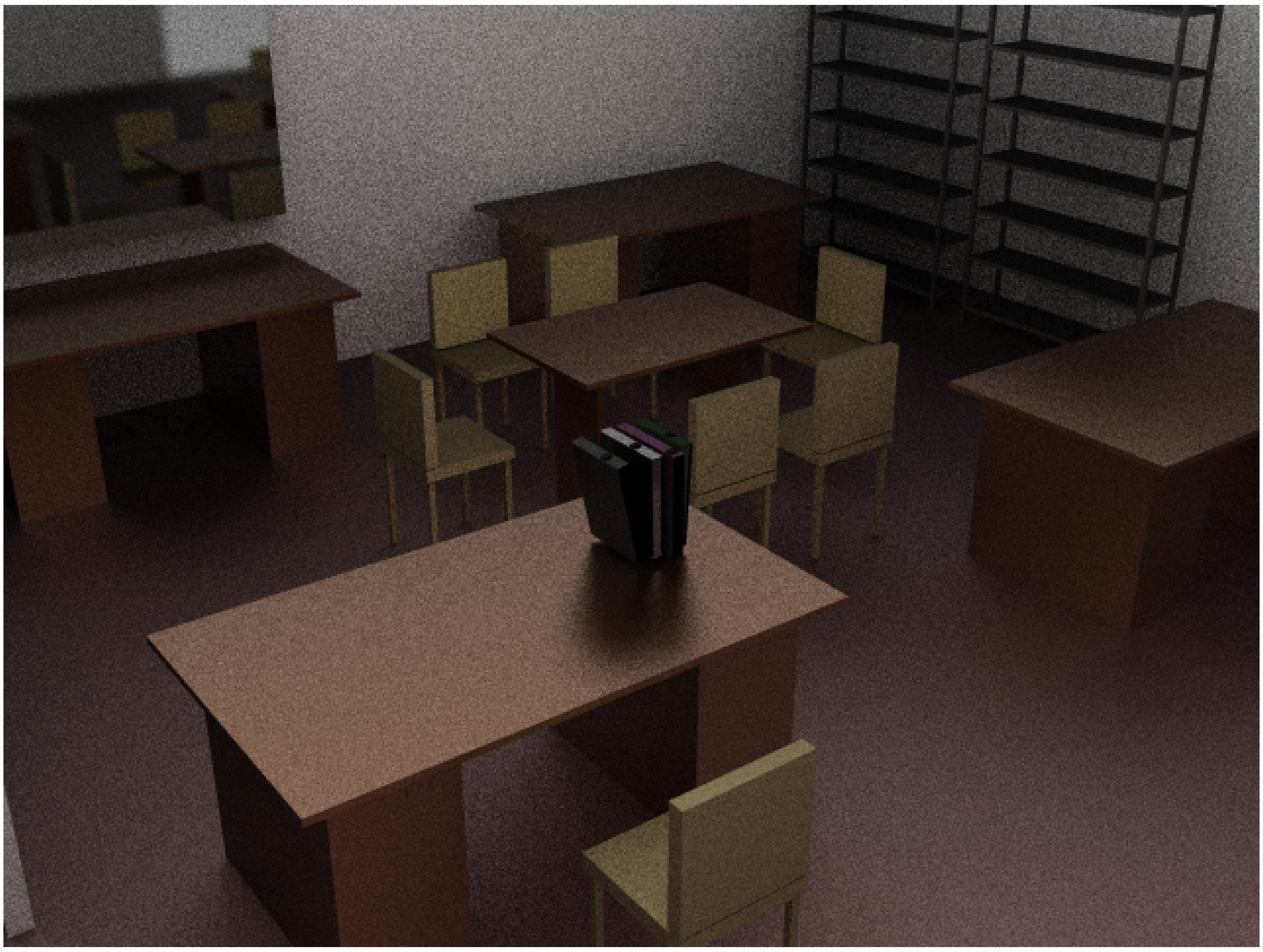


Figure 7: Selecting a region with random number r.

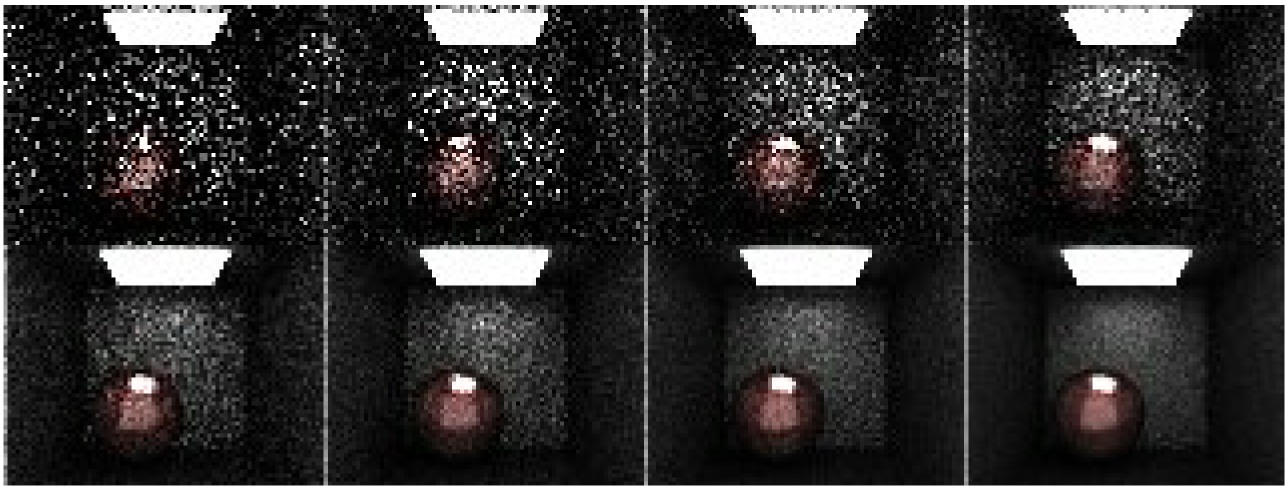
Our work

We currently implemented a particle tracer and an inportance sampler. The importance sampler can generate directions using only the BRDF of the surface or using the data provided by the particle tracer. The implementation was easy using our library we are developing.

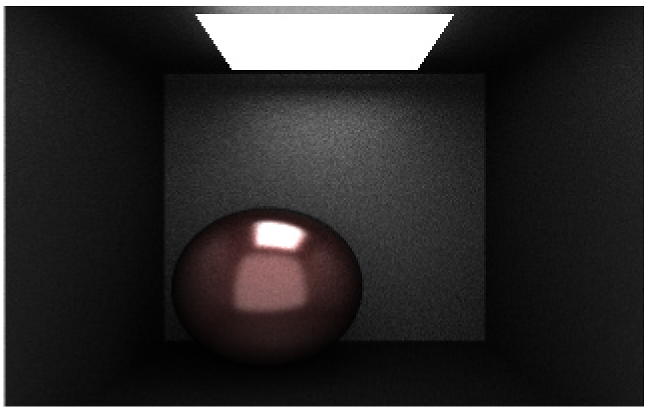
These scenes are the test scenes for our library. There is an empty office room, and another one with books on the table and a mirror on the wall. We have a box with one lightsource on the ceiling and one behind the camera.



An office room with mirror and books



On this picture there is the box scene rendered with different number of samples taken at each pixel. The number of samples from left to right are: 1, 2, 4, 8, 16, 32, 64 and 128.



A bigger version of the box.

Acknowledgements

This work has been supported by the National Scientific Research Fund (OTKA), ref.no. F015884, and by the Austrian-Hungarian Action Fund, ref.No.: 29ö4 and 32öu9.

References

[1] Peter Shirley, Bretton Wade, Philip M. Hubbard, David Zareski, Bruce Walter, Donald P. Greenberg: Global Illumination via Density-Estimation, *Eurographics Rendering Workshop*, 1995.

[2] Wolfgang Stûrzlinger, Rui Bastos: Interactive Rendering of Globally Illuminated Glossy Scenes, *Eighth Workshop on Rendering (Eurographics)*, 1997 June.

[3] Karol Myszkowski: Lighting Reconstruction Using Fast Adaptive Density Estimation Techniques, in *Rendering Techniques '97,* Springer-Verlag, pp. 251-262, 1997.

[4] Henrik Wann Jensen: Importance Driven Path Tracing using the Photon Map, in *Rendering Techniques '95*, Springer-Verlag, pp. 326-335, 1995.

[5] László Szirmay-Kalos, Balázs Csébfalvi, Werner Purgathofer: Importance-driven quasi-Random walk solution of the rendering equation, Winter School of Computer Graphics, 1998

[6] László Szirmay-Kalos: Theory of three-dimensional computer graphics, Akadémiai Kiadó, Budapest, 1995.

[7] László Szirmay-Kalos, Tibor Fóris, Werner Purgathofer: Quasi-Monte Carlo Global Light Tracing with Infinite Number of rays, Winter School of Computer Graphics, 1998

[8] László Szirmay-Kalos, Tibor Fóris, Werner Purgathofer: Non-diffuse, Random-walk Radiosity Algorithm with Linear Basis Functions, Journal of Machine Graphics and Vision, 1998 May

[9] László Szirmay-Kalos, Tibor Fóris, László Neumann, Balázs Csébfalvi: An Analysis of quasi-Monte Carlo Integration Applied to the Transillumination Radiosity Methods, Computer Graphics Forum, Vol 16, No 3, 1997

[10] László Szirmay-Kalos, Gábor Márton: Construction and Analysis of Worst-case Optimal Ray-shooting Algorithms, Computers and Graphics, 1998