Computer Graphics Project

Distributed Ray Tracing

Jakub Cupisz
Michael Friis Lippert

Abstract

As a topic of our project we have chosen Distributed Ray Tracing. It uses Monte Carlo integration to solve the rendering equation.
We have extended our Photon Mapping exercise.

Idea

Ray traced images are overly sharp, because ray directions are determined from the geometry. We can distribute the directions of the rays according to the analytic function they sample, this will incorporate fuzzy phenomena into the solution.
Basic idea as proposed by Cook [COOK1] was that once we accept the cost of oversampling in space, a few effects can be easily caught with no additional rays.
Those effects are:
- depth of field - sampling camera lens area
- gloss - sampling the reflected ray according to the specular distribution function
- translucency - sampling transmitted ray
- penumbra (soft shadows) - sampling the solid angle of the light source
- motion blur - sampling in time

We have implemented four first effects (DOF, gloss, translucency, soft shadows) and we hope to add the last one in the near future. Soft shadows has been added very recently and hasn't been tested thoroughly in more complex scenes then scene3.
When running the program you can pass as a first parameter number from 1 till 3 to choose from scenes made by us.

Monte Carlo integration

Random points with some distribution are used to find and approximate value for the integral. For an integral I of a function f over some space S, estimate of I can be generated by using set of random points e_1 through e_n, where each e_i is distributed according to a probability density function pdf:
I = Sum for i = 1 to N (f(e_i) / pdf(e_i))

It seems to us that a difficult question in any application of Monte Carlo integration is what probability density function should be used to generate sample points on the domain of integration. We report our findings in the following sections.

Depth of Field

Depth of field has been implemented in DOFCamera class. Instead of the model of pinhole camera we have used the camera with thin lens.
We have placed the lens between the image plane and the scene. Image plane is an equivalent of the pixels in the screen but in the world coordinates.
For the given pixel P in the screen we determine the pixel on the image plane P', then we shoot a ray from P' through the centre of the lens (camera_position) and intersect it with focal plane to get a point Q. Then a random point P2 on the lens is chosen (we are using rejection sampling to get the point in the given radius). Lens radius is calculated from the focal distance and the given aperture. A new ray is shot from P2 in the direction of Q. We repeat that for raysPerPixel times (usually 8 or 64). Colours collected with this method for each pixel are averaged uniformly.

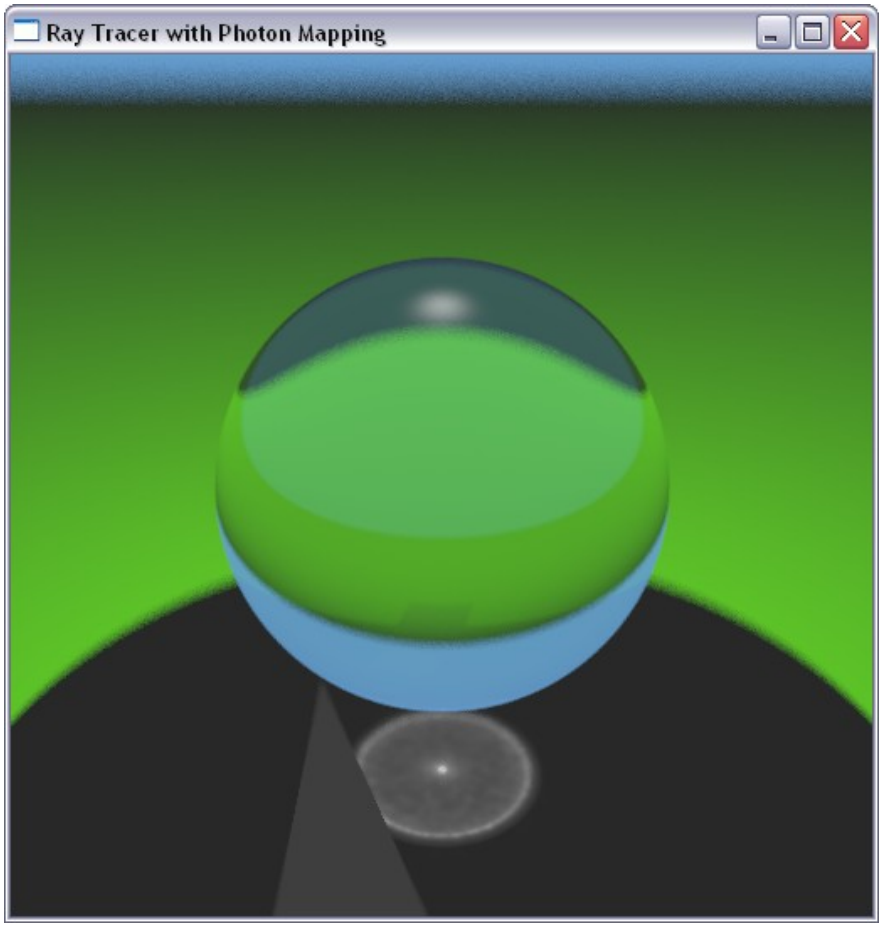

Figure 1: Depth of Field shown in a scene with transparent sphere.

Gloss

Mirror-like reflections are not that often in the real life. We usually perceive blurred ones.
To catch this phenomena we sample Phong-like BRDF. We use the probability scattering function for a direction as proposed in [SHIRLEY1]:
pdf(theta, phi) = (N + 2 / 8 * PI) * (cos(theta/2)^N)
where theta is the angle relative to the ideal reflection direction, phi is an angle around the reflection direction and N is Phong-like exponent.
A scattered direction:
(theta, phi) = (2 * arccos[(1 - u_1)^(1/(N + 2)), 2 * PI * u_2)
where u_1 and u_2 are random numbers on [0,1].

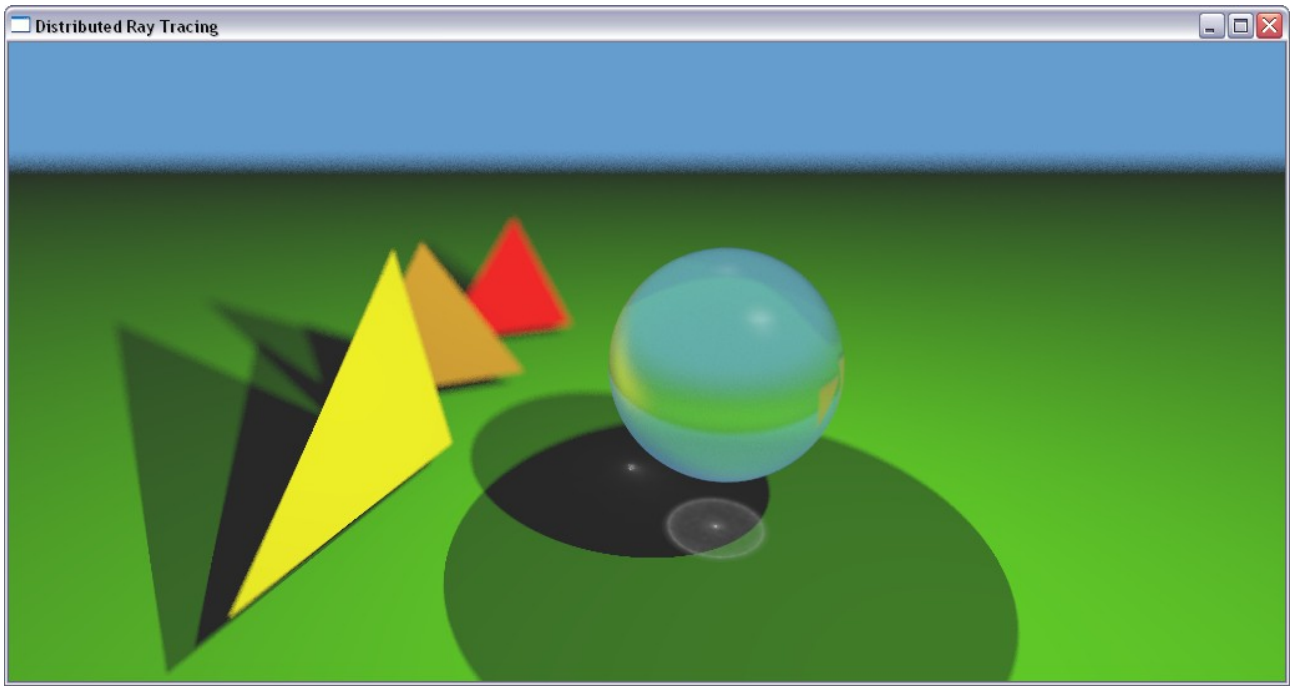

Figure 2: Scene1: Depth of Field and Gloss effects. Sphere has a k_reflected = k_transmitted = 0.5

Translucency

Here we perturb ray directions in similar manner like in the gloss. No particular pdf function was given in various papers, so we are using the one described in [LAWRENCE1]. It is very similar to the gloss, but computes a little faster and gives also nice results.
pdf(theta, phi) = (N + 1 / 2 * PI) * (cos(theta)^N)
(theta, phi) = (arccos[(u_1)^(1/(N + 1)), 2 * PI * u_2)

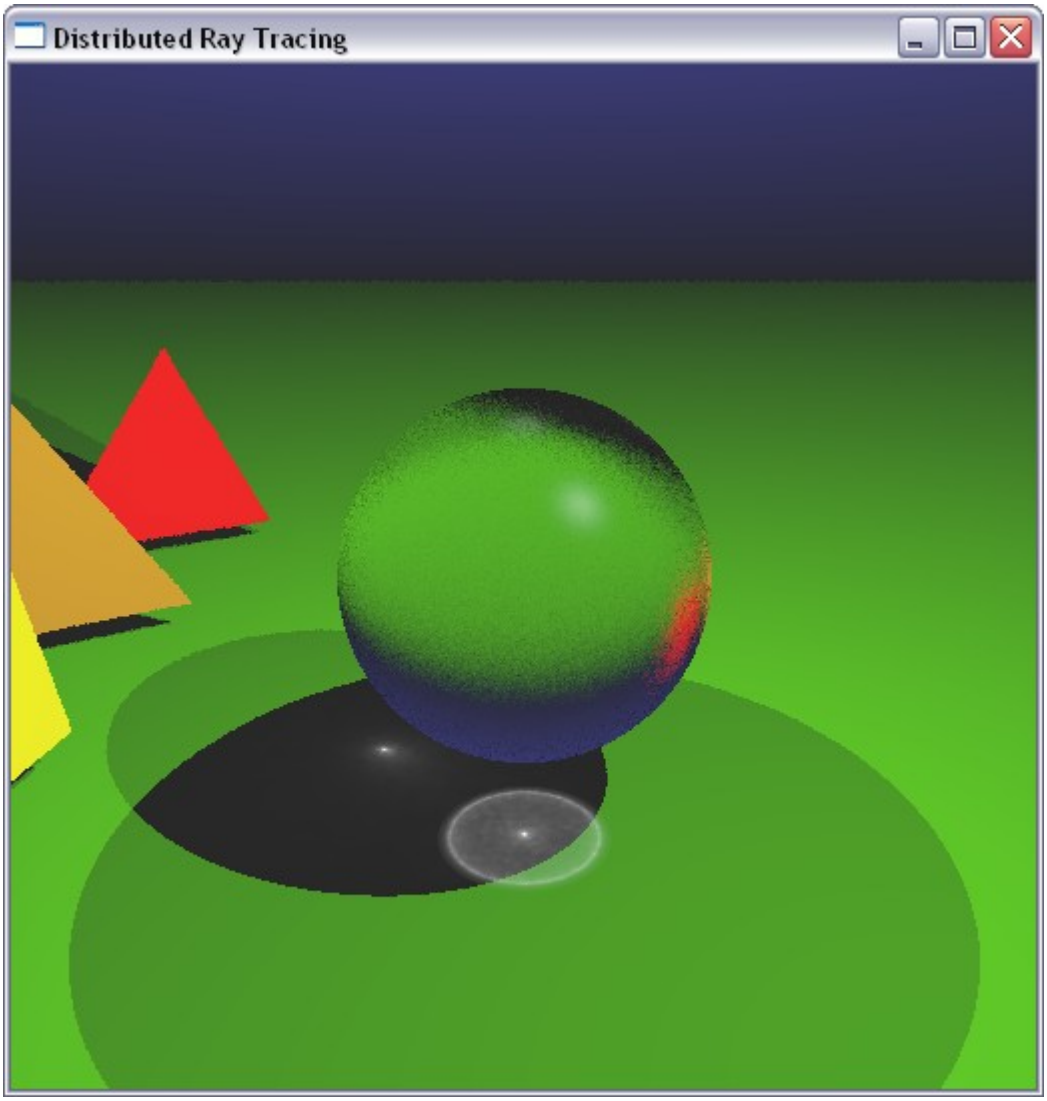

Figure 3: Translucency on a sphere from Scene1. Blue plane instead of just background color in the sky.

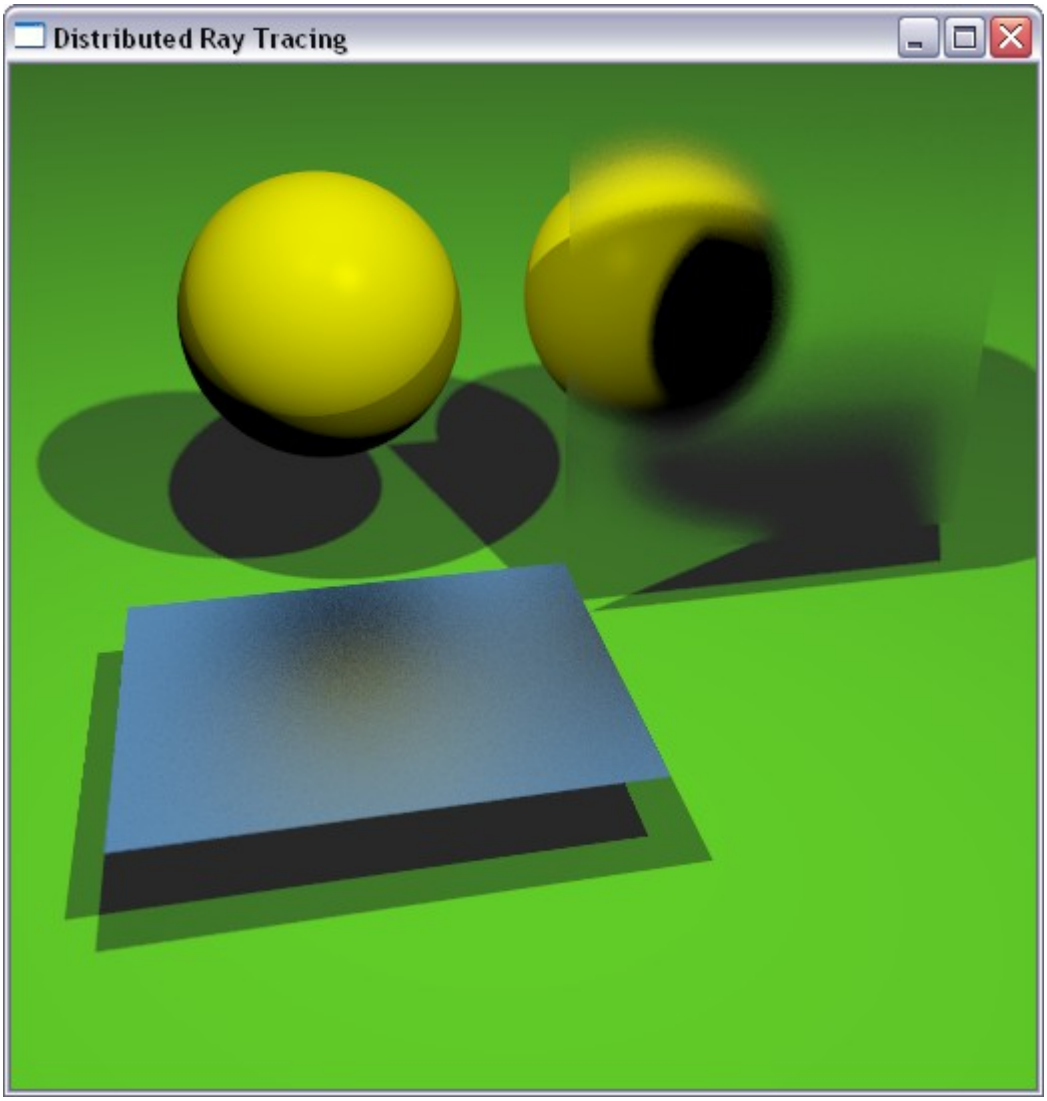

Figure 4: Scene2: Gloss and Translucency.

Soft Shadows

We are using sphere luminaries for making soft shadows.
For testing whether point x is in the shadow we use a shadow ray slightly perturbed around the direction of the light source. We have used the probability density function, which describes the "probability" of choosing a particular point on the sphere, from [SHIRLEY2]. It takes into account: the distance between point x in the scene and x' on the sphere, the angle between vector xx' and the normal of the sphere in the point x' and the ratio between light radius and the distance between x and light.
But there was no particular explanation on how to apply it. We have done it then as follows:
when point x is in the shadow set the lightIntensity coming onto the point to: lightIntensity * (1 - 1/pdf).
This gives a soft shadow but with dark centre, when the light is close, and more crisp shadow, but less dark when the light is far away.
Scene 3 was used to produce images.

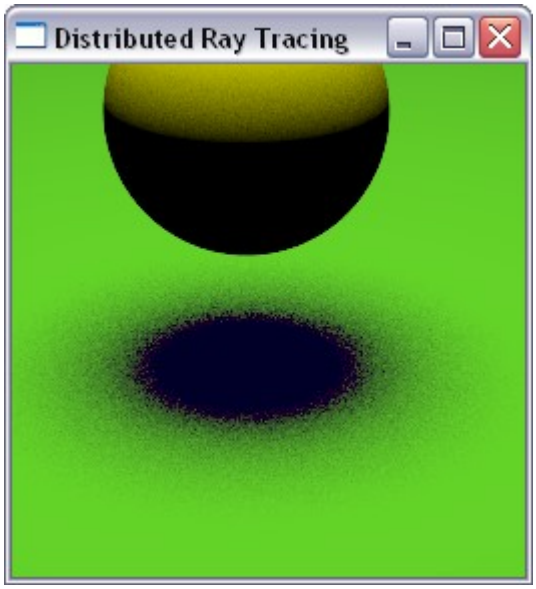

Figure 5: Scene3: Penumbra. Light position y = 2. Soft shadow but with dark center.

Figure 6: Scene3: Penumbra. Light position y = 4.

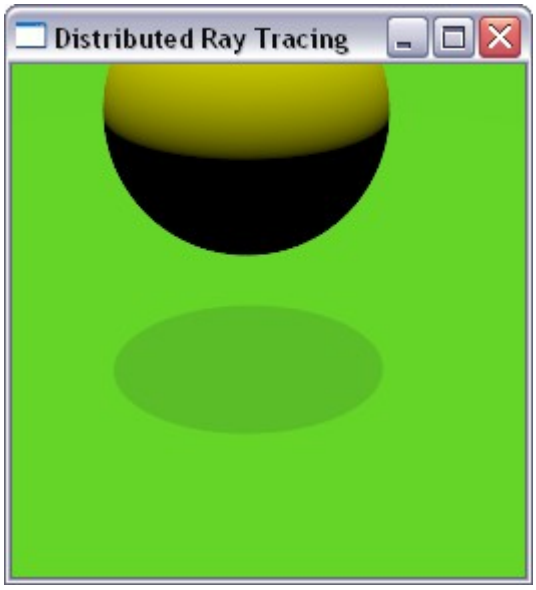

Figure 7: Scene3: Penumbra. Light position y = 15. More crisp shadow, but less dark.

Random Numbers Generation

We looked for some optimization. The first thing that we encountered when searching in the internet was random numbers generation.
We are using Mersenne Twister, it has the period of 2^19937-1 and the order of equidistribution. And it's way faster then rand() in generating double random values from [0,1]. We have notice almost 40% speed-up for the early version of Scene2.
We have used the implementation from [MERSENNE].

Summary

We are satisfied with the project. It was nice to finally get hands on the code that produces nicer images then ordinary ray tracing. It was also a pleasure to extend it - when the basic idea has been understood, other futures have been added easily (not without slight implementational problems). Please look at the images and judge by yourself.

Bibliography

[COOK1] : Cook et al, "Distributed Ray Tracing",
[SHIRLEY1] : Shirley, Wang, "Distribution Ray Tracing: Theory and Practice",
[LAWRENCE1] : Jason Lawrence, "Importance Sampling of the Phong Reflectance Model",
[SHIRLEY2] : Shirley, Wang, "Monte Carlo Techniques for Direct Lighting Calculations",
[MERSENNE] : , Mersenne Twister Random Number Generator, ,
http://www-personal.umich.edu/~wagnerr/ChE.html