Advanced Computer Graphics

Projet: Bidirectional Path Tracing

Lauren De Meyer
François Fonteyn
Fabian Brix

Professor:
Prof. Mark Pauly

December 2013

Step 1 : Implementing refractive and reflective materials using the fresnel formula

Fresnel reflection We use the fresnel equations to compute the fraction of light reflected from the surface following Section 8.2 of the book "Physically based Rendering" [1] The physically accurate fresnel equations depend on the polarization of light. In physically based rendering however the assumption is made that light is unpolarized simplifying the Fresnel reflectance to the average of parallel and perpendicular polarization terms.

$$F_r = \frac{1}{2}(r_\perp^2 + r_\parallel^2)$$

$$r_\parallel = \frac{cos(\theta_i) - \eta cos(\theta_t)}{cos(\theta_i) - \eta * cos(\theta_t)}$$
$$r_\perp = \frac{\eta cos(\theta_i) - cos(\theta_t)}{\eta cos(\theta_i) + \cos(\theta_t)}$$

Computing the transmission angle In order to compute these equations we need the transmission angle according to the surface normal at every point we are tracing. Given the incident angle with respect to the surface normal θ_i we can compute the sine of the transmission angle using Snell's refraction law.

$$\frac{sin(\theta_t)}{sin(\theta_i)} = \frac{\eta_i}{\eta_t} = \eta$$

The identity $sin(\theta_t)^2 + cos(\theta_t)^2 = 1$ gives us the expression for the transmission angle: $\theta_t = \arcsin(\frac{\eta_i}{\eta_t}\sqrt{1 - cos(\theta_i)^2})$.

Detecting the incident side Before we compute the transmission direction it is essential to determine if the incident ray hits the medium surface from the inside or the outside. The ray is coming from outside the medium if $cos(\theta_i) > 0$.

Choosing reflection or refraction For each sample ray that reaches the dielectric surface, we must decide whether to reflect or refract it. We do this by comparing a random sample with the Fresnel Reflectance. Since the Fresnel reflectance is the ratio of the power of the light that is reflected, this is also the probability that a ray of light will be reflected.

BSDF For reflection the BSDF-value is weighted with the Fresnel reflectance. For refraction, except for the Fresnel refractance there is also a factor $\frac{1}{\eta^2}$ as a result of Snell's law. Note that the BSDF-function is discrete as it is only nonzero for 2 directions.

Step 2 : Implementation of the bidirectional path tracing

The light path is the first thing to do. We trace it similarly to the eye path, starting at a random point of a random light in a random direction. We also use Russian Roulette for the length of this path to get an unbiased rendering. We trace a new light path before every

eye path. It is like a *precalculation*. Since the light can be seen as a precalculation, we need to keep track of every computed intersection and throughput. Note that in the eye path, the transmittance and the color of the light are taken into account at the end of the path. In the light path, it is the opposite since we start from the light. Here is the step the follow to trace the light path :

1. Choose a random point on a random light of the scene ;
2. Trace a ray starting from the point in a random direction ;
3. Iterate as long as Russian Roulette is passed
   (a) Compute the intersection of the ray with the scene, if any ;
   (b) Update the current throughput ;
   (c) Sample the BSDF to get a new direction and so a new ray.

After this, we produce the eye path in the same way as for normal path tracing with the following extras in every step of the Russian Roulette:

Connecting the two paths is the second step. We try to connect all the points of the light path to the current newest point of the eye path. This requires to make a visibility test to check whether linking these two points is possible or if there is an obstacle in between.

Computing the weights is the third step. Indeed, as we add more contribution to the result, we should weight them accordingly.

Complete light paths are paths such that their last point is the eye/camera itself. This requires specific treatment but is a very efficient way to render caustics among others. Imagine a cornell box with a light on the ceiling and a glass sphere. Let us develop one possible path: the light path starts at the light then goes to the sphere, gets refracted in the sphere, hits the sphere from the inside, gets refracted leaving the sphere and goes to the floor. The probability that this intersection point on the floor in the same one as the intersection with the eye path is very small because when leaving the sphere, the light path has only one direction it can go to. This implies that many light paths will be computed "for nothing". A way to use them is to try to connect the point on the floor to the eye. The problem then becomes that the concerned pixel is - most probably - not the current one. This means that we need to get track of a second image called the light image. At the end of the rendering, we combine them together (our program asks whether you want to do or not in the command line) to get better caustics and other light effects.

Step 3 : Making a representative scene and comparison between bidirectional and unidirectional path tracing

See figure 1. A good scene to illustrate the benefits of bidirectionals path tracing needs of course caustics. Also, if the light source is blocked by one or more objects, the difference in brightness with normal path tracing is very clear. Our final scene is a variation on the Cornell box, in which the light was hidden behind a sphere. Also, extra reflective materials were added to achieve many caustics. Some caustics are seen trough the mirror and this effect shows much less variance with bidirectional path tracing.

References

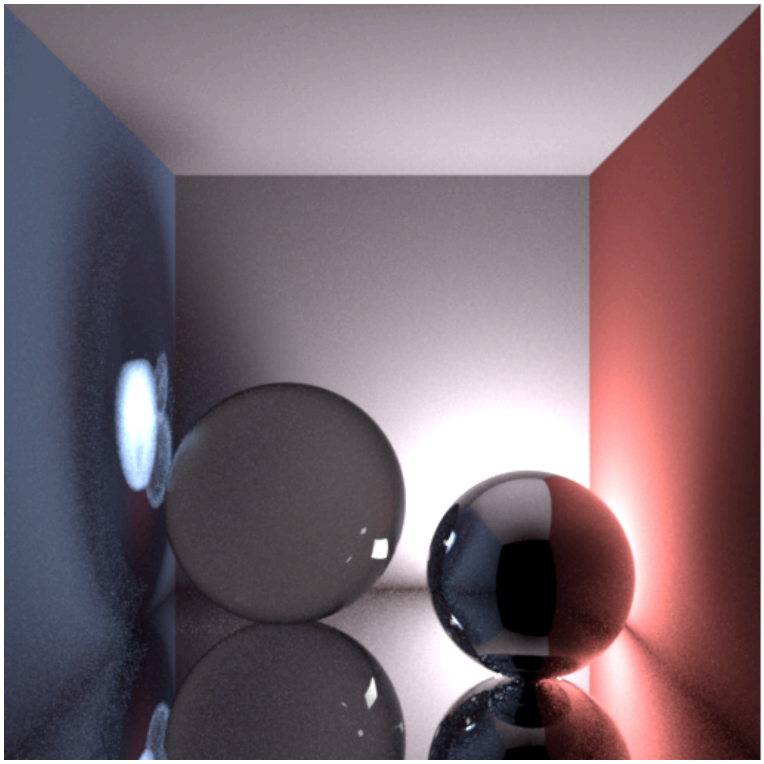
[1] Pharr, M., Humphreys, G., "Physically Based Rendering: From Theory to Implementation", Second Edition

[2] Lafortune, E., Willens, Y., "Bi-directional path tracing", Report cw 197, Department of Computer Science, K.U.Leuven, 1994.
Available at
http://www.cs.princeton.edu/courses/archive/fall04/cos526/papers/lafortune93.pdf

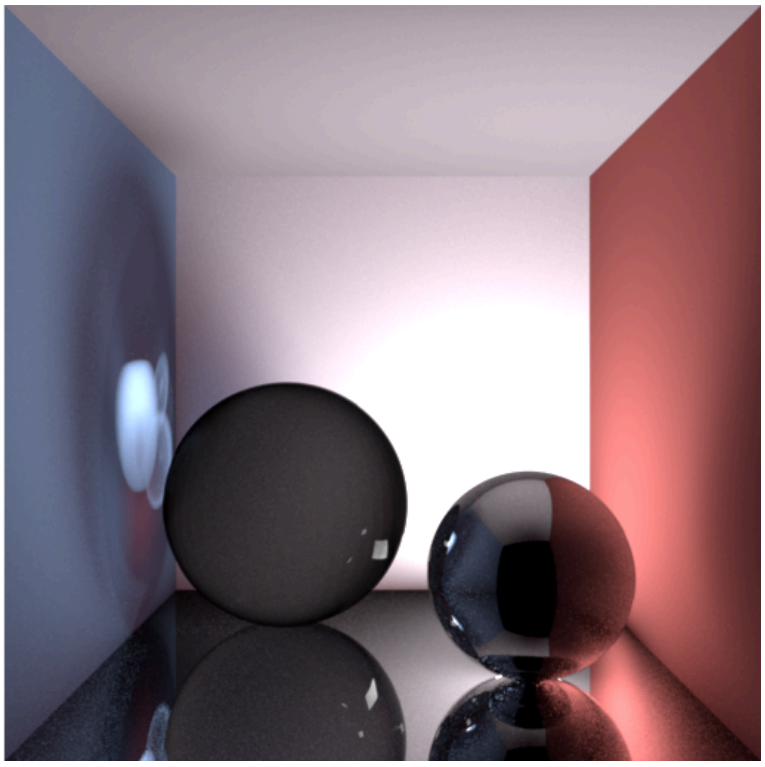
[3] Veach, E., "Robust Monte Carlo methods for light transport simulation", Ph.D. thesis, Stanford University, December 1997.
Available at
http://graphics.stanford.edu/papers/veach_thesis/

Useful link
https://www.youtube.com/watch?v=QhJhVkbCgVU

Images

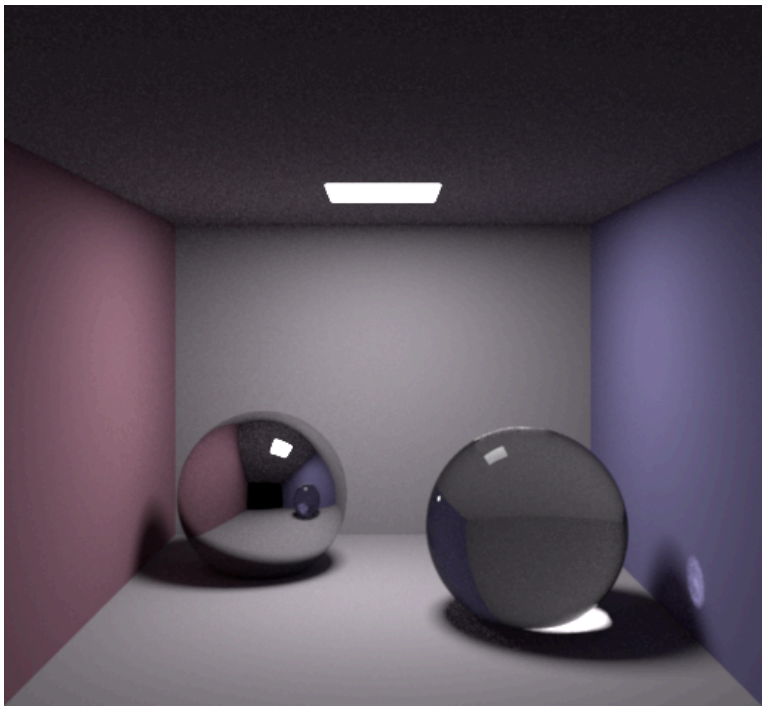


(a) without the light image

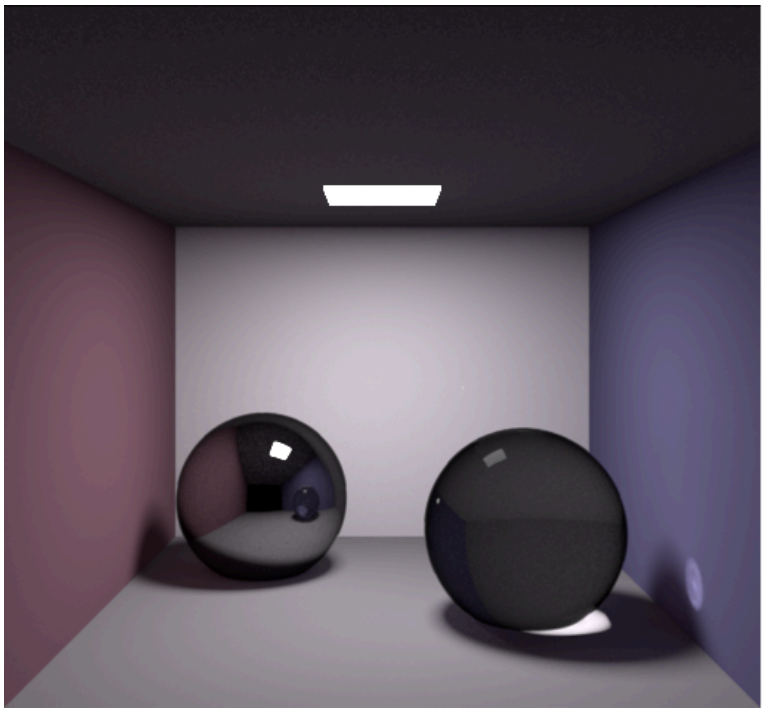


(b) with the light image

Figure 1: Our final scene shows many nice effects like the multiple caustics on the left wall and the many reflections between the right sphere and the floor. The bidirectional path tracing makes the caustics a lot sharper and the room brighter. (5000 samples - 4 hours)

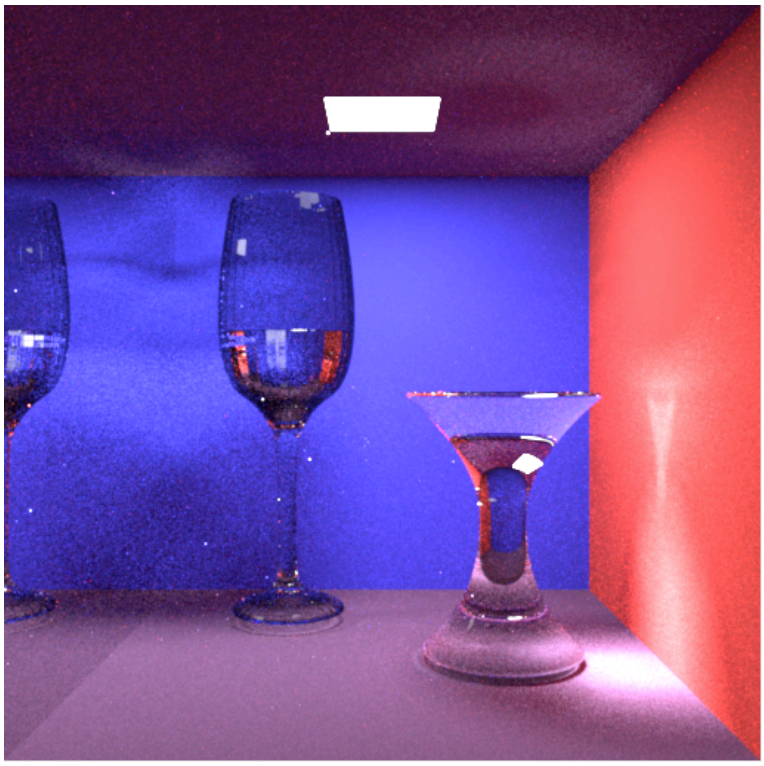


(a) without the light image



(b) with the light image

Figure 2: Replication of the Cornell box to see if all necessary light effects were recreated. (5000 samples - 3.4 hours)



(a) without the light image

(b) with the light image

Figure 3: Scene with new special meshes that give nice and realistic effects when made of glass. Notice the light ring on the ceiling, the glow in the glass of water, the special caustic on the right wall and the caustics beneith the objects. (5000 samples - 3 hours)

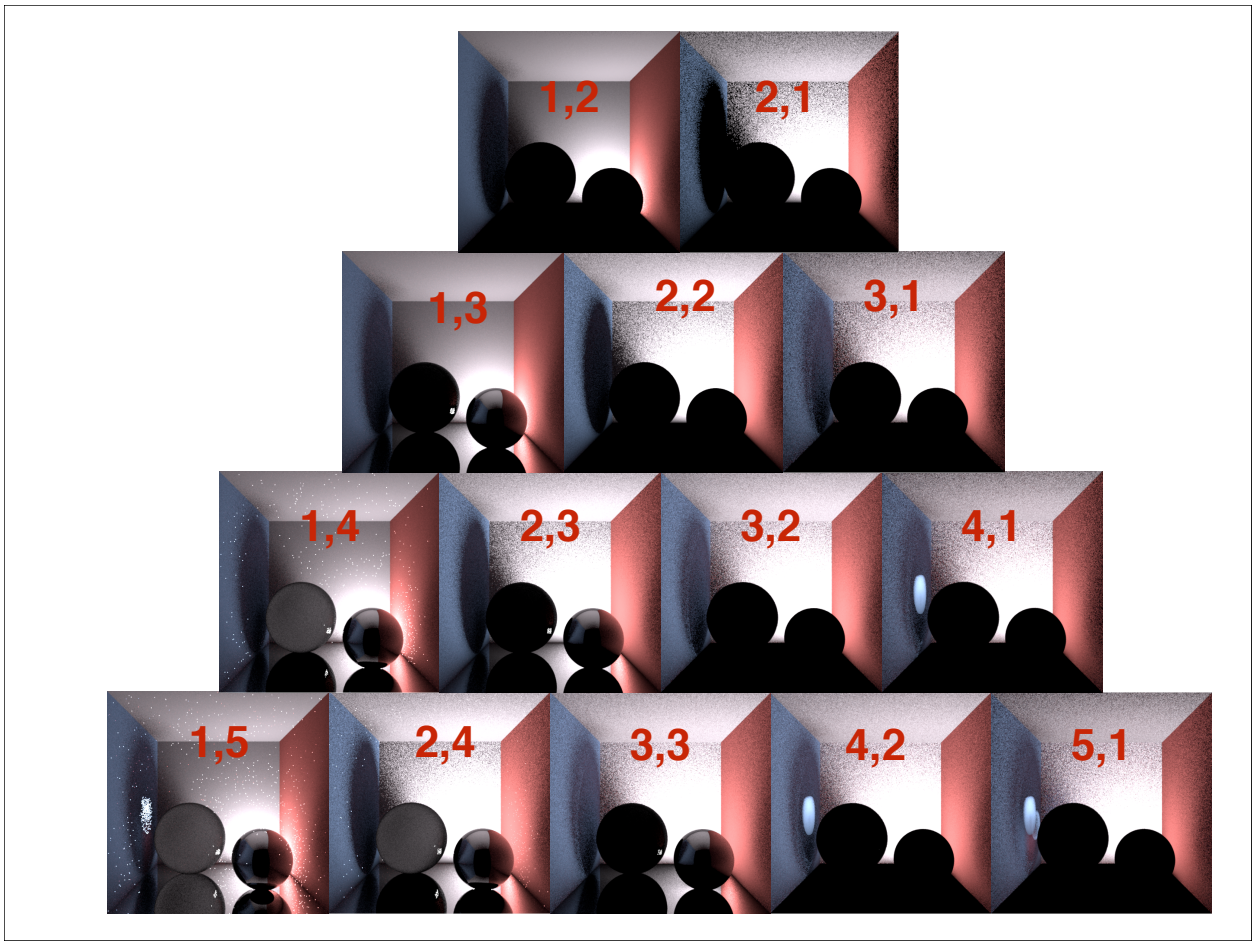


Figure 4: Pyramid of bidirectional path tracing of final scene with (light path length, eye path length) (128 samples)

Videos

Some videos were attached that show adding the light image to a rendering. They show very clearly the difference that bidirectional path tracing makes for caustics, variance and brightness. The scenes were rendered with 500 samples.

- **video 1: Scene in which the ligth must travel far before reaching a dielectric sphere. Shows clearly the difference in brightness.**
- **video 2: Rendering of final scene: lots of reflections which create multiple caustics behind the dielectric material. Accentuates differences for caustics and variance.**
- **video 3: Scene with 3 spheres for which the indices of refraction were inverted (1.5 on outside, 1 on inside). Shows very well the differences in brightness and variance.**

Code

For the new imlemented features, see the following files:

- **path.cpp**
- **dielectric.cpp**
- **common.cpp**
- **perspective.cpp**
- **block.cpp**
- **main.cpp**
- **light_integrator.cpp**
- **depth.cpp (for the header of Li-function)**
- **directional.cpp (for the header of Li-function)**
- **naive.cpp (for the header of Li-function)**

and corresponding header files