Bidirectional Path Tracing

Michal Vlnas*
Supervised by: Pavel Zemcik†

Faculty of Information Technology
Brno University of Technology
Brno / Czech Republic

Abstract

This paper discusses an experimental implementation of the bidirectional path tracing algorithm. The mathematical derivation of the bidirectional estimator using the Monte Carlo method is shown. Moreover, an explanation on how path tracing and light tracing are subsets of the bidirectional approach. Furthermore, a comparison between the bidirectional and the naive path tracing algorithm is shown. Bidirectional path tracing is used to create images of 3D scenes, such that the global illumination is faithful to reality. The naive algorithm is quite inefficient, so many optimized modifications have been developed where one of the most efficient and important extensions is the bidirectional approach. It combines the ideas of shooting and gathering light to create a photorealistic images. Finally, this paper shows a comparison of proposed implementation and state-of-art methods.

Keywords: bidirectional path tracing, global illumination, path tracing, light tracing, rendering equation

1 Introduction

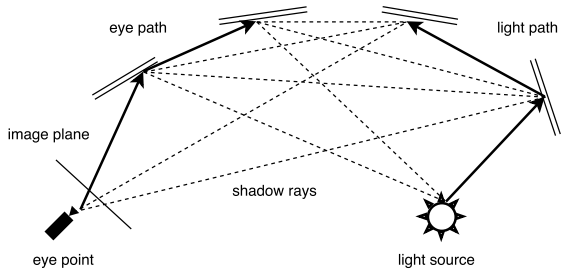


Figure 1: Schematic representation of BDPT

The goal of global illumination algorithms is to render images faithful to reality. Many methods focusing on solving such problems exist. One of such methods is path tracing, which is quite popular nowadays. By shooting rays from the eye into the scene one can compute how much light reaches the eye. When the naive algorithm is used, some scenes can be really slow to render, especially when they contain caustics.

Another possibility to solve global illumination is to use the light tracing algorithm which shoots rays directly from light sources. As it turns out, light tracing can render caustics quite efficiently, but it is quite weak in rendering other details.

A bidirectional path tracer (BDPT) combines these two techniques mentioned above. It shoots rays from the eye and the light towards each other. These paths are interconnected with each other under certain circumstances. The diagram is shown in Figure 1.

2 Background

In 1986, James Kajiya introduced the rendering equation [2] and presented a new form of ray-tracing, called path tracing, opening new solutions for light transport simulation using random sampling with the Monte Carlo method. The basic idea was to sample the flux through the pixels, gathering light by following paths back to the light sources. It was shown, Monte Carlo techniques can make the most general lighting effects but they require a lot of effort and resources.

Later in 1993, Lafortune continued with his work and presented a new technique – bidirectional path tracing [3] which combines the eye path together with the light path, which enormously reduces number of rays with zero contribution. The basic idea is that rays are shot at the same time from the viewing point and from a selected light source. After that, all hit points on their paths are interconnected with a shadow ray, causing the appropriate contribution.

In 1995, Erich Veach with L. J. Guibas published the idea of multiple importance sampling [7] showing that a proper weight function for each traced path can reduce noise very significantly. Also they defined multiple effective weight functions such as the balance or the power heuristic. Another extension of path tracing was presented in 1997 by Eric Veach and L. J. Guibas as metropolis light transport [8], which was focused on re-using already sampled paths.

At the beginning of the 21st century, a new direction

*xvlnas00@stud.fit.vutbr.cz
†zemcik@fit.vutbr.cz

was focused on programmable GPUs. In 2002 Timothy Purcell introduced the first ray tracing based algorithm using the GPU [4].

3 Mathematical representation

This section describes the mathematical formulations of the basic rendering equation, the path tracing, as well as Monte Carlo method in solving integral equations; finally it defines a bidirectional estimator.

3.1 Rendering equation

The rendering equation can be used to describe outgoing radiance on any surface point. The amount of outgoing radiance $L(\mathbf{x}, \omega_0)$ from point $\mathbf{x}$ in direction ω_0 can be computed as the sum of emitted radiance and reflected radiance [2].

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + L_r(\mathbf{x}, \omega_o) \tag{1}$$

Emitted radiance $L_e(\mathbf{x}, \omega_o)$ from point $\mathbf{x}$ in direction ω_o is defined only in light sources, otherwise it is zero.

$$L_r(\mathbf{x}, \omega_o) = \int_\Omega L(\mathbf{x}', -\omega_i) f_r(\mathbf{x}, \omega_i, \omega_0) \, |N_\mathbf{x} \cdot \omega_i| \, \mathrm{d}\omega_i \tag{2}$$

Reflected radiance $L_r(\mathbf{x}, \omega_o)$ is computed as all incoming light in the point $\mathbf{x}$, reflected in the direction ω_o, where $L(\mathbf{x}', -\omega_i)$ represents all incoming radiance from the direction ω_i. To find out how much radiance is actually reflected, it is multiplied by the bidirectional reflectance distribution function (BRDF) $f_r(\mathbf{x}, \omega_i, \omega_o)$. Finally it is multiplied with the dot product between the normal vector in the point $\mathbf{x}$ and the direction ω_i.

3.2 Path tracing

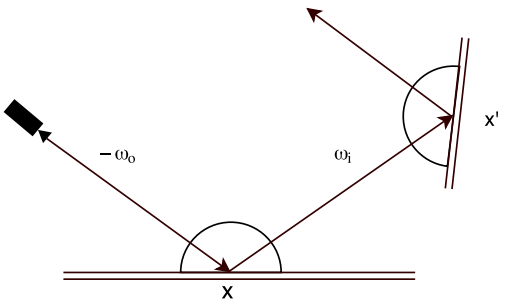


Figure 2: Path tracing schema – focused in point $\mathbf{x}$

Path tracing, illustrated in Figure 2, solves the rendering equation (1) by using Monte Carlo integration. Instead of integrating over the whole hemisphere, this algorithm samples the hemisphere to get the single direction ω_i. The radiance reflected from ω_i is then divided by a probability density function (PDF) of the sampling ω_i, e. g. using a uniform distribution over the hemisphere gives a PDF equal to $\frac{1}{2\pi}$. An intuitive form of hemisphere sampling is illustrated in Figure 3.

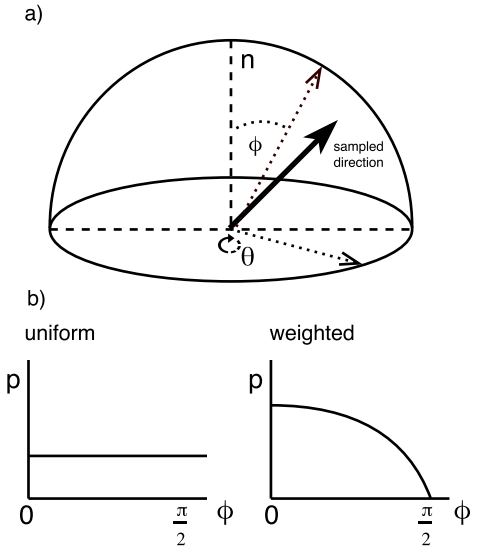


Figure 3: **a)** Illustration of hemisphere sampling. **b)** Uniform versus cosine weighted probability density function of upper hemisphere sampling.

The rendering equation estimated with Monte Carlo integration is the following:

$$L(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \frac{L(\mathbf{x}', -\omega_i) f_r(\mathbf{x}, \omega_i, \omega_o) \, |N_\mathbf{x} \cdot \omega_i|}{p(\omega_i)} \tag{3}$$

Obviously, the previous equation (3) can be evaluated in a recursive manner. A ray can be traced from the camera eye into the nearest hit point $\mathbf{x}$, then sample a new direction ω_i above $\mathbf{x}$ and repeat these steps until a defined maximal path length or by using some kind of termination condition, such as Russian roulette, described in Veach thesis [6].

The significant problem of the algorithm mentioned above is that many paths will never hit a light source, therefore their contribution will be zero. For example, when a point light is used, the probability of hitting the light source is close to zero. This can be solved by separating the direct and the indirect component from (2), such as Shirley et al. [5]. It gives us:

$$L_r(\mathbf{x}, \omega_o) = L_{direct} + L_{indirect} \tag{4}$$

The first part of the right side of the previous equation, representing direct lighting, can be computed as:

$$L_{direct} = \int_A L_e(\mathbf{x}' \to \mathbf{x}) f_r(\mathbf{x}, \overrightarrow{\mathbf{x}'\mathbf{x}}) G(\mathbf{x} \leftrightarrow \mathbf{x}') dA_i \tag{5}$$

where $L_e(\mathbf{x}' \to \mathbf{x})$ is emitted light and $G(\mathbf{x} \leftrightarrow \mathbf{x}')$ is a geometric coupling term, which is described below. It is integrated over the whole area of the light. Indirect lighting

corresponds to (2). $G(\mathbf{x} \leftrightarrow \mathbf{x}')$ is a geometric term which is defined as:

$$G(\mathbf{x} \leftrightarrow \mathbf{x}') = V(\mathbf{x} \leftrightarrow \mathbf{x}') \frac{|N_x \cdot \overrightarrow{\mathbf{x}'\mathbf{x}}||N_{x'} \cdot \overrightarrow{\mathbf{x}\mathbf{x}'}|}{||\mathbf{x} - \mathbf{x}'||^2} \tag{6}$$

where $V(\mathbf{x} \leftrightarrow \mathbf{x}')$ is a visibility function, which is equal to 1 if a ray can be shot directly from point $\mathbf{x}$ to $\mathbf{x}'$ without hitting anything else, otherwise it is equal to 0.

3.3 Applying path integral

As it may be seen, the rendering equation (1), which is integrated over all directions, can be transformed with the path integral formulation of the light, as presented in Veach thesis [6], into a finite equation which is integrated over surface area. So the overall problem can be rewritten as:

$$L_p = \int_D f_j(\bar{p}) dD \tag{7}$$

which represents radiance L_p, that flow through a pixel, where D are all the possible light paths in the scene, $\bar{p}$ is a single light path, an example of the path can be seen in Figure 4, and f_j is a measurement function of light contribution.

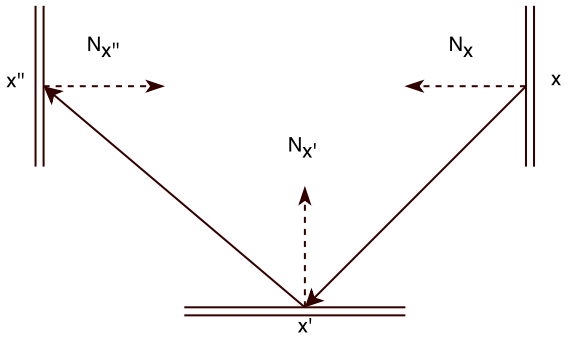


Figure 4: Light path $\bar{p}$: light coming from $\mathbf{x}''$ is reflected in point $\mathbf{x}'$ towards $\mathbf{x}$ (arrows oriented in the direction of tracing)

Also, the previous equation (7) can be approximated using Monte Carlo integration by taking the average of N randomly sampled paths:

$$L_p = \frac{1}{N} \sum_{i=0}^{N} \frac{f_j(\bar{p}_i)}{p(\bar{p}_i)} \tag{8}$$

where $p(\bar{p}_i)$ is the PDF of sampling path $\bar{p}_i$. As written in Veach thesis [6], the PDF is usually given in respect to the solid angle $p(\omega)$, for example, when sampling a new direction using the BRDF. The conversion into a PDF with respect to surface area $p(\mathbf{x})$ is written as:

$$p(\mathbf{x}) = p(\omega) \left(\frac{|N_{\mathbf{x}'} \cdot \overrightarrow{\mathbf{x}'\mathbf{x}''}|}{||\mathbf{x}'' - \mathbf{x}'||^2} \right) \tag{9}$$

Similarly, if we apply Monte Carlo integration on a concrete measurement (4) with conversion to the surface area, then we get:

$$L_p = \frac{1}{N} \sum_{i=0}^{N} L_{p_i} \tag{10}$$

where L_p is a radiance measurement of pixel and L_{p_i} is the radiance of a single sample. After applying the path integral on L_{p_i} we get:

$$L_{p_i} = \sum_{t=0}^{N_E} C_t \tag{11}$$

where C_t is the contribution of a single path of length t and N_E is the maximum path length.

Now let us apply Monte Carlo integration on the contribution C_t, which gives us:

$$C_t = \frac{L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{x}_t})}{p(\mathbf{y}_0)} f_r(\mathbf{x}_t, \overrightarrow{\mathbf{x}_t\mathbf{y}_0}, \overrightarrow{\mathbf{x}_t\mathbf{x}_{t-1}}) G(\mathbf{x}_t \leftrightarrow \mathbf{y}_0) \prod_{i=1}^{t-1} \frac{f_r(\mathbf{x}_i, \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}}, \overrightarrow{\mathbf{x}_i\mathbf{x}_{i-1}}) \left| N_{\mathbf{x}_i} \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}} \right|}{p(\overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}})} \tag{12}$$

The first part of the equation corresponds to the direct lighting from $\mathbf{y}_0$ onto $\mathbf{x}_t$, where $L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{x}_t})$ is the amount of radiance from point $\mathbf{y}_0$ going in the direction towards $\mathbf{x}_t$. Then it is multiplied with the geometric term, defined above and also with the BRDF. The rest of the equation belongs to indirect lighting. It is computed for each point on the path as the product of BRDF and the dot product, divided by the PDF relative to the BRDF.

3.4 Bidirectional approach

This approach combines the strategies of gathering and shooting rays. Gathering rays from a point on a surface refers to path tracing, which is defined in previous equation (12). The principle of shooting rays is called light tracing, which is necessary to be defined to finish the BDPT relation.

As proved in Veach thesis [6], each measurement can be written in form of equation (7) using the path integral framework. Then light tracing, which represents a light path from a point on the surface of a randomly selected light, can be written as:

$$L_p = \sum_{s=0}^{N_L} C_s \frac{\left|\left| P_{pixel} - P_{eye} \right|\right|^2}{cos(\theta)^2 A_{pixel}} \tag{13}$$

where the sum represents path, from point $\mathbf{y}_0$ to the maximum path length N_L, C_s is the single path contribution and the rest of equation is the conversion from flux to radiance. Contribution C_s can be computed as:

$$C_s = \frac{L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})}{p(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})} f_r(\mathbf{y}_s, \overrightarrow{\mathbf{y}_s\mathbf{y}_{s-1}}, \overrightarrow{\mathbf{y}_s\mathbf{y}_{s+1}}) G(\mathbf{y}_s \leftrightarrow \mathbf{x}_0) \frac{W_e(\mathbf{x}_0, \overrightarrow{\mathbf{y}_t\mathbf{x}_0})}{p(\mathbf{x}_0)} \left(\prod_{i=1}^{s-1} \frac{f_r(\mathbf{y}_i, \overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}}, \overrightarrow{\mathbf{y}_i\mathbf{y}_{i-1}}) \left| N_{\mathbf{y}_i} \cdot \overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}} \right|}{p(\overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}})} \right) \tag{14}$$

The first part is the emitted light $L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})$ from the light source at point $\mathbf{y}_0$. The PDF $p(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})$ is probability of selecting a ray in the direction $\overrightarrow{\mathbf{y}_0\mathbf{y}_1}$. Considering a light that has an uniform distribution of the emitted light, then probability density function is:

$$p(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1}) = \frac{1}{A_{light} 2\pi}$$

because $\mathbf{y}_0$ can be sampled over the light area A_{light} in all directions over the hemisphere above the point $\mathbf{y}_0$. $W_e(\mathbf{x}_0, \overrightarrow{\mathbf{y}_t\mathbf{x}_0})$ is the potential of the light ray going straight from the light source to the pixel, which is equal to 1, when the ray is in front of the camera, and when using a pinhole camera, where only one direction exists for each point $\mathbf{x}$. Also when using a pinhole camera $p(\mathbf{x}_0)$ is equal to 1 as well, because only $\mathbf{x}_0$ can be chosen as point.

The next part $f_r(\mathbf{y}_s, \overrightarrow{\mathbf{y}_s\mathbf{y}_{s-1}}, \overrightarrow{\mathbf{y}_s\mathbf{y}_{s+1}}) G(\mathbf{y}_s \leftrightarrow \mathbf{x}_0)$ represents a direct ray from $\mathbf{y}_s$ to the eye $\mathbf{x}_0$. The third part of the equation is the product of: the BRDF, multiplied with the dot product between normal and direction $\overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}}$ and normalized by the PDF.

3.5 BDPT estimator

We have already defined both strategies of ray evaluation. Bidirectional path tracing can now be defined as:

$$L_p = \sum_{s=0}^{N_L} \sum_{t=0}^{N_E} w_{s,t} \cdot C_{s,t} \tag{15}$$

where $C_{s,t}$ is the unweighted contribution of a path with s vertices on the light path and t vertices on the eye path. A visibility check between s$^{\text{th}}$ point on the eye path and t$^{\text{th}}$ point on the light path is a part of the geometric term in contribution function. And finally, $w_{s,t}$ is the weight function.

As it can be seen, evaluation of $C_{s,t}$ is dependent on the eye and light path; therefore, there are four important cases:

- $s = 0, t = 0$ – light is directly visible from the eye, the evaluation can be done with direct path between eye the point $\mathbf{x}_0$ and $\mathbf{y}_0$:

$$C_{0,0} = L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{x}_0}) G(\mathbf{y}_0 \leftrightarrow \mathbf{x}_0)$$

- $s > 0, t = 0$ – this term is the classic light tracing algorithm, defined in (14)

- $s = 0, t > 0$ – this case corresponds to the classic path tracing, which is defined in (12)

- $s > 0, t > 0$ – the final case is the evaluation of radiance, from the eye path t connected with s vertices of the light path, reaching the pixel, defined in the Equation (16).

$$\begin{aligned} C_{s,t} = & \frac{L_e(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})}{p(\mathbf{y}_0, \overrightarrow{\mathbf{y}_0\mathbf{y}_1})} G(\mathbf{y}_s \leftrightarrow \mathbf{x}_t) \\ & f_r(\mathbf{x}_t, \overrightarrow{\mathbf{x}_t\mathbf{y}_s}, \overrightarrow{\mathbf{x}_t\mathbf{x}_{t-1}}) f_r(\mathbf{y}_s, \overrightarrow{\mathbf{y}_s\mathbf{y}_{s+1}}, \overrightarrow{\mathbf{y}_s\mathbf{x}_t}) \\ & \left(\prod_{i=1}^{t-1} \frac{f_r(\mathbf{x}_i, \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}}, \overrightarrow{\mathbf{x}_i\mathbf{x}_{i-1}}) \left| N_{\mathbf{x}_i} \cdot \overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}} \right|}{p(\overrightarrow{\mathbf{x}_i\mathbf{x}_{i+1}})} \right) \\ & \left(\prod_{i=1}^{s-1} \frac{f_r(\mathbf{y}_i, \overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}}, \overrightarrow{\mathbf{y}_i\mathbf{y}_{i-1}}) \left| N_{\mathbf{y}_i} \cdot \overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}} \right|}{p(\overrightarrow{\mathbf{y}_i\mathbf{y}_{i+1}})} \right) \end{aligned} \tag{16}$$

3.6 Weight function

Many ways to create a weight function $w_{s,t}$ are possible. One approach could be to define such a weight function that strictly uses only eye/light paths, however this way is quite wasteful, since it throws away many already sampled paths. A better approach is to use multiple importance sampling.

It is obvious that each path with $s+t$ points can be sampled in $s+t-1$ different ways. Each path should have such weight, that together all weights sum to 1. In E. Veach work [6, 7] it is described, that the most effective way to use multiple importance sampling, in order to create a weight function is with the power heuristic:

$$w_{s,t} = \frac{p_s^\beta}{\sum_{i=0}^{s+t-1} p_i^\beta} \tag{17}$$

where recommended value is $\beta = 2$, according to E. Veach [6].

$$w_{s,t} = \frac{p_s^2}{\sum_{i=0}^{s+t-1} p_i^2} = \frac{1}{\sum_{i=0}^{s+t-1} (p_i/p_s)^2} \tag{18}$$

where p_i is defined as the density for generating path $\bar{x}_{s,t}$ using i sub light path vertices and $s+t-i$ eye sub path vertices:

$$p_i = p_{i,s+t-i}(\bar{x}_{s,t})$$

and p_s is the actual probability with which the path was generated. According to Veach [6], the current value can be set to $p_s = 1$ and the values of the other p_i relative to p_s can be computed using ratio:

$$\frac{p_{i+1}}{p_i} = \frac{\overleftarrow{p_i}(x)}{\overrightarrow{p_i}(x)} \tag{19}$$

According to Gerogiev [1], the power heuristic equation (18) can be split into two parts, determining camera and light weight independently.

$$\begin{aligned} w_{s,t} &= \frac{1}{\sum_{i=0}^{s-1} (p_{i,s+t-i}/p_{s,t})^2 + 1 + \sum_{i=s+1}^{s+t} (p_{i,s+t-i}/p_{s,t})^2} \\ &= \frac{1}{w_{light,s-1} + 1 + w_{camera,t-1}} \end{aligned} \tag{20}$$

The camera and light weights are defined as:

$$w_{camera,i} = \frac{\overleftarrow{p_i}(x)}{\overrightarrow{p_i}(x)}(w_{camera,i-1}+1)$$
$$w_{light,i} = \frac{\overleftarrow{p_i}(x)}{\overrightarrow{p_i}(x)}(w_{light,i-1}+1) \quad (21)$$

where these weights can be evaluated progressively during ray tracing from their own perspective.

4 Implementation

The implementation is focused on light and eye path generation, direct illumination, vertex connections, and also a basic implementation of multiple importance sampling is shown in form of weight functions.

4.1 BDPT algorithm

First, we define the main algorithm, which is the crucial part of the implementation.

Algorithm 1 BDPT

```
1:  function RUN(Scene)
2:      define number of light and eye paths to be proceed
3:      for all light paths do                    ▷ light path processing
4:          GenerateLightSample()
5:          while tracing do
6:              if !Trace then
7:                  break
8:              BRDF ← initialize BRDF on hit point
9:              if BRDF is not delta then
10:                 vertex ← fill light vertex
11:                 store vertex into light path storage
12:                 JoinWithCamera()
13:             if path is too long then
14:                 break
15:             Sample() ← sample new point or end path
16:         store light path length
17:     for all eye paths do                      ▷ eye path processing
18:         GenerateEyeSample()
19:         color ← fill with zero
20:         while tracing do
21:             if !Trace then
22:                 break
23:             BRDF ← initialize BRDF on hit point
24:             if BRDF is not delta then
25:                 color += DirectIllumination()
26:                 for all corresponding light paths do
27:                     color += ConnectVertices()
28:             if path is too long then
29:                 break
30:             Sample() ← sample new point or end path
31:         store color in frame buffer
```

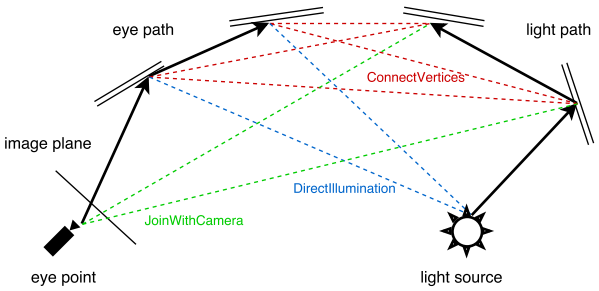


Figure 5: Schematic representation of Algorithm 1

In Algorithm 1, at first, the path count to be processed is defined, which is the same for light and eye paths, because it is important to have only one corresponding path for the eye path. Then in the iteration over all light paths, it is important to sample the first point on the path, from a random light source in the scene. From this point the algorithm traces the path till the end, which can be defined by a hard limit or for example by Russian roulette, or both. The vertices are then joined into the camera, unless the BRDF is a delta function, which is typical for purely specular surfaces, which would create a lot of variance. Sampling of the new point on the light path depends on the hit points BRDF. Different sampling optimization techniques can be used, the importance sampling is used in this implementation. More about this is discussed in Section 4.2.

With the eye path, it is really similar with the light path, with the difference, that for non-delta surfaces it is necessary to compute direction illumination on the hit points and create connections between corresponding light path vertices and the current point. More about this is shown in section 4.5. Also, the eye path sample has its origin on the camera point in direction towards the scene. Each part of the algorithm is illustrated in Figure 5.

From this algorithm, it is obvious that it creates only a single sample per pixel, In practice, it is recommended to run it more times and divide the frame buffer by the number of iterations, to create a less noisy image.

4.2 Light path sampling

Algorithm 2 Generating light sample

```
1: function GENERATELIGHTSAMPLE
2:     light ← choose random light in the scene
3:     prob ← compute probability of light picking
4:     prob ← scale prob with sampling probability
5:     cast ray in random direction from sampled point
```

As shown in Algorithm 2, at first, one needs to choose a random light and then sample a point on its surface. It is substantially important to compute the correct probabilities, in respect to the light count and the light surface probability density function.

4.3 Eye path sampling

The origin of the ray is in the camera, when using a pinhole camera, it is always the same point. The ray is then cast into the selected pixel direction. The count of all eye paths is strictly equal to the pixel count on the image plane.

Algorithm 3 Generating eye sample

```
1: function GENERATEEYESAMPLE
2:     xy ← get XY coords for current path
3:     cast ray from camera into point xy
```

4.4 Direct illumination

As shown in Algorithm 1, direct illumination is computed only when the surface BRDF is no delta function, so it is not evaluated for purely specular surfaces, which must be strictly omitted.

Algorithm 4 Direct illumination

```
1: function DIRECTILLUMINATION
2:     light ← pick a light
3:     occlude test
4:     compute light intensity illuminating shaded point
```

Therefore, the implementation consists only of shadow ray casting, which directly transports light from the source to the point of interest.

4.5 Vertex connections

The connection between vertices is a straightforward implementation of the equation (16), which basically is the product between eye and light path BRDF scaled with geometric coupling term. Also the occlusion test is required to be done, to prevent connecting vertices which are invisible to each other.

4.6 Multiple importance sampling

Obviously, the implementation of (21) still cannot be done with the algorithm defined above. As partly shown by Georgiev [1], one can rewrite this equation into the form:

$$\begin{aligned} w_i &= \overleftarrow{p_i}(x)(vc_i + vcm_i) \\ vc_i &= \frac{1}{\overrightarrow{p_i}} \\ vcm_i &= \frac{\overleftarrow{g_{i-1}}}{\overrightarrow{p_i}}(vcm_{i-1} + \overleftarrow{p_{\sigma,i-2}}vc_{i-1}) \end{aligned} \tag{22}$$

where $\overleftarrow{g_{i-1}}$ is the conversion factor from solid angle to area surface measure. In that form it can now be implemented, it is just necessary to store probabilities during the ray tracing iterations.

5 Results

The described algorithm was written in standard C++11. Based on this implementation, tests were divided into two parts. The first tests (see Figure 6) have shown a comparison between BDPT and naive PT/LT, also a difference of using BDPT with and without multiple importance sampling. The test scene consist of the simple cornell box cube with a glossy floor and three spheres – a perfect mirror, a textured diffuse and a smooth dielectric; and the light source is an area light. From the results can be seen the big impact of the light source size which increases number of PT zero contribution rays (about 35 percent, see Table 1). For the obvious reasons, LT is totally ineffective for such scene.

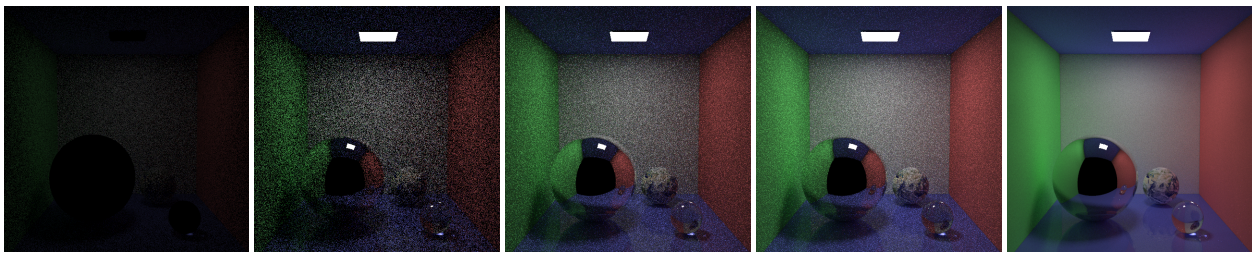
A quite considerable difference is also between the BDPT with uniform weighted paths and the BDPT with multiple importance sampling, using the power heuristic (Figure 6d and Figure 6e).

The second part of the test if focused on comparing the proposed implementation with existing state-of-art methods, using the Mitsuba renderer. The scene in Figure 7 is the cornell box with a textured diffuse, a perfect mirror and a smooth dielectric glass sphere, illuminated with the directional sun light. It shows that the proposed implementation achieves a little bit better quality in the whole image measured with mean-square-error against Mitsuba, see Table 2, and in a scene with detailed caustics stays a little behind, according to MSE, see Table 3. All these measures are also quite burdened with the inaccuracy of MSE measurement. In the most cases proposed impl. was faster than Mitsuba, or equally fast.

6 Conclusions

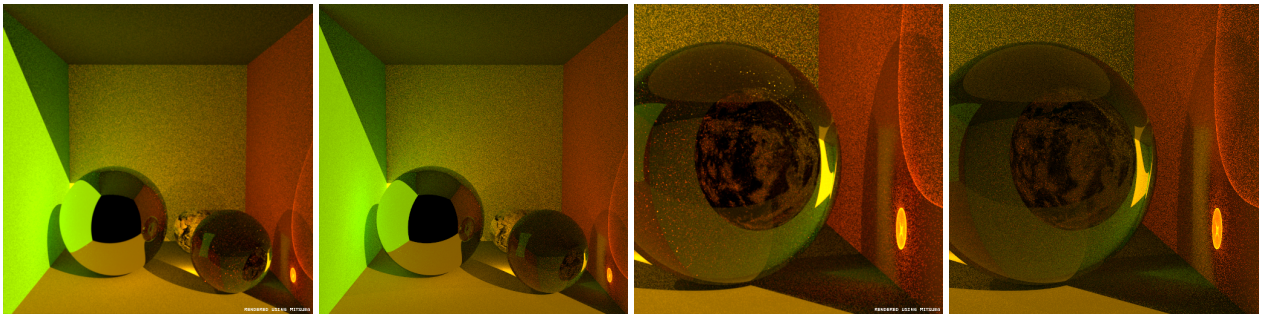
An experimental implementation of bidirectional path tracing is presented in this article, showing comparison of the naive methods with the BDPT and with Mitsuba renderer. Even though the BDPT algorithm uses more rays during the processing than PT, so that it takes longer to render final image, for still the most computational time is spent by ray intersection, it is more effective for complicated scenes with any type of the light source, as it significantly reduces the number of the rays with zero contribution. It reduces variance at all in less samples per pixel. Also with using the effective weight function, it converges even faster. In comparison with the other state-of-art methods such as Mitsuba BDPT module, the proposed implementation provides a similar quality of the final image and it is slightly faster.

Further research can be focused on techniques of efficient sampling such as an adaptive importance sampling. Another way is making a real-time path tracer, accelerated on the GPU. The third way is the integration of denoising filters during processing.



(a) Light tracing (b) Path tracing (c) PT with DI (d) BDPT (e) BDPT with MIS

Figure 6: DI stands for the direct illumination. All images were rendered with 60 samples per pixel



(a) Mitsuba test scene (b) Proposed impl. test scene (c) Mitsuba caustics (d) Proposed alg. caustics

Figure 7: Comparison between the proposed implementation and Mitsuba; a), b) 64 s. p. pixel; c), d) 128 s. p. pixel

	PT	MIS-BDPT
rays	65 mil.	251 mil.
zero contrib.	22 mil.	15 mil.
time	12s	30s

Table 1: PT and BDPT comparison

Cornell box scene

	MSE	time
Mitsuba	99.57	32.4s
Proposed impl.	64.15	27.6s

Table 2: Proposed impl. and Mitsuba MSE comparison in cornell box scene

Caustic detail

	MSE	time
Mitsuba	59.94	58.1s
Proposed impl.	80.27	56.4s

Table 3: Proposed impl. and Mitsuba MSE comparison in scene focused on caustic details

References

[1] Iliyan Georgiev. Implementing vertex connection and merging. Technical report, Saarland University, 2012.

[2] James T. Kajiya. The rendering equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM.

[3] Eric P. Lafortune and Yves D. Willems. Bi-directional path tracing. In *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[4] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. *ACM Trans. Graph.*, 21(3):703–712, July 2002.

[5] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. Monte carlo techniques for direct lighting calculations. *ACM Trans. Graph.*, 15(1):1–36, January 1996.

[6] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford, CA, USA, 1998. AAI9837162.

[7] Eric Veach and Leonidas J. Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 419–428, New York, NY, USA, 1995. ACM.

[8] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.