Small Project: A streaming bidirectional path tracer based on Light House 2

Lu Guowei (6374395)

October 11, 2020

1 Introduction

Lighthouse 2 (LH2) is a rendering framework for real-time ray tracing / path tracing experiments which uses a state-of-the-art wavefront / streaming ray tracing implementation to reach high ray throughput on RTX hardware and pre-RTX hardware.

Bidirectional path tracer (BDPT) algorithm [1] is a generalization of the standard path-tracing algorithm by constructing paths that start from the eye on one end and from the light on the other end and are connected in the middle with a visibility ray.

Based on LH2 framework, I build a BDPT core by which difficult lighting settings such as caustics can be handled more effectively. First, it is an Optix Prime[2] wavefront path tracer on the GPU[3]; second, I apply the recursive MIS computation [4], [5] for this streaming BDPT(SBDPT); third, the visibility rays are traced outside the wavefront loop, batching the visibility rays as large as possible is a good strategy for the performance optimization; finally, this BDPT also handles non-symmetric scattering situations including refraction and shading normals by using adjoint BSDFs[6].

This report contains the following three parts:

1. **Implementation:** The design and optimization of the BDPT render system
2. **Result:** The ground truth, quality, performance, MIS and other aspect
3. **Conclusion:** My understanding and reviews about this SBDPT render system

2 Implementation

2.1 Process

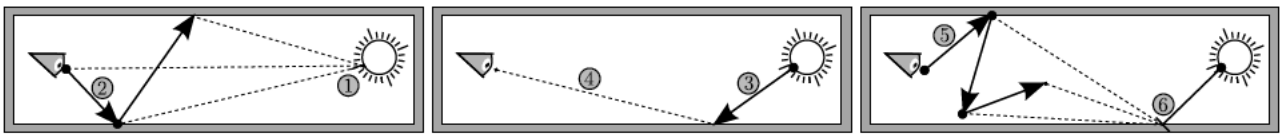


Figure 1: Construction of an SBDPT sample[4]

Figure 1 shows how one SBDPT sample is constructed: (1) first, we start out with generating a vertex on the light source. (2) Then, the first eye path is generated, during the extension of the eye path, each vertex is connected to the light vertex resulting in complete light transport paths. (3) When the eye path is terminated (hit missing or reaching the maximum length), the light path is extended and (4) connected to the eye. (5) Then, a new eye path is generated and extended as

step (2). This process is repeated until the light path is terminated (hit missing or reaching the maximum length). During the extension of eye path or light path, there is a connection between the eye path and light path and a visibility ray is generated. If it is not occluded, there is a contribution to the corresponding pixel. This scheme makes it possible to compute MIS weight for each connection by using only two edges from the eye and light paths.

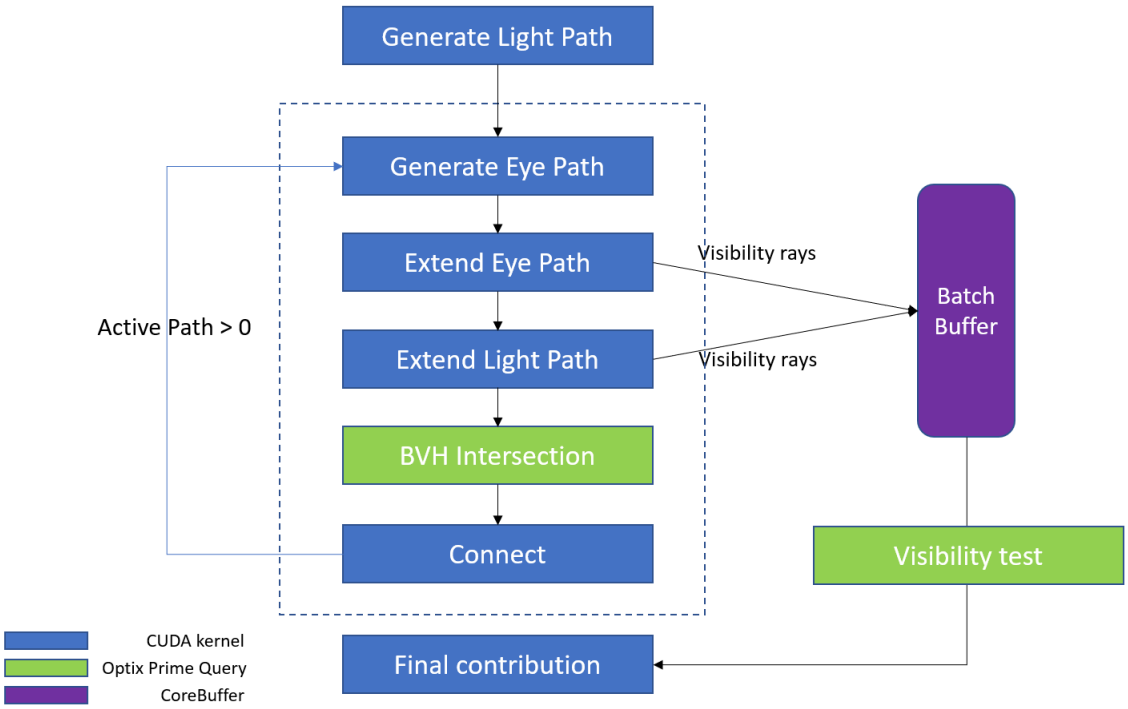


Figure 2: Wavefront pipeline of SBDPT

Then, this SBDPT algorithm is designed to fit the GPU architecture through six CUDA kernels:

1. **constructionLightPos.h**
2. **constructionEyePos.h**
3. **extendEyePath.h**
4. **extendLightPath.h**
5. **connectionPath.h**
6. **finalizeContribution.h**

As Figure 2 shows, first, for all the pixels on the image plane, the light vertices are generated. Then, there is a loop, either eye path or light path is extended for each pixel, we call it as random walk. Meanwhile, the potential contribution of the connection between the current edges of eye and light paths is measured and pushed into the batch buffer. After that, BVH intersection is executed for all random walks. Based on the hitting results, in the 'connect' stage, the path state for each pixel is re-arranged (4 possible states: $DEAD$, NEW, EYE_EXTEND and $LIGHT\_EXTEND$). The loop is terminated when there is no active path for all pixels, active path here means it can extend the eye path or light path to the current pixel. Finally, the visibility test is executed and we measure the final contribution accurately. By batching and querying all visibility rays, the final contribution is measured with a high performance. Unfortunately, for the path with N vertices, the upper bound on the visibility ray count is $(N-1)(N-2)/2$, $O(N^2)$.

2.2 Connection Type

We set s as the length of eye path and t as the length of light path and there are six different path connection types:

1. $t == 0$: Standard path tracing without direct illumination sampling (Implicit path)
2. $t == 1$: Path tracing with direct illumination sampling (Explicit path)
3. $s == 0$: Particle tracing from a light source with an explicit visibility test between a point on a surface and the camera (Particle path)
4. $s > 0$ && $t > 1$: Sub-path connection
5. **S-Connection**: the material of sub-path connection is specular, ignore this contribution
6. **No hitting**: The random walk misses objects, and the contribution is from skydome

2.3 Non-symmetric Scattering

In this BDPT core, I handle two situations which can lead to non-symmetric behavior in light transport simulations, one is about radiance path (eye path) and another one is about importance path (light path).

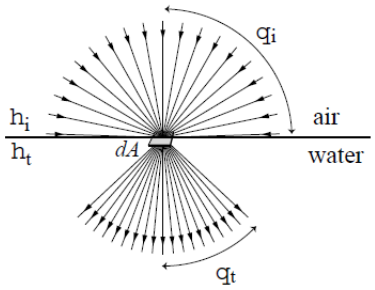


Figure 3: Refraction with energy conservation[1]

As Figure 3 shows, when there is a refraction along the radiance path, the energy is squeezed into a smaller volume, this causes the radiance along each ray to increase. So, when the BSDF of the material is measured, we need to consider the type of the path and adjust its radiance.

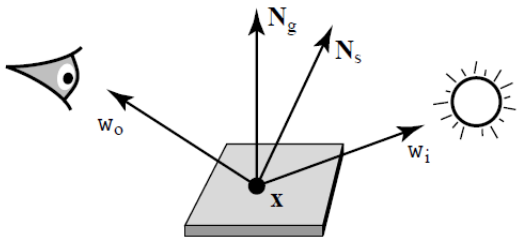


Figure 4: Shading normal[1]

Shading normals are mainly used to make polygonal surfaces appear smoother by replacing the "true" geometric normal N_g with an interpolated shading normal N_s, as Figure 4 shows. So, we need to adjust the adjoint BSDF to avoid artifacts and inconsistencies of the particle path in BDPT algorithms.

2.4 Others

1. **Batching of visibility rays**: Optimize the query performance
2. **Path regularization**: When the path contains the substring DD (two consecutive diffuse), the extension of eye path is terminated. This method has an influence on the ground truth slightly and improves the performance obviously
3. **blueNoiseSampler**: We apply this random method in the eye path which can improve the convergence
4. **GPU Memory Optimization**: We reduce the size of path state and compact it although there are still 14 float4 variables, and we reduce the buffer needed in the wavefront.
5. **Sampling the light rays**: In BDPT render system, it supports area light, point light, spotlight and directional light

3 Result

3.1 Ground Truth

There are three different render systems in this comparative study. I use Resemble.js for the image analysis and comparison.

1. **BDPT**: BDPT render system, *rendercore_optixprime_bdpt*
2. **Path Tracer**: A path tracer with NEE, IS and MIS, *rendercore_optixprime_b*
3. **Ref**: A path tracer with NEE, *rendercore_primeref*

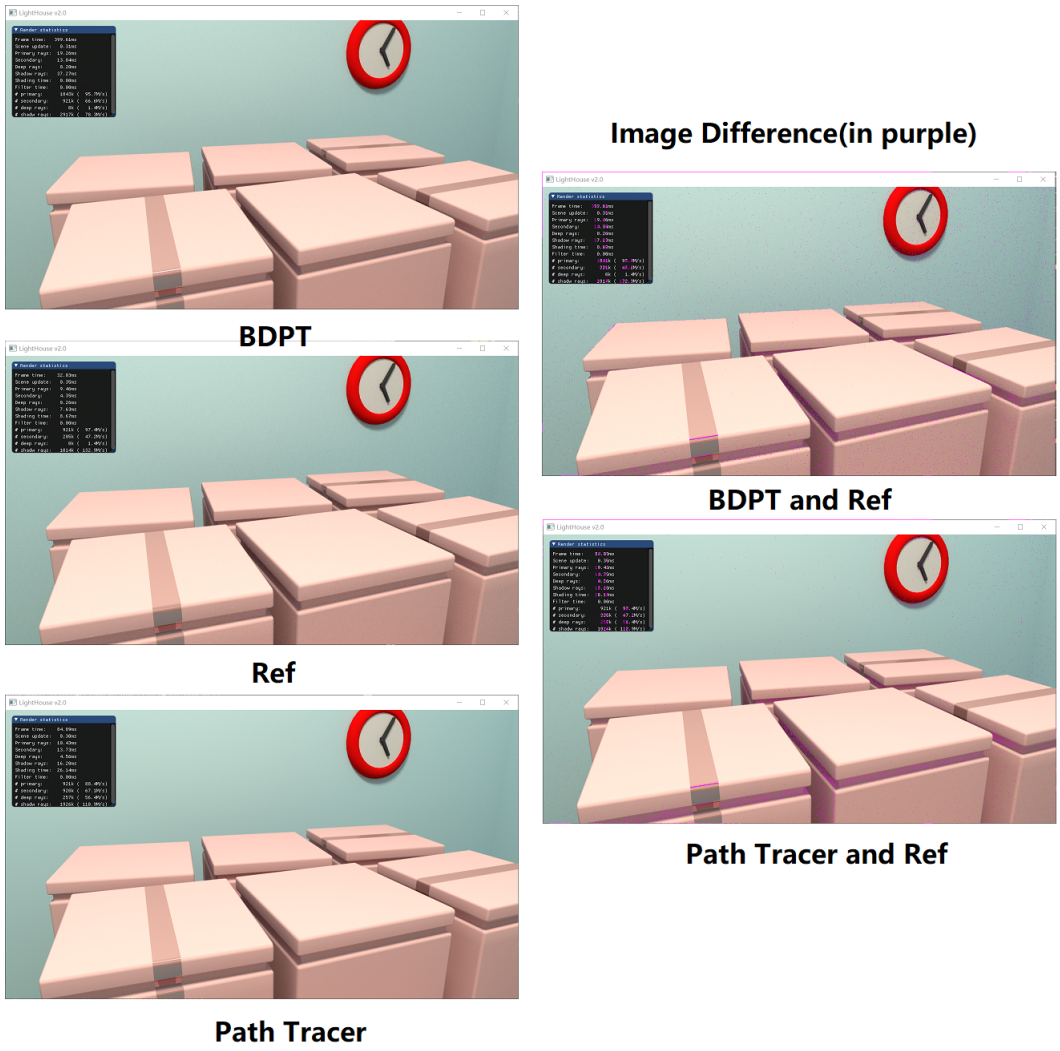


Figure 5: Box Scene

Figure 5 shows the detail of this scene. These three images on the left side look almost the same, the right side shows the image difference. The *Ref* render system does not consider the emissive contribution.

Figure 6 is a panoramic view of the same scene. And the comparison shows two differences. One is the shadow region in the hole (Path Tracer), another one is the boundary of the box (BDPT). In my opinion, there are two possible reasons for the boundary difference, normal precision and anti-aliasing.

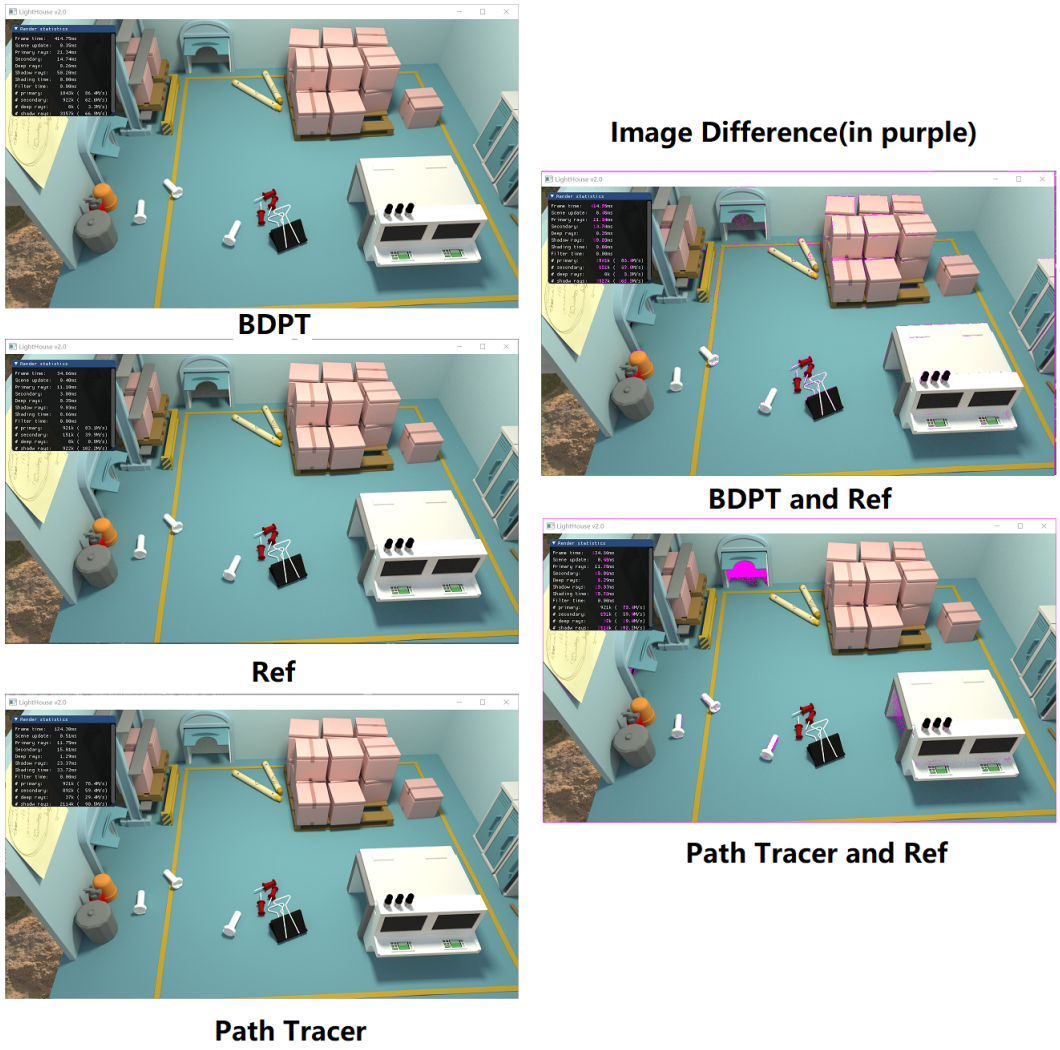


Figure 6: Panoramic view

The last one is diverse materials in an outdoor scene, as Figure 7 shows. First, Ref render system needs too much time to get convergence, we only compare the difference between $BDPT$ and $PathTracer$. Second, in BPDT, if either material is specular, there is no contribution from the sub-path connection. So, I apply the same principle in the $PathTracer$.

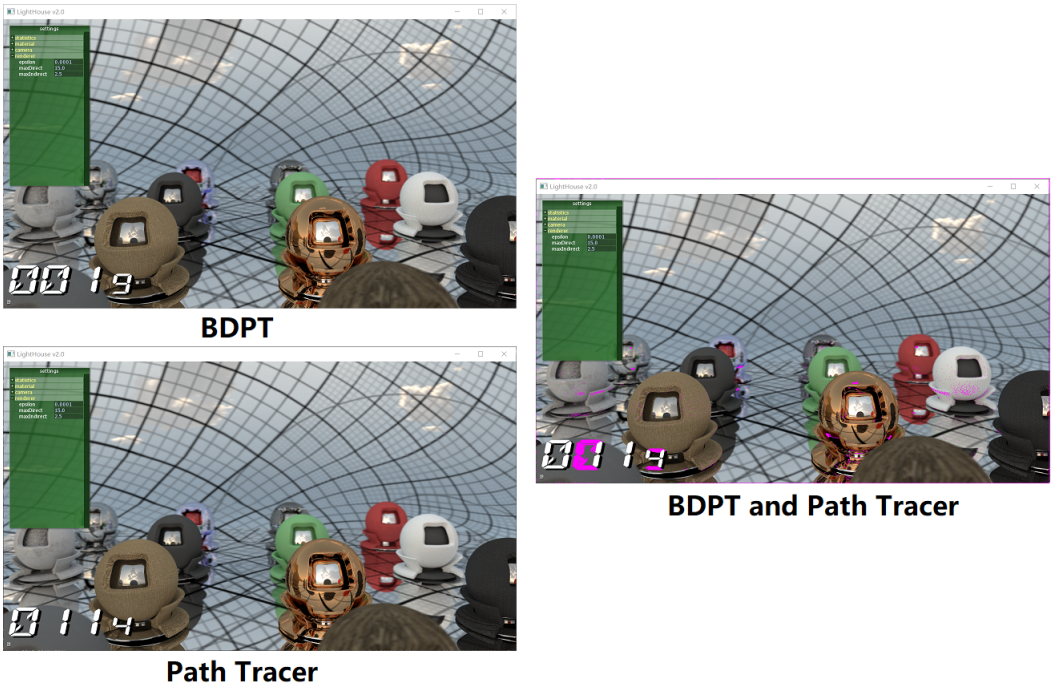


Figure 7: Diverse material

Notice the difference is from emissive contribution, BDPT's contribution is larger than Path Tracer's, as Figure 8 shows.

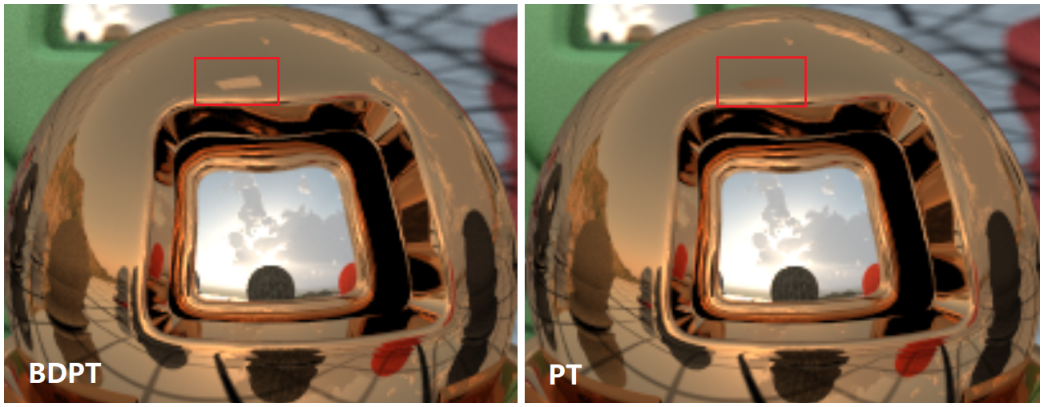


Figure 8: Emission contribution

3.2 Quality

Figure 9 shows an outdoor scene. Obviously, right side is better than the left one.

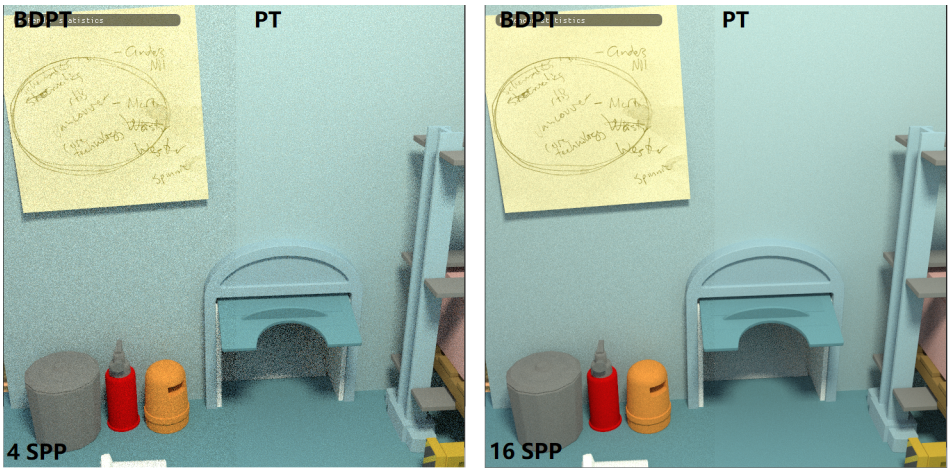


Figure 9: Outdoor environment

I tried to figure out the reason and this is my explanation. As Figure 10 shows, both eye and light are in an open position, it is possible to sample the same direction using both eye path and light path, and the only different is *pdf*. The *pdf* of the eye path (orange line) when connecting to the light (explicit path) is fine, so, the different in *pdf* is not obvious. Unfortunately, many light paths can not make contributions (red line with cross) because of the limitation of the eye direction and fov. In one sample, there are more than one million visibility rays from explicit path while there are only sixty thousand visibility rays from particle path. The right side of Figure 10 shows the comparison between explicit path and particle path. Particle path increases the variance of the final result due to its inefficiency. It is the low-variance problem of MIS[1], we can apply the power heuristic to handle it.

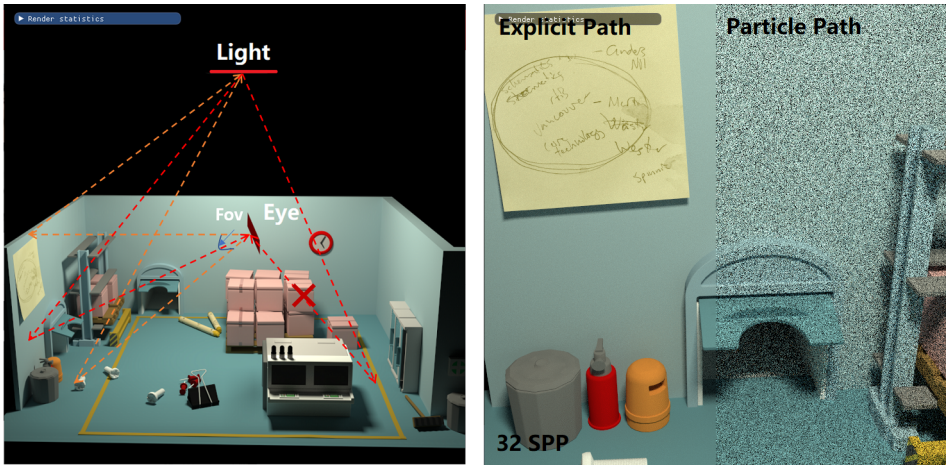


Figure 10: Outdoor layout

Figure 11 is slightly different. In this scene, the light in the hole is difficult to reach. So, the *BDPT* has lower variance and better confidence.

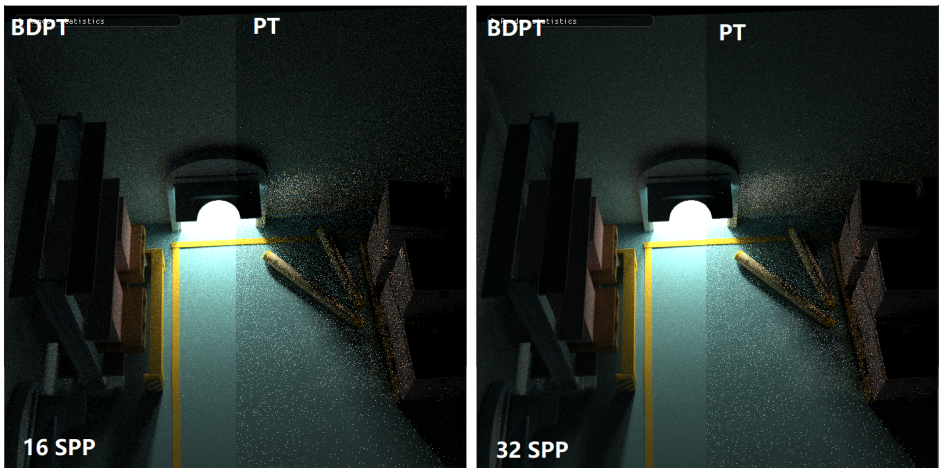


Figure 11: Dark scene

Figure 12 shows the result of a caustic scene, there is a light source in the bottle and only $BDPT$ can get fast convergence.

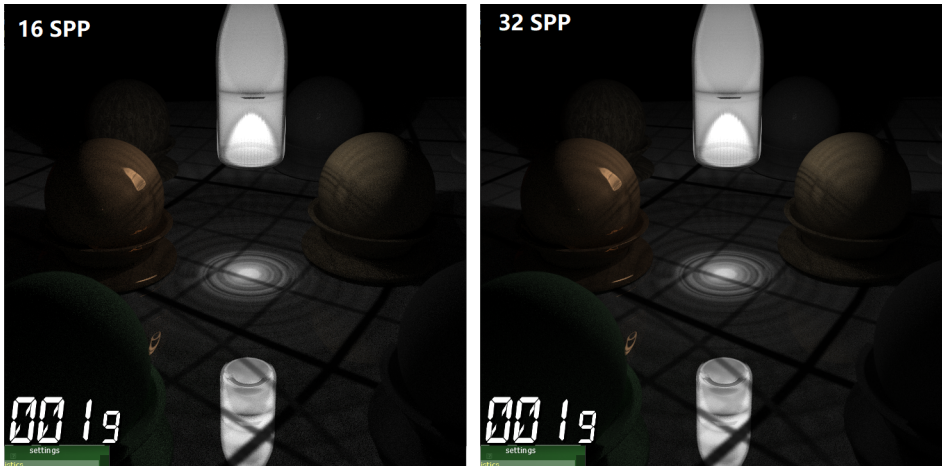


Figure 12: Caustic scene

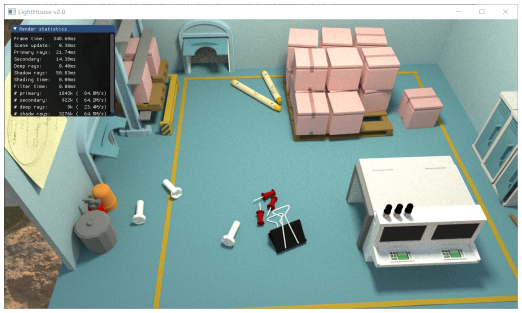
3.3 Performance

Since my laptop does not support RTX render system, I test the performance between $BDPT$ and $PathTracer$ render systems. The basic information of my laptop is:

CPU	**GPU**	**CPU Memory**	OS
i7-8550U 1.80 GHz	MX150 2GB	16GB	Windows 10

About the performance, there are two strategies: with $FLAGS$ or without $FLAGS$, you can see the details below. Generally, $BDPT$ is four to five times slower than $PathTracer$; $FLAGS$ is practical when there are many diffuse materials, but these regions will be a little darker; random walk rays and visibility rays are in a high performance. If I will continue to optimize the performance, I will try to apply compaction, split the struct of path into eye path and light path, and modify the pipeline to reduce divergence. Theoretically, in one sample, the number of random walk rays is $1 + N$ times of $PathTracer$, and the visibility rays is $O(N^2)$(N is the depth of the path).

	Frame time (ms)	Primary (Mrays/s)	Second (Mrays/s)	Deep (Mrays/s)	Visibility/Shadow (Mrays/s)
BDPT	282.37	117.4	95.9	5.5	54.6
BDPT_FLAGS	261.21	115.2	95.7	0.7	57.3
PT	63.42	95.9	99	24.8	114.1
PT_FLAGS	54.88	95.3	99.3	20.8	121.4



	Frame time (ms)	Primary (Mrays/s)	Second (Mrays/s)	Deep (Mrays/s)	Visibility/Shadow (Mrays/s)
BDPT	348.6	84.8	64.2	23.4	64.5
BDPT_FLAGS	275.82	87.3	65.3	28.4	68
PT	84.9	76.2	66.4	47.7	100.6
PT_FLAGS	52.54	77.5	67	0	111.1

3.4 MIS

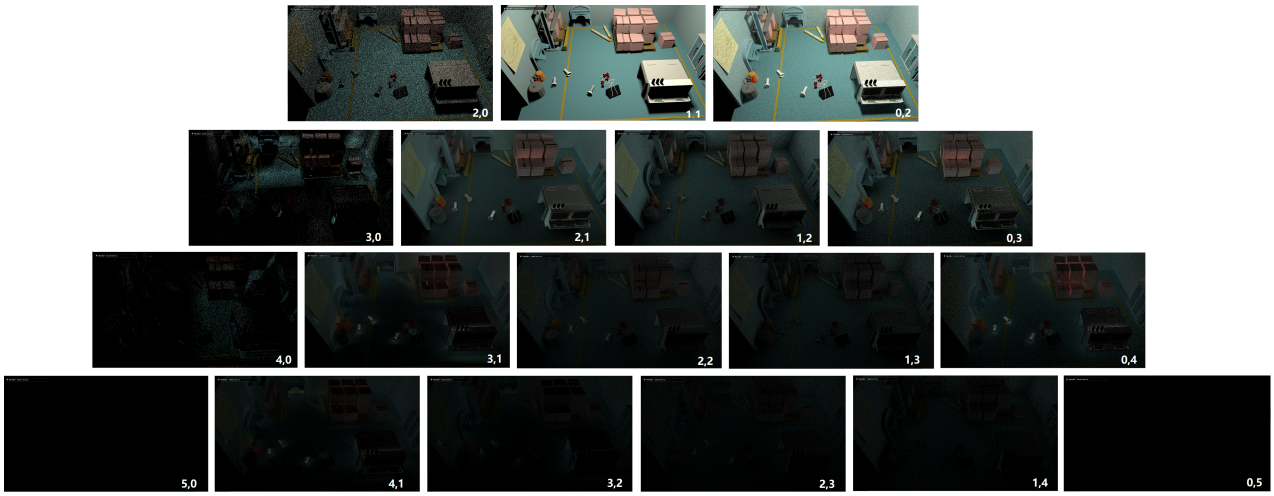


Figure 13: The Individual BDPT Strategies

In this part, we will see how the MIS works. Figure 13 is the result after 32 SPP, each row corresponds to light paths of a certain length. The number (s, t) means the length of eye path and light path. Path Tracer only samples the t = 1 paths. When t = 0, it is the implicit path, when s = 0, it is the particle path. With the MIS weight, the final result is computed by summing all these images in Figure 14.

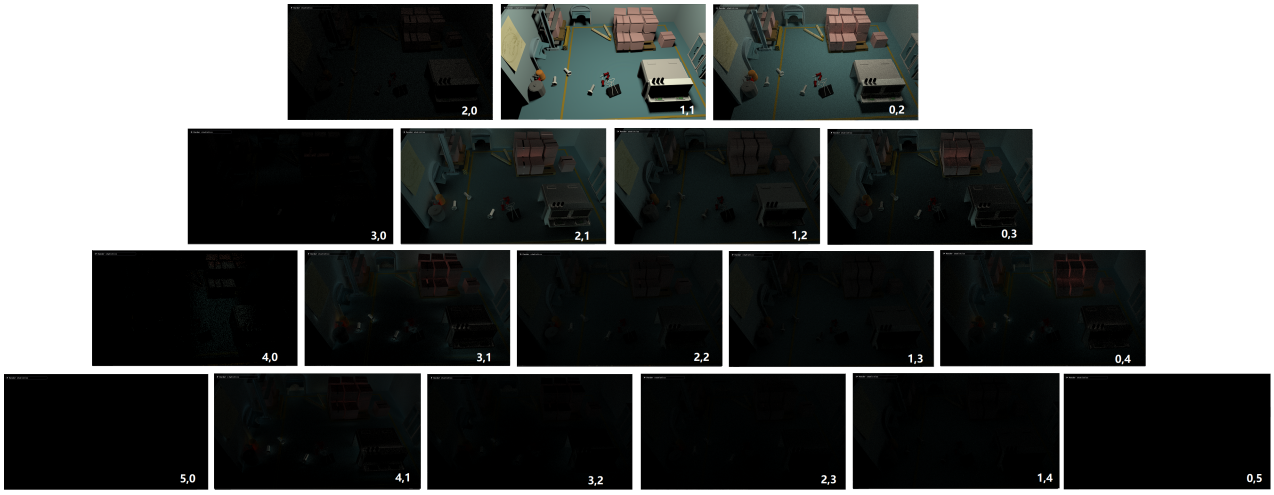


Figure 14: Variance Reduction due to MIS

Figure 15 shows how MIS can reduce variance. In this scene, it shows MIS can effectively "turn off" the contribution where it does not perform well. PP here means particle path.

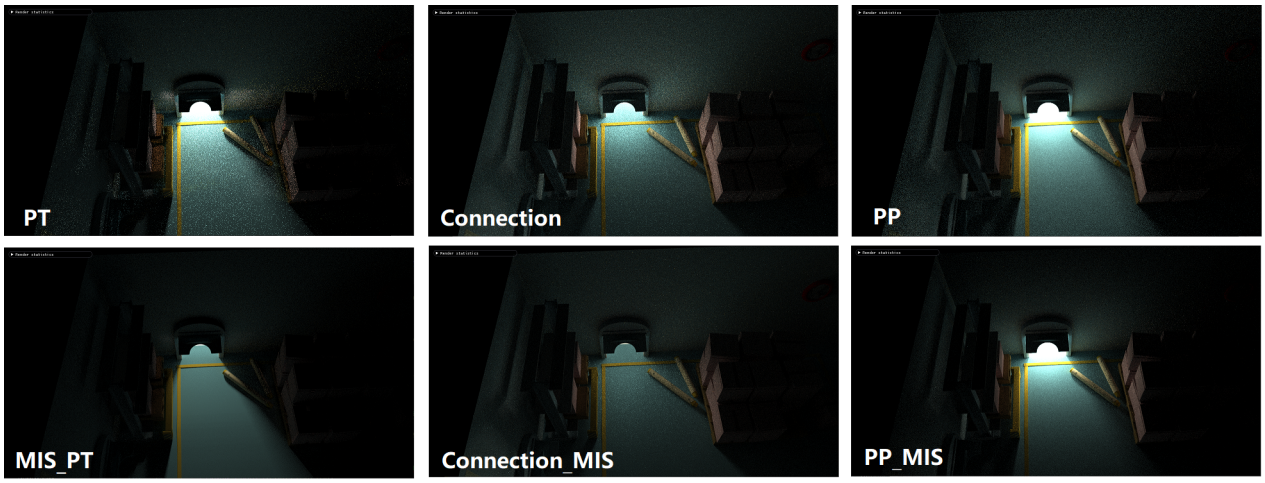


Figure 15: Variance Reduction due to MIS(2)

Figure 16 shows the comparison of $BDPT$ with MIS or average weight. MIS is an effective way to reduce the variance.

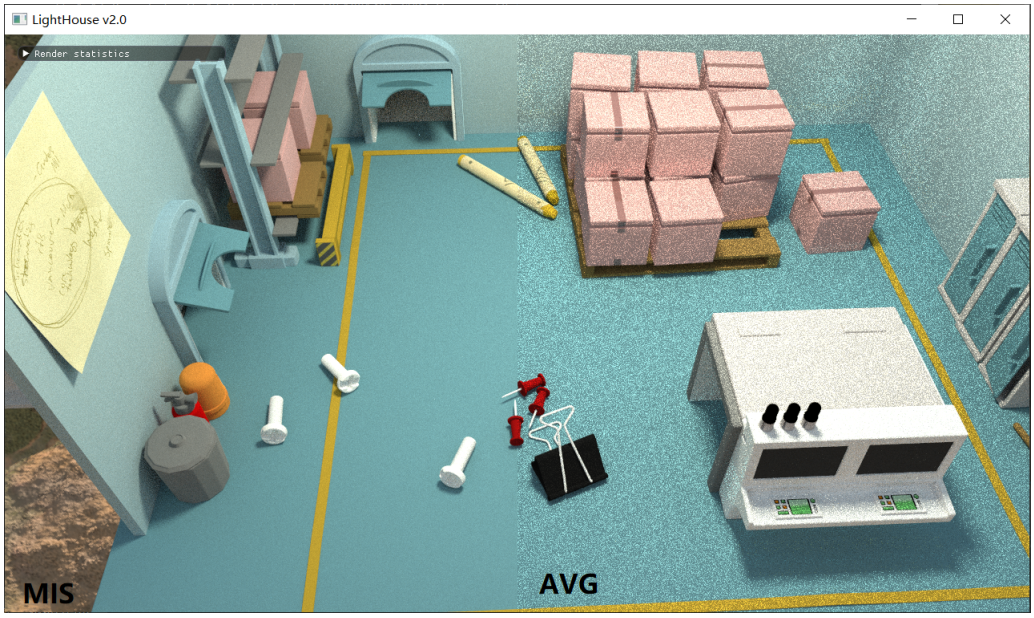


Figure 16: BDPT with MIS or not

3.5 Others

I also test other aspects such as $BlueNoise$, $atomicAdd$ and energy conservation.

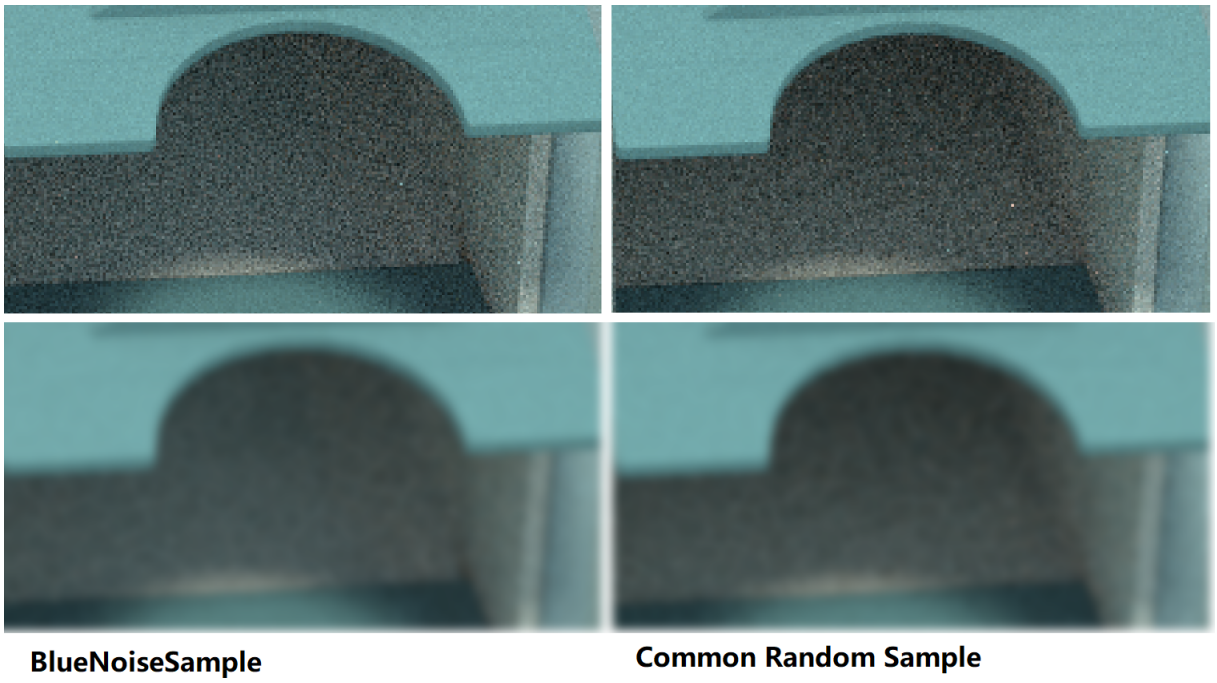


Figure 17: BDPT with blueNoise or not

In Figure 17, the first row is the screenshot of Figure 16 after zooming 8 times. And the second row is the result after applying Gaussian Blur. The one in the left side is smoother.

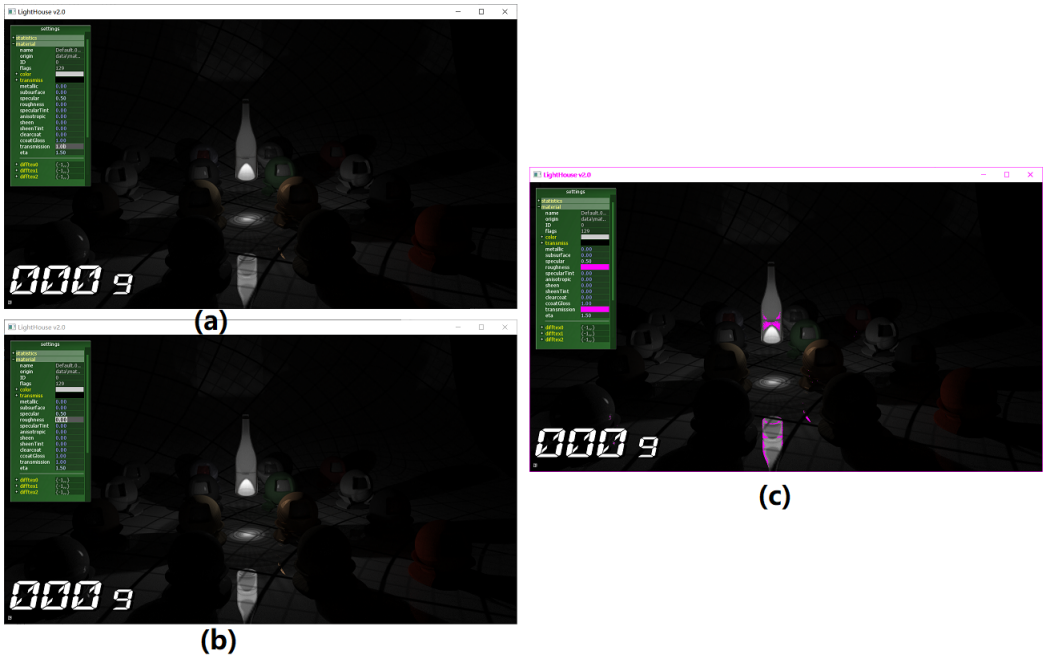


Figure 18: BDPT with refraction correction or not

In order to keep energy conservation, we need to consider non-symmetry due to refraction. Figure 18(a) is the correct result considering the energy conservation. Figure 18(b) is too brighter, caused by assuming that refraction between glass bottle and air is modeled by a symmetric BSDF.

The pink region in Figure 18(c) is the difference between (a) and (b).

Figure 19: *atomicAdd*

Because we use batch for the visibility rays, strictly we should accumulate the contribution to the corresponding pixel by atomic operation. Figure 19 shows the difference between *operator+* and *atomicAdd*, the pink part shows there is a loss of energy without applying *atomicAdd*. I also test this in another scene without *caustic* and there is no difference. So, we should use *atomicAdd* for the contribution from particle path or design a struct to avoid *atomicAdd* because of its poor performance.

4 Conclusion

Based on the implementation and result analysis, this SBDPT algorithm is robust. The performance is not good enough, but it is an implementation problem, we should find many possible ways to optimize the performance.

About the quality, there are several factors to determine whether it is good to choose $BDPT$: is it easy to reach the light source; are there many diffuse materials; is it easy to reach the eye and do we need to simulate caustic. If it is difficult to find the light source and there are many diffuse materials, it means the *pdf* of particle path is high and the variance is low. Meanwhile, if the eye fov is big enough to accept most light paths, it means the particle path can make many contributions to the rendering result. For example, in an indoor scene, path tracer ray needs to be bounced more than once before connecting to the light source, and there will be more noise, in this situation, $BDPT$ is a better choice.

About the ground truth, this recursive MIS computation is not perfect, there are two disadvantages. First, when we extend the light path, we need to build a new eye path corresponding to each vertex in the light path, but the depth of the light path depends on random walk, we do not know the accurate number of eye paths we build, so we need an experienced value for the MIS calculation. [4] mentioned this issue, and the test in this report shows the result is not sensitive to this factor. Second, the BDPT algorithm is a path to connect one eye path and one light path, and we measure the MIS value of each path strategy. For example, when we measure the MIS value for a path with length 4, there are 4 possible sub-path connections such as Figure 14, the second row. The sum of the MIS weights in this row equals to one:

$MIS_{sum} = MIS_{(3,0)} + MIS_{(2,1)} + MIS_{(1,2)} + MIS_{(0,3)} = 1$, but in this recursive MIS computation algorithm, it is a sub-path connection between one light path and different eye paths, so the MIS_{sum} is not guaranteed to be 1. From statistical perspective, it is an acceptable approximation, but to the algorithm itself, it is not accurate.

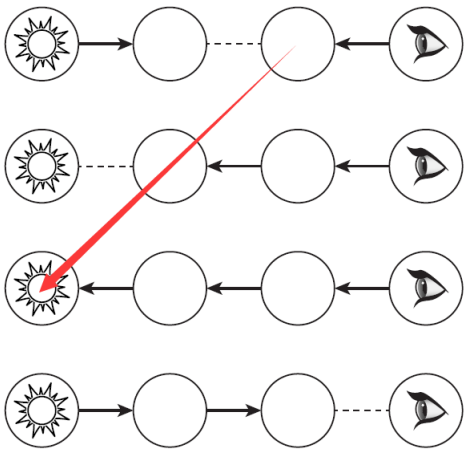


Figure 20: Sub-path connection

About the performance, because in this recursive algorithm, we need to build a new eye path for each extension of light path, it is a much more expensive algorithm than $PathTracer$. Another point I find, as Figure 20 shows, during the extension of the light path, there are less vertices in the eye path which are useful to the connection(the upper left corner of the red line). So, we can terminate these random walks from the eye path. But it has a potential issue, because if the vertices hit the light source luckily, it can make contributions as an implicit path. So, we need to adjust N_{kk} to calculate the correct MIS in this situation.

References

[1] Eric Veach. *Robust Monte Carlo methods for light transport simulation*, volume 1610. Stanford University PhD thesis, 1997.

[2] Steven G Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison, et al. Optix: a general purpose ray tracing engine. In *Acm transactions on graphics (tog)*, volume 29, page 66. ACM, 2010.

[3] Samuli Laine, Tero Karras, and Timo Aila. Megakernels considered harmful: wavefront path tracing on gpus. In *Proceedings of the 5th High-Performance Graphics Conference*, pages 137–143. ACM, 2013.

[4] Dietger Van Antwerpen. Recursive mis computation for streaming bdpt on the gpu. Technical report, Citeseer, 2011.

[5] Iliyan Georgiev. Implementing vertex connection and merging. *Technical Re-port. Saarland University. Accessed May*, 22:2018, 2012.

[6] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016.

5 Appendix

This BDPT link is here, and the modification of the LH2 is:

1. **RenderCore_OptixPrime_BDPT**: the whole project
2. **Sample_Le**: the method in the *lights_shared.h* file for sampling light
3. **UniformSampleSphere and UniformSampleCone**: two sample methods in the *tools_shared.h*
4. **Camera::GetView()**: add two attributes (*imagePlane* and $focalDistance$) for sampling camera, in *common_classes.h* and *camera.cpp* files