The Two-Level Semi-Synchronous Parallelization Method for the Caustic and Indirect Luminance Calculation in Realistic Rendering[*]

Andrey Zhdanov [0000-0002-2569-1982], Dmitry Zhdanov [0000-0001-7346-8155]

ITMO University, 49 Kronverksky Pr., St. Petersburg, 197101, Russia

adzhdanov@itmo.ru, ddzhdanov@mail.ru

Abstract. The paper considers an original approach to the semi-synchronous calculation of the luminance of caustic and indirect illumination for the group of methods based on the bidirectional stochastic ray tracing with backward photon maps. The designed parallelization method uses the two-level threads hierarchy. The low level of this thread hierarchy is synchronous calculations of the part of the whole image defined by a randomly generated pixel mask which is applied to the whole image. The top level is semi-synchronous parallelization level that consists groups of the low level threads which of them calculate own part of the whole image in a way similar to asynchronous calculations. As the top level is semi-synchronous it means that when calculating the luminance of the caustic and indirect illumination, the threads of the low level have access to the data accumulated in the backward photon maps of the other parallel threads of the semi-synchronous level. A special algorithm for organizing an access to data of the upper-level threads avoids delays associated with data synchronization. The comparison of the developed solution with purely synchronous and asynchronous parallelization methods is presented.

Keywords: Ray Tracing, Photon Maps, Backward Photon Maps, Parallel Computing.

1 Introduction

Realistic rendering is a significant component that is commonly used in the modern realistic visualization, virtual prototyping and virtual reality systems. In addition, it is used to solve a wide range of applied problems, including the realistic images forming, optical effects simulation, virtual prototyping of complex optical systems, etc. With increasing computing power and computation architecture complexity of modern computer systems, both the complexity of tasks for virtual prototyping and the required

[*] This work was funded by the Russian Science Foundation (project No. 18-79-10190).

accuracy of calculations raise. The tasks of virtual prototyping, solved by physically correct realistic visualization methods, include modeling the illumination of optical systems, modeling the human perception of synthesized images formed by complex optical systems, such as, for example, virtual or mixed reality systems, indicators on the windshield, and others.

Traditionally realistic rendering algorithms are based on the Monte-Carlo ray tracing methods which are used to calculate the luminance. These ray tracing methods can be forward, backward or bidirectional. The most universal of these method for calculating the physically correct luminance of indirect and caustic illumination are the methods based on the use of photon maps [1, 2]. These rendering methods have a good parallelization capability, however effective usage of the photon map in a multi-core environment is a challenging task. Despite there are existing solutions aimed at the effective CPU or GPU ray tracing as, for example, Intel Embree [3], Nvidia RTX [4] or AMD RadeonRays, they do not solve the problem of the effective processing of the traced rays for the needs of the realistic rendering with photon mapping. So, the research and development of effective rendering methods using all available computation resources of modern multi-core CPUs with continuously raising number of cores is still an urgent challenge.

There are existing solutions aimed on parallelizing the photon mapping rendering algorithms using synchronous calculations on multi-core CPU [5], out-of-core photon mapping [6, 7], massive parallel calculations with GPUs [8, 9], distributed simulations [6, 10] and cloud rendering [11]. At the same time the rendering algorithm might be aimed either at the speed to achieve the real-time rendering or at the physical correctness of the simulated image. In the scope of the current article we concentrated on the effective usage of the backward photon maps in a multi-thread environment when rendering with the stochastic progressive backward photon mapping (SPBPM) method on a single-CPU workstation with multiple cores with aim at the physical correctness of simulation.

2 Rendering parallelization

2.1 Rendering method

In the scope of the current research we used the method based on bidirectional ray tracing with backward photon mapping [12, 13]. Opposite to the traditional photon mapping based methods the backward photon maps are formed by the backward rays emitted from the camera and luminance transferred by forward rays is accumulated in these maps to form the final image. The general workflow of the rendering method is shown on the Fig. 1. The rendering method consist of four main steps:

1. Backward path tracing when the backward paths from the camera are generated and traced in the scene with the direct light and BDF samplings to account the direct luminance.
2. Backward photon maps forming along with creation of the acceleration structures.

3. Forward ray tracing when forward rays from light sources are generated and traced in the scene. By intersecting with previously formed backward photon maps, the indirect and caustic luminance is accumulated.
4. Final image forming when the luminance accumulated in backward photon maps is added to corresponding pixels of the image and weighted. The image accuracy is estimated and if the required accuracy is not achieved then calculation continue from the first step.

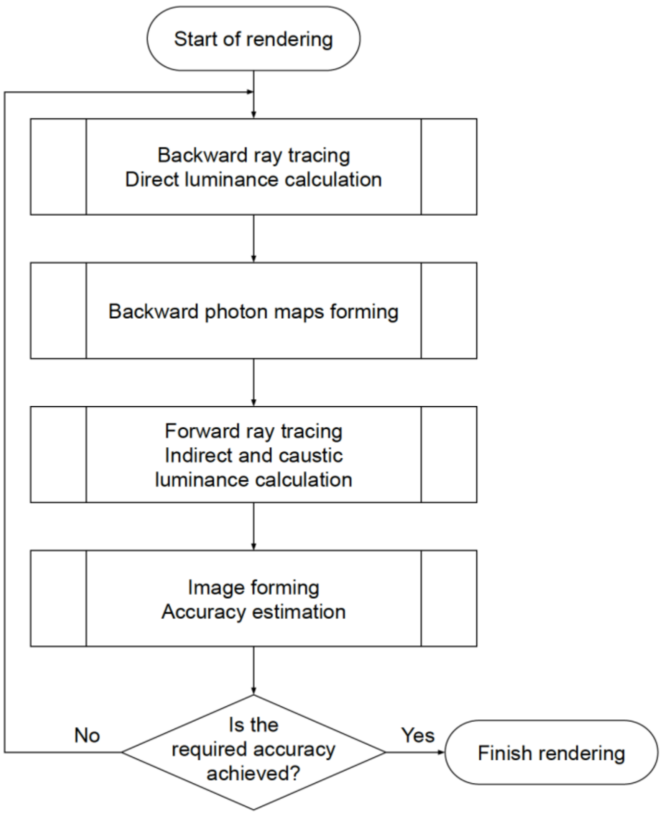

Fig. 1. General workflow of the used backward photon mapping rendering method.

2.2 Traditional parallelization methods

The traditional parallelization methods used for ray tracing-based rendering are synchronous and asynchronous calculations. Both traditional methods were implemented and tested on 12 cores of the Intel Xeon 6230 CPU and standard Cornell box scene. Ray tracing and processing speedup test results are presented on Fig. 2 and Tables 1 and 2.

Synchronous calculations method for parallelizing the rendering process on a multi-core system is using all available cores in a synchronous way. In this case all threads share the same memory pool and render the same scene in synchronous way. The main problem of this approach is presence of the non-parallelizable parts of the algorithm that according to the Amdahl's law [14] cause the significant ray tracing and processing slowdown when increasing the number of cores. As it can be seen from the presented graph the rays tracing and processing speed growth almost stops at some point when increasing the number of used computation cores.

In opposite to the synchronous calculations method the asynchronous calculations use one main thread and a group of computation threads. Each of these computation thread performs the independent rendering of the whole image, and the main thread is used to control the computation threads and gather rendering results from the computation threads. Due to asynchronous nature of calculation all computation threads use their own private memory and as result it multiplies the memory usage of the whole rendering process by number of computation threads. As result as it can be seen from the presented graph the number of traced and processed rays growth linearly when increasing the number of the used cores.

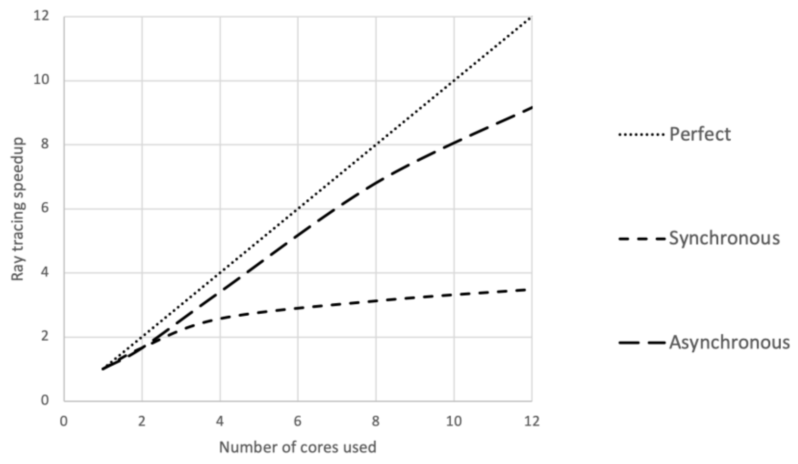

Fig. 2. Test results for traditional synchronous and asynchronous parallelization methods.

At the same time after 10 minutes of calculations using all computation cores the synchronous method achieved the accuracy of 3.9%, while asynchronous method achieved the accuracy of 5.1%. So even if asynchronous method traces and processes more rays the accuracy attained after 10 minutes of calculations of the same scene is lower comparing to synchronous calculations. The main reason of this slowdown is that in case of the asynchronous calculations the whole available memory is split between threads and as result rays are processed in smaller groups resulting in less connections made between light sources and camera pixels.

2.3 Semi-synchronous parallelization method

The research goal was to unite the scalability of the asynchronous calculations with higher accuracy of the synchronous calculations in the most effective way. So, in the scope of the current research the semi-synchronous parallelization method was developed. This method consists of two parallelization levels: synchronous and semi-synchronous.

The first parallelization level is synchronous rendering of the part of the whole scene image defined by the random mask, mainly of 32x32 pixels size. This mask defines image pixels that are used at the backward ray tracing step for the backward photon map forming. Each rendering step is parallelized independently in a synchronous way with one main thread that controls the rendering method workflow. The synchronous parallelization level is shown on the Fig. 3.

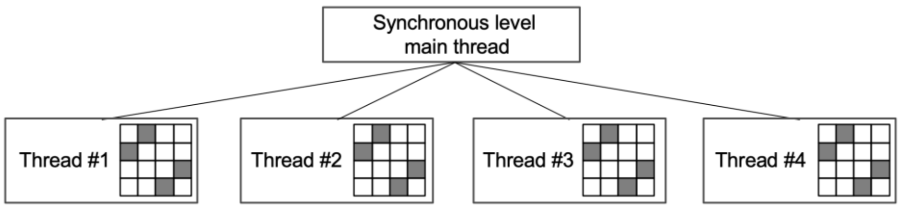

Fig. 3. Synchronous parallelization level.

The second parallelization level is semi-synchronous that unites several of synchronous thread groups to render the whole scene in semi-synchronous way. Each group of synchronous threads have their own random mask. These masks are generated by the main thread so that they cover all image pixels and at the same time do not intersect with each other. Once in a several number of rendering phases all threads are synchronized, the whole rendered image is formed, and masks are re-randomized. This re-randomization is required to guarantee that all synchronous thread groups have about equal load and all image pixels are processed equally. Each of the synchronous thread groups uses their own backward photon maps as result multiplying the memory usage required to store maps by the number of synchronous thread groups, however the size of each of these maps can be made smaller comparing to a map built for the whole image. The semi-synchronous parallelization level is shown on the Fig. 4.

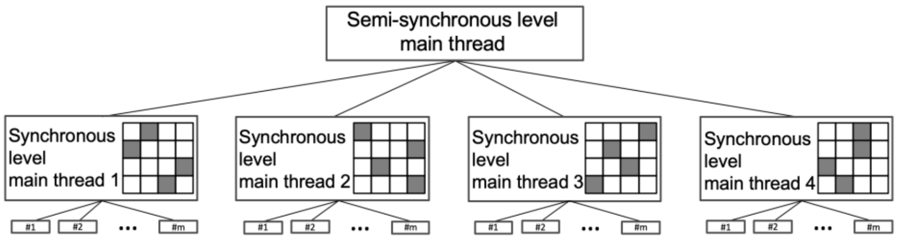
Semi-synchronous level main thread
Synchronous level main thread 1
Synchronous level main thread 2
Synchronous level main thread 3
Synchronous level main thread 4
#1
#2
...
#m

Fig. 4. Semi-synchronous parallelization level.

It is possible that at the forward ray tracing step of the rendering method the forward ray traced by one synchronous group of threads do not find an intersection with the own backward photon maps, however this ray might result in the luminance forming in backward photon maps of the other synchronous groups of threads. To increase the ray processing ratio of the traced forward rays it is required provide an access to all backward photon maps that exist at the moment. This means that some flag should be maintained by the backward photon maps owner indicating that maps are available and open for other threads to be used. At the same time these maps should not be closed and deleted while some other thread is using them. The workflow of the rendering process in a synchronous group of threads with opening and closing an access to the own backward photon maps is shown on the Fig. 5.

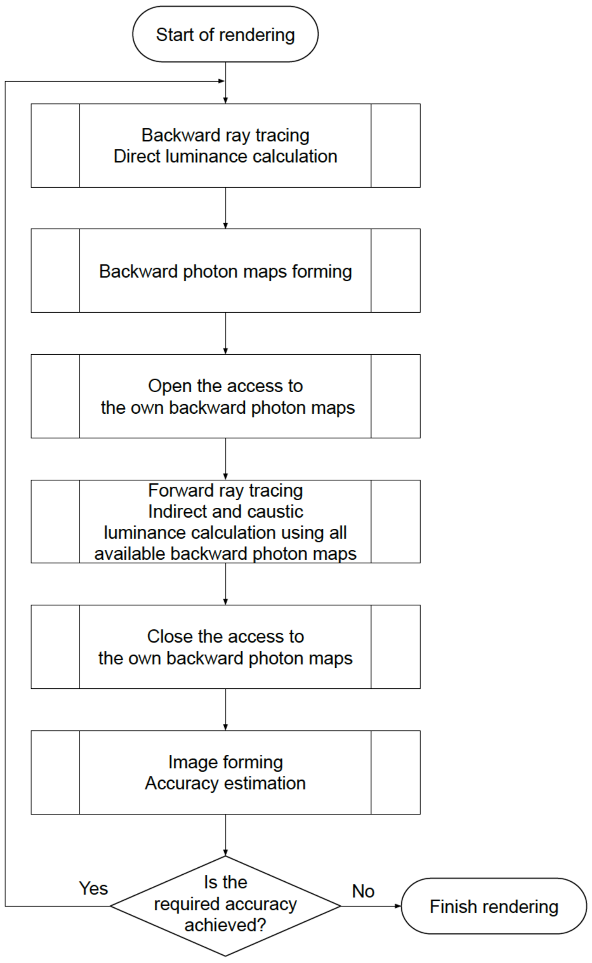

Fig. 5. General workflow of the single thread of the backward photon mapping rendering method with shared photon maps.

This opening and closing the access to the own backward photon maps should be performed without interrupting the calculation process and without using the critical sections. So, in the scope of the current research the algorithm of asynchronous access to the backward photon maps was developed. This algorithm uses only atomic operations both to open and close the access to the own backward photon maps and to gain and release access to the other thread's backward photon maps.

For these needs the backward photon maps active thread counter is stored along with along with each of the backward photon maps. This counter shows how many independent threads are currently accessing corresponding map. If this counter is equal to zero that means that maps either do not exist or are not open for access by not-owner thread. If the counter is more than zero, then maps are available for other threads to be used. Only the thread that created photon maps can increase the counter from zero state. The owner thread can decrease the corresponding counter to zero only if it is equal to 1 that means that no other thread is currently using these maps. To open and close the access to the own backward photon maps at the beginning and end of the forward ray tracing step the following algorithm is used:

1. Increase the own backward photon maps active thread counter by 1 with an atomic increment operation. This would mark the own backward photon maps as ready to be used during forward ray tracing step for indirect and caustic luminance calculation by other threads.
2. Proceed to the forward ray tracing along with indirect and caustic illumination calculation using both own backward photon maps and backward photon maps of the other threads that are open for access at the moment.
3. When the desired number of forward rays are traced and processed try to perform the atomic compare-and-swap operation to decrease the own backward photon maps active thread counter from 1 to 0.
   a. If the compare-and-swap operation succeeds, then it means that the current thread was the last one accessing these backward photon maps and new access is successfully closed for other threads.
   b. If the compare-and-swap operation fails it means that some other thread is still accessing current backward photon maps and they cannot be closed at the moment. In this case more forward rays are traced and processed until this compare-and-swap operation can be successfully completed.

As result the thread that created the backward photon maps performs the forward ray tracing and processing until two conditions are fulfilled: first the desired number of forward rays are traced and second the access to the own backward photon map can be successfully closed. To guarantee that thread do not get stuck in the forward ray tracing step the forward ray tracing step the thread stops using other backward photon maps after the required number of forward rays is reached. The workflow of the algorithm of opening ang closing an access to the own backward photon maps at the forward ray tracing step is shown on the Fig. 6.

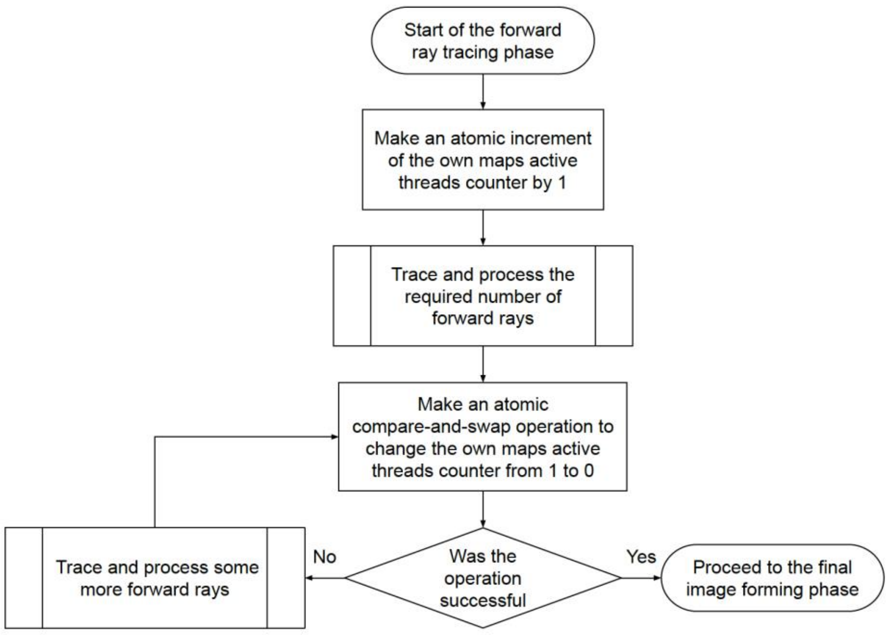
Start of the forward ray tracing phase
Make an atomic increment of the own maps active threads counter by 1
Trace and process the required number of forward rays
Make an atomic compare-and-swap operation to change the own maps active threads counter from 1 to 0
Was the operation successful
No
Yes
Trace and process some more forward rays
Proceed to the final image forming phase

Fig. 6. Opening and closing the access to the own backward photon maps at the forward ray tracing step.

The forward ray tracing step is also modified to account the indirect and caustic luminance that might be formed in backward photon maps of the other threads that are available at the moment of the forward ray tracing. To gain the access to the backward photon maps of the other threads it should temporary increase the other thread’s backward photon maps active thread counter to let it know that it is currently being used. The following operations are performed:

1. First of all, it should check the required backward photon maps active thread counter. If it is equal to zero, then maps are not available and should not be processed when calculating the indirect and caustic luminance formed by the current ray.
2. If the backward photon maps active thread counter is more than zero, then try to increment it by 1 with an atomic compare-and-swap operation.
3. If the compare-and-swap operation failed, then access was not granted, and this map is ignored when processing current forward ray.
4. If the compare-and-swap operation was successful, that means that access to the other thread’s backward photon maps was successfully granted.
5. If an access was granted, then intersection of forward ray with other thread’s backward photon maps is analyzed and in case of success the indirect and caustic luminance are accounted in corresponding backward photons. Also, the forward ray is accounted in the total energy processed by this backward photon map.

6. After backward photon map processing is finished the corresponding backward photon maps active thread counter is decreased by 1 with an atomic decrement operation to release the access.

All operations related to increasing the luminance accumulated in the backward photon map and corresponding ray counters are implemented using solely atomic operations to ensure correct luminance accumulation from concurrent threads. The workflow of the forward ray tracing with accounting luminance in all available backward photon maps is shown on the Fig. 7.

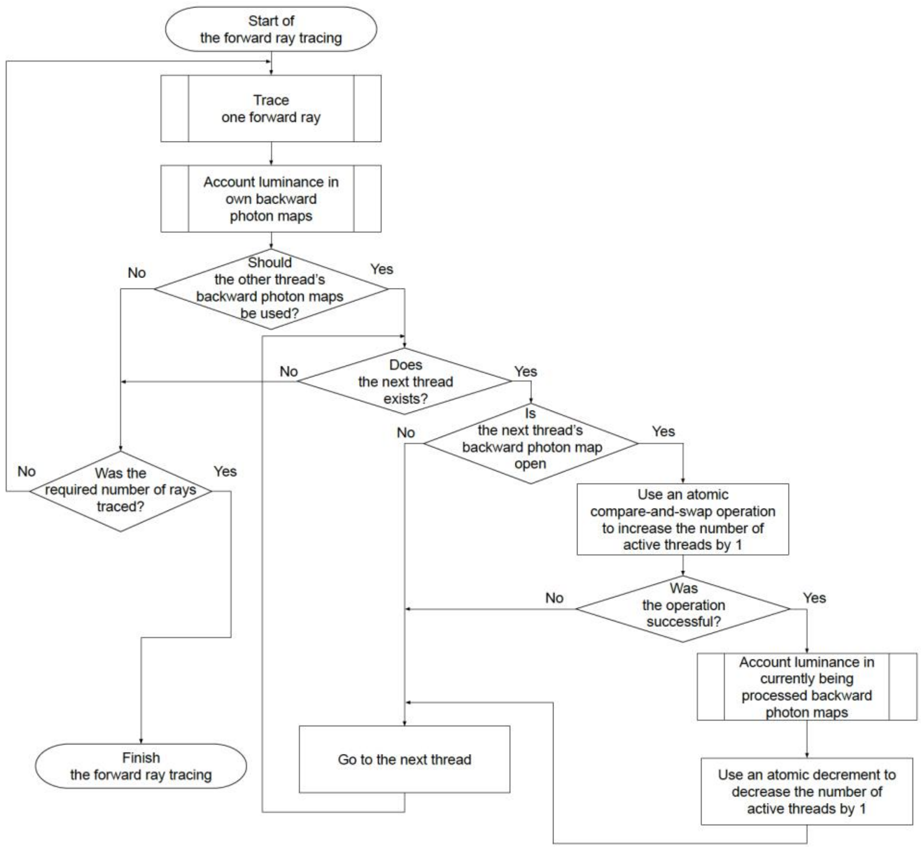
Start of the forward ray tracing
Trace one forward ray
Account luminance in own backward photon maps
Should the other thread's backward photon maps be used?
No
Yes
Does the next thread exists?
No
Yes
Is the next thread's backward photon map open
No
Yes
Was the required number of rays traced?
No
Yes
Use an atomic compare-and-swap operation to increase the number of active threads by 1
Was the operation successful?
No
Yes
Account luminance in currently being processed backward photon maps
Finish the forward ray tracing
Go to the next thread
Use an atomic decrement to decrease the number of active threads by 1

Fig. 7. The forward ray tracing with accounting luminance both in own backward photon maps and in backward photon maps of other threads.

As result at the step when the forward ray tracing is performed the thread's own backward photon maps are available for other threads to accumulate indirect and caustic luminance and all backward photon maps that are available at the moment of tracing the forward ray are used to account the ray's luminance. As only atomic operations are used in the backward photon maps access algorithms no special synchronization between threads is required that gives linear scalability of the rendering process when increasing the number of cores. Due to the uniform task distribution between different synchronous thread groups this overlapping is quite high and results in more effective forward ray tracing along with indirect and caustic luminance accumulation.

3 Results

The presented parallelization method was implemented and integrated in Lumicept light simulation software [15] and tested along with traditional synchronous and asynchronous methods on the same PC with the 12 cores of the Intel Xeon 6230 CPU and standard Cornell box scene. Ray tracing and processing speedup test results are presented in Tables 1-3 and on Fig. 9. Corresponding rendering results are shown on Fig. 8.

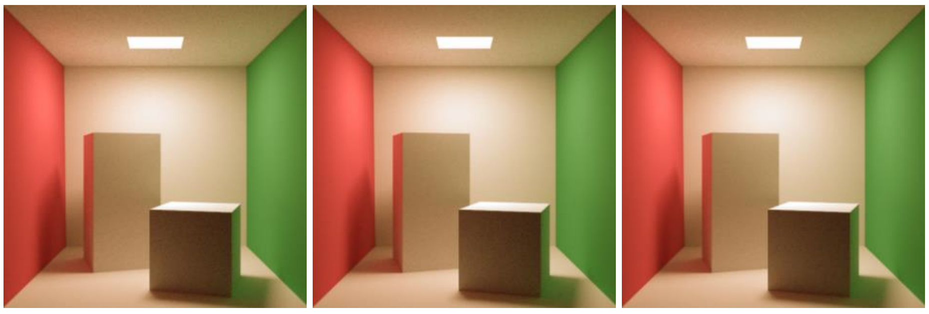

Fig. 8. Rendering results attained with synchronous, asynchronous and semi-synchronous methods (from left to right) after 10 minutes.

Table 1. Test results for synchronous parallelization method.

Cores used	1 core	2 cores	4 cores	8 cores	12 cores
Forward rays	20689305	34199691	53129762	64440303	71715295
Backward paths	26788848	44482520	68986597	83987647	93785374
Speedup	1	1.66	2.57	3.13	3.49

Table 2. Test results for asynchronous parallelization method.

Cores used	1 core	2 cores	4 cores	8 cores	12 cores
Forward rays	20717568	34101290	89476702	220730378	315318894
Backward paths	26790396	44481495	72347407	102960933	120367282
Speedup	1	1.65	3.41	7.81	9.17

Table 3. Test results for semi-synchronous parallelization method.

Cores used	1 core	2 cores	4 cores	8 cores	12 cores
Forward rays	20647944	53498805	120166069	220579897	338469540
Backward paths	26859521	43273242	55467718	72378193	93121907
Speedup	1	2.03	3.7	6.17	9.08

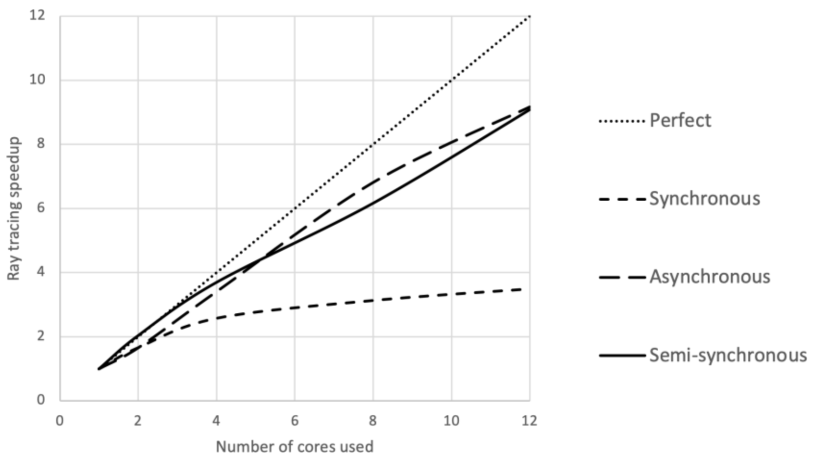

Fig. 9. Comparison of test results for synchronous, asynchronous and semi-synchronous parallelization methods.

As it can be seen the proposed approach have the same linear scalability of traced and process rays with the asynchronous calculation. After 10 minutes of calculations with the Cornell box test scene the synchronous calculations method achieved the accuracy of 3.9%, asynchronous calculation achieved the accuracy of 5.1% and the proposed semi-synchronous calculations achieved the accuracy of 2.6% as result of more effective usage of the backward photon maps formed by different computation threads.

The similar acceleration was achieved not only on Cornell box test scene, but also on scenes with light guiding optical systems, scenes containing volume scattering and others. Corresponding test scenes rendered images are shown on Fig. 10.

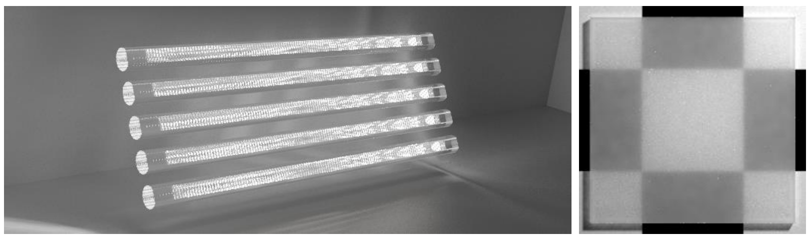

Fig. 10. Light guiding optical systems and volume scattering test scenes.

4 Conclusion

The developed two-level semi-synchronous parallelization method for the caustic and indirect luminance calculation can significantly increase the efficiency of calculating these luminance components on multicore systems. This approach allowed us to increase the rendering efficiency in non-parallelizable or poorly parallelizable algorithm sections and, when using small number of lower-level computation threads, to achieve an almost linear dependence of the rendering performance on the number of cores used. In addition, two-level semi-synchronous parallelization allows reducing the size of the backward photon maps in the upper-level threads, which makes it possible to speed up the process of maps voxelization and search of intersections with traced rays. The developed approach can be used as a part of a distributed rendering system, providing high performance on a remote server service.

References

1. Jensen, H. W.: Global illumination using photon maps. In: Proceedings of the eurographics workshop on Rendering techniques ’96, pp. 21–30. Springer-Verlag, London, UK (1996).
2. Kang, C. et al:. A survey of photon mapping state-of-the-art research and future challenges. In: Frontiers Inf Technol Electronic Vol 17, pp. 185–199. (2016).
3. Wald, I.: Embree ray tracing kernels: overview and new features. In: SIGGRAPH’16, New York, NY, USA, Art. 52. (2016).
4. Frolov, V.A.: Examination of the Nvidia RTX. In: Proceedings of Computer Graphics and Vision 2019, pp. 7-12. Bryansk, Russia. (2019).
5. He, H., Wang, T., Xu, Q., Xing, Y.: Multi-core Parallel of Photon Mapping In: Visual Information Communication, pp. 365-374, Springer, Boston, MA (2009).
6. Gunther, T., Grosch, T.: Distributed Out-of-Core Stochastic Progressive Photon Mapping. In: Computer Graphics Forum, vol. 33, pp. 154-166. (2014).
7. Schregle, R., Grobe, L.O., Wittkopf, S.: An out-of-core photon mapping approach to daylight coefficients. In: Journal of Building Performance Simulation 9:6, pp. 620-632. (2016).
8. Carlberg, K.: Stochastic Progressive Photon Mapping Using Parallel Hashing. Lund University, Sweden. (2011).
9. Fabianowski, B.: Interactive Manycore Photon Mapping. University of Dublin, Irleand. (2011).
10. Günther, J., Waldy, I., Slusallek, P.: Realtime Caustics Using Distributed Photon Mapping. In: Proceedings of the 15th Eurographics Workshop on Rendering Techniques, Norkoping, Sweden. (2004).
11. Jacobsson, M.: Distributed Progressive Photon Mapping. Lund University, Sweden. (2014).
12. Hachisuka, T., Jensen, H. W.: Stochastic Progressive Photon Mapping. In: Proceedings of SIGGRAPH Asia '09, Nr. 41, pp. 1-8, New York, NY, USA. (2009).
13. Havran, V., Herzog, G., Seidel, H.-P.: Fast Final Gathering via Reverse Photon Mapping. In: Proceedings of the Eurographics 2005, Vol. 24, Nr. 5, pp. 323-333. Blackwell Publishing, Oxford, UK (2005).
14. Amdahl, G.M.: Validity of the single processor approach to achieving large scale computing capabilities. In: Proceedings of the April 18-20, 1967, spring joint computer conference (AFIPS ’67 (Spring)), pp. 483–485. New York, NY, USA (1967).
15. Lumicept – Hybrid Light Simulation Software, http://www.integra.jp/en, last accessed 2020/07/01.