THE EFFECTIVE BIDIRECTIONAL RAY TRACING ON MULTIPROCESSOR WORKSTATIONS

Andrey Zhdanov\(^1\), Dmitry Zhdanov\(^1\), Vadim Sokolov\(^1\) and Vladimir Galaktionov\(^2\)

\(^1\)ITMO University, Russia
\(^2\)Keldysh Institute of Applied Mathematics, Russia

ABSTRACT

The computing power and complexity of computation architecture of modern computer systems used for image rendering is constantly rising, so effective parallelization of rendering methods is an urgent challenge. Modern workstations might have several CPUs with NUMA that makes traditional parallelization methods ineffective and requires research of the new approaches. In the scope of the current research, authors present a CPU parallelization algorithm for bidirectional ray tracing with photon maps that use three-level parallelization thread architecture and asynchronous calculations to achieve the high rendering scalability.

KEYWORDS

Photon Mapping, Rendering, Ray Tracing, Parallel Computing

1. INTRODUCTION

The problem of physically correct light propagation modeling and luminance calculation arise while solving a wide range of applied problems, including synthesis of the photorealistic images, optical effects modeling, virtual prototyping of the complex optical systems, and so on. With increasing computing power and complexity of the computation architecture of the modern computer systems, both the complexity of image synthesis tasks and the required calculation accuracy raise. The photon mapping method introduced by Jensen (1996) allows us to solve the rendering equation and account all types of illumination. Methods based on photon mapping are being developed by various groups of researchers. In their work, Kang et al. (2016) presented the current state of research in the area of the photon mapping methods. There are approaches aimed at using GPU, CPU, or both these types of calculators. Effective parallelization of photon mapping methods is an urgent challenge for the modern workstations which might have several CPUs with Non-Uniform Memory Access and dozens of cores, GPUs, and other types of calculators. Despite there are CPU and GPU manufacturers solutions aimed at the effective ray tracing as, for example, Intel Embree, presented by Wald et al. (2016), Nvidia RTX, examined by Frolov et al. (2019) or RadeonRays by AMD, the problem of the effective processing of the traced rays still exits. According to Amdahl's law (1967), the theoretical speedup of algorithm parallelization is limited and depends on a percent of work that cannot be parallelized. In the scope of the current research, authors have developed a CPU parallelization algorithm for bidirectional ray tracing with photon maps that pretend to minimize the non-parallelizable part of the calculation process by using three-level parallelization architecture and asynchronous calculations.

2. PARALLELIZATION METHOD

One of the methods to solve the rendering equation is to use the method of stochastic bidirectional ray tracing with photon maps. In the scope of the current research, we used the modified method of bidirectional ray tracing with photon mapping, where instead of photon map an imphoton (or visibility) map is formed. The general method algorithm consists of 3 main steps:
1. Backward paths from the camera are generated and traced in the scene. The direct light and BDF samplings are performed at path points with diffuse properties to account the direct luminance. The imphoton (visibility) map is formed at the points of paths with diffuse properties and an acceleration structure is built to speed up the further maps access.

2. Forward rays from light sources are generated and traced in the scene. By intersecting with previously formed imphoton maps, the indirect and caustic luminance is accumulated in the corresponding pixels of the intermediate image.

3. An intermediate image formed at the current phase is added to the final image and accuracy is estimated. If attained accuracy is not enough, then calculations continue from the first step.

The bidirectional ray tracing with imphoton maps is hardly parallelized and requires special techniques to use all computation resources of a modern workstation in the most effective way. The main problems arise when implementing this algorithm to be used on a modern workstation with several CPUs and a big number of computation cores. Traditionally, the rendering parallelization method is as following: the high-resolution image is split into blocks (mainly of 32x32 pixels size) and these blocks are rendered on CPU cores one by one in a single task scheduler queue. If the number of CPU cores is relatively high and is close to the number of computation blocks, then it is highly possible that some threads would finish their job earlier and would stay unoccupied until all threads finish their jobs as all blocks are already either finished or being rendered. Taking into account that most computer graphics scenes are not uniform, these blocks would require different time to be calculated which would also result in a significant difference in the computation load for different cores. When using single-level parallelization, it is not possible to proceed to the next step before the previous step is finished. For example, it is not possible to start imphoton map forming step before all backward rays are traced. Also, the single level parallelization would result in a huge size of the imphoton map that requires advanced parallelization algorithms for processing and optimization. So, the authors developed the three-level parallelization method that allows performing part of the operations asynchronously, as a result, reducing the synchronization time loss and speeding up the algorithm performance on multi-CPU workstations.

The first parallelization level utilizes several computation threads that perform rendering of the whole scene in parallel and renders only the part of the whole image. This part is defined by a random 32x32 pixel mask that is applied to the whole image. Each of the first level group thread uses the same shared memory, shared scene, shared maps, and runs synchronously through the whole rendering process steps. In general, it is simple single-level parallelization method for a random part of the whole synthesized image.

Each of the second-level group threads is the main thread of the first level group, each of them computing their own part of the image basing on the random mask assigned by the main thread. Once in several predefined number of phases, the whole intermediate image is formed, added to the final image, and masks are re-randomized. The randomization of masks leads to the more uniform task distribution between the level 1 thread groups. The level 2 thread groups run partly synchronously. These threads use the same shared memory, however, each of them uses their own maps and render their own part of the whole image at the same time having access to the other threads maps when they do exist to calculate the indirect and caustic luminance components. The group of first and second level threads is shown in Figure 1.

![Figure 1. The first and second level thread groups](image)

By using synchronous and asynchronous calculation at the same time authors have achieved almost full utilization of single CPU resources with shared memory. However modern workstations might have Non-Uniform Memory Access (NUMA) architecture. In NUMA architecture the CPU cores are split between several NUMA nodes and each of these NUMA nodes has its own local memory. At the same time, NUMA nodes have low-speed access to other NUMA nodes memories. In this case, the shared memory usage is no
longer effective, so authors have introduced the third level of parallelization that is aimed at effective usage of the whole resources of a workstation that has several NUMA nodes. The third level uses groups of the second level threads and binds each of them to a specific NUMA node. These groups have no shared resources and do not communicate with each other directly in any way. Each of these groups works solely on their own NUMA node and utilizes only local memory. The main thread of the third level group is at the same time the main thread of the second level group. Once in several, relatively big number of phases, the main thread fetches the current state of the image from all other second level main threads, forms the total image, and updates the total accuracy. This is performed without interrupting the calculations of the second level groups and keeping the asynchronous nature of calculations. The group of the third level threads is shown in Figure 2.

![Figure 2. The three levels of threads](image)

As this computation model is highly asynchronous it can utilize all the CPU resources in a quite effective way and have high scalability when changing the number of cores used for computation.

To sum up, the computation process parallelization levels are:

1st. Completely synchronous level. Threads compute the same image with the same random mask.

2nd. Partly synchronous level. Thread groups of the first level compute their own parts of the whole image with partly shared memory (scene and imphoton maps) and synchronize after several phases. At synchronization points masks are re-randomized.

3rd. Completely asynchronous level. Thread groups of the second level compute their own whole images and the main thread accumulates current computed images from these groups over several phases interval to form the final image without interrupting the computation process.

### 3. SIMULATION RESULTS

We have implemented the proposed parallelization method and tested it on the Cornell Box scene and the light guides placed in the Judge II viewing booth scene. The test equipment was a workstation with double AMD EPYC 7281 16 cores CPUs resulting in a total of 32 cores. All simulations were performed with the spectral color model. The speedup estimation was based on the number of rays traced and processed when rendering a scene with the 10 minutes time limit. Table 1 shows the bidirectional ray tracing with the imphoton maps speedup test for the Cornell box scene using the developed method. Table 2 shows the speedup test of a simple single-level parallelization method. Figure 3 shows the speedup graph based on the number of cores used and the final synthesized image when all 32 cores were used.

**Table 1.** The number of rays traced when varying the number of used cores. Three-level method. Cornell box scene

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>1 core</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
<th>24 cores</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward paths</td>
<td>35 936 168</td>
<td>71 797 941</td>
<td>138 834 968</td>
<td>263 419 810</td>
<td>403 687 733</td>
<td>514 062 777</td>
</tr>
<tr>
<td>Forward rays</td>
<td>27 682 024</td>
<td>155 974 036</td>
<td>300 251 153</td>
<td>564 103 834</td>
<td>867 447 398</td>
<td>110 115 351</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>3.58</td>
<td>6.90</td>
<td>13.01</td>
<td>19.98</td>
<td>25.39</td>
</tr>
</tbody>
</table>

**Table 2.** The number of rays traced when varying the number of cores used. Single-level method. Cornell box scene

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>1 core</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
<th>24 cores</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward paths</td>
<td>36 495 918</td>
<td>80 342 047</td>
<td>112 646 486</td>
<td>98 916 448</td>
<td>70 904 587</td>
<td>61 734 479</td>
</tr>
<tr>
<td>Forward rays</td>
<td>28 104 980</td>
<td>173 091 798</td>
<td>339 242 857</td>
<td>392 582 996</td>
<td>409 392 886</td>
<td>455 235 368</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>3.92</td>
<td>7.00</td>
<td>7.61</td>
<td>7.43</td>
<td>8.00</td>
</tr>
</tbody>
</table>
Figure 3. To the left: the speedup of the calculations depending on the number of used cores. To the right: synthesized image with 32 cores and three-level method. Cornell box scene.

Tables 3 and 4 show the speed test for lightguides in the Judge II viewing box scene. Figure 4 shows the speedup graph based on the number of cores used and the synthesized image when all 32 cores were used.

Table 3. The number of rays traced when varying the number of cores used. Three-level method. Light guides scene

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>1 core</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
<th>24 cores</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward paths</td>
<td>12 933 960</td>
<td>24 560 276</td>
<td>47 597 995</td>
<td>93 309 854</td>
<td>147 415 145</td>
<td>191 567 253</td>
</tr>
<tr>
<td>Forward rays</td>
<td>1 241 843</td>
<td>22 673 023</td>
<td>44 330 204</td>
<td>81 280 703</td>
<td>133 191 853</td>
<td>176 613 858</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>3.33</td>
<td>6.48</td>
<td>12.31</td>
<td>19.79</td>
<td>25.97</td>
</tr>
</tbody>
</table>

Table 4. The number of rays traced when varying the number of cores used. Single-level method. Light guides scene

<table>
<thead>
<tr>
<th>Number of cores</th>
<th>1 core</th>
<th>4 cores</th>
<th>8 cores</th>
<th>16 cores</th>
<th>24 cores</th>
<th>32 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backward paths</td>
<td>13 566 182</td>
<td>30 072 348</td>
<td>31 856 759</td>
<td>24 031 276</td>
<td>20 370 604</td>
<td>16 356 663</td>
</tr>
<tr>
<td>Forward rays</td>
<td>1 308 936</td>
<td>27 687 893</td>
<td>71 095 408</td>
<td>79 462 634</td>
<td>93 799 644</td>
<td>98 508 600</td>
</tr>
<tr>
<td>Speedup</td>
<td>1</td>
<td>3.88</td>
<td>6.96</td>
<td>7.57</td>
<td>7.68</td>
<td>7.72</td>
</tr>
</tbody>
</table>

Figure 4. To the left: the speedup of the calculations depending on the number of used cores. To the right: synthesized image with 32 cores and three level method. Light guides scene.

From the presented simulations results the advantages of the presented method can be seen clearly. The speedup of the proposed approach on test scenes is close to the linear one. The authors have also tested the proposed approach on workstations with up to 64 cores with similar results.

4. CONCLUSION

In the scope of the current research, the parallelization of the bidirectional ray tracing with imphoton maps was developed that allows keeping the calculation speedup at an almost linear rate when rising the number of used computation cores on modern multiprocessor workstations. In the following research, authors are going to extend this method to use modern heterogeneous workstations with both several CPUs and GPUs.
ACKNOWLEDGEMENT

This work was funded by the Russian Science Foundation (project No. 18-79-10190). The design and calculation of the light guides scene were done by V. G. Sokolov and V. A. Galaktionov.

REFERENCES


