Constructing K-d Tree on GPU through Treelet Restructuring [*]

Vadim Bulavintsev[1][0000-0003-3099-1442], Dmitry Zhdanov[2][0000-0001-7346-8155]

[1] ITMO University, 49 Kronverksky Pr., St. Petersburg, 197101, Russia
v.g.bulavintsev@gmail.com
2 ITMO University, 49 Kronverksky Pr., St. Petersburg, 197101, Russia
ddzhdanov@mail.ru

Abstract. With every new generation of graphics processing units (GPUs), offloading ray-tracing algorithms to GPUs becomes more feasible. Software-hardware solutions for ray-tracing focus on implementing its basic components, such as building and traversing bounding volume hierarchies (BVH). However, global illumination algorithms, such as photon mapping method, depend on another kind of acceleration structure, namely *k*-d trees. In this work, we adapt state-of-the-art GPU-based BVH-building algorithm of treelet restructuring to *k*-d trees. By evaluating the performance of the resulting *k*-d tree, we show that treelet optimisation heuristic suitable for BVHs of triangles is inadequate for *k*-d trees of points.

Keywords: Graphics Processing Unit, Kd Tree, Bounding Volume Hierarchy, Photon Mapping, Ray-tracing.

1 Introduction

Over the last three decades, physically correct rendering became a ubiquitous method of data visualization in scientific, educational, industrial design and digital entertainment fields [1,2]. Meanwhile, advances in GPU design brought consumer computational devices enough processing power to make physically correct rendering feasible in interactive applications.

In this paper we discuss recent advancements in acceleration structures – building algorithms used to facilitate physically-correct rendering with GPUs. We apply a method of building an acceleration structure for ray-tracing for GPUs originally used with Bounding Volume Hierarchies (BVHs) to k-d trees.

The paper is organised as follows. In the rest of the "Introduction" section we describe the necessary preliminaries, such as definitions of acceleration structures and

[*] Publication is supported by RFBR grant № 18-08-01484.

heuristics, as well as specifics of GPU programming. In the “Related Works” section we review the development of GPU-based acceleration structures construction and traversal methods over the years. In the "Methods" section, we describe the choice of the original BVH-building method and the changes that were necessary to adapt it to k-d trees. Also, we describe the choice of test scenes and experimental setup. In the "Results" section, we compare the performance of our method to state-of-the-art top-down methods for building k-d trees on CPU and to the corresponding BVH-building method. In the "Discussion" section, we analyse the problems we encountered while adapting the method and propose the directions for further research.

1.1 Photon mapping

Photon mapping is a robust method for solving the rendering equation [3]. The method is able to reproduce complex optical effects, such as caustics [4]. It consists of three stages:

- cast light rays from into the scene and note the luminance at the rays' points of intersection ("photons") with the scene surfaces;
- do the same for visibility rays emitted by the camera;
- for each visibility point, gather the photons in its vicinity to calculate the approximate luminance.

1.2 Acceleration structures

To perform the photons gathering in the third stage of the photon mapping algorithm, it is essential to organise the photons into an acceleration structure, such as the k-d tree or a Bounding Volume Hierarchy.

***K*-d tree.** *K*-d tree is a binary tree that partitions the scene volume into smaller volumes by assigning an axis-aligned split plane for one of the scene's k dimensions at each internal node [5]. The process of building a tree of volume elements is called *voxelization*.

Bounding volumes hierarchy. Another important acceleration structure used in rendering is the bounding volumes hierarchy (BVH) [6]. BVH partitions the scene space into a tree of axis-aligned bounding boxes. Building BVHs using massively parallel architectures (e.g. GPUs) got much attention from researchers and industry [7, 8, 9] during the last decade. The goal of the present work is to apply one of the more successful methods for building BVHs with GPUs to k-d trees.

Top down vs bottom-up building. When building a tree-like acceleration structure, there are two primary strategies. Splitting the whole space according to some heuristic, then proceeding to apply the same heuristic to the resulting subspaces is called top-down approach. Combining the smallest elements of the set (e.g. photons or triangles) into gradually bigger units, effectively building the tree from its leaves is called bottom-up approach.

Top-down methods are known to produce the most effective acceleration structures [10]. However, top-down methods are not suitable for massively-parallel architectures, such as GPU. This led to development of alternative, bottom-up construction methods.

Over the years, several bottom-up methods were suggested by researchers. The basic method was coined in by Lauterbach et al. in [8]. The method's idea is to sort the primitives based on their coordinates on a Morton curve [11], then interpret them as a tree. The resulting structure is called Linear BVH (LBVH). This method was further refined in [12].

The next generation of bottom-up methods uses LBVH as a starting point, then building up to optimize it by e.g. changing the internal nodes [9] or iteratively merging them bottom-up [13].

It is important to note that top-down methods, while producing more efficient acceleration structures, tend to be much more computationally intensive than bottom-up methods, therefore requiring choosing a trade-off between the structure's quality and building time. This makes finding a fitting trade-off a critical part of building a real-time rendering system.

1.3 Surface area heuristic

Beside the actual tree construction method, the most important thing in an acceleration-structure-building algorithm is the heuristic that selects the splitting point. It is important to note that the actual effectiveness of the acceleration structure is highly scene-dependent [10]. Nonetheless the most popular heuristics are the median heuristic for k-d trees and the Surface Area Heuristic (SAH) [14] for BVHs.

Surface Area Heuristic. The idea of the Surface Area Heuristic is to optimize for a general case of rays traversing the scene at random angles. At each splitting the heuristic minimizes the summary surface area of the resulting bounding volumes in any given subtree.

1.4 GPU architecture

Modern GPUs are SIMT [15] devices, which stands for "Single Instruction Multiple Threads" architecture, a variant of SIMD architecture [16] with hardware control of branch divergence. SIMD architecture features an asymmetric ratio of control (CUs) to processing units (PUs), making it efficient at processing extensive collections of homogenous data [17]. A GPU consists of independent multiprocessors, each of which has a single CU controlling multiple PUs. To provide data to all PUs of a single multiprocessor at once, GPU memory controller is optimized to fetch long words. Typically, GPU's on-chip cache size is small in regards to the number of PUs. To hide memory access latencies a single multiprocessor pipelines execution of several thread groups, switching between them when the data from GPU memory becomes available. Threads running on a single multiprocessor share the register file. Thus, register pressure limits the number of threads running on a single multiprocessor.

These quirks of GPU architecture result in the following rules of efficient GPU programming [17]:

- avoid branch divergence;
- avoid non-coalesced memory access (gather) in a thread group;
- minimize register usage and device memory access.

Modern GPUs have recently received support of hardware acceleration for some ray-tracing operations, such as BVH tree traversal. Unfortunately, the underlying hardware instructions remain undocumented [18] and are only usable through a closed-source API [19].

2 Related works

Being an "embarrassingly parallel" problem, ray-tracing is well suited to GPU architecture. However, following the rules mentioned above for such stages of a ray-tracing algorithm as building and traversal of acceleration structures is non-trivial.

Firstly, top-down methods for constructing BVHs and kd-trees are bottlenecked by the low parallelism during the beginning of the build process. Researchers proposed several ways to alleviate this problem, such as using trees and BVHs of higher arity or using hybrid CPU/GPU algorithm [20]. Alternatively, it is possible to use bottom-up building algorithms [8, 12] as mentioned in the previous section.

Secondly, tree traversal can result in high branch divergence when, for example, a single thread in a thread group proceeds into a deep branch of the tree when the other threads in the same group have already terminated. The problem can be tackled both from the side of traversal algorithm by modifying it to use "persistent threads" technique [21] and from the side of tree construction algorithm by avoiding creating deep leaves.

Thirdly, while persistent threads method solves the problem of branch divergence, it can result in scattered memory access. One way to solve this problem is to use prefix scans, and regularly sort the threads based on locality [22].

Fourthly, GPU cache/on-chip memory proved to be too small to support stack-based tree traversal methods, which motivated the development of stackless methods as an alternative [23]. Though, in recent hardware generations the cache constraints were somewhat relieved.

A detailed analysis of the problem can be found in [24, 25].

3 Methods

One of the reasons ray-tracing became feasible on GPUs is the rapid development of algorithms for building high-quality BVHs on massively parallel processors, such as GPUs [7, 9]. We decided to base our algorithm on TRBVH algorithm [9] that is natively implemented in NVIDIA Optix API [19], has some open-source implementations [13] and is readily suitable to be converted to work with k-d trees [12]. We reimplemented the algorithm in Python programming language to facilitate rapid prototyping. As the Python prototype does not run on a GPU, we studied relative and hardware-independent performance metrics, such as the average number of traversed nodes and SAH cost.

3.1 TRBVH algorithm

The idea of TRBVH algorithm is to first build a lower-quality BVH by utilizing Morton codes (the so-called Linear BVH) [12] and then enhance its quality selecting small (e.g. 7-9 nodes) "treelets" from it and optimize them by SAH heuristic [9].

The LBVH is built in the following way. Coordinates of the primitives (e.g. a photons or triangle centroids) are converted to 30-bit Morton codes. The resulting array is sorted using Radix sort [17]. Then, the sorted points can be readily interpreted as a k-d tree by comparing the radix prefix of their Morton codes as described in [12]. The algorithm is very efficient on GPU as it uses the fact that operations for individual elements are independent of each other.

At the optimization stage, the algorithm starts to traverse the LBVH bottom-up, selecting 7-nodes breadth-first subtrees (“treelets”) to optimize. For each treelet, the algorithm compares all possible variants of organizing four of its leaves into a new subtree, comparing the results by SAH. To avoid repeated SAH recalculations, the algorithm employs methods of dynamic programming [9]. Treelet optimization processes run in parallel, using atomic write instructions to avoid data races.

4 Results

4.1 Algorithm modifications

Every k-d tree can be interpreted as a BVH, but not every BVH can be interpreted as a k-d tree. BVH nodes' volumes can overlap, which is impossible for k-d tree nodes. Thus, converting the linear part of the TRBVH algorithm to produce k-d trees is straightforward because of Morton codes natively splitting the scene space into non-overlapping volumes. However, converting the treelet optimisation part requires modifying the original SAH heuristic to avoid overlapping volumes. To achieve this, we assign ∞ heuristic cost to permutations that would result in overlapping child volumes. Also, TRBVH works with triangles, while the photon map consists of points (spheres), so we had to change the algorithm accordingly.

4.2 Algorithm performance

We tested the performance of the algorithm with Sibenik, Stanford Dragon and Conference scenes [26] and Lumicept light modeling system [27]. We rendered each scene using Lumicept’s reverse path tracing mode. In this mode, instead of emitting light rays (called photons) from light sources, Lumicept emits image photons from the camera. For each scene, we randomly selected 105 of the generated image photons, then built two k-d trees by using our Python-based voxelizer prototype. The first tree was built using linear Morton codes (LBVH) [9]. The second tree was built by refining the first tree using our treelet-based method. We tested the tree performance by counting the total number of tree node traversals necessary to find 1000 image photons randomly selected from the original set.

Table 1 shows improvements in SAH cost and ray-tracing performance for k-d trees built with our treelet-based method relative to k-d trees built with linear Morton codes (LBVH). Usage of relative values allows us to compare results obtained by using our Python-based emulation of a k-d tree GPU algorithm to the results reported by Karras for BVHs in [9]. The data was averaged over two camera positions facing opposing sides of a scene.

Table 1. k-d tree and TRBVH performance compared to linear BVH (100%)

Performance in comparison to linear Morton codes // LBVH	Dragon		Sibenik		Conference	
	SAH surface	Search speed	SAH surface	Search speed	SAH surface	Search speed
TRBVH (BVH) [9]	86%	113%	78%	129%	59%	153%
Our method (k-d tree)	88%	107%	89%	102%	90%	98%

5 Discussion

Table 1 indicates that treelet restructuring does not increase the quality of the resulting k-d tree tangibly. We can attribute this effect to the following. First, forbidding volume overlaps cuts off more than 80% of the search space. Second, SAH heuristic is known to be better suited to BVHs of triangles, not k-d trees of points. This can be clearly seen with the Conference scene: after applying treelet restructuring the SAH cost goes down for both k-d tree (our algorithm) and TRBVH. However, the traversal performance goes up only for the BVH case, while for the k-d tree it actually decreases. Thus, the direct translation of BVH-building methods to the domain of k-d trees is not feasible. This observation gives us clear direction for future research: voxelizers based on the idea of tree optimization should employ an alternative, more suitable heuristic.

References

1. Barladian, B., Voloboy, A., Galaktionov, V., Shapiro, L.: Integration of realistic computer graphics into computer-aided design and product lifecycle management systems. Programming and Computer Software, 44(4), 225-232 (2018)
2. Zhdanov, D., et al.: Photorealistic rendering of images formed by augmented reality optical systems, Programming and Computer Software, 44(4), pp. 213-224 (2018)
3. Jensen, H.: Realistic image synthesis using photon mapping. AK Peters/CRC Press (2001)
4. Papadopoulos, C., Papaioannou, G.: Realistic real-time underwater caustics and godrays. In Proc. GraphiCon 2009, vol. 9, pp. 89-95 (2009).
5. Shevtsov, M., Soupikov, A., Kapustin, A.: Highly parallel fast KD-tree construction for interactive ray tracing of dynamic scenes. In Computer Graphics Forum, vol. 26(3), pp. 395-404, Oxford, UK: Blackwell Publishing Ltd (2007).
6. Gunther, J., Popov, S., Seidel, H., Slusallek, P.: Realtime ray tracing on GPU with BVH-based packet traversal. In 2007 IEEE Symposium on Interactive Ray Tracing, pp. 113-118, IEEE (2007)
7. Garanzha, K., Premože, S., Bely, A., Galaktionov, V.: Grid-based SAH BVH construction on a GPU. The Visual Computer, 27(6-8), pp. 697-706 (2011)
8. Lauterbach, C., et al.: Fast BVH construction on GPUs., Computer Graphics Forum., vol. 28(2)., Oxford, UK: Blackwell Publishing Ltd (2009)
9. Karras, T., Aila, T.: Fast parallel construction of high-quality bounding volume hierarchies. In Proc. of the 5th High-Performance Graphics Conference, pp. 89-99 (2013)
10. Aila, T., Karras, T., Laine, S.: On quality metrics of bounding volume hierarchies. In Proc. of the 5th High-Performance Graphics Conference, pp. 101-107 (2013)
11. Morton, G.: A computer oriented geodetic data base and a new technique in file sequencing, Technical Report, Ottawa, Canada: IBM Ltd (1966)
12. Karras, T.: Maximising parallelism in the construction of BVHs, octrees, and k-d trees., In Proc. of the Fourth ACM SIGGRAPH/Eurographics conference on High-Performance Graphics (2012)
13. Domingues, L., Pedrini H.: Bounding volume hierarchy optimisation through agglomerative treelet restructuring, in Proc. of the 7th Conference on High-Performance Graphics (2015)
14. MacDonald, J., Booth, K.: Heuristics for ray tracing using space subdivision, The Visual Computer, 6(3), pp.153-166 (1990)
15. Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J.: NVIDIA Tesla: A unified graphics and computing architecture., IEEE micro, 28(2), pp. 39-55 (2008)
16. Flynn, M.: Some computer organizations and their effectiveness., IEEE transactions on computers, 100(9), pp. 948-960 (1972)
17. Bulavintsev, V.: An evaluation of CPU vs. GPU performance of some combinatorial algorithms for cryptoanalysis, Vestn. YuUrGU. Ser. Vych. Matem. Inform., 4:3, pp.67–84 (2015)
18. Sanzharov, V., Gorbonosov, A., Frolov, V., Voloboy, A.: Examination of the Nvidia RTX. In Proceedings of the 29th International Conference on Computer Graphics and Vision (GraphiCon 2019), vol. 2485, p. 7 (2019)
19. Parker, S., et al.: OptiX: a general purpose ray tracing engine. ACM transactions on graphics (TOG) vol. 29.4, p1-13 (2010)
20. Roccia, J. P., Coustet, C., Paulin, M.: Hybrid CPU/GPU KD-Tree construction for versatile ray tracing, Eurographics (Short Papers), Euro-graphics Association, pp. 13–16. (2012)
21. Gupta, K., Stuart, J. A., Owens, J. D.: A study of persistent threads style GPU programming for GPGPU workloads, IEEE, pp. 1-14., (2012)

22. Garanzha, K., Loop, C.: Fast ray sorting and breadth-first packet traversal for GPU ray tracing., Computer Graphics Forum, Vol. 29, No. 2, pp. 289-298. Oxford, UK: Blackwell Publishing Ltd. (2010)
23. Popov, S., Günther, J., Seidel, H. P., Slusallek, P.: Stackless KD-tree traversal for high performance GPU ray tracing., Computer Graphics Forum, Vol. 26, No. 3, pp. 415-424. Oxford, UK: Blackwell Publishing Ltd. (2007)
24. Aila, T., Laine, S.: Understanding the efficiency of ray traversal on GPUs, In Proceedings of the conference on high performance graphics 2009, pp. 145-149 (2009)
25. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for manycore GPUs., In 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1-10, IEEE (2009)
26. McGuire, M.: ”Computer Graphics Archive”, URL https://casual-effects.com/data (2017).
27. Integra Inc., Lumicept – Hybrid Light Simulation Software, URL http://www.integra.jp/en/products/lumicept (2020)