SIGRAD 2015
L. Kjelldahl and C. Peters (Editors)

Complete Quadtree Based Construction of Bounding Volume Hierarchies for Ray Tracing

Ulises Olivares[†1] Arturo García[2] and Félix F. Ramos[1]

[1] Departament of Electrical Engineering Center for Research and Advanced Studies of the National Polytechnic Institute, México
[2] Intel Corporation

Abstract

This paper presents an efficient space partitioning approach for building high quality Bounding Volume Hierarchies using x86 CPU architectures. Using this approach a structure can be built faster than a binned-SAH heuristic while the structure preserves its quality. This method consists of a hybrid implementation that uses binned-SAH for the top level and a binary partitioning approach for the rest of the levels. As a result, this method produces more regular axis-aligned bounding boxes (AABB) into a complete quadtree. Additionally, this approach takes advantage of the 4-wide vector units and exploits the SIMD extensions available for current CPU architectures. Using our construction approach a structure can be built up to three times faster than binned-SAH.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Ray tracing is a rendering technique that produces high quality images. Nevertheless, it requires a high computational cost to produce a single image. This have generated special interest in using compute-intensive architectures such CPUs [WIK*06, SSK07], GPUs [LGS*09, ZHWG08] based implementations or even specialized architectures [NPP*11, DFM13]. Current improvements have been proved that is feasible to get interactive frame-rates even for complex scenes [Wal12, GMOR14, PFL*13].

However, although an acceleration structure decreases the complexity of ray tracing, it requires some time to be built, and this time depends on the quality of the construction and consequently on the heuristic that a structure uses. The use of sophisticated heuristics to estimate the cost of ray traversal tends to have a negative impact in the construction times but a positive impact in ray traversal performance, in an analogue way simpler algorithms tend to have faster construction times but a poorer ray traversal performance. The approach proposed in this paper tries to find a trade-off between complex split methods and fast split methods (get faster construction times while it maintains the quality of the structure).

Currently, most of the research oriented to computer graphics uses GPU architectures to get lower bounds of execution times. Since it is possible to adapt complex structures to a linear representation [LGS*09], GPUs have been the compute-intensive architecture for excellence. Nonetheless, Wald et al. [WWB*14] have demonstrate that is possible to get competitive frame-rates by exploiting full compute capability of current x86 CPU architectures.

This paper proposes a method for building Bounding Volume Hierarchies (BVHs) for ray tracing based on a hybrid splitting approach. This approach uses binned-SAH for the top level in order organize primitives coherently and then it uses a binary split approach for the rest of the levels. This partitioning scheme produces a regular data structure into a complete quadtree, which ensures that all levels of the tree will be completed. This construction takes advantage of the 4-wide vector units and exploits the SIMD extensions available for current x86 CPU architectures. Additionally, this construction uses the integrated GPU on x86 architectures to perform compute-intensive operations efficiently.

† PhD Student of Computer Science at CINVESTAV México

2. Construction Overview

This section addresses a new approach for the construction of BVHs using a complete tree.

2.1. Complete Quadtree

In the construction approach presented by Garcia et al. [GMOR14] one of the main problems was the overhead produced by sorting primitives in each level of the tree and it produced lower construction times. In this paper, we proposed an approach that avoids this overhead by using arithmetic operations to select which primitives will be contained in the left or right internal sub-trees. This partition scheme is recursively applied until each level of the quadtree is filled. Finally, this partition approach produces a complete quadtree, this structure takes advantage of the 4-wide vector units of x86 CPU architectures.

In order to have a complete tree, it is necessary to determine how many primitives will be stored on every branch of the tree ensuring that only the deepest level could be incomplete.

Assuming that the number of leaf nodes N, is greater or equal than 4 (branching factor), where k represents the radix of the most significant bit of the binary representation of N, and r represents the complement $r = |N| - 2^k$ (Figure 1 depicts the binary representation of the number of primitives). Then, it can be two different cases: the binary representation of N is $10..._2$ or is $11..._2$, This binary representation follows a big endian order where the most significant bit is on the left side. In the first case, there are not enough leafs to fully complete the left side and thus the right side should be full, if $N = 2^k + r$ then we should have $2^{k-1} + r$ leafs on the left side and 2^{k-1} on the right side In the second case, there are enough leafs to complete the left side and thus will have 2^k leafs while the right side will hold r leafs, for this case, $r \geq 2^{k-1}$.

The process just described provides the rules to determine the spatial partition (cut) in a given axis, it is represented by these two cases:

$$|N| = 2^k + r \tag{1}$$

$$|N| = \begin{cases} 10..._2, & \text{(2a)} \\ 11..._2, & \text{(2b)} \end{cases}$$

$$10..._2 = \begin{cases} 2^{k-1} + r, & \text{for } left \quad \text{(3a)} \\ 2^{k-1}, & \text{for } right \quad \text{(3b)} \end{cases}$$

$$11..._2 = \begin{cases} 2^k, & \text{for } left \quad \text{(4a)} \\ r, & \text{for } right \text{ with } r \geq 2^{k-1} \quad \text{(4b)} \end{cases}$$

Lets assume there exist 13 primitives in a scene, so the number of primitives is given by $|N| = 2^k + r = 13$ from this formula we can obtain the most significant radix of the binary representation $2^k = 2^3 = 8$ and Finally, it is easy to obtain the r value from the original equation 1, $r = |N| - 2^k = 5$. Following the partition rules, we can calculate the number of primitives that will be contained in the left and right sides respectively.

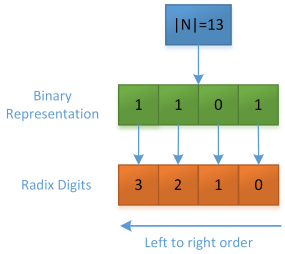

Figure 1: *Binary representation of primitives to obtain 2^k, where k = radix digit.*

2.2. Hybrid Construction

We employed a hybrid construction to take advantage of a complex split method that produce high quality Bounding Volume Hierarchies [Wal07] binned-SAH and a fast binary split approach that produces a complete tree just described above. We employed a binned-SAH on the top of the tree and the binary partitioning for the rest of the tree. The first partition ensures that coherent primitives are contained in the same bounding box and then, the binary split approach ensures that the tree structure will be a complete tree.

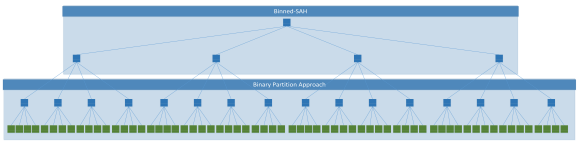

Figure 2: *Hybrid Construction Approach. It uses binned-SAH for the top of the tree and our binary partitioning approach for the rest of the tree.*

2.3. Space Partitioning

Once we have selected the number of primitives that every side will have, the next step is to decide which primitives will be stored in the left side and which will be stored int the right side. To do so, we find the lowest and highest values of the coordinates of the primitive centers using the axis that has the longest range, we select the primitives with the lowest value on that axis to be in the left side and the rest in the right side. Geometrically, we are selecting the longest side of the bounding box that contains all the primitive centers,

and splitting the primitives based on the order given by the projection of the centers on that side. Using this method will generate boxes that tend to be more regular, by reducing the length of the longest side.

Once the complete tree is built we know how many primitives will be contained in each node, then we need to perform the space partition in order to assign the primitives to each node. The space partitions are executed on the longest axis in order to improve the traversal performance by localizing the major density of adjacent primitives in the left branch of the tree.

The space partitioning is done following these steps:

1. Find the longest axis
2. Map Primitives
3. Split primitives

2.3.1. Find the Longest Axis

This is determined by doing a parallel reduction based on an algorithm presented by Hillis et al. [HS86] to sum an array of n numbers that can be computed in time $O(logn)$ by organizing the addends at the leaves of a binary tree and performing the sums bottom-up at each level of the tree in parallel. Our function returns the longest axis by calculating the *max* and *min* values in x, y and z of an array of primitives, then the function calculates the distance between the *max* and *min* points for each axis and returns the axis that has the longest distance.

This compute-intensive operation was parallelized using Open CL and the integrated Graphics Processor Unit on x86 Intel current architectures (see Figure 3). Using the integrated GPU avoids communication latency which is implied when information is being transferred from the CPU to GPU using the PCI communication bus.

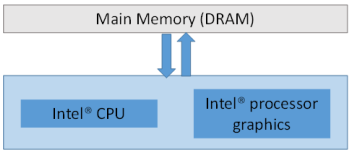

Figure 3: *Shared memory on Intel architecture [Cor14].*

2.3.2. Map Primitives

Garcia et al. [GMOR14] proposed a fast data parallel radix sort implementation to sort the primitives against the split axis in ascending order based on the centroids. Nonetheless, this sorting caused an overhead during the construction. As an alternative we proposed a simplest approach that map a primitive to a BVH node using arithmetic operations.

Assuming that it is necessary to take the first L primitives among N and the range of the values is $[a,c]$, it is selected a value to be the pivot that makes a cut proportional to L/N and move all the primitives that are lower or equal to the pivot to the left and the rest to the right. As a result primitives are partitioned in two ranges $[a,b]$ and $(b,c]$. If L denotes the number of primitives contained on the first range, and L' denotes the number of primitives contained on the second range, then there exist two cases, this method guarantees that the left side is always complete:

$$L' = L \tag{5}$$

$$L > L' \tag{6}$$

In the first case, no further movements are needed, in the second case, it is necessary to update the lowest limit of our current range as the lowest value that we moved to the right and update L as $L - L'$ and N as $N - L'$. In the third case, b has to be updated as the highest value that we moved to the left and N has to be updated as L'. It is important to note that in every step we are removing at least one element from the array of primitives, this ensures that the algorithm will finish. Also, it can be observed that the complexity of the expected time will be similar to the expected cost for quickselect algorithms, which is $O(N)$ for every level.

2.3.3. Split Primitives

This step is straight forward because the split was calculated previously when the tree was pre-built to determine the number of primitives for each sibling node. Then, we apply the partition process described in the equations 2a and 2b, we get that the first primitives of the array will be assigned to the right node and the rest to the left node. After that, this rule is applied again in a recursive way until the four nodes in the quadtree are completed.

3. Results

This section exhibits the performance numbers of the Complete BVH structure rendering 3D models in a ray tracing application. The metrics taken compared the construction time and rendering frame rate against benchmarks obtained by State of the art acceleration structures on x86 architectures [WWB*14]. The tests were executed in a 2.5 GHz 8X Intel Core i7 CPU with 16Gb of DDR3 RAM compiled as 32-bit application using the ispc intel compiler.

The models used are Stanford Bunny (69,451 primitives), Crytek Sponza (279,163 primitives) and Happy Buddha (1,087,716 primitives). Figure 4 shows three-dimensional models rendered at 1024x1024 pixels. Tables 1 and 2 show that the proposed method offers lower construction times up to three times faster than binned-SAH. As a counterpart, binned-SAH construction has better ray traversal performance.

Table 1: *Construction time for diverse scenes.*

Scene	Hybrid Builder [Our Method]		Binned-SAH Builder Embree BVH4. [WWB*14]	
Ring(6 K)	2.829 ms	2.17 Mprims/s	7.840 ms	0.77 Mprims/s
Stanford Bunny(69 K)	10.850.48 ms	7.05 Mprims/s	33.083 ms	2.09 Mprims/s
Crytek Sponza(279 K)	43.657 ms	6.39 Mprims/s	179.598 ms	1.55 Mprims/s
Stanford Buddha(1 M)	174.170 ms	6.24 Mprims/s	403.810 ms	2.69 Mprims/s

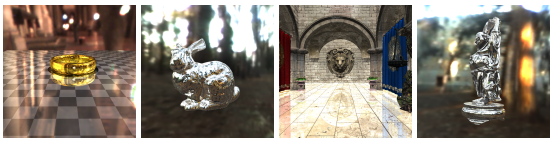

Figure 4: *Ring Model (6K), Stanford Bunny (69K), Crytek Sponza (279K) and Stanford Happy Buddha (1M) ray-traced using a Complete BVH4 acceleration structure at 1024 X 1024 pixels.*

Table 2: *Ray traversal performance for diverse scenes.*

Scene	Hybrid Builder [Our Method]	Binned-SAH [WWB*14]
Ring(6 K)	154.075 fps	156.213 fps
Stanford Bunny(69 K)	120.523 fps	136.254 fps
Crytek Sponza(279 K)	5.03 fps	20.624 fps
Stanford Buddha(1 M)	2.01 fps	6.05 fps

4. Conclusions

This work presented a novel hybrid partitioning approach to build high quality BVH structure. It also presented an efficient process using SIMD extension to build the structure. The partition method took full advantage of current x86 CPU and integrated GPU architecture. We also presented very competitive build times of this construction. Ray traversal performance of this structure needs some improvements in order to increase the current frame-rate. The Complete BVH partitioning offers fast build times for high quality structures, efficient memory storage and provides high performance traversal for rigid objects. For future work we will apply this acceleration method in combination with predictive and adaptive strategies for accelerating the rendering of dynamic scenes.

Acknowledgments

The authors would like to thank the Stanford Computer Laboratoryfor the Happy Buddha and the Bunny models, Crytek for the Sponza model. Last but not least the CONACYT for its financial support.

References

[Cor14] CORPORATION I.: Getting the most from opencl 1.2: How to increase performance by minimizing buffer copies on intel processor graphics, 2014. 3

[DFM13] DOYLE M. J., FOWLER C., MANZKE M.: A hardware unit for fast sah-optimised bvh construction. *ACM Trans. Graph. 32*, 4 (July 2013), 139:1–139:10. 1

[GMOR14] GARCÍA A., MURGUIA S., OLIVARES U., RAMOS F. F.: Fast parallel construction of stack-less complete lbvh trees with efficient bit-trail traversal for ray tracing. In *Proceedings of the 13th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry* (New York, NY, USA, 2014), VRCAI '14, ACM, pp. 151–158. 1, 2, 3

[HS86] HILLIS W. D., STEELE JR. G. L.: Data parallel algorithms. *Commun. ACM 29*, 12 (1986), 1170–1183. 3

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast bvh construction on gpus. In *In Proceedings of the EUROGRAPHICS âĂŹ09* (2009). 1

[NPP*11] NAH J.-H., PARK J.-S., PARK C., KIM J.-W., JUNG Y.-H., PARK W.-C., HAN T.-D.: T&ıengine: Traversal and intersection engine for hardware accelerated ray tracing. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (New York, NY, USA, 2011), SA '11, ACM, pp. 160:1–160:10.

[PFL*13] PARKER S. G., FRIEDRICH H., LUEBKE D., MORLEY K., BIGLER J., HOBEROCK J., MCALLISTER D., ROBISON A., DIETRICH A., HUMPHREYS G., MCGUIRE M., STICH M.: Gpu ray tracing. *Commun. ACM 56*, 5 (May 2013), 93–101. 1

[SSK07] SHEVTSOV M., SOUPIKOV A., KAPUSTIN A.: Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Comput. Graph. Forum 26*, 3 (2007), 395–404. 1

[Wal07] WALD I.: On fast construction of sah-based bounding volume hierarchies. In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing* (Washington, DC, USA, 2007), RT '07, IEEE Computer Society, pp. 33–40. 2

[Wal12] WALD I.: Fast construction of sah bvhs on the intel many integrated core (mic) architecture. *IEEE Trans. Vis. Comput. Graph.* (2012), 47–57. 1

[WIK*06] WALD I., IZE T., KENSLER A., KNOLL A., PARKER S. G.: Ray tracing animated scenes using coherent grid traversal. *ACM Trans. Graph. 25*, 3 (July 2006), 485–493. 1

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph. 33*, 4 (July 2014), 143:1–143:8. 1, 3, 4

[ZHWG08] ZHOU K., HOU Q., WANG R., GUO B.: Real-time kd-tree construction on graphics hardware. *ACM Trans. Graph. 27*, 5 (Dec. 2008), 126:1–126:11. 1