Agglomerative Building and Traversal of an Object Aligned Bounding Volume Heirarchy

Dan Aston 300406690

October 2018

Abstract

This report comprises an investigation of the topic of acceleration structures, specifically bounding volume heirarchies for real time raytracing, and covers an implementation of a provided example using a simple bottom up agglomerative technique based on object aligned bounding boxes (OABBs). The design challenge is discussed in two aspects: designing a structure and algorithm for an efficient ray traversal, and designing an efficient algorithm for building this structure. For both aspects, precedents are discussed, solutions are proposed, and our implementation is summarised. We will compare the performance of this technique to prior work, and discuss how they perform differently. We then propose directions for further research, such as alternative abstraction hulls, multi trees, and different heuristics to assist the build process. We will propose and discuss a parallelised version of the agglomerative build process.

Design Discussion

Introduction

Raytracing is a rendering technique that is agnostic to screen space, and the view frustrum. A ray may be cast in any direction, originating from any point. In the screenspace world, one knows an eye ray intersects a surface at depth t by sampler the z-buffer only after all primitives have been rasterised. This is convenient if the primitives had to be rasterised anyway. What this tells us is that the length of the ray depends on some organisation of the scene that depends on the ray's origin. This means that for each ray origin, there is a different set of salient scene features to be processed in order to find the depth of its closest intersection. The direct analogy to rasterisation in the context of ray tracing is checking the intersection against all primitives. For a single ray, the scene cost is O(n). With rays requiring multiple samples, and sampling algorithms budgeted to tens of steps, this is prohibitive for a scene with an expressive level of detail. The performance constraints should be fine, given that a single ray will only have a handful of intersections with the scene.

Acceleration Structures

What is needed is a means by which to organise a scene in a way that abstracts non salient primitives, based on the origin and orientation of the ray. A range of structures have been conceived for doing this. But the details of the rules which define them make them function differently.

Acceleration structures can be built from the top-down using abstractions derived from pre existing high level knowledge of the scene. The subdivision methods implemented by Karras, T.[2012] and Timo Aila[2013], and Popov, S. et al [2009] work are examples of this. Top-down approaches seem to be more common in the literature.

Alternatively, the scene may be considered a disordered "triangle soup" and the higher level nodes are all new. This is an "agglomerative", or bottom-up approach. Examples are given by Gu Yan et al. [2013].

This is a useful dichotomy because both perspectives provide different intuitions for parallelisation.

Reasoning about Topology and Traversal

A Heirarchy is something with a simple end and a complex bunch of other ends. An intersection is one of these complex other ends. If there is a valid one, we want to find it as soon as possible. We want the tree to be organised in a way that allows us to do this. We must decide, what kind of tree will do that for our scene? What are the tradeoffs between the different topologies?

On the one hand, we're used to thinking that shorter trees are better trees. If multiple cores are working on one ray, a multi tree would facilitate the dispatch of more threads at earlier steps. Also, a ray with many potential intersections will tend to have more vertical traversal up and down the tree as it climbs in and out of near misses. A binary tree is conceptually the simplest, but it is the tallest non unary tree. If we have an intersection, it may take more steps to get there. A multi tree may facilitate a shorter string of decisions along the traversal.

However, while using a fragment shader, we assume there are more fragments than there are cores. On a multi tree, the net computational saving it affords depends on how flow control is implemented at each visitation. If assessing siblings neccessitates branched logic, the node becomes a tree within a tree, and can easily become more costly to evaluate than

the comparisons it abstracts. Generally speaking, when all nodes are stored in the same buffer, the cost of choosing a path outweighs the cost of navigating it. Navigating a wide tree involves more possibilities, and ray tracing is about eliminating possibilities. Potentially, precomputed relationships between siblings may inform faster elimination, but this is something to investigate later.

The depth of the tree is only as relevant as it is correlated with the amount of intersections it necessitates. We just want as few checks as possible, with minimal branching. For the investigative purposes of this project, we propose an acceleration structure with the topology of a simple binary tree, with a single branch to distinguish leaf nodes and inner nodes.

Abstraction Hulls

We introduce the concept of an "abstraction hull". An abstraction hull (or in this context just, "hull") is something that is easier to compute an intersection with than the set of its contents.

Surface Area and Emptiness

Hulls only save computation when they are missed, and when testing them is cheaper than testing their contents. The more "emptiness" there is within a given node, the more likely it is that it will escape without more checks. A good BVH maximises ray thoroughfare. While checking within an intersected containing hull hA, the likelyhood that a child hull hC will intersect the ray is proportional to the area of the child's profile as projected onto the image plane divided by the area of the profile of the parent. A BVH that contains all scene geometry can be simpler but not emptier than the scene itself. A good BVH simplifies the scene while filling it out as little as possible. The literature on BVHs makes frequent reference to the surface area heuristic (SAH) (see Wald. I(2007)). The idea is that the best possible hull to enclose a set is the one with minimum surface area. It is worth noting that this is not the same concept as maximising the likelyhood of early escape, but it is a useful concept that balances the requirements of simple abstractions with conservation of emptiness.

Slab, k-DOPs, Spheres

The presented implementation uses slab, which is a well known technique for defining bounding boxes. It can also be generalised to arbitrary convex hulls. If a shape is convex, each intersecting ray will have one entry and one exit point. Entry points will have normals facing towards the ray's origin, and vice versa. Therefore, faces may be divided into entry and exit planes depending on the dot product between their normals and the ray direction. If a ray misses the hull, at least one pair of these planes will the "flipped", and the nearest exit point will be closer than the furtherest entry. Otherwise, the furtherest entry point is a close valid hit.

AABB drawbacks

There are two things AABBs struggle with: large polygons, and diagonal surfaces. Consider the case of a corrugated iron roof. There are many long triangles along each wave of the iron, and each pushes out the corners of their bounding boxes far beyond the plane of the roof's panel. Even if they are subdivided, the resulting parent nodes will take up a needless amount of negative space in the direction normal to the roof. Rays that glance along the surface of the roof tangentially will intersect bounding boxes as if they were participating media, which is far costlier than testing against an aligned abstraction.

Traversal pseudocode

The traversal algorithm is very simple. Most of the work goes into intersection checks against the bounding volumes with slabBox(), and the primitive triangles with testTriIndex(); The loop walks through the tree testing intersections. If a box intersects closer than the best hit, its bigger child is visited, and its small child is pushed to the check stack. In this way, the algorithm improves the chance of the closer hits being tested first. When close hits are found early, there are more abstractions may be eliminated based on distance. The effects of this are quite significant. If you turn off the primitive intersections and render the whole bounding structure you will notice a drop in framerate. There are two phases. No intersection tests are made until visiting the node that contains the origin, although siblings are pushed to the stack on the way.

```
bestHit = ESCAPE;
// PUSH TO ORIGIN PHASE
while( boxcontains (visit,r.o) ){
  //triangle case
  if (topo[visit-1] == -1)  {break;}
  //parent node case
  visitStack[stackCounter] = topo[visit*2];
  stackCounter ++;
  visit = topo[visit*2-1];
}
```

Once the stack has been preloaded with the siblings along the path to the containing node, the node itself is added to the stack so that its children can be tested first. Then, the array tries to escape the stack through intersection tests.

```
// INTERSECTION PHASE
for (int  STEP = 0; STEP < MAX_TRAVERSAL_STEPS;
     STEP++){
  if (stackCounter<=0) break;
  visit = visitStack[-1+stackCounter--];

  slabBox(visit,r);
```

```
  if (!theBvhHit.hits) continue;
  if (theBvhHit.t > leafRay.bestHit.t){
    continue;
  }

  leafRay.boxAccum += 0.03; // Visualise the BVH

  // Primitive case
  if (topo[visit*2-1] < 0){
  testTriIndex(r,topo[(visit)*2]);
    if( theTri.hits
        && theTri.t < leafRay.bestHit.t //){
        && theTri.t > 0){
      leafRay.bestHit.hits=true;
      leafRay.bestHit = theTri;
    }
  continue;
  }
   int small = topo[visit * 2 - 1];
   int large = topo[visit * 2 ];//+ 1];
  visitStack[stackCounter++] = small;
  visitStack[stackCounter++] = large;
 }
}
// Traversal escaped the tree.
//Got the best hit.
```

Memory Usage

The current implementation requires significant state for traversal. The biggest part of it is the visit stack. Each stack destination uses an unsigned int, and there are many sites to visit in a complicated scene. Afta et. al propose a bitstack based implementation that can also be applied to multi-trees.

Building

Recall that, there are top down techniques, and there are bottom up techniques. Our implementation is a bottom up technique. In the planning of this project, we identified the child hulls as dependencies for that of the parent. No hull can be finalised without its children's hulls also being finalised. Therefore, we build the heirarchy from the bottom up.

Pre Sorting

The tradeoff with bottom up approaches is that you start off with each primitive having every other primitive as possible hull siblings. Efficient building neccessitates sorting the geometry into a structure to facilitate a faster nearest neighbor search, which is a whole different preprocessing step.

Morton Coding

A fast way to do this is to put all primitives into a 3D space filling Z-order curve. The are well documented techniques for doing this with bitwise operators (see Lauterbach et al [2009] for the Linear BVH, or 'LBVH'). The positions of primitives are quantised to an integer representation, and the bits of each dimensional component are simply interleaved to write the position in the curve. When the geometry is sorted, primitives can search their nearby 3D neighbourhood by checking a range around their index in the sorted morton curve.

Summary of Implementation

The app loads arbitrary wavefront .obj meshes and paints their fragments as normals, while counting the bvh intersection cost. The user may fly around with w-a-s-d, and look and pan with the mouse. The user can toggle the brute force triangle tracing for the purposes of comparison, and they can decide how many triangles they want to brute force with a slider. There is another slider that decrements the index of the bvh's head node, so that the user can visualise the bvh's subtrees and inspect how well they fit the geometry, and to support recognition of emergent phenomena in the tree's construction.

The implemented BVH is an oriented bounding box volume heirarchy (OBBVH), built using an agglomerative technique. Building happens on the CPU, and the OBBVH is sent to the gpu into a set of shader storage buffer objects. The current builder implementation is very brute force. It uses a while loop with no nearest neighbor search. The build has three stages:

- Primitive node creation
- Topology creation loop
- Topology string rendering

First, a vector for leaf nodes is filled with a new node containing each primitive in the scene. In contrast to the subdivision of the axis aligned techniques, here each primitive occupies only one leaf. Then the topology is built. The loop has two phases:

- Picking phase
- Matching phase

In the first, each loose node checks every other loose node for the best potential match. The best match is one with whom the resulting bounding box containing both nodes would be the smallest. Nodes who propose hulls hold pointers to the nodes they would like to partner with in a tiny home and save copies of the specifications of the hull to share. Proposees recieve proposals during their turn in the loop and update their match pointer and copy of the hull plan to the best proposition. Nodes make and recieve proposals in the same pass. Once nodes are satisfied that they have chosen the best candidate, a matching pass iteratively goes through the list, constructing new nodes from the indices of

mutually matching pairs, and the bounding hull that they agree on. Matching nodes are marked for two reasons. It means that the loop will not create multiple copies of the same parent node upon subsequent visits to matched nodes. It also means prospective testers can skip them. The number of free primitives is known at the beginning of the build. This count is decremented upon each union, and the loop ends when it is equal to 1.

Results

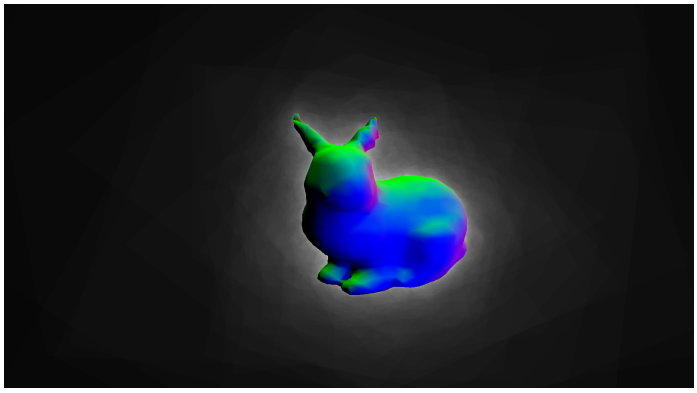

Figure 1: The Obligatory Hero Image

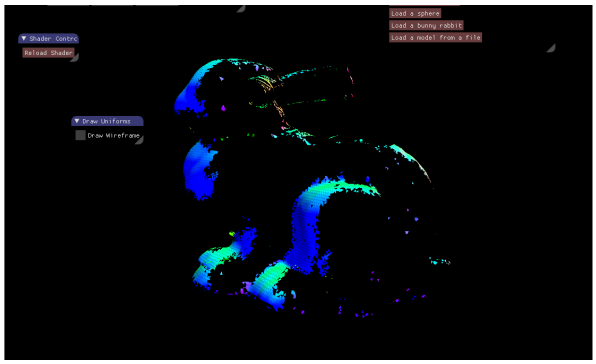

Figure 2: Brute Force Triangle testing: 10000 triangles @ < 1 fps

Test Hardware

These results were recorded on this system: > i5 4690k > Gigabyte z-97n Gaming 5 > 16GB DDR3 PC3-10600 > Zotac GTX 660 3GB

All hardware running standard voltages and clock speeds.

Building

The single threaded brute force build is, as expected, comically slow. It builds the 80 triangle ball mesh in 2 seconds. To load a 4000 triangle mesh, it takes several minutes.

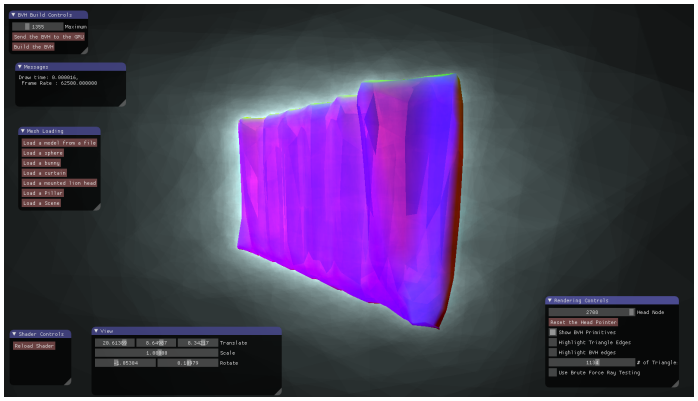

Figure 3: OABs fail to align to integrated orientation, the span diverges

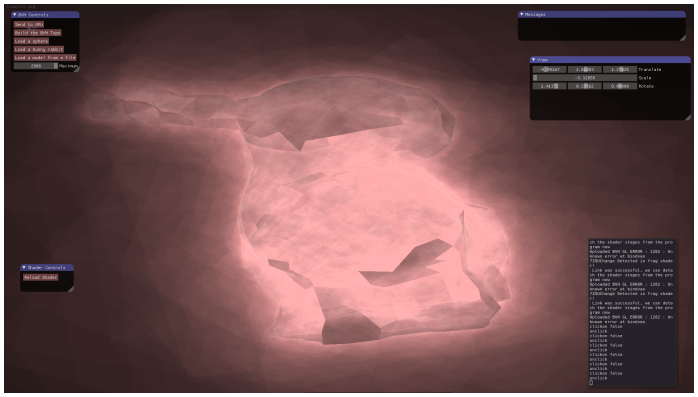

Figure 4: BVH Boxes Only - No early termination, therefore the cost is volumetric.

BVH Quality

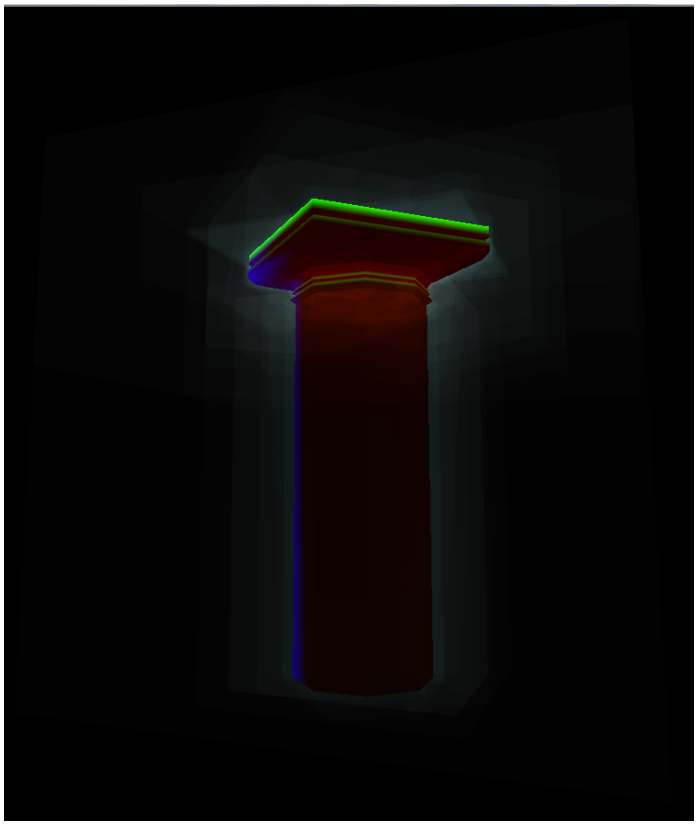

Figure 5: The pillar is a great test case, with a lot of long and co planar surfaces. The red highlights are emitted by the occluded intersections the tracer tests while 'climbing out' of the stack after finding the closest hit.

Enough encapsulation is happening to facilitate a consistent order of magnitude performance benefit over the n^2 brute force case with no culling.
It is obvious that improvements can be made to the efficiency of OBB construction. For one thing, the builder needs to find better bases for inner nodes. Currently all parent nodes are in the tree are aligned to the basis of some particular leaf node, instead of the overall basis of the containing set. Where unused far corners push out the centres of the faces of their parents, where their parents bases' are relatively diagonal.
This eats up a lot of negative space. It also stops the higher level nodes from converging toward overall alignments, which is evident for the bvh of the curtain. Thankfully, finding a better basis for each parent will be trivial, and there are plenty of ways to cull corners.

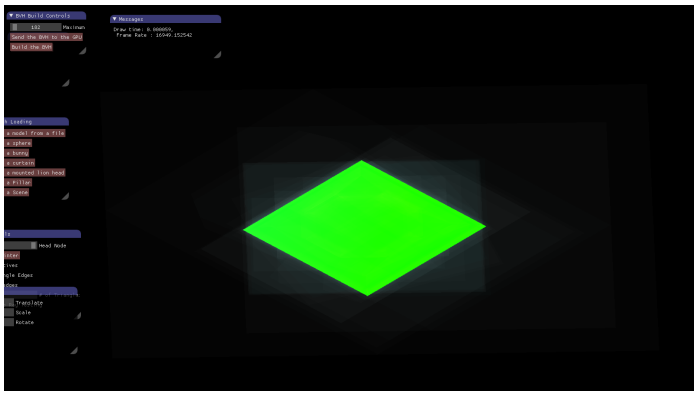

Figure 6: Here we can see how the hulls are quickly diverging. Each new orientation is not sensitive to the orientation of best fit. Snapping to one basis or the other sets up a divergent process where the cost of snapping between bases gets bigger as they are larger every time it happens.

Conclusion

We have discussed the design challenge of acceleration structures. We have discussed the conservation of emptiness, and shown how there are a range of convex abstraction hulls to use for this purpose, which have different costs and use cases. We have discussed issues of diagonal aligment and and large polygons, and the structure of the tree. We have implemented a simple object aligned bounding volume heirarchy with a single threaded build process. The demo visualises the span of a rudimentary OABVH, and allows the user to interactively inspect its structure. and demonstrates how even a sub optimal bvh still performs orders of magnitude better than the naive algorithm. We have used this to observe the limitations of a simple bvh, we have identified how they manifest, and we are now in a position to introduce alternative abstractions with a sense of purpose.

References

Áfra, Attila T., and László Szirmay-Kalos. 2014. "Stackless Multi-BVH Traversal for CPU, MIC and GPU Ray Tracing." *Comput. Graph. Forum* 33 (1): 129–40. https://doi.org/10.1111/cgf.12259.

An Introduction to Ray Tracing. n.d.a. Accessed November 1, 2018. https://books.google.com/books/about/An_Introduction_to_Ray_Tracing.html?id=YPblYyLqBM4C.

An Introduction to Ray Tracing. n.d.b. Accessed November 1, 2018. https://books.google.com/books/about/An_Introduction_to_Ray_Tracing.html?id=YPblYyLqBM4C.

Article. n.d.a.

Article. n.d.b.

Chajdas, Matthäus G., and Rüdiger Westermann. 2014. *Quantitative Analysis of Voxel Raytracing Acceleration Structures*. The Eurographics Association. https://doi.org/http://dx.doi.org/10.2312/pgs.20141257.

Dinas, JosÃ M., Simena AND BaÃ±Ã. 2015. "A Literature Review of Bounding Volumes Hierarchy Focused on Collision Detection." *IngenierÃa Y Competitividad* 17 (June): 49–62. http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0123-30332015000100005&nrm=iso.

"F11b32e45393a356d5eae40c8eab7b8e1152.Pdf." n.d. Accessed November 1, 2018. https://pdfs.semanticscholar.org/5979/f11b32e45393a356d5eae40c8eab7b8e1152.pdf.

Ganestam, Per, and Michael Doggett. 2016a. "SAH Guided Spatial Split Partitioning for Fast BVH Construction." *Computer Graphics Forum* 35 (2): 285–93. https://doi.org/10.1111/cgf.12831.

———. 2016b. "SAH Guided Spatial Split Partitioning for Fast BVH Construction." *Comput. Graph. Forum* 35 (2): 285–93. https://doi.org/10.1111/cgf.12831.

García, Arturo, Sergio Murguia, Ulises Olivares, and Félix F. Ramos. 2014. "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 - VRCAI '14*, 151–58. Shenzhen, China: ACM Press. https://doi.org/10.1145/2670473.2670488.

Glassner, Andrew S. 1989. *An Introduction to Ray Tracing*. Morgan Kaufmann.

"GPU-Based Parallel Stackless BVH Traversal for Animated Di Stributed Ray Tracing - Semantic Scholar." n.d. Accessed November 1, 2018. https://www.semanticscholar.org/paper/GPU-Based-Parallel-Stackless-BVH-Traversal-for-Di-UMBC/a143df2abbdc1b562da4e4a268f736b167865f0c.

Gu, Yan, Yong Jun He, Kayvon Fatahalian, and Guy E. Blelloch. 2013. "Efficient BVH Construction via Approximate Agglomerative Clustering." In *High Performance Graphics*. https://doi.org/10.1145/2492045.2492054.

Hapala, Michal, Tomáš Davidovič, Ingo Wald, Vlastimil Havran, and Philipp Slusallek. 2013a. "Efficient Stack-Less BVH Traversal for Ray Tracing." In *Proceedings of the 27th Spring Conference on Computer Graphics*, 7–12. SCCG '11. New York, NY, USA: ACM. https://doi.org/10.1145/2461217.2461219.

———. 2013b. "Efficient Stack-Less BVH Traversal for Ray Tracing." In *Proceedings of the 27th Spring Conference on Computer Graphics*, 7–12. SCCG '11. New York, NY, USA: ACM. https://doi.org/10.1145/2461217.2461219.

Ize, Thiago. 2009. "Efficient Acceleration Structures for Ray Tracing Static and Dynamic Scenes." PhD Thesis, Salt Lake City, UT, USA: University of Utah.

Ize, Thiago, Ingo Wald, and Steven G. Parker. 2007. "Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures." In *Proceedings of the 7th Eurographics Conference on Parallel Graphics and Visualization*, 101–8. EGPGV '07. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association. https://doi.org/10.2312/EGPGV/EGPGV07/101-108.

Karras, Tero. 2012. *Maximizing Parallelism in the Construction of BVHs, Octrees, and K-d Trees*. The Eurographics Association. https://doi.org/http://dx.doi.org/10.2312/EGGH/HPG12/033-037.

Karras, Tero, and Timo Aila. 2013. "Fast Parallel Construction of High-Quality Bounding Volume Hierarchies." In *Proceedings of the 5th High-Performance Graphics Conference*, 89–99. HPG '13. New York, NY, USA: ACM. https://doi.org/10.1145/2492045.2492055.

"Karras2012hpg_paper.Pdf." n.d. Accessed November 1, 2018. https://research.nvidia.com/sites/default/files/pubs/2012-06_Maximizing-Parallelism-in/karras2012hpg_paper.pdf.

Lauterbach, C., M. Garland, S. Sengupta, D. Luebke, and D. Manocha. 2009a. "Fast BVH Construction on GPUs." https://diglib.eg.org:443/xmlui/handle/10.2312/CGF.v28i2pp375-384.

———. 2009b. "Fast BVH Construction on GPUs." https://diglib.eg.org:443/xmlui/handle/10.2312/CGF.v28i2pp375-384.

Legrand, Hélène, and Tamy Boubekeur. 2015. "Morton Integrals for High Speed Geometry Simplification." In *Proceedings of the 7th Conference on High-Performance Graphics*, 105–12. HPG '15. New York, NY, USA: ACM. https://doi.org/10.1145/2790060.2790071.

Nguyen, An, Leonidas J. Guibas, Jean-Claude Latombe, Tim Roughgarden, Pankaj Agawal, Li Zhang, Jie Gao, et al. 2006a. "Implicit Bounding Volumes and Bounding Volume Hierarchies." In.

———. 2006b. "Implicit Bounding Volumes and Bounding Volume Hierarchies." In.

Novák, Jan, and Carsten Dachsbacher. 2012. "Rasterized Bounding Volume Hierarchies." *Comput. Graph. Forum* 31 (2pt2): 403–12. https://doi.org/10.1111/j.1467-8659.2012.03019.x.

"PLOCTree." n.d. Accessed November 1, 2018. https://dl-acm-org.helicon.vuw.ac.nz/citation.cfm?id=3233309.

Popov, Stefan, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek. 2009. "Object Partitioning Considered Harmful: Space Subdivision for BVHs." In *Proceedings of the Conference on High Performance Graphics 2009*, 15–22. HPG '09. New York, NY, USA: ACM. https://doi.org/10.1145/1572769.1572772.

"Projects." n.d. Accessed November 1, 2018. http://www.bwfischer.com/oculusRayTracer.html.

Report. n.d.

Stibora, Bc Radek. n.d. "Building of SBVH on Graphical Hardware."

Stich, Martin, Heiko Friedrich, and Andreas Dietrich. 2009. "Spatial Splits in Bounding Volume Hierarchies." In *Proceedings of the Conference on High Performance Graphics 2009*, 7–13. HPG '09. New York, NY, USA: ACM. https://doi.org/10.1145/1572769.1572771.

UMBC, Charles Lohr. 2009. "GPU-Based Parallel Stackless BVH Traversal for Animated Di Stributed Ray Tracing." In.

Vinkler, Marek, Jiri Bittner, and Vlastimil Havran. 2017. "Extended Morton Codes for High Performance Bounding Volume Hierarchy Construction." In *Proceedings of High Performance Graphics*, 9:1–9:8. HPG '17. New York, NY, USA: ACM. https://doi.org/10.1145/3105762.3105782.

Wald, Ingo. 2007. "On Fast Construction of SAH-Based Bounding Volume Hierarchies." In *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, 33–40. RT '07. Washington, DC, USA: IEEE Computer Society. https://doi.org/10.1109/RT.2007.4342588.

Wald, Ingo, Solomon Boulos, and Peter Shirley. 2007. "Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies." *ACM Trans. Graph.* 26 (1). https://doi.org/10.1145/1189762.1206075.

Ylitie, Henri, Tero Karras, and Samuli Laine. 2017. "Efficient Incoherent Ray Traversal on GPUs Through Compressed Wide BVHs." In *Proceedings of High Performance Graphics*, 4:1–4:13. HPG '17. New York, NY, USA: ACM. https://doi.org/10.1145/3105762.3105773.