ME-BVH: Memory Efficient Bounding Volume Hierarchies

Evan Shellshear, Yi Li, and Johan S. Carlson

The Geometry and Motion Planning Department

Fraunhofer Chalmers Centre

SE-412 88 Gothenburg, Sweden

E-mail address of the corresponding author: evan.shellshear@fcc.chalmers.se

Abstract

Collision detection and distance computation algorithms often form the bottlenecks of many digital human modelling simulations in industrial processes. When designing vehicle assembly lines or cobot assembly cells it is essential to be able to accurately simulate collision free interactions both for efficient and safe operations. Hence, any attempt to improve such algorithms can have a broad and significant impact. Most of the focus is typically on speeding up the queries, however, with models becoming larger as scenarios become more realistic and simulations include more elements such as musculoskeletal models and 3D human body modelling, other parts of the proximity query performance are becoming important such as the management of memory. In this paper, we demonstrate a new technique called ME-BVH (Memory Efficient Bounding Volume Hierarchies) to improve memory usage for proximity queries with bounding volume hierarchies. The approach utilizes a simple and effective way of grouping primitives together at the leaf level and building the bounding volume hierarchy top down to the grouped primitive leaves. The paper then shows ways of efficiently carrying out primitive and bounding volume queries to offset the greater number of potential queries. In addition, the modifications taken are simple enough to be easily applied to most bounding volume hierarchies. By using these approaches, we demonstrate on a number of real-life assembly scenarios with millions of primitives that, compared to existing approaches, our proposed method is able to save up to half of the memory used and can reduce the build times at little cost to the query performance. In addition, the methods developed here are compatible with all BVH types and queries used in ergonomic simulations, unlike many other approaches. The developed algorithms present advantages for proximity queries for deformable meshes used in digital human modelling by reducing the time it takes to build a bounding volume hierarchy which often must be rebuilt or updated many times during simulations due to mesh deformations.

Keywords: Bounding Volume Hierarchies, Collision Detection, Proximity Queries, Automated Assembly, Simulation

Introduction

Collision queries (CQ) and distance queries (DQ), proximity queries, are key processes used in many applications (Shellshear & Bohlin, 2014) (Eriksson & Shellshear, 2014). The field of proximity queries is a well-researched area of study and there exist many effective algorithms for proximity queries in a variety of scenarios. However, given these queries are a key bottleneck in many areas, such as motion planning for virtual manikin assembly (Li, Delfs, Mårdberg, Bohlin, & Carlson, 2018), any improvements to the algorithms underpinning these techniques can benefit many industries beyond digital human modelling and reveal new approaches to solving well-known problems.

In this paper, we analyze algorithmic improvements to traditional Bounding Volume Hierarchies (BVHs) to quickly perform both collision queries and distance queries between a moving rigid body and static obstacles in digital human modelling scenarios such as vehicle assembly. The focus of our algorithmic improvements is on removing bounding volumes around primitives (triangles, lines, or points (Tafuri, Shellshear, Bohlin, & Carlson, 2012)) and then adapting the build and query phases to correctly handle the missing bounding volumes. By doing so, we are able to theoretically remove up to half of all bounding volumes required in a BVH, leading to faster build times and less memory requirements. To ensure the proximity queries are not impaired, we develop specific BV-primitive distance queries, where BV stands for a Bounding Volume.

Related Work

The foundational technology which we use to solve the problem of interest is the well-known Bounding Volume Hierarchy solution (Hubbard, 1993), with Rectangular Swept Spheres (RSSs) (Larsen, Gottschalk, Lin, & Manocha, 2000) as our chosen bounding volumes. RSSs are constructed by taking the Minkowski sum of some core primitive shape and a sphere. To create the set of bounding volumes that can be defined by RSSs one can have a point, line, or rectangle as the core primitive (resulting in a sphere, lozenge, or RSS), the first two cases being considered as degenerate rectangles (i.e., with one or two side lengths equal to zero). By enclosing an entire triangle mesh in an RSS one is then able to build a BVH top-down using well-known algorithms. The superiority of computing proximity queries with RSSs over other types of bounding volumes has been demonstrated in papers such as (Larsen, Gottschalk, Lin, & Manocha, 2000), (Lauterbach, Mo, & Manocha, 2010), and (Pan, Chitta, & Manocha, 2012).

However, as shown in (Terdiman, 2001), although RSS BVHs produce the best performance for distance queries, they also consume the greatest amount of memory by a factor of four over BVHs built on simpler shapes such as spheres. Hence, for large scenes with millions of triangles, any optimization of RSS BVHs which doesn't sacrifice (or minimally impacts) performance is a welcome advance.

Since the creation of BVHs, there have been numerous attempts at finding ways to reduce the memory footprint via quantization (Terdiman, 2001), simplifying the representation of shapes (Zachmann, 1995), levels of detail (Yoon, Salomon, Lin, & Manocha, 2004), lazy building approaches (Krishnan, Pattekar, Lin, & Manocha, 1997) and many more. Although many of these methods produce some advantages, they are either not general enough to apply to all types of BVHs, not applicable to the intended applications here (exact calculations with fast query times) or are only relevant for collision queries and not distance queries. In the case of lazy build strategies (i.e., incrementally updating especially for deformable meshes), these can result in the update phases causing proximity queries to be up to 10x slower (Shellshear, Bitar, & Assarsson, 2013).

There is a known approach that is not only applicable to all BVHs and query types but also significantly reduces memory usage by removing unnecessary BVHs and does not result in slowing down the proximity queries by orders of magnitude. This approach either increases the number of primitive shapes (e.g., triangles) in each leaf or simply removes BVs around primitives, to a similar effect.

In the first case of increasing the number of primitive shapes in a leaf, this has been carried out and applied in many scenarios that use BVHs (Dammertz, Hanika, & Keller, 2008) (Ericson, 2004). The results, however, typically require using low-level SIMD (single instruction multiple data streams) instructions such as SSE (streaming SIMD extensions) or AVX (advanced vector extensions) (Shellshear & Ytterlid, Fast distance queries for triangles, lines, and points using SSE instructions, 2014) to speed up the slowdowns caused by unnecessary primitive tests.

In this paper we avoid this approach due to the additional complexity of coding SIMD or AVX instructions and also due to the fact that compilers are typically able to carry out SIMD and AVX optimizations better than manually coded instructions (Shellshear & Ytterlid, Fast distance queries for triangles, lines, and points using SSE instructions, 2014). Hence, existing approaches that are compiled in a modern compiler will likely be benefiting from this already.

Although increasing the number of primitive shapes in a leaf may lead to individual primitives without a specific BV surrounding them, it is not the same as the second approach mentioned above (i.e., removing BVs around primitives). The second approach leads to just having to test a single primitive when a leaf is reached, hence preserving the benefit of traditional BVHs. However, it requires developing BV-primitive collision and distance queries to be able to work which are not always a part of many BVH packages.

This paper focuses on the second approach of simply removing the BV around the final primitive to potentially reduce the memory footprint by up to fifty percent. Surprisingly there are very few papers which have addressed this and the one relevant paper (Terdiman, 2001), ended up carrying out a comparison which does not lead to any obvious conclusions (due to comparing two different types of BVHs) and also does not test the improvements for distance queries which can have fundamentally different performance profiles.

The ME-BVH Algorithm

In this section we introduce the ME-BVH (Memory Efficient Bounding Volume Hierarchies) algorithm via discussing the changes to traditional BVHs. When building a BVH for a triangle mesh, the typical process is top-down, whereby a BV is built around the entire triangle mesh, which is then split recursively into smaller disjoint subsets around which new BVs are built until the algorithm reaches a single triangle around which a final BV is built, and the process stops.

If we begin with a triangle mesh with N triangles, this leads to $2N - 1$ BVs. When dealing with millions of triangles, this then requires memory to store twice as many BVs as triangles and for certain types of BVs (such as RSSs), the storage required per BV is greater than the storage requirements per triangle. If we remove unnecessary BVs from the BVH leaves, we immediately avoid building N BVs. Additionally, by not having to build these BVs, the construction phase would be faster. However, doing so requires changes to existing query routines to handle the leaf cases which no longer have BVs around them, which typically means creating a BV-triangle collision or distance routine. As mentioned in (Terdiman, 2001), such BV-triangle routines exist for simple shapes like Box-triangle, however, in our case of an RSS BVH we were required to implement a new routine.

Algorithm 1: The traditional BVH build process

```
1 RecurseBuild ($T$)
  Input: $T$ a non-empty set of triangles
  Output: BVH
2 Build a BV around triangle(s) $T$
3 if $|T| > 1$ then
4     Split triangles into two sets $T_1$ and $T_2$
5     RecurseBuild($T_1$), RecurseBuild($T_2$)
6 else
7     return
```

Build Phase in ME-BVH

In many current top-down BVH construction algorithms, at each stage, criteria are used to determine whether a further split is required and when a single triangle is reached the criteria return false (e.g., the traditional BVH build process detailed in Algorithm 1. In our case the simple adaptation was to change the criteria to false as soon as we reached two or less triangles (i.e., in line 3 in Algorithm 1, we changed the condition to $|T| > 2$).

Query Routine in ME-BVH

Compared to standard BVHs, the ME-BVH contains leaves without BVs. Consequently, the standard traversal routine has to be updated in order to handle the following four cases: the usual RSS-RSS traversal, the RSS-triangle traversal, the triangle-RSS traversal and the usual triangle-triangle test.

The first part of the traversal where distances or collisions are computed between RSSs can use existing approaches (Larsen, Gottschalk, Lin, & Manocha, 2000). For the RSS-triangle and triangle-RSS cases we had to come up with a new algorithm. The main part of this was to create a new RSS-triangle distance query as well as logic during the query traversal for the ability to handle when a BV was to be tested against a leaf.

Due to the way the build phase is carried out, the algorithm now needs to handle “leaves” which may have two triangles in them. If we reach a leaf with two triangles, we traverse one triangle against the other BVH first, then traverse the second triangle. This effectively achieves the goal of having a single triangle tested against another hierarchy unlike approaches which simply have multiple triangles in a leaf with a BV around them. In the case that we reach two leaves with multiple triangles, we test all pairs of triangles against each other (either two or four pairs).

The creation of an RSS-triangle distance query was straightforward and used the same approach for the Triangle-Triangle distance query in the PQP package (Larsen, Gottschalk, Lin, & Manocha, 2000). This algorithm works by first finding the minimum distance between each pair of triangle edges (9 total combinations), then it examines the off-edge vertices and checks if either are within the slab between the two planes defined by the line connecting the closest points on the current triangle edges being tested. In the triangle case, there is only one off vertex to check for each triangle, however, with an RSS, we need to test both off-edge vertices, see Figure 1, which demonstrates the scenario where we need to test if the off-edge vertices A, D and G lie in the slab defined by the lines L_1 and L_2.

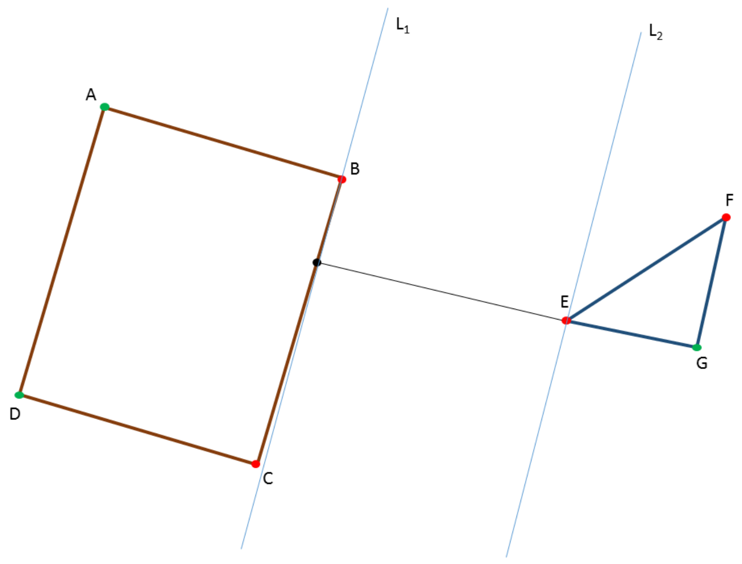

Figure 1. Testing off-edge vertices A and D and G between a rectangle and a triangle when testing edges BC and EF.

If the minimum distance is not found due to vertices being contained in the slab between the closest points on a given edge-edge test, then we need to check four other cases:

1. one of the closest points is a vertex of one triangle and the other closest point is on the face of the other RSS (or vice versa).
2. the triangle and RSS intersect.
3. an edge of one triangle is parallel to the face of the other RSS or vice versa.
4. one or both triangles and/or RSS are degenerate.

The adaptations were straightforward to the existing Triangle-Triangle distance query in the PQP package (Larsen, Gottschalk, Lin, & Manocha, 2000) and led to good performance. We also tested a simpler version of the above RSS-Triangle routine by splitting a rectangle into two triangles and then simply using two Triangle-Triangle distance routines to compute the shortest distance, however, this turned out to be much slower than the above more tailored version, even with optimizations to avoid one of the Triangle-Triangle routines in obvious situations.

Results

To test the performance of the ME-BVH, we developed a number of experiments based on real life, virtual manikin assembly scenarios with millions of triangles. We not only tested collision and distance query timings but also tested the effect of the ME-BVH on the performance of IPS Path Planner (Li, Shellshear, Bohlin, & Carlson, 2020). This planner is widely by industry to plan collision-free disassembly paths and it relies on the underlying collision and distance computation module (IPS CDC) instead of the PQP package (Larsen, Gottschalk, Lin, & Manocha, 2000) to perform collision queries and distance queries. Both IPS Path Planner and IPS CDC have been presented in a number of papers (Li, Shellshear, Bohlin, & Carlson, 2020) (Spensieri, Carlson, Bohlin, Kressin, & Shi, 2016) (Forsberg, Mårdberg, Roll, Rilby, & Carlson, 2018) and we refer the reader to those papers for more details.

Currently, IPS CDC uses the traditional approach to construct BVHs. To demonstrate the effectiveness of our algorithm (i.e., the ME-BVH), we implemented it on top of IPS CDC in lieu of the traditional approach. The source code was written in C++ and compiled in Microsoft Visual Studio Professional 2019 (Version 16.7.7) on Windows 10 Pro 64 bit. The computer that we used contained an Intel® Core™ i7-7700K CPU @ 4.20 GHz (4 cores, 8 threads), a NVIDIA GeForce GTX 970 (4 GB) GPU, and 32 GB RAM.

To compare the memory required by BVHs constructed with the traditional approach and the ME-BVH, we collected five different test cases (TC) from the manufacturing industries:

1. TC Air Filter Assembly, the models are proprietary, so we are unable to show an image. This test case involves a virtual manikin having to move an air filter assembly out of a motorcycle frame where the initial assembled state of the frame and air filter assembly are very close to each other so that the movements are very constrained. The total number of triangles in this test case is more than 1.5 million.
2. TC Cylinder Head, as above the models are also proprietary so we are unable to show an image. This test case involves moving the cylinder head cover of an engine block from the top of the engine block in the frame of the motorbike to a location outside of the frame. Again, in the final assembled state there are many parts of both the rigid and static geometry which are very close to each other and almost being in collision. The total number of triangles in this test case is more than 650,000.
3. TC CEM (central electronic module) Box, see Figure 2 and Table 1 for details.
4. TC Large Environment, see Figure 3 and Table 1 for details.
5. TC Tunnel Console, see Figure 4 and Table 1 for details.

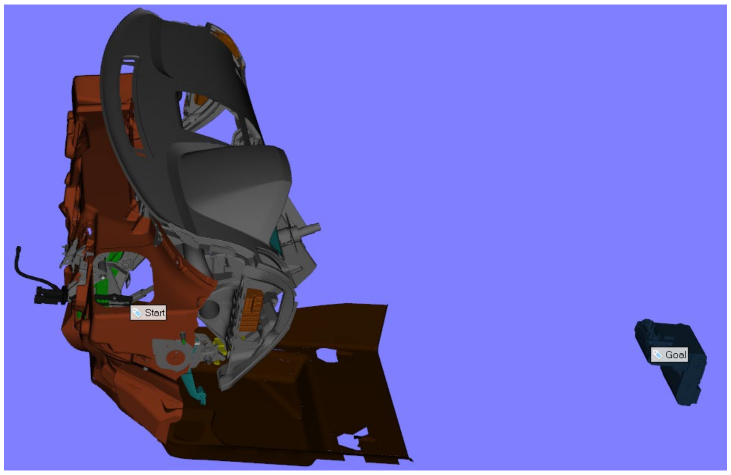

Figure 2. TC CEM Box: The CEM (central electronic module) box on the right side of the figure is the rigid body, whereas the parts of the dashboard on the left is considered to obstacles. Labels *Start* and *Goal* mark the rigid body's positions when it is assembled and disassembled, respectively.

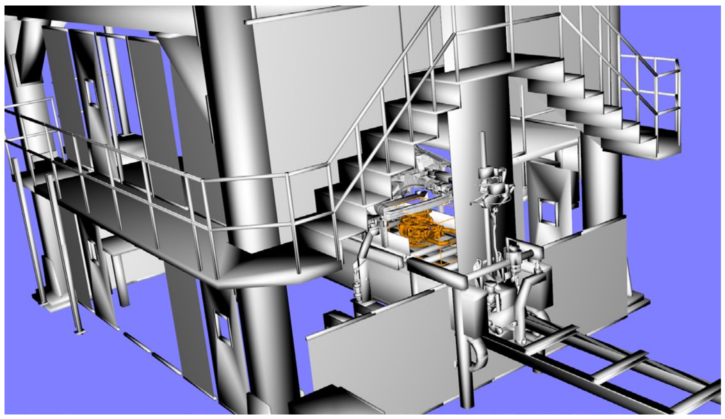

Figure 3. TC Large Environment: The spot-welding machine (highlighted in brown) is the rigid body, whereas the structures around it are considered to be the obstacles.

We evaluate the performance of IPS Path Planner using IPC CDC with the ME-BVH by using it to solve the following three test cases that are the more challenging ones (for path planners) among the test cases listed in Table 1: TC Air Filter Assembly, TC Cylinder Head, and TC Tunnel Console. In each test case, parts to be disassembled are grouped into a single rigid body and the goal of IPS Path Planner is to compute a collision-free disassembly path for the rigid body as fast as possible. For comparison, we also solved these path planning problems using IPS CDC with the traditional approach to construct BVHs. In addition, given the focus is on the performance of the BVH we didn't add human dynamic or simulation elements to the computations.

Table 1. The number of vertices and triangles in all 3D triangular meshes used by the test cases. Models with superscript † represent the triangular meshes of the rigid bodies, whereas models with superscript ‡ represent the triangular meshes of the obstacles. Bolded numbers are for the ME-BVH.

3D Models	No. of Vertices	No. of Triangles	No. of BVs	Memory (kb)	Build Time (msec)
TC Air Filter Assembly†	797,076	1,253,476	2,506,951 vs **1,525,577**	190,507 vs **125,337**	8,272 vs **7,900**
TC Air Filter Aassembly‡	290,433	415,378	830,755 vs **500,159**	63,438 vs **41,484**	2,195 vs **2,056**
TC Cylinder Head†	264,495	246,584	493,167 vs **305,365**	38,738 vs **26,267**	1,375 vs **1,266**
TC Cylinder Head‡	423,742	413,742	827,483 vs **512,659**	64,764 vs **43,858**	2,218 vs **2,094**
TC CEM Box†	23,691	32,615	65,229 vs **39,817**	4,991 vs **3,308**	209 vs **193**
TC CEM Box‡	312,349	456,782	913,563 vs **568,917**	69,679 vs **46,806**	3,134 vs **2,896**
TC Large Environment†	64,184	127,265	254,529 vs **156,349**	19,145 vs **12,626**	771 vs **753**
TC Large Environment‡	1,161,904	1,742,755	3,485,509 vs **2,104,553**	265,498 vs **173,794**	14,706 vs **14,135**
TC Tunnel Console†	65,997	73,469	146,937 vs **90,269**	11,391 vs **7,628**	421 vs **375**
TC Tunnel Console‡	288,484	300,567	601,133 vs **374,789**	46,821 vs **31,791**	1,688 vs **1,578**

Table 2. The average numbers of Collision Queries (CQ) and Distance Queries (DQ) that IPS CDC is able to perform per second with BVHs built by the traditional approach and the ME-BVH, respectively. Averaged over 100 runs.

Test Cases	Traditional BVH		ME-BVH	
	CQ	DQ	CQ	DQ
CEM Box	14,103	2,259	12,177	949
Large Environment	57,009	2,201	55,454	1,184
Tunnel Console	7,891	4,754	6,376	1,519

Before running the path planning problems, we tested the number of possible distance queries and collision queries that ME-BVH could carry out each second compared to the default algorithm in a number of different configurations arising during the path planning exercise. These results are presented in Table 2.

After solving the three path planning problems (i.e., TC Air Filter Assembly, TC Cylinder Head, and TC Tunnel Console) with IPS Path Planner using IPS CDC with the ME-BVH algorithm, the experimental running times and the speedups (compared to IPS Path Planner using IPS CDC with the traditional approach to construct BVHs) are listed in Table 3.

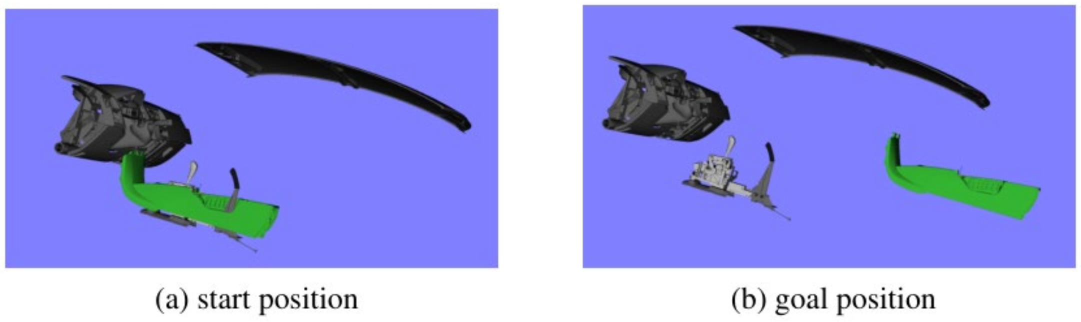

Figure 4. TC Tunnel Console: The tunnel console (in green) in start and goal configurations.

Table 3. The experimental running times (in seconds) of IPS Path Planner using IPS CDC with the traditional approach to construct BVHs and IPS CDC with the ME-BVH. The speedups (compared with the traditional approach to construct BVHs) are listed in the parentheses.

Test Cases	Traditional BVH	ME-BVH
Air Filter Assembly	16 (1.00)	20 (0.80)
Cylinder Head	136 (1.00)	149 (0.91)
Tunnel Console	33 (1.00)	33 (1.00)

The results in this section demonstrated on the five case studies that ME-BVH reduced memory consumption by an average 33% and ME-BVH reduced build time by an average 6%. These benefits are gained for less than a 10% average reduction in collision query performance and up to 66% reduction in distance query performance. It should be noted that the BVH build phase in IPS CDC is highly parallelized with each branch being built with a separate thread and so if a user is unable to use a parallel build strategy, we expect even larger improvements to the build times for non-threaded scenarios.

Although the ME-BVH led to slower distance queries, overall, the ME-BVH algorithm did not lead to significantly slower planning times, which is the key benchmark. This is due to several heuristics in the IPS CDC path planner which depend on the pairs of nodes in the BVH hierarchies tested against each other during the proximity queries and so the path planning timings don't always correspond 1-1 with distance query performance.

Discussion and Conclusions

In this paper, we have presented a new technique called ME-BVH with the main aim to substantially reduce memory requirements by not building unnecessary BVs. Overall, we see the approach achieving its desired goals of memory reduction and build time reduction at little or no cost to path planning execution times.

Digital human modelling typically requires modelling deformable meshes due to the movements carried out by virtual manikins during assembly activities. The ME-BVH has advantages for deformable meshes. A common approach to accelerating distance and collision queries for deformable meshes is to rebuild BVHs at specific points in time. A key advantage of the ME-BVH is that it has half as many BVs in the BVH and hence faster rebuild times as seen above. Building a whole BVH takes 10,000 to 100,000 times longer than a distance query or collision query which is why BVH designs for deformable meshes balance BVH query performance with rebuild time (Shellshear, Bitar, & Assarsson, 2013).

If we save 6% on the build time, as the results here demonstrate, then for BVH algorithms for deformable meshes like in (Shellshear, Bitar, & Assarsson, 2013), the benefit of faster build times will outweigh the slightly slower distance and collision queries. (Shellshear, Bitar, & Assarsson, 2013) showed that even a smarter build phase which intelligently balances the need to rebuild with query performance, still means the build phase makes the distance queries ten times slower on average (collision queries would take an even larger performance hit than that). So, a 6% improvement in the build phase would offset a 60% slowdown of the average collision query or distance query time. This implies using the ME-BVH in these contexts saves us memory without any penalty to the average distance query performance and faster collision queries on average.

Further avenues for investigation would be to test balancing the BVH before building to ensure more nodes have two leaves as children instead of the typical one third, although challenges exist due to worse performance in hierarchies which are forced to be balanced (Held, Klosowski, & Mitchell, 1995). Further research could also look at the performance of ME-BVH with other bounding volume types instead of RSSs.

Acknowledgments

This work is part of the Sustainable Production Initiative and the Production Area of Advance at the Chalmers University of Technology. We would like to acknowledge the generosity of Volvo Cars in providing the models that were used for the experiments in this work.

References

Dammertz, H., Hanika, J., & Keller, A. (2008). Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. In *Computer Graphics Forum* (pp. 1225-1233). Wiley.

Ericson, C. (2004). *Real-time collision detection.* CRC Press.

Eriksson, D., & Shellshear, E. (2014). Approximate distance queries for path-planning in massive point clouds. *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, (pp. 20-28).

Forsberg, T., Mårdberg, P., Roll, R., Rilby, E., & Carlson, J. S. (2018). Demonstration of simulation software industrial path solutions IPS. *Proceedings of the 2018 Winter Simulation Conference*, (pp. 4259-4259).

Held, M., Klosowski, J. T., & Mitchell, J. S. (1995). *Speed comparison of generalized bounding box hierarchies.* Department of Applied Math, SUNY Stony Brook.

Hubbard, P. M. (1993). Interactive collision detection. *Proceedings of 1993 IEEE Research Properties in Virtual Reality Symposium*, (pp. 24-31).

Krishnan, S., Pattekar, A., Lin, M. C., & Manocha, D. (1997). Spherical Shell: A Higher Order Bounding Volume for Fast Proximity Queries. *Proceedings of Third International Workshop on Algorithmic Foundations of Robotics.*

Larsen, E., Gottschalk, S., Lin, M. C., & Manocha, D. (2000). Fast distance queries with rectangular swept sphere volumes. *Proceedings of 2000 IEEE International Conference on Robotics and Automation*, (pp. 3719-3726).

Lauterbach, C., Mo, Q., & Manocha, D. (2010). gProximity: hierarchical GPU-based operations for collision and distance queries. *Proceedings of 2010 Computer Graphics Forum*, (pp. 419-428).

Li, Y., Delfs, N., Mårdberg, P., Bohlin, R., & Carlson, J. S. (2018). On motion planning for narrow-clearance assemblies using virtual manikins. *Procedia CIRP*, (pp. 790-795).

Li, Y., Shellshear, E., Bohlin, R., & Carlson, J. S. (2020). Construction of Bounding Volume Hierarchies for Triangle Meshes with Mixed Face Sizes. *Proceedings of 2020 IEEE International Conference on Robotics and Automation*, (pp. 9191-9195).

Pan, J., Chitta, S., & Manocha, D. (2012). FCL: A General Purpose Library for Collision and Proximity Queries. *Proceedings of 2012 IEEE Int. Conference on Robotics and Automation.*

Shellshear, E., & Bohlin, R. (2014). A heuristic framework for path planning the largest volume object from a start to goal configuration. *Proceedings of of the 11th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, (pp. 264-271).

Shellshear, E., & Ytterlid, R. (2014). Fast distance queries for triangles, lines, and points using SSE instructions. *Journal of Computer Graphics Techniques*.

Shellshear, E., Bitar, F., & Assarsson, U. (2013). PDQ: Parallel Distance Queries for deformable meshes. *Graphical Models*, 69-78.

Spensieri, D., Carlson, J. S., Bohlin, R., Kressin, J., & Shi, J. (2016). Optimal robot placement for tasks execution. *Procedia CIRP*, (pp. 395-400).

Tafuri, S., Shellshear, E., Bohlin, R., & Carlson, J. S. (2012). Automatic collision free path planning in hybrid triangle and point models: a case study. *Proceedings of the 2012 Winter Simulation Conference (WSC)*, (pp. 1-11).

Terdiman, P. (2001). *Memory-optimized bounding-volume hierarchies*. Retrieved from www.codercorner.com/Opcode.pdf

Yoon, S.-E., Salomon, B., Lin, M. C., & Manocha, D. (2004). Fast collision detection between massive models using dynamic simplification. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on geometry processing*, (pp. 136-146).

Zachmann, G. (1995). The BoxTree: Exact and fast collision detection of arbitrary polyhedra. *First Workshop on Simulation and Interaction in Virtual Environments (SIVE 95).*