Accelerating Line-of-Sight Computations in Large OneSAF Terrains with Dynamic Events

C. Lauterbach, M. C. Lin, D. Manocha
Univeristy of North Carolina
Chapel Hill, NC
{cl,lin,dm}@cs.unc.edu

S. Borkman, E. LaFave, G. Peele
Applied Research Associates
Orlando, FL
{sborkman,elafave,gpeele}@ara.com

M. Bauer
STTC, RDECOM
Orlando, FL
maria.bauer@us.army.mil

ABSTRACT

We describe novel algorithms to accelerate the performance of line-of-sight (LOS) computations in large terrains with dynamic entities and events. The underlying approach can handle all kinds of dynamic environments with large number of moving entities, modifiable and dense urban features and may include environmental elements such as terrain skin and features (trees, roads, buildings) an entities such as smoke, clouds, etc. Our algorithm makes use of dynamic bounding volume hierarchies (D-BVHs), which are represented in terms of axis-aligned-bounding-boxes (AABBs). We describe efficient algorithms to compute the D-BHVs by using a combination of refitting and restructuring techniques and perform fast intersection tests between AABBs and the LOS to improve the runtime performance. We have integrated our algorithm with the OneSAF Objective System (Versions 1.1 and 1.5) and created an LOS services library inside of the OneSAF Environment Runtime Component (ERC). We have measured the LOS performance of the existing OneSAF algorithm and our novel D-BVH algorithm on many test suites including JNTC and JRTC databases. Our new integrated LOS algorithm executes the query in 18 micro-seconds per call on a current desktop PC within high-resolution, urban exercises. In practice, our D-BVH algorithm is about 3X faster than the current OneSAF v1.5 LOS routines and about 10X faster than OneSAF v1.1 LOS routines. To the best of our knowledge, this is the first efficient approach to accelerate LOS computations in large terrains with dynamic entities and events. Our formulation can also be used to accelerate route planning, collision detection and physics-based simulations in dynamic terrains.

Accelerating Line-of-Sight Computations in Large OneSAF Terrains with Dynamic Events

C. Lauterbach, M. C. Lin, D. Manocha
Univeristy of North Carolina
Chapel Hill, NC
{cl,lin,dm}@cs.unc.edu

S. Borkman, E. LaFave, G. Peele
Applied Research Associates
Orlando, FL
{sborkman,elafave,gpeele}@ara.com

M. Bauer
STTC, RDECOM
Orlando, FL
maria.bauer@us.army.mil

BACKGROUND

In this paper, we address the problem of accelerating the computation of line-of-sight (LOS) in terrains with dynamic entities and geometric features. Our driving application is the OneSAF Objective System (OOS) Environment Runtime Component (ERC), where the prior implementation of Line-of-Sight is relatively expensive and can take a significant percentage of CPU cycles. Most prior methods used to accelerate the performance of LOS computations are mainly limited to static terrains [Verdesca et al. 2005] and may not extend to large terrains with dynamic events. The simplest algorithms for LOS computations in dynamic terrain datasets walk every triangle along the LOS ray to see if any of them blocks the ray. Checking the ray against the terrain features, both integrated and non-integrated, complicates the process. Integrated features are features used in triangulating the terrain skin, ensuring that all terrain triangles are associated with exactly one feature. If the integrated feature has volumetric properties, the ray is also checked against the volume created by the integrated feature. Non-integrated features are placed on the terrain surface using an anchor point. All of the point features in an area are loaded, converted into a volume, and then checked against the LOS ray. The advantage of integrated features versus non-integrated point features is that only the features that lie directly beneath an LOS ray are checked in an LOS call. The point features are stored in blocks—basically sub-pages of a geotile. If a ray intersects one of these blocks, all of the features in that block have to be checked for intersection. These blocks are large, and the majority of these features will not be anywhere near the LOS ray. In this manner, the LOS computation can be expensive when the terrain is represented using a high number of triangles and features.

BOUNDING VOLUME HIERARCHIES

Our algorithm to accelerate this computation is based on use of bounding volume hierarchies (BVHs). A BVH is a versatile and general data structure that have been widely used to accelerate intersection computations for ray tracing (Rubin and Whitted, 1980), collision detection (Ericson, 2004) and visibility computations. One of the main advantages of BVHs is that they can support deformable models by adjusting the hierarchy to account for changes in geometric objects, which sets them apart from competing data structures such as spatial partitioning hierarchies such as a KD-tree. In the past, hierarchical approaches have been mainly limited to static scenes and terrains. In this paper, we introduce the notion of dynamic-BVHs that can accelerate many computations in dynamic terrains. Our major contributions are novel formulation to compute dynamic-BVHs, using them for fast LOS computations and finally integrating these technologies into the OneSAF system.

A BVH denotes a tree of nodes where every node is associated with some kind of bounding volume (BV). The main property of a BVH then is that each node's volume always fully contains all the bounding volumes of all its children (see Fig. 1 for illustration). At the bottom of the tree, nodes then contain the actual geometric objects.

Bounding volumes

The principal decision when using BVHs is the choice of underlying bounding volume (BV). Many alternatives such as axis-aligned or oriented boxes, spheres, k-DOPs and many more exist. There are multiple factors involved when picking a bounding volume:

- *Fit of the BV*: how closely it conforms to the geometric objects without leaving empty space. This is usually closely correlated with the culling performance.
- *Intersection performance*: the most common operation on BVs is intersection either with other BVs or objects such as rays. This has the largest impact on runtime query performance.
- *Construction performance*: finding the BV that encapsulates a set of objects the most very frequent operation during the construction of the hierarchy.
- *Storage complexity*: the memory footprint of the BVH depends on how much memory it costs to store each BV, and can be an important factor for large scenes.

We use axis-aligned bounding boxes (AABBs) in our algorithm to speed up LOS computations. In general, AABBs present a good balance in terms of the factors described above. Even though an AABB may not offer as close a fit to objects as more complex BVs for some scenes, their improve performance in terms of fast intersection computation and storage overhead makes them a worthwhile candidate.

Ray Intersection using BVHs

One of the most interesting applications of a BVH is that for a set of geometric objects it allows to quickly answer queries such as testing visibility between two points or finding an intersection of any line and the objects along a ray. The naïve way to test this would be to test the ray against every object until an intersection is found or all objects have been tested. Of course, as the number of objects grows, this becomes very inefficient. The BVH provides an alternative that will usually only have to test a small number of objects against the ray.

The idea for intersecting a ray with a BVH is the following: given the property that the bounding volume of each BVH node contains all its children, it can be concluded that if a ray does not intersect the BV of a node, it also cannot intersect any of its children and thus also cannot intersect any of the geometric objects in that part of the tree. Therefore, just by performing one intersection it is possible to rule out or *cull* a very large number of geometric objects at a time.

An implementation of this approach involves intersection computation of a ray with a BVH node (starting with the root of the tree). If the ray hits the BV, then all the children of the node can also potentially be intersected, so they are put in a work list. If it does not hit, then all the children are ignored. The algorithm continues by fetching a node from the work list until it is empty. Every time it reaches the bottom of the tree, the ray needs to be intersected with the actual geometric object. If a viable intersection is found at this point, then the algorithm has the option of either returning this information (for example, if testing for visibility between points, as soon as an occluding object is found) or it can continue to find additional intersections.

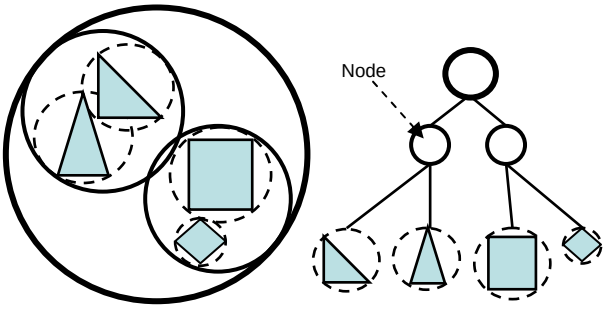

Figure 1: BVH example. Left: Four geometric object are encapsulated by spherical BVs. Right: The BVH representing this shown as a tree.

In practice, our algorithm can reduce the number of intersection with geometric objects to only a few. The overall complexity increases sub-linearlyas a function of terrain size and geometric features. Moreover, it only accesses a logarithmic number of tree nodes of the BVH.We refer the reader to [Lauterbach et al. 2006] for more details.

BVH construction and update

The actual hierarchy for the BVH is built as a preprocessing step. In our approach, we use a binary tree, but it is equally possible to choose a higher branching factor. The construction proceeds by recursively partitioning the set of objects in the scene into two subsets until each set only has one object in it. This approach creates a tree of nodes in the form as described above and also assigns the BVs to each node.

The partitioning step makes use of a metric called the surface area heuristic (SAH) [Goldsmith and Salmon 1987, MacDonald and Booth 1990, Havran 2000] to decide the manner in which to split up the set of objects. The SAH metric has been widely used in the interactive ray tracing literature to construct tight fitting hierarchies that can minimize the number of runtime intersections between the ray and the nodes of the tree. In this paper, we use the same metric for fast LOS computation. While in theory any top-down partitioning of the BVH is possible, the actual choice has a strong influence on the actual intersection performance with the BVH (as correlated with the

number of nodes needed to be tested for a given ray). The SAH uses a cost function based on the probabilities that a ray or LOS can hit a bounding volume and then locally minimizes this cost during each partition. Results have shown that the effect of SAH partitioning can more than double the performance of ray intersections (Wald et al. 2007).

Handling Dynamic Events with BVHs

The real advantage of BVHs is that they can support dynamic scenes where individual geometric objects deform or move or explode by recomputing the BVs. Therefore, BVHs only have to be built once before being used and can then be reused even after object changes.

One of the simplest BVH recomputing algorithms is based on *refitting*. The refitting step for BVHs essentially processes each node and changes its bounding volume so that it can tightly bound the contained geometric objects. If starting at the bottom levels, all the changes to the boxes are propagated upwards in the tree until they reach the root and visit all the nodes on the way. Note that this process means that the actual structure of the tree is never changed and only the BV information needs to be updated. This is much faster than rebuilding a hierarchy and in practice only takes milliseconds even for complex environments (Lauterbach et al. 2006).

Multi-threading and scalability

Since current processors exploit parallelism for higher performance, it is imperative that the underlying algorithms perform well on such systems. Fortunately, ray tracing in general is embarrassingly parallel in that it is possible to answer any number of ray-BVH queries in parallel, e.g. by using multiple threads. Thus, it is perfectly scalable in most environments. Previous work has shown that ray tracing with hierarchies is also very cache-coherent and therefore works well in the memory environment of current CPUs [Lauterbach et al. 2006].

The refitting step for deformable models can also be parallelized between multiple threads. Since the tree structure does not change after the preprocessing BVH construction step, we can find several sub-trees of comparable size and refit them in parallel on several cores or processors. Afterwards, one processor or core can quickly propagate the results from the sub-trees for the few nodes that contain them. While this is relatively scalable in computation, this parallel update usually becomes limited by available memory bandwidth on commodity hardware and in our experience works well on systems with 2-4 cores.

LOS COMPUTATION USING BVHs

One of our goals is to implement the dynamic BVH algorithm within the OneSAF system. The BVH LOS Service is a layer developed to exist between the new BVH LOS suite and the existing OOS ERC code. This layer essentially handles the communication between OOS and the BVH structures. It is also responsible for the storage and translation of important data types and information.

In terms of design choices, the BVH LOS Services layer has to meet the following requirements:

- Be capable of retrieving terrain triangle and feature data from the OneSAF Objective System.
- Convert the OneSAF data into BVH specific data types.
- Compile, serialize, and write to disk BVH pages in a pre-exercise application.
- Be capable of using both the BVH LOS routine along with the current OneSAF ray-trace LOS routine.

We designed an application layer to meet those requirements. The layer exists between OneSAF and the BVH library and consists of the following components:

- BVH Tree Raytracer Services – The interface to the new BVH-based LOS calls. Interfaces between the LOS Controller and BVH Tree Raytracer.
- BVH Tree Raytracer – BVH library being developed by the BVH development team.
- Feature Manager – Component responsible for managing feature and terrain data. Converts the data from OneSAF data types into BVH data types. Also responsible for handling dynamic changes and translating them to the BVH component.
- BVH Tree Builder – Responsible for retrieving terrain data from OneSAF and converting it into BVH data types.
- BVH Tree Cache – Responsible for managing the BVH structures currently in memory. If a BVH is needed and not currently in memory, it either loads it from disk or creates it using the tree builder.
- BVH Tree Disk Format – Compiled form of the BVH tree. We originally planned to write

all of the BVH structures out to disk and load them, but we did not receive an API to do this.

- Feature Change Set – Responsible for storing dynamic changes that occur after a compiled BVH tree is created.

This is highlighted in the overall architecture diagram shown in Figure 2.

Los Computations

In this section, we describe various changes and new algorithms that were implemented to support BVH-based LOS computation within OneSAF.

Coordinate systems

The BVH representation needs the OneSAF data to be converted from OneSAF data types into their unique data formats/structures. This data includes the terrain skin triangles, the integrated features, and point features. OTF stores all of the terrain data in round earth representation: every stored coordinate can be referenced either in geodetic coordinates – latitude, longitude, and elevation - or in geocentric coordinates – Cartesian coordinate system with its origin at the center of the Earth.

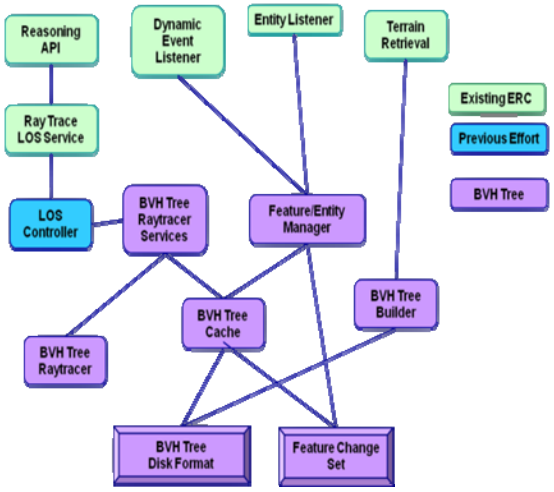

Figure 2: This figure gives an overview of our D-BVH based LOS computation algorithm integrated into OneSAF.

Cartesian coordinate systems are necessary for LOS computations, which makes using geocentric coordinates a necessity. However, geocentric values are extremely large and may need 64-bit floating point values to accurately model the terrain of the entire Earth. Instead of using Geocentric values, we transformed these coordinates into local tangent planes. By converting these values over to a local tangent system for each terrain page, Our approach is able to use less expensive 32-bit floating point values to accurately model the terrain data.

Terrain retrieval and conversion

The Terrain Retriever class was developed to be the interface between OOS data types and the BVH LOS library. It is responsible for extracting terrain data from the OTF database both during pre-exercise terrain compilation and during page loads mid-exercise. It retrieves all of the terrain triangles through the ERC Terrain Retrieval API and passes them to the BVH Builder class.

Some special processing is performed to create the triangles for integrated features. As the terrain triangles are processed they are checked to determine if they are associated with a feature that could block an LOS query. If they are capable of blocking the LOS ray, they need to be extruded from the terrain surface using their height attribute. This requires making an elevated triangle for the top of the feature, and the "face" triangles that span from the original triangle elevation to the raised triangle height.

Non-integrated point features are handled differently. The Terrain Retriever requests all of the non-integrated features for the compilation region and builds a bounding volume for them based on their attributes including the length, width, and height.

Feature Management

The fundamental structure for BVH - OOS integration is the Feature Manager. The LOS Controller associates a Feature Manager to a BVH page. Each Feature Manager contains a BVH Wrapper—the interface to the external BVH code.

The purpose of the Feature Manager is to maintain a BVH Wrapper that can be used to perform LOS calculations on all of the features in a page and to serve as the connection between OneSAF features and BVH Wrapper Model Instances. It contains all of the triangles that create both the terrain skin and terrain features.

The LOS Controller uses the Feature Manager to add, modify, and remove features. When a feature is added to the Feature Manager, the Feature Manager takes control of the feature and is responsible for the management of its used memory. When the Feature Manager receives each feature (including terrain triangles) it transforms them into the local Cartesian coordinate system appropriate for the compilation

region and adds them to the BVH structure through the BVH Wrapper.

Management of Dynamic Terrain

OneSAF Dynamic Feature Capabilities: The driving force for the BVH solution is to implement a high-performance LOS solution that can handle dynamic environments. Unfortunately, OneSAF v1.5 contains limited support for dynamic events. In fact, this version currently only support the addition, deletion, and attribute modification of terrain features). OneSAF currently has no ability to dynamically change the terrain skin through events such as cratering. Nevertheless, our BVH solution will be able to handle more of these dynamic features in the future versions of OneSAF.

The LOS Controller and each page's Feature Manager are responsible for handling thedynamic events. When the LOS Controller receives a new dynamic feature event (either add, delete, or modify) it identifies the proper Feature Manager for that Feature's page and passes the call to the Feature Manager.

The Feature Manager identifies the feature being modified and successfully communicates the event to the BVH Wrapper. The Feature Manager also conducts some "bookkeeping" exercises to properly maintain its current state and allocated memory for the BVH Wrapper. One important note: Since the BVH tree cannot properly modify a feature at the current time, a "modify" event is treated as a deletion of the feature's previous state and a creation of a new feature (i.e. delete followed by an add).

Integrated LOS Services

The integration of the dynamic BVH into the ERC LOS code is reasonably straightforward. The LOS services library manages the BVH services. When an attenuated ray-traced LOS is requested within ERC, the call is delegated to the LOS controller, a class in the LOS services library, which checks to see if the BVH LOS is enabled. If so, it sends the ray to the BVH based LOS and processes it.

On the first LOS, the BVH for the page that contains the LOS ray is loaded from the terrain database into the cache. After that, if an LOS query occurs on an already loaded page, then the BVH-based LOS is called for that particular page. If it is not currently in the cache, then that page is loaded off disk and added to the cache. If space is not available for the new page, the least-recently-used page is unloaded to create space for the new page.

Other OneSAF elements such as entities and obscurants must always be checked even if the BVH LOS determines a clear LOS, since they are not included in the BVH structure. The algorithms to perform these computations were embedded in the existing ERC ray tracer LOS method and thus we had to factor them out so that they could be used separately.

INTEGRATION & RESULTS

After implementation and integration of the BVH code was complete, we were able to analyze their performance timings and result accuracy on different benchmarks.

Testing

Our major goals for this effort were to find significant performance improvements compared to the traditional OOS ray-trace LOS routine, while not sacrificing accuracy of the results. Because of this goal, we placed a strong emphasis on the integration and test phase of this effort. The major focuses for integration were:

- Accurate results
- BVH performance
- Overall LOS performance improvements

To achieve these objectives we designed a test suite to assure correlation and to capture performance results. We set up our test suite to run actual exercised captured LOS rays on both the BVH LOS and the original ERC LOS calls. Also we chose to test not only against multiple exercises, but also on multiple terrain databases. OneSAF is delivered with many pre-developed exercises on two different databases: JNTC and JRTC.

The two databases, specifically where the exercises occur on the databases, are vastly different from one another. The JRTC database is a feature-rich database using a high fidelity urban inset, with a lot of feature information, including buildings and trees. The JNTC database is of a more barren area, with a low feature count. Having multiple datasets allowed us to test the system on a wide range of terrain data, from high fidelity urban databases and low fidelity rural areas.

We captured "real" line-of-sight rays from executing exercises to accurately test the system. A listener class that records LOS rays to a file was attached to the LOS routine. Four different exercises were chosen for testing, two from both the JRTC and JNTC databases.

		Number of Tests	ERC v 1.5 Time (µs)	BVH Time (µs)	% of ERC Time
JRTC	Ambush	507840	47.311	17.679	37.37%
	Raid	400420	50.217	18.785	37.41%
JNTC	Conduct Direct Air Fire	97160	33.884	19.666	58.04%
	Massacre	71580	11.572	15.892	137.33%

Table 1- Performance results between the original OOS ERC LOS routine and the BVH based LOS routine. On higher resolution databases, the BVH based LOS routine was significantly faster.

		Number of Tests	Errors	False Positives	False Negatives	Correlation
JRTC	Ambush	25392	11	6	5	99.957%
	Raid	20021	8	0	8	99.960%
JNTC	Conduct Direct Air Fire	4858	1	0	1	99.979%
	Massacre	3579	0	0	0	100.000%

Table 2 – Correlation test results between the original OOS ERC LOS routine and the BVH based routine. With over 99% accuracy, the BVH based LOS and the ERC ray trace LOS have very high correlation.

These exercises were JNTC Massacre, JNTC Direct Air Fire, JRTC Ambush, and JRTC Raid. Using the MCT, the exercises were run as normal and the LOS rays were captured to text files.

The ray files were then used to test and verify the system. Using the four different exercises became an invaluable testing system. Many times, tests would execute correctly on one dataset and have correlation issues on another dataset. The test suite allowed us to identify many potential issues, and using it we were able to integrate the culling code and meet our goals of complete correlation along with reasonable performance and effectiveness.

Benchmarks

Performance

We collected performance benchmarks of the BVH solution and OneSAF ERC LOS on our four standard test exercises. The OneSAF v1.5 ray-trace LOS calls average approximately 48 µs per call. The performance of the BVH solution was significantly faster than the ERC solution on three of the four test cases. When ran on the JRTC database in the urban inset area, the BVH code took almost a third of the time as ERC. On the JNTC, a rural, desert area, the BVH had mixed results. As the timing numbers show, the BVH technique achieves near constant results independent of terrain resolution with all of the timings within a four microsecond range. The ERC tests are more volatile ranging from eleven to fifty microseconds and suffer on high resolution terrain. The performance of the ERC LOS degrades with triangle count, the more triangles that it processes the slower the traversal. The JNTC database is far lower resolution than the JRTC database. This means that each of the LOS queries process less triangles on JNTC, allowing the significant difference in performance when compared to the higher resolution JRTC database.

BVH - OneSAF Correlation

Table 2 shows the correlation between OneSAF ERC's LOS routine and the integrated BVH code. Although the methods do not show 100% correlation, the results far exceed our expectations. Explanations for the discrepancies include UHRB interiors—which we do not model—floating point inaccuracies, triangulation of point feature cylinders, among many other things. The fact that we have replaced an entire terrain model with another one and achieved 99.9% correlation for every database— and 100% correlation for one particular test highlights the accuracy of our system. We hope that this would be used for complex terrains in the future.

Memory

This effort was conducted as a research effort to study how novel algorithms could be used to enhance performance in OneSAF, particularly in dynamic

environments. Our approach has focused on integrating the BVH LOS solution into OneSAF with the smallest amount of disturbance to baselined LOS code. In our current implementation, we chose to integrate the BVH data structures in parallel with the current OneSAF data structures.

Due to the parallel solution it is a little difficult to get an accurate analysis of the memory overhead of our BVH-based solution. However, BVH-based representation can also be useful for many other computations within OOS including avatar simulation, collision checking and route planning. Currently,we duplicate all of the terrain data: a set in the original OneSAF data structures, and a set in the BVH data structure. In the future, such data structures can be easily merged and thereby, reduce the memory overhead

CONCLUSIONS AND FUTURE WORK

Our solution for fast LOS computations in dynamic terrains is quite effective. Moveover, we made a focused effort to minimize modifications to existing ERC code. All of the BVH-related code is separated into its own library and only three existing non-test classes are modified. The implementation is intended to be very configurable to be able to handle varying hardware capabilities and user wants.

The BVH solution proved to be valuable and significantly faster than the current OneSAF approach. The fact that it will be capable to handle a repolygonalized terrain surface with very little modification demonstrates its effectiveness. These dynamic BVHs can also be used to accelerate many other computations including route planning, collision detection, physics-based simulation and crowd simulations in dynamic terrains.

There are many avenues for future work. We will like to apply our algorithms to more complex terrains with a higher degree of dynamic events. We can exploit the capabilities of multi-core CPUs or many-core GPUs to further accelerate the computations. We will also like to compare the performance of our LOS algorithm with the LOS routines in the future versions of OneSAF (e.g. Version 2.0).

ACKNOWLEDGEMENTS

The work described in this paper is supported in part by RDECOM Contracts N61339-05-C-0145 and N61339-04-C-0043. We would like to thank Kent Pickett, John Logdson, Bruce Robbins, Angel Rodriguez, Oanh Tran and Buck Surdu for their feedback and support. We are also grateful to Michael Proctor for his feedback on an earlier draft of this manuscript.

REFERENCES

ERICSON, C. (2004). Real-Time Collision Detection. Morgan Kaufmann.

GOLDSMITH, J., & SALMON, J. (1987). Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl. 7, 5*, 14–20.

HAVRAN, V. (2000). Heuristic Ray Shooting Algorithms. PhD thesis, Czech Technical University in Prague.

LAUTERBACH, C., YOON, S., TUFT, D., AND MANOCHA, D. (2006). RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. *IEEE Symposium on Interactive Ray Tracing '06.*

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D. AND MANOCHA, D. (2008). Fast BVH Computations on GPUs. Technical Report, Department of Computer Science, UNC Chapel Hill.

VERDESCA,M., MUNRO, J., HOFFMAN, M., BAUER, M. AND MANOCHA, D. (2005). Using Graphics Processor Units to Accelerate OneSAF: A Case Study in Technology Transition. *Proc. of I/ITSEC.*

MACDONALD, J. D., AND BOOTH, K. S. (1990). Heuristics for ray tracing using space subdivision. *The Visual Computer*.

RUBIN, S. M., AND WHITTED, T. 1980. A 3-dimensional representation for fast rendering of complex scenes. *Computer Graphics 14, 3 (July)*, 110–116.

WALD, I., BOULOS, S., AND SHIRLEY, P. (2007). Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26, 1*.

YOON, S., CURTIS, S., AND MANOCHA, D. (2007). Ray Tracing Dynamic Scenes using Selective Restructuring. *Eurographics Symposium on Rendering.*