Geometry Query Optimizations in CAD-based Tessellations for Monte Carlo Radiation Transport

by

Patrick C. Shriwise

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Nuclear Engineering and Engineering Physics)

at the

UNIVERSITY OF WISCONSIN – MADISON

2018

Date of final oral examination: 06/08/2018

The dissertation is approved by the following members of the Final Oral Committee:

Paul Wilson, Professor, Nuclear Engineering & Engineering Physics
Tim Tautges, Adjunct Professor, Nuclear Engineering & Engineering Physics
Eftychios Sifakis, Professor, Computer Science
Douglass Henderson, Professor, Nuclear Engineering & Engineering Physics
Xiaoping Qian, Professor, Mechanical Engineering

Acknowledgments

This work was funded in part by the Nuclear Regulatory Commission. My thanks as well for their support and their contributions to the nuclear engineering community at large. Additionally I'd like to acknowledge a number of mentors, colleagues, friends, and family members who have supported me throughout the course of my PhD work.

To Dr. Paul Wilson goes my extreme gratitude for taking me on as a student in the Computational Engineering Research Group (CNERG). He is a shining example of integrity, wisdom, and intelligence for his students. These qualities along with a dedicated work ethic are reflected in all of the work that comes out of our research. His holistic view of the graduate student experience and ability to develop an inclusive culture in his group is something of great value to his students, whether they know it or not. I couldn't have asked for a better advisor.

To Dr. Tim Tautges, thank you for all of the conversations in your office about life, politics, research, and otherwise. I'll cherish those times greatly. Your wealth and breadth of knowledge in the field and academic rigor have been of immense value to me as a researcher, and your friendship even more so. Additionally, thank you for providing me with the caffeine necessary to survive the long weeks of writing and research.

I'd like to thank Dr. Andrew Davis for being a daily source of support and positivity in his time at UW. His uplifting attitude and generally helpful nature are missed here at UW. The hours you've saved students

through your willingness to aide and impart knowledge cannot be counted. I thank Andy also for rather enjoyable discussions on ideas about somewhat extreme research concepts and his knowledge about the landscapes of computational nuclear engineering.

My thanks to all of my colleagues in CNERG who are not only coworkers but friends to me. In particular, thank you to Kalin Kiesling for a variety of entertaining discussions along with insight into what it is to be a woman in STEM today. Lucas Jacobson, thank you for all of the hard work you've put in maintaining and extending the DAGMC toolkit. Alex Swenson and Baptiste Mouginot, thank you for making the fourth floor such an enjoyable place to work, full of laughter and fun.

My heartfelt thanks goes out to all of my ultimate frisbee teammates and coaches. This sport has been a critical source of stress relief, grounding, perspective, and personal development for me throughout my time in Madison. I've met so many wonderful people across the globe through this sport, making life-changing and life-long friendships in the process.

To my parents, Rod and Susan, thank you for all of your love and support. You've ingrained in me a set of principles, values, and work ethic which have served me will for many years and many more to come. To Amanda, my sister, thank you for being a strong academic role model to me and for helping me to bet on myself. My gratitude as well to Marcia Bosscher for countless delicious meals, hosted holidays, and general support in my time at UW.

Finally, I'd like to thank my wife, Georgia. You are a constant source of support and joy in my life. Your dedication to your profession inspires me to show the same to mine. I will forever look back fondly on the time we've spent walking Monk along the water on the edge of the arboretum while sharing our hopes and dreams, our fears and frustrations.

Contents

List of Tables

List of Figures

List of Algorithms

DISCARD THIS PAGE

Nomenclature

AABB	Axis-Aligned Bounding Box
ATR	Advanced Test Reactor
AVX	Advanced Vector Extensions
BVH	Bounding Volume Hierarchy
BREP	Boundary Representation
CAD	Computer-Aided Design
CSG	Constructive Solid Geometry
DAGMC	Direct Accelerated Geometry Monte Carlo
DAG-MCNP	DAGMC couplded with MCNP 5 or 6
ERH	Entity Ratio Heuristic
FNG	Frascatti Neutron Generator

GUI	Graphic User Interface
HPC	High Performance Computing
HV	High Valence
Interior Node	Node connected to both parent and child nodes in a tree
Leaf Node	Node connected to only parent nodes, located at the bottom of a tree
OBB	Oriented Bounding Box
MCRT	Monte Carlo Radiation Transport
Meshset	A group of mesh entities in MOAB
MOAB	Mesh Oriented dAtaBase
MPBVH	Mixed Precision Bounding Volume Hierarchy
MPI	Message Passing Interface
nTOF	Neutron Time of Flight model
SAH	Surface Area Heuristic
SHINE	Medical isotope facility

SIMD	Single Instruction Multiple Data
SDF	Signed Distance Field
SDV	Signed Distance Value
SNS	Spallation Neutron Source
SSE	Streaming SIMD Extensions
SSAH	Simplified Surface Area Heuristic
ITER	International Thermonuclear Experimental Reactor
UWNR	University of Wisconsin Nuclear Reactor

Abstract

The performance of direct CAD-based Monte Carlo Radiation Transport (MCRT) relies heavily on its ability to return geometric queries robustly via ray tracing methods. Current applications of ray tracing for MCRT are robust given that certain requirements are met [48], but cause simulations to run much longer than native code geometry representations. This work explores alternate geometry query methods aimed at reducing the complexity of these operations as well as algorithmic optimization by adapting recent developments in CPU ray tracing for use in engineering analysis. A preconditioning scheme is presented aimed at avoiding unnecessary ray queries for volumes with high collision densities. A model is also developed to inform the application of the preconditioning data structure based on a *post facto* analysis. Next, a specialized ray tracing kernel for MCRT is presented. As new ray tracing kernels are developed for real-time, photo-realistic rendering, algorithmic approaches have appeared which are demonstrated to be advantageous when applied in radiation transport. In particular, the application of data parallelism in ray tracing for Monte Carlo is demonstrated - resulting in significant performance improvements. Finally, model features resulting in systematic performance degradation commonly found in CAD models for MCRT are studied. Methods are proposed and demonstrated to improve performance of ray tracing kernels in models with these features. The combination of this work is shown to provide improvement factors ranging from 1.1 to 9.54 in simulation run

time without loss of robustness for several production analysis models. The final impact of this work is the alleviation of concern for additional computational time in using CAD geometries for MCRT while maintaining the benefit of reduced human time and effort in model preparation and design.

Chapter 1

Introduction

This work focuses on methods for modeling radiation transport to determine particle flux, or derived quantities, across the dimensions of space, angle, energy and time known as *phase space*. The behavior of these particles is described by the linear Boltzmann transport equation [37]. Deterministic codes solve this transport equation by discretizing the phase space of the problem domain, but time and memory constraints often limit the resolution of phase space in practical problems.

Monte Carlo Radiation transport (MCRT) simulates the interaction of individual particles across phase space [33]. This method was developed at Los Alamos National Laboratory (LANL) during World War II by Fermi, von Neumann, Ulam, Metropolis, and Richtmyer [53]. It uses a random walk process to solve the transport equation. Pseudo-random numbers are used to sample probability distribution functions representing properties of the virtual medium and in turn determine the particle interaction outcomes. This stochastic approach requires the simulation of many particles to reduce the statistical uncertainty of the solution, where the uncertainty is inversely proportional to the square root of the number of particles simulated. As the number of simulated particles approaches infinity, tallied quantities in the simulation approach the value of the continuous solution.

The pros and cons of the deterministic and Monte Carlo approaches complement each other. While deterministic approaches inherently calculate a solution over the entire problem domain, they take on additional error by discretizing phase space. In contrast, Monte Carlo methods only incur error associated with input parameters such as cross sections or geometry specifications, but it is challenging to achieve a global solution with uniform statistical error using this approach. Computationally, deterministic methods typically suffer memory and run time costs that scale with the resolution of the discretized phase space whereas Monte Carlo methods are typically limited by the run time needed to achieve satisfactory uncertainty in a region of interest.

1.1 Monte Carlo Geometry

Historically, Monte Carlo codes use Constructive Solid Geometry (CSG) as their native geometry representation. CSG represents 3D regions of virtual space using Boolean combinations of half spaces defined by quadratic surfaces. To define the geometry, the surface and volume definitions are typically entered into a text file. This format for geometry is robust once defined properly, but it is difficult to manage for large models and limited in representation compared to more modern geometric modeling tools available in Computer-Aided Design (CAD).

In contrast to CSG, CAD allows for increased accuracy in model representation and better human efficiency. CAD engines are capable of representing higher-order surfaces and provide access to models used for analysis in other engineering domains. These shared models allow for a common problem domain in coupled simulations. CAD systems also provide a rich set of tools for model generation, topological representation, and design iteration. For highly complex, well-developed models, these tools are more intuitive and efficient for human use over alteration of native

codes' text-based formats. Several tools exist for converting native CSG models to and from CAD systems. A few have the capability to perform simulations directly on CAD geometries as well [32].

The Direct Accelerated Geometry Monte Carlo (DAGMC) [52] toolkit is one of several software packages which enables Monte Carlo Radiation Transport on tessellated surfaces (MCRT-t) originating from CAD geometries. DAGMC's design allows it to serve as a particle tracking and geometry kernel for all Monte Carlo codes listed in Table 1.1.

Monte Carlo Code	DAGMC Implementation
MCNP5[64]	DAG-MCNP5
MCNP6[21]	DAG-MCNP6
Fluka[11]	FluDAG
Tripoli4[35]	DAG-Tripoli4
Geant4[6]	DagSolid
Shift[40]	DAG-Shift

Table 1.1: A list of Monte Carlo codes and the names of their corresponding DAGMC implementations.

1.2 Statement of Problem

While the use of CAD geometries brings the benefits outlined above, it also adds complexity to particle tracking during Monte Carlo simulations. Particle crossings with higher-order surfaces are difficult and sometimes impossible to compute analytically. One method of addressing this problem is to discretize the analytic CAD surfaces into a triangle surface mesh. This generalizes intersections with surfaces to intersections with a set of planar surfaces. However, a large number of triangles are needed to maintain an accurate representation of surfaces throughout the model. As a result, the costly search for surface crossings and other fundamental geometry queries cause simulations on CAD-based tessellations to take much longer than native CSG models.

The intersection of a particle and trajectory with a triangulated surface is a well-researched problem in the area of ray tracing. In this field, geometries are also triangulated for visualization and rendering purposes. DAGMC currently employs techniques from this field to accelerate geometric queries, but it is still slower than native geometry implementations in CSG. DAGMC's simulations are anywhere from 2.5 to 10 times longer than those of their native counterparts.

Table 1.2 represents a comparison for several representative MCNP problems between the native geometry representations and DAGMC coupled with MCNP, or DAG-MCNP. For smaller problems with simple geometries and relatively low numbers of histories required to reach a low level of statistical uncertainty, this might not pose as much of a problem to users. As problem geometries become more complicated and the number of histories becomes larger, however, the difference in computing time becomes important when run times extend to days or weeks longer than they would using the native MCNP CSG geometry representation. These models include the Frascati Neutron Generator (FNG), the Advanced Test Reactor (ATR), the University of Wisconsin Nuclear Reactor (UWNR), and the neutron Time of Flight (nTOF) device at the Institute for Science and International Security. It should be noted that only time spent simulating particle histories is reported in Table 1.2, not including setup operations for simulation such as nuclear cross section loading and acceleration data structure construction.

Model	Native run time (min)	DAG-MCNP run time (min)	Timing Ratio
FNG	5871.92	22769.33	3.9
ATR	901.68	8627.80	9.6
UWNR	8767.29	69429.60	7.9
Nerf	167.52	295.13	1.8

Table 1.2: Table comparing the performance of DAG-MCNP to native MCNP for the same models after translation to a CAD-based tessellated surface mesh.

As a part of this study, these runs were repeated using the profiling tool, Callgrind [41], to determine where computing time is being spent in both the DAG-MCNP and native MCNP runs. Because the geometry representation and query system is the only difference between the two models, it is expected that the time is being spent there, but it is practical to confirm this and useful to know more specifically where in the query system this is occurring. All callgraphs are displayed with the MCNP *history_* subroutine at the top. It is inside this subroutine that MCNP calls DAGMC methods to fulfill geometric queries as particles move through the model.

In Figure 1.1 a callgraph for a profiling run of FNG for 1×10^7 histories is provided. In this callgraph it is shown that the time spent in transport is dominated by calls in DAGMC used to track particles as they move through the geometry. About 60% of the run time is spent in DAGMC's *ray_fire* while in native MCNP the relative time spent on this process is reduced to 5% with the majority of the time spent in calculating cross sections under the *acetot_* subroutine.

Figure 1.1: Callgraph of DAG-MCNP FNG run for 1×10^7 histories. Processes taking <10% of the run time are filtered out in order to simplify the graph.

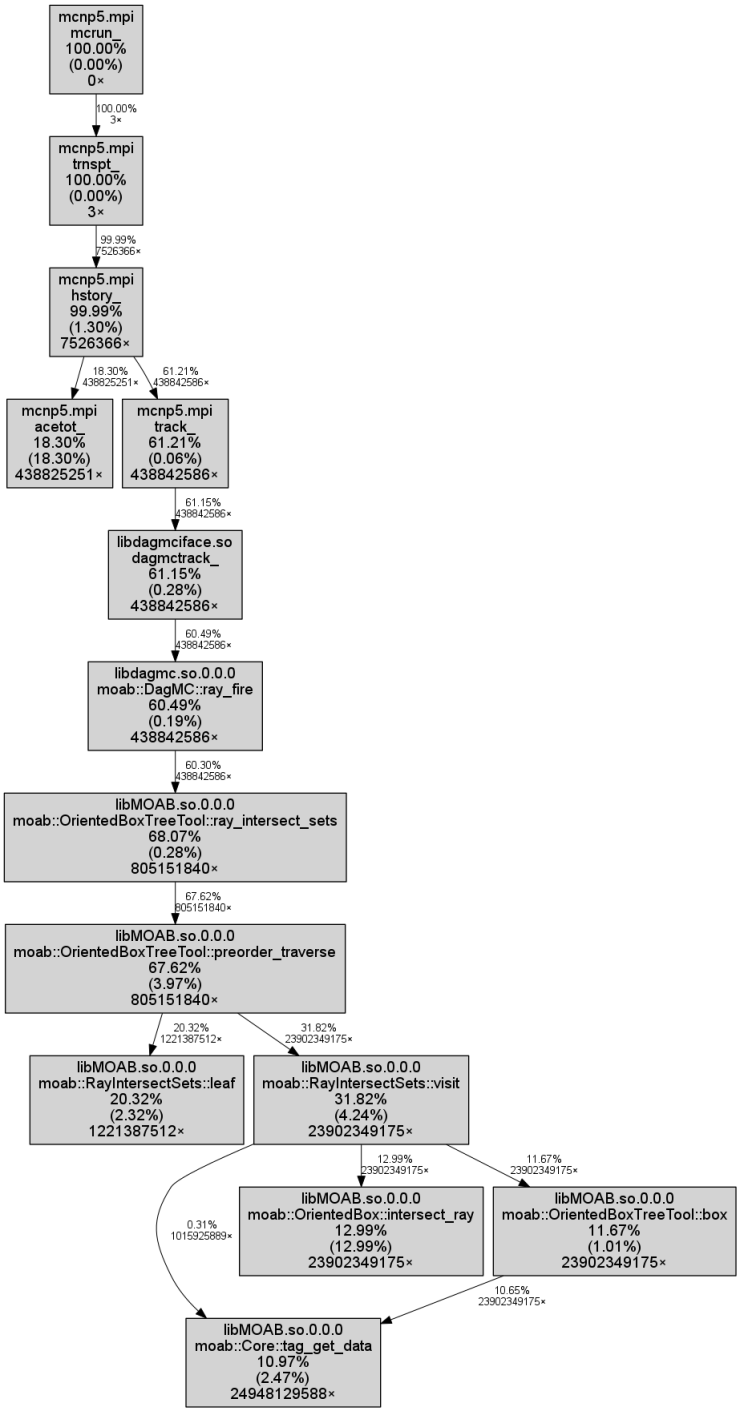
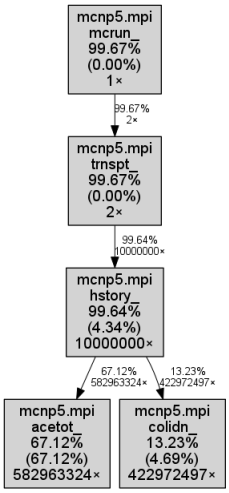


Figure 1.2: Callgraph of native MCNP FNG run for 1×10^7 histories. Processes taking <10% of the run time are filtered out in order to simplify the graph.



The combination of the profiling results indicating how much time is spent in tracking particles in DAGMC along with the difference in absolute simulation times confirms that the performance bottleneck of DAGMC lies in its ability to quickly satisfy the geometric queries of the underlying Monte Carlo code it is coupled to. Looking more closely at the underlying calls in DAGMC, one can see that this time is collectively spent in the DAGMC *ray_fire* method, indicating that a significant amount of additional time is spent in MCRT-t geometry queries.

1.3 Statement of Thesis

The purpose of this dissertation is to discover new pathways toward improving the performance of Monte Carlo radiation transport in CAD-based tessellations in a manner that is widely accessible to analysts. Toward this end, this work aims to improve performance of spatial queries for general purpose mesh data structures through adaptation of rendering techniques

to suit the purposes of engineering analysis and robust particle transport methods. The effect of these adaptations is demonstrated in the DAGMC toolkit for the purpose of radiation transport to significantly reduce simulation run times in this engineering domain.

A background and literature review of CAD-based transport methods and associated acceleration techniques is provided in Chapter 2. First, it outlines required capabilities for particle tracking in MCRT. Next it describes commonly used geometry representations in MCRT codes followed by a description of the MCRT-t particle tracking systems of interest for this work. Lastly, associated acceleration data structures and techniques which enable highly performant computation of transport on CAD models are discussed.

Chapter 3 presents a novel acceleration method for CAD-Based MCRT and its application for each of the relevant geometry query types outlined in Chapter 2. An analytic model is then described to guide the application of this technique based on simple parameters of the problem geometry and physics. Finally, results of this method's application in both contrived and production models are discussed along with limitations of the method and aforementioned analytic model.

Chapter 4 demonstrates the integration of nuclear engineering simulation with state-of-the-art computer graphics tools. Significant improvements in performance are demonstrated in both test and production models by adapting data structures discussed in Chapter 2 for optimal efficiency on modern CPU architectures. Robustness limitations of the computer graphics tools as applied to engineering analysis are discussed, and extensions of these tools are presented to address those limitations. Critical implementation details and algorithmic adjustments of these extensions are outlined, and performance comparisons are drawn in terms of the raw query speed between all particle tracking implementations. Finally results, of the extended system are presented for several production models, in-

cluding several of the models used in benchmark comparison found in Chapter 1.

Chapter 5 addresses a long-standing issue in CAD-based MCRT-t. Performance degradation caused by problematic features of the CAD-based tessellation are characterized using a contrived model for all ray tracing implementations in Chapter 4. A solution using on-the-fly detection of and adaptation to of this feature during construction of particle tracking acceleration data structures is presented. Characterization of the performance both with and without the adaptive construction technique are presented and discussed. Finally, results of this technique as applied to the same set of production models shown in Chapter 4 are presented and discussed.

The conclusion statements in Chapter 6 discuss the contributions of this work to MCRT for CAD geometries, its broader impacts, and possible future directions for various aspects of the work.

Chapter 2

Background

2.1 Monte Carlo Geometry Queries

A Monte Carlo geometry kernel must provide robust support for the geometry queries shown in Figures 2.1 - 2.6.

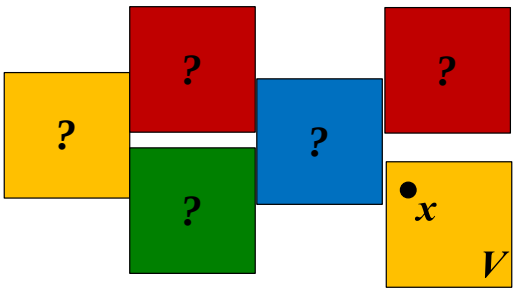


Figure 2.1: **Point Containment:** Given a set of volumes and particle location, **x**, determine if the point is inside, outside, or on the boundary of a volume.

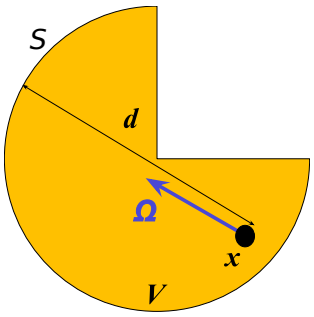


Figure 2.2: **Next Surface:** Given a volume, **V**, particle location, **x**, and particle trajectory, **Ω**, determine the next surface, **S**, of the volume that the particle intersects with along **Ω** and the distance, **d** to that intersection.

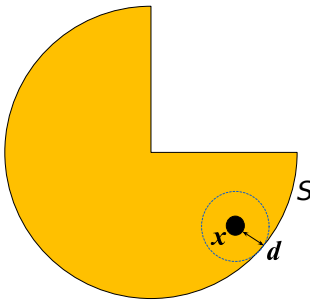


Figure 2.3: **Closest Surface:** Given a volume, **V**, and particle location, **x**, determine the distance, **d**, to to the nearest surface, **S**, of the volume in any direction.

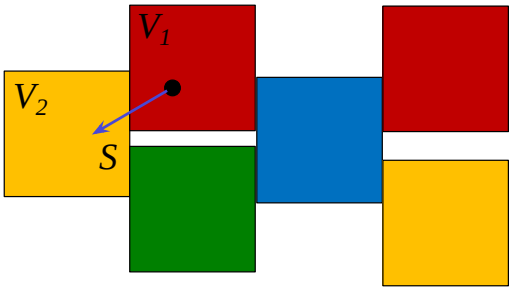


Figure 2.4: **Next Volume:** Given a particle in volume, $\mathbf{V_1}$, and crossing a surface, $\mathbf{S}$, determine the correct adjacent volume, $\mathbf{V_2}$.

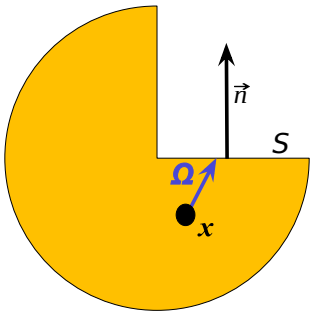


Figure 2.5: **Surface Normal:** Given a surface, $\mathbf{S}$ and particle location, $\boldsymbol{x}$ determine the normal vector, $\vec{\mathbf{n}}$, of the surface where the particle crossing occurs.

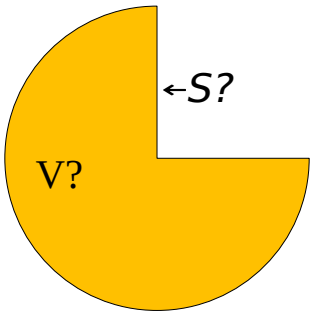


Figure 2.6: **Measure:** Given a volume, ***V***, or surface, ***S*** in the geometry, determine properties of that entity such as the volume or area.

Failure to accurately satisfy geometry queries or calculate geometric values can result in biased simulation results. Once such failure mode is lost particles during simulation. Lost particles occur when the geometry kernel cannot determine the particle's logical position, or which cell/volume contains the particle, based on it's numerical position. Lost particles remove statistical information from the simulation, and may influence results if particles are not uniformly lost throughout phase space. Inaccurate measurement of volume or surface area can result in incorrect values derived from simulation results, such as surface current density tallies or k-effective values in criticality simulations.

2.2 Analytic Geometry Representations

This section contains a discussion of common analytic geometry representations which are often used as native representations of Monte Carlo geometry.

2.2.1 Implicit Surfaces

An implicit surface is a multivariate function defined over an R^3 domain onto the one-dimensional space, R:

$$\Omega(R^3) \rightarrow R \tag{2.1}$$

These geometric representations are a rich and versatile representation of closed manifolds used for modeling, simulation, and rendering. Implicit surfaces are defined using the isocontour of a scalar function defined over all space - unlike an *explicit* representation of a surface which defines the subset of space which the boundary occupies. Intuitively it might seem wasteful for a definition to be true for all space considering the relatively small amount of space the object will occupy, however a number of powerful tools for geometric modeling using these representations will be discussed in this section.

An isocontour of this function with the value, ν, can be described as:

$$\Omega(\vec{x}) - \nu = 0 \tag{2.2}$$

For simplicity, the boundary of an implicit surface is conventionally defined as the isocontour for which $\nu = 0$. As a result, any point inside of the surface will have a negative value while any point outside of the surface will have a positive value.

Unlike their explicit counterparts, implicit representations allow complex topologies of surfaces to be integrated into a single representation. This is in part because the function is defined for all space, allowing them to naturally represent the merging and separation of disparate volumes. These properties allow for straightforward representation of dynamic surfaces such as fluids, though this is not yet of concern in the area of radiation transport. In practice, implicit surfaces are often used to re-sample the model into some other proxy for the geometry or render models via

ray tracing. Additionally, implicit surfaces can be used to generate triangle meshes for rasterization or rendering on GPUs [45] and can also be constructed from arbitrary triangle meshes or point clouds [46]. Implicit surfaces are well-suited to these applications due to the integrated geometric properties that can be quickly recovered from their analytic forms.

Geometric information needed for visualization and simulation can be readily recovered from implicit surface representations. For example, a common operation in particle transport is the determination of its containment by a volume in the model. A quick evaluation of the implicit function for this point will indicate its containment by the sign of the function. Additionally, the distance to nearest intersection with the surface from any point in space can quickly be determined via the definition of a signed distance function which is formally defined as:

$$d(\vec{x}) = \min(|\vec{x} - \vec{x_I}|) \tag{2.3}$$

$$\Omega(\vec{x}) \text{ s.t. } |\Omega(\vec{x})| = d(\vec{x}) \tag{2.4}$$

$$d - \text{distance function} \tag{2.5}$$

$$\vec{x}_I - \text{surface interface} \tag{2.6}$$

These forms of implicit surface functions can be modified or selected such that the following conditions are met:[†]

[†]The sign convention shown here is opposed to the sign convention commonly used for interior and exterior points in space. This sign convention is arbitrary and has been reversed for clarity in radiation transport where physical phenomena are simulated in the interior of objects rather than the space between them in rendering applications.

- $\Omega(\vec{x}) = d(\vec{x}) = 0$ for all x on the surface boundary
- $\Omega(\vec{x}) = d(\vec{x})$ for all x inside the surface boundary
- $\Omega(\vec{x}) = -d(\vec{x})$ for all x outside the surface boundary

Implicit surfaces are often used in time-dependent simulations due to their natural extension into a fourth dimension ($\Omega(\vec{x}, t) - \nu = 0$) and in turn their support for moving boundaries and changing topologies. Dynamic geometries are not yet of concern for CAD-based Monte Carlo work, but specialized forms of implicit surfaces are used as native geometry representations in most Monte Carlo simulation codes.

2.2.2 Constructive Solid Geometry

Native Monte Carlo geometries are commonly formed from a standard set of well-behaved implicit surfaces known as general quadratics. Constructive Solid Geometry (CSG) representations combine these surfaces using Boolean operations to form more complex objects as shown in Figure 2.7.

It is possible to construct complex geometries using CSG, but, as mentioned in Chapter 1, the interface for this work is typically text-based. This format makes defining complex volumes a tedious and time-consuming task. Detecting problems with the geometry definition is straightforward for the same reason that particles can be robustly tracked through the analytic description of the surfaces. Fixing undefined regions of the geometry or detecting invalid volume definitions is more difficult however.

The detection of intersections and particle containment queries in CSG geometries is computationally inexpensive for volume definitions constructed from a small number of surfaces, but, due to the logical combinations of surfaces used to create volumes, the number of evaluations necessary to satisfy these queries is linear with the number of surfaces in

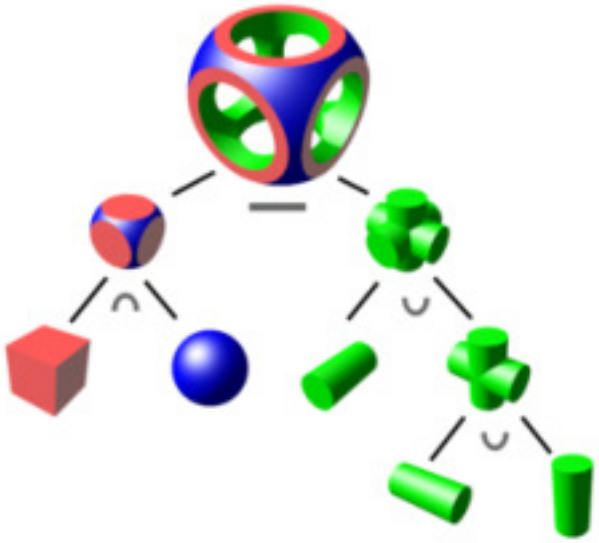


Figure 2.7: An example of how CSG volumes are created using Boolean combinations of objects. The union of the three orthogonal cylinders is subtracted from the intersection of the box and sphere on the left to form the final volume at the top of the figure.

the definition. For sufficiently large and complex models, it is not uncommon for volumes with many surfaces to be artificially separated by planes to create multiple volumes with fewer surfaces in their definition.

Visualization of CSG models is also somewhat limited. Because native formats for CSG differ greatly between each Monte Carlo code, each typically provides its own geometry visualization tools. These tools are commonly restricted to 2D images of the model representing a user-specified slice through the geometry.

2.3 CAD Geometry

Computer-Aided Design (CAD) systems allow for efficient and accurate representation of geometrically complex domains. An example such a

model is shown in Figure 2.8. Models can be created using interactive visualization tools represented in 3D space with a rich tool set for volume and surface creation. These creation and modification tools along with the immediate visual verification of a user's work reduces human error in model generation and design iteration.

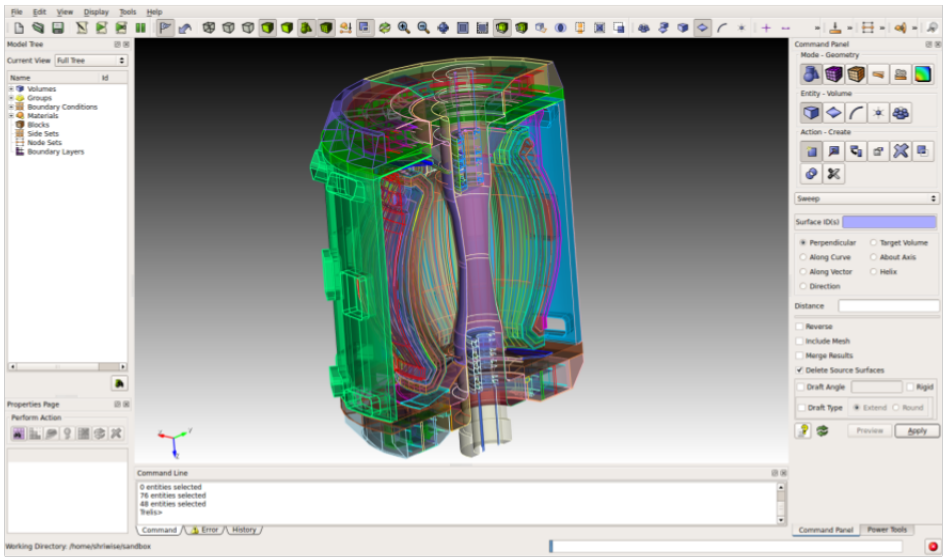


Figure 2.8: The Fusion Neutron Science Facility (FNSF)[29] model displayed in the CUBIT geometry and mesh generation toolkit [10].

In addition to reducing human error and effort, CAD models provide a common domain for analysis in other engineering domains such as fluid dynamics, heat transfer, and structural engineering. This shared domain enables ease of parametric studies and iterative design in coupled physics simulations. CAD engines also provide the ability to represent free-form or higher-order surfaces. The use of representations like splines, Bezier curves, and subdivision surfaces allow for accurate representation of these arbitrarily complex forms which would be impossible to accurately represent using CSG.

2.4 CAD-Based Monte Carlo Radiation Transport

2.4.1 Translation-Based Methods

One approach to CAD-Based MCRT is to leverage the native geometry capabilities of Monte Carlo codes by translating CAD geometries into CSG representations. In this approach, the fidelity of the geometric representation is limited to the geometric primitives available in the supported Monte Carlo code. Most codes support only first and second order surfaces, so higher-order surfaces of the CAD model must be approximated to accommodate the limited representation of the native code. As a result, many of these methods will simplify single volumes into smaller, less complex volumes able to be represented in CSG. This can be problematic when tally information related to CAD entities is desired - surface or volumetric fluxes for a single entity in the CAD model must now be mapped to several entities in the CSG representation. The lack of one-to-one correlation between CAD and CSG entities makes model manipulation difficult in that a change to the CAD model may result more or fewer CSG entities after translation, adding difficulty to tracking tally mapping. Several translation approaches have been developed oriented toward the Monte Carlo N-Particle (MCNP) input format developed at Los Alamos National Laboratory [64]. A few of these tools are described below:

MCAM: CAD support for the Super Monte Carlo (SuperMC) code, developed at the Chinese Academy of Sciences, is provided by the Monte Carlo Automatic Modeling (MCAM) tool [63, 65]. It supports both CAD-to-CSG and CSG-to-CAD conversion, relying on the ACIS [5] geometry engine. It is capable of fixing gaps and overlaps in models and supports geometry creation. Volumes are decomposed into simplified,

convex volumes during CAD-to-CSG translation.

McCad: McCad was developed at the Karlsruhe Institute of Technology in Germany and supports both CAD-to-CSG and CSG-to-CAD model conversion, relying on the Open CASCADE modeling engine to do so [13, 23]. Void spaces in the model are automatically filled in McCad. McCad also decomposes single volumes into multiple volumes with simplified CAD definitions.

Though other translation-based methods exist, these tools were selected for discussion based on usage in analysis and development efforts at the time of this work.

2.4.2 Direct Transport Methods

In the direct approach to CAD-Based MCRT, geometric queries are performed directly on the CAD geometry. Unlike translation-based approaches, physics codes must be modified to direct geometric operations to the CAD-Based interfaces. Some direct methods perform queries on analytic CAD representations while others use discretized forms, but in either case no simplification of the geometry is involved. Direct computation on CAD geometries tends to be slower than using native geometry because particle tracking is more computationally expensive on complicated surface representations. Several recent implementations of direct methods are listed below:

SERPENT: Serpent [32] provides CAD-based transport using a geometry represented by individual stereo-lithography (STL) files or a ABAQUS mesh [4] generated using CUBIT [10]. It is expected that the volumes

in these files do not occupy the same space and that each file contains a single, closed volume of triangles. Geometries can also be defined using both CSG and STL representations in this format [50]. The use of separate files to represent volumes independently in this system makes interchanging parts in a geometry straightforward, but ensuring that these parts do not have gaps or overlaps can be a difficult task for users.

MCNP6: MCNP6 [21], developed at Los Alamos National Laboratory, has added a native capability for transport on a volumetric mesh of unstructured tetrehedra. Thus a conformal mesh can be generated using a CAD model for transport in this mode to achieve good geometry fidelity. Generating a geometrically constrained volumetric mesh is difficult (and sometimes impossible) for arbitrary geometries, however. Non-conformal meshes can also be used where tetrahedra overlapping volume boundaries will contain a homogenized mixture of the materials in the mesh element, but the effects of this approximation are difficult to quantify and depend greatly on the importance of the materials homogenized and mesh element characteristics.

DAGMC: The Direct Accelerated Geometry Monte Carlo (DAGMC) software toolkit enables MCRT directly on CAD geometries [51] using surface tessellations to represent volume boundaries. DAGMC was developed at the University of Wisconsin - Madison and has been coupled with many Monte Carlo codes including MCNP (see Table 1.1). DAGMC relies on CUBIT (and its commercial counterpart, Trelis[1]) for model importing, cleaning, and tessellation. Surface meshses are stored in the Mesh Oriented dAtaBase (MOAB) [52], developed at Argonne National Laboratory. CUBIT and Trelis features are used to ensure volumes

do not overlap before discretization, and preprocessing steps ensure models are topologically watertight before transport[47].

Though each of the above methods has its merits, the work of this thesis has been implemented in DAGMC. DAGMC has demonstrated robust transport capability on models of varying geometric complexity for a number of physics applications. As mentioned above, it accomplishes this by discretizing CAD surfaces into sets of triangles representing surface boundaries. Volumes are then defined by all sets of triangles which represent bounding surfaces of a given volume. This surface mesh and the geometric relationships between sets of mesh elements, also known as Meshsets, are stored in MOAB. These relationships, which preserve the geometric topology, are stored in a hierarchical structure within MOAB, relating volumes to their surfaces, surfaces to curves, and curves to geometric vertices. For the purposes of this work, only the relationship between volumes and surfaces are of concern. A depiction of the relationship between Meshsets in DAGMC geometries can be seen in Figure 2.9.

It is important that topological relationships of the geometry-based Meshsets are maintained to accelerate certain geometric queries on the surface mesh. For example, *Next Volume* (see Figure 2.4) queries are accelerated by using these relationships to directly determine which volume a particle is passing into upon crossing a surface. In CSG, a surface crossing can require a series of point containment checks for each volume to update the logical position of a particle in another volume. Other queries become more complicated, however, due to the sheer number of triangles needed to properly define volumes with detailed features.

Next Surface and *Closest Surface* (see Figures 2.2 and 2.3) geometry queries, for example, can be computationally expensive for volumes often composed of hundreds of thousands or even millions of triangles. A con-

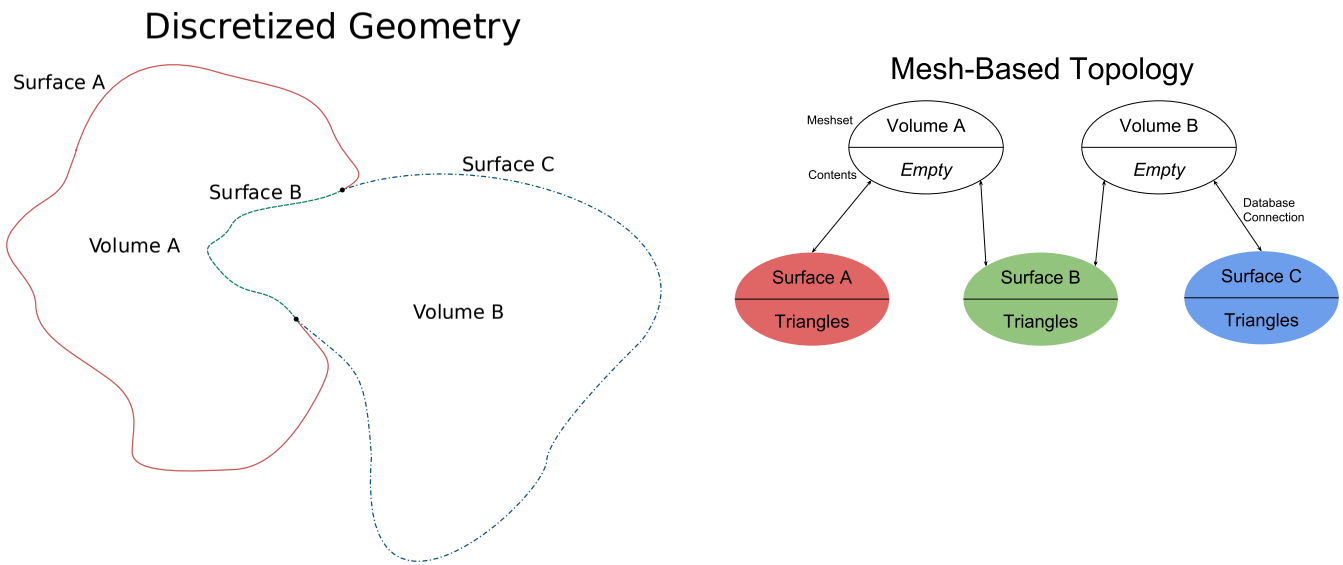


Figure 2.9: Left: Partial representation of a discretized DAGMC geometry. Right: A representation of the mesh-based hierarchy used to maintain topological information about the CAD geometry in MOAB.

venient way to conceptualize geometric queries on triangulated surfaces or volumes is to consider an equivalent CSG representation constructed using a planar surface in place of each triangle in the DAGMC model. The structure imposed by the Boolean combinations used to define such volumes require that each surface be checked for an intersection with the particle trajectory, resulting in a somewhat naive search for the nearest intersection. This intersection can then be used to determine the location of surface crossings.

The problem of finding a surface intersection for a given particle location and trajectory for a set of geometric primitives is a well-researched problem in the field of ray tracing. In this field, data structures designed to accelerate the location of the nearest ray intersection are used to render animations and images in real time.

2.5 Ray Tracing Acceleration Data Structures

Acceleration data structures for ray tracing are designed to rapidly narrow the search for an intersection in 3D virtual space given a starting position and trajectory used to construct a ray. This is accomplished by partitioning the space and associating geometric primitives, usually triangles, with that bounding partition. A search is performed by checking for an intersection with this bounding partition. If the ray does not intersect with the partition, then the set of primitives contained within that partition can be removed from the search. If the ray does intersect with the partition, then the set of associated primitives must be checked for intersection. Because a single separation into two spatial partitions is often insufficient to increase search efficiency, this partitioning process is performed recursively. The result is a tree, or hierarchy, in which partitions at the top of tree are associated other partitions, known as child nodes, rather than primitives. Partitions at the bottom of the tree are known as leaf nodes and are directly associated

with geometric primitives representing the geometric boundary.

The search for a ray intersection then becomes a traversal of the tree in which the children of the root node are checked for intersection. If an intersection is found with one or both of the nodes, then the corresponding child nodes are checked for intersection as well. This process is repeated until leaf nodes are reached at which point primitives are checked for intersection. A simplified version of this method can be found in Algorithm 2.1.

Any primitives underneath a node with which the ray does not intersect can immediately be rejected as a possibility for a hit. This allows many primitives to rapidly be removed from the search, limiting the number of primitive intersection checks to a small number compared to checking each primitive individually. This technique reduces the algorithmic complexity of the search from a brute force, or linear $O(N_{primitives})$, search to a logarithmic $O(\log(N_{primitives}))$ search.

```
def intersect_ray(tree_root, ray):

    stack.append(tree_root)

    while not stack.empty():

        node = stack.pop()

        if is_leaf(node):
            primitives = get_node_primitives(node)

            for primitive in primitives:
                d = primitive_intersect(primitive, ray)
                if d < dist: dist = d

            continue

        if node_intersect(node, ray):
            children = get_node_children(node)

            stack.append(children)

return dist
```

Algorithm 2.1: General algorithm for spatial hierarchy traversal to return the nearest intersection along a ray.

The remainder of this section reviews heuristics, bounding constructs, and data structure relationships used in rendering and scientific simulation today. Next, two splitting heuristics used to build these data structures, the Entity Ratio Heuristic (ERH) and Surface Area Heuristic (SAH) are discussed. After that, several common spatial hierarchies are described including the KD-Tree, Bounding Volume Hierarchy (BVH), and the octree. The remainder of this chapter describes CPU architecture characteristics which allow adaptations of these hierarchical data structures to achieve improved performance. This description includes a commentary on the relevance of specific design elements in these accelerations to MCRT.

2.5.1 Splitting Heuristics

There are two critical components that go into the creation of spatial partitions in ray tracing hierarchies. The first is the selection of a candidate splitting plane which is used to separate entities into one partition or another. The second is the evaluation of the "cost" of that split. This cost is a purely relative measure used to compare candidate splits for selection of a best split curing construction. Because there is no way to know exactly how expensive or inexpensive the cost of a split will be for the particular simulation at hand, heuristics are used to estimate this cost and determine the optimal splitting plane using limited information about the local nodes in the tree. More specifically, this information is typically limited to the number of primitives being split, bounds of the parent partition, and bounds of the candidate child partitions. Split costs are compared to the relative cost of forming a leaf node as well. If the cost of all splits is larger than that of the current node, then it is declared a leaf node and the build process continues in other areas of the hierarchy.

A virtually infinite number of planes could be tested to find the optimal plane for dividing the entities between the child bounding volumes, but even if one were to encounter such a splitting plane, it can be difficult to identify the plane as such without more knowledge about the final tree structure. As a result, a limited set of planes is tested for the best split based on a set of assumptions about the problem at hand and the heuristics being used to evaluate split costs. The most common method for split plane candidates is median plane splitting in which the current partition is split in half along each axes of the current bounding volume. Splitting plane costs are then evaluated and the split with the lowest cost is selected. Other methods for plane selection exist, but will not be discussed here as this work is more focused on traversal performance.

Two heuristics will now be discussed - the Entity Ratio Heuristic (ERH) and the Surface Area Heuristic (SAH). The ERH uses the resulting number

of primitives in each child node to determine the cost of a split. The philosophy behind this heuristic is to maintain the expected $O(log(N_{primitives}))$ cost of a ray traversal by ensuring that the number of primitives are split as evenly as possible from parent to child node. A form of this heuristic is presented in Equation (2.7). The ERH cost is unit-less and bounded by zero and one. This heuristic provides a finite limit on the expected cost, and makes it possible to set both an upper and lower bound as both an unacceptably high and a "good enough" cost, respectively.

$$C = \frac{|P_R - P_L|}{(P_R + P_L)} \tag{2.7}$$

$C -$ final cost evaluation
$P_L -$ primitives contained by the left child
$P_R -$ primitives contained by the right child

Figure 2.10: An example of the ERH calculation for a binary hierarchy.

The SAH applies spatial information as well as division of primitives to the cost evaluation. Its full form is found in Equation (2.8). The SAH uses the surface area of candidate child partitions relative to the parent partition's surface area as an approximation for the probability that the children will be visited after the parent volume. This evaluation relies on the assumption that rays in the problem are globally isotropic. The explicit form of the surface area heuristic was introduced in 1987 by Goldsmith and Salmon [20] and later formalized by MacDonald and Booth in 1990 [34].

$$C = C_t + \frac{SA_L}{SA_P} P_L C_i + \frac{SA_R}{SA_P} P_R C_i \tag{2.8}$$

C_t – cost of traversal to child nodes
C_i – cost of primitive intersection check
SA_L – surface area of left child
P_L – primitives contained by the left child
SA_R – surface area of right child
P_R – primitives contained by the right child
SA_P – parent node's surface area

Figure 2.11: A form of the surface area heuristic for a binary tree.

For the general case, ERH has not proved to be as effective as the SAH [9], but as seen in Chapter 5 it is a useful tool in correcting the surface area heuristic for triangle mesh features of a specific type. This scenario will be discussed further in Chapter 5.

2.5.2 KD-Trees

The KD-Tree or multidimensional binary search tree was originally developed as an acceleration data structure for querying records in databases and has since found use in other applications including speech recognition, global information systems, and ray tracing [8]. KD-Trees operate by using single values to represent divisions in a dimension of the problem space. A different dimension is split in each level of the tree, and the process is recursively repeated until a sufficiently small number of records or entities exist within the resulting partitions of a split, resulting in leaf nodes of the tree.

This data structure is also commonly applied to virtual 3D space for ray intersection queries. A different dimension of space is split in each

level of the tree just as is done in the context of an arbitrary database with different indices of data. The values used for separation of entities in the tree now represent a coordinate of a plane in that dimension; entities are sorted to either side of that plane to perform a split. First, the problem space is divided evenly in the x dimension. The two child partitions are divided along the y axis and the resulting children of this division are subdivided along the z axis. Divisions are typically selected in such a way that the largest extent of the current candidate is bisected.

It is possible that by doing this primitive entities are divided by the plane as well. There are a couple of ways in which this problem is addressed. The first is to simply reference any intersected primitives in both of the children. This means that some primitives may be checked for intersection more than once, but requires no changes to the original model. The other solution is to divide the primitive entities using the partition plane. This solution requires alterations to the model which may be undesirable under certain conditions and violates the description of the KD-Tree as a pure querying structure by altering the model.

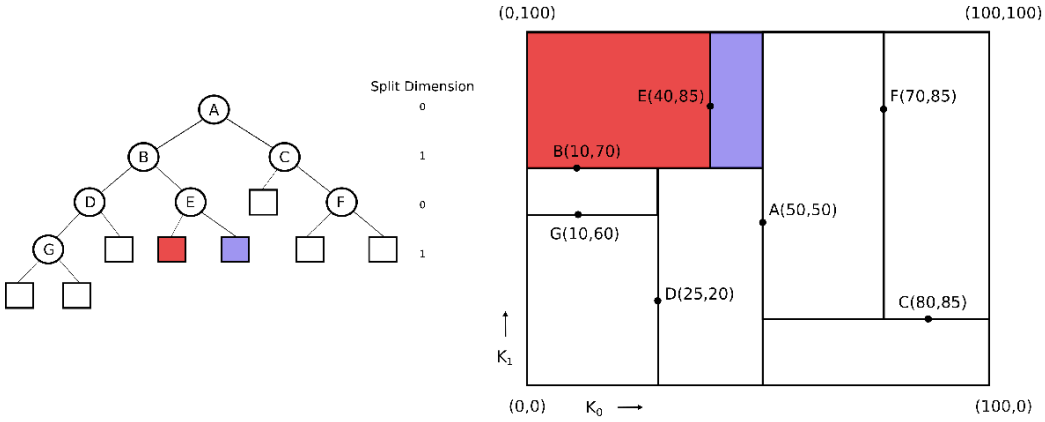


Figure 2.12: Depiction of a two dimensional KD-Tree. Left: Graph representation of the KD-Tree with boxes representing leaf nodes. Right: Two dimensional space partitioned in the graph. Boxes represent the range of their respective sub-tree nodes. (Adapted from Bently et. al. [8])

KD-Trees are able to perform highly efficient spatial searches due to traversal schemes developed based on the inherent characteristic of KD-Trees' non-overlapping sibling nodes. As a KD-Tree is being constructed, an ever-shrinking bounding box is being defined as one moves deeper into the tree. At the leaves of a KD-Tree, a well-resolved bounding box can be conceptualized using the coordinates of the last six splitting planes visited and possibly the domain boundary. The conceptual construction of this box is one way to move from node to node in a more efficient way than a more standard depth-first approach shown in Algorithm 2.1. The partitions whose planes are used to construct this conceptual box can be linked to the current partition in order to maintain a spatially localized search within the hierarchy. These links are referred to as neighbor links and, as shown by Samet et. al.[44], can be used to significantly reduce traversal costs in the KD-Tree. After a leaf is visited, neighbor links can be used to direct the traversal to either the next adjacent leaf or a nearby interior node in the tree, thus avoiding a depth-first style traversal in which the next step upon visiting a leaf node is to return to the root node of the tree and continue. By using these links to move directly to nearby leaf nodes, unnecessary shallow and mid-level tree traversal steps can be avoided.

KD-Trees are frequently cited as being able to provide the best ray tracing performance to date for certain geometries [16, 24, 27]. In particular, KD-Trees are noted as being better equipped to handle models with highly varying triangle sizes/densities. In practice, KD-Trees also tend to be very deep, taking a long time to construct, and can consume relatively high amounts of memory compared to other acceleration data structures, however.

2.5.3 Bounding Volume Hierarchies

A bounding volume hierarchy (BVH) uses a series of linked bounding volumes to enclose sets of geometric primitives. The uppermost node of the hierarchy, or the root, is a bounding volume which contains all primitives in the tree. These primitives are then recursively split into subsets which are bounded by their own volumes deeper in the tree until leaf conditions are met.

The initial concept of using the bounding volume construct as a pre-check for ray-intersection with CSG objects was introduced by Weghorst in 1984 [61]. Weghorst explored the possibility of using bounding spheres and bounding boxes to contain geometric objects. This work also went so far as to create a hierarchy out of the object-based bounding volumes, noting the importance of hierarchically joining bounding volumes near to each other in space so as not to have parent volumes containing large amounts of empty space.

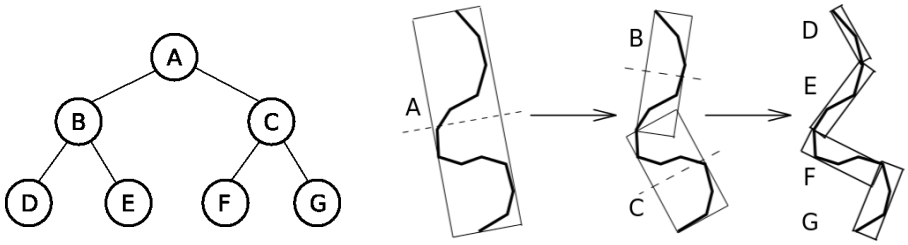


Figure 2.13: Depiction of a two dimensional BVH example adapted from Gottschalk 1996 [22]

In Weghorst's exploration of sphere and box bounding volumes it was found that while spherical bounding volumes are not as computationally expensive to check for ray intersections compared to bounding boxes, the latter generally provide a tighter fit to the objects they contain. This decreases the chance of wasted ray-volume intersection checks for rays which intersect the bounding volume but not the object it contains. When

considering the application of bounding volumes to a discretized analytic surface represented by a triangle mesh, this becomes more important as BVHs become deeper and more ray-volume intersection checks are performed to reach leaf nodes. Even applied to analytic objects, this effect was reflected in the results of Weghorts's work strongly enough to show that bounding boxes provided better performance in accelerating the ray intersection process than bounding spheres.

Two forms of BVHs are commonly applied in ray tracing problems: Axis-Aligned Bounding Boxes (AABBs) and Oriented Bounding Boxes (OBBs). AABBs are boxes whose orientation is restricted such that their faces are parallel to the problem's global coordinate axes. Given a set of points to contain, aligned boxes are straightforward to construct. Their simple representation results in a relatively low memory footprint and computationally inexpensive ray-intersection tests. Unlike AABBs, the axes of OBBs are allowed take any orientation relative to the global axes in order to enclose their set of primitives as tightly as possible. Due to the freedom in orientation, OBBs are able to bound sets of primitives at least as tightly as AABBs, resulting in fewer unnecessary intersection checks during the hierarchy traversal. Several robust methods for determining the orientation of a box for best fit to a set of primitives have been developed [22, 38]. OBBs are better for avoiding superfluous ray-box intersections that might otherwise occur for an AABB. They also more quickly conform to the full set of enclosed primitives as the boxes are recursively divided. By orienting their axes with the local set of primitives they are bounding, candidate splitting planes, usually selected in the reference frame of the parent node's oriented axes, are more effective at separating primitive entities and reaching leaf conditions quickly. This leads to more shallow hierarchies making the worst-case number of intersection tests lower than for an AABB hierarchy on average. While a shallow hierarchy might indicate a smaller memory footprint, OBBs require one to store some extra

information about their orientation relative to the global axes making this assumption difficult to prove consistently.

One disadvantage of using OBBs is that the ray-box intersection check requires an extra step in transformation of the ray to the oriented axes of the box in question at each level of the hierarchy traversal. The information needed for transformation of the ray basis must be applied to the ray before the box intersection can continue as it would for an axis aligned box. Thus, for a given ray query, an OBB hierarchy may have fewer intersection checks to perform than an AABB hierarchy, but the intersection checks are inherently more expensive than in the case of OBBs. In practice, AABBs are commonly used in BVHs for their simplicity of implementation and well-researched ray intersection algorithms. Other reasons for this preference will be discussed later in Section 2.5.5.

There are multiple approaches to constructing a BVH around a set of geometric primitives, but only "top-down" approaches will be discussed here. A top-down approach begins with the construction of a single bounding volume enclosing all primitives which will be part of the tree. At this point, child boxes of this root node are created by selecting a splitting plane for the box which divides the primitives contained by the current bounding volume into two subsets. This process is then repeated until leaf conditions are met. The selection of candidate splitting planes and the selection of a final plane for splitting based on its estimated worth can greatly affect the performance of the data structure.

One difficulty that BVHs face is that of overlapping sibling bounding volumes. Overlapping sibling volumes can cause additional box intersection checks in a similar manner to loosely fitting bounding volumes. If a ray passes through a region of overlapping sibling volumes, this causes the children of both boxes to be checked despite the fact that the desired nearest intersection will be found in only one of those boxes. Overlaps are difficult to avoid, however, due to the reality that volumes are required to

contain discrete elements, not just a section of the virtual space. Simply put, if the splitting plane of a bounding volume goes through one of the geometric primitives, there will be an overlap in the resulting child bounding volumes. Overlaps of sibling AABBs are typically limited to the size of perhaps one or two geometric primitives whereas overlaps between sibling OBBs are more difficult to characterize as they may overlap regardless of the splitting plane used. This inefficient characteristic of BVHs can be exacerbated by the structure of triangulated objects the BVHs are being formed around. One such feature which will be addressed in Chapter 5.

The spatial BVH variant (SBVH) was introduced by Stich et. al. in 2009 [49] with an additional complexity to the node splitting step. As candidate split planes are considered, triangles (or geometric primitives) are duplicated and contained in both resulting nodes. As a result, box boundaries do not need to be re-calculated. They can be created directly by dividing the parent bounding volume using the selected splitting plane. If a ray incident on one of the duplicated primitives misses one of the boxes, the sibling box will be intersected and the correct primitive intersection location will still be found. This method grants much more freedom when considering how a node should be partitioned. The relatively simple application of this method is performed by considering both splits in which triangle duplication is prohibited and splits in which it is allowed. In the scenario for which triangle duplication is prohibited, the set of candidate planes is equivalent to that of a standard BVH building algorithm. In the case where reference duplication is allowed, the search for a splitting plane is much more open - as previously mentioned. In fact, the search becomes closely aligned with the search for a spatial split as might be found in KD-Tree construction. The optimal splitting plane is then selected via a comparison of the cost for all candidate split planes - spatial or "traditional". Secondary heuristics are used to limit reference splitting in an effort to manage the data structure's memory footprint. The result of

the SBVH is a hierarchy which can be traversed just like any other BVH but with significantly reduced sibling volume overlaps. The SBVH results consistently show significant improvement over other methods [49].

In summary, bounding volume hierarchies are favored in the field of ray tracing for their relatively low memory footprints and well-developed parallel building schemes while providing high performance ray tracing capabilities. These features are of great import for systems with limited memory, such as GPUs, and applications with intent for real-time viewing or interaction. These data structures are particularly performant for *Next Surface* intersections and are currently the most commonly employed acceleration data structure for ray tracing. They are relatively simple to implement for the performance they provide and have smaller memory footprints relative to other ray tracing acceleration data structures, making them attractive to memory-limited environments such as GPUs.

2.5.4 Octrees

Octrees are a partitioning scheme in which cuboid bounding boxes, also known as voxels, are used to partition the 3D problem space into eight octants defined by the global axes and extents of the parent voxel. These eight voxels are then linked as children of the parent voxel. This process is repeated recursively until leaf conditions are met in which a sufficiently small number of primitives is contained within the current voxel. This spatial subdivision technique is commonly used to efficiently index data in 3-D space [19]. Octrees are somewhat like KD-trees in that their divisions are purely spatial, their partitions contain no overlaps, and the placement of entities in nodes occurs in a similar manner to that of KD-trees, but each node is represented by a closed bounding volume as in a BVH.

Octrees can consume a large amount of memory relative to the data structures previously discussed in this chapter. It is often possible that voxels may be completely devoid of underlying entities. There are typically

many voxels containing no primitive references but may be required to exist as part of the data structure, depending on its design. This results in many voxels being stored in memory which aren't useful other than to verify that the space they contain is empty. Additionally, geometric primitives may be referenced multiple times if they intersect multiple nodes thus increasing the required memory storage with the same consequence as seen in KD-Tree traversal with the possibility of a primitive being checked more than once upon traversal. The memory footprint is mostly of concern in applications for visualization on GPUs - though specialized methods for octree applications on these architectures do exist.

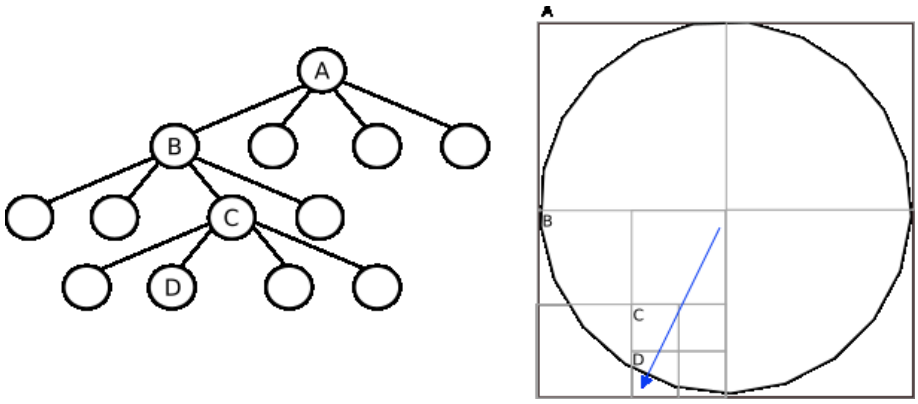


Figure 2.14: Depiction of a two dimensional octree example of a top-down ray fire traversal for a simple geometric object.

One advantage of octrees is the regular nature of the partitions. The value of a node in hierarchies such as these (or in the BVH for that matter) lies in its ability to remove candidate space from the query, yet a voxel can only accomplish this if rays strike the voxel. The result is that one measure of a voxel's value can be described by the ratio of its probability of an intersection check to the space it will exclude from the query search, represented by its volume. In a problem with a uniform ray distribution, the probability of a ray to intersect a given voxel can be related to a voxel's surface area as seen in the SAH. Thus cubic voxels have the most favorable

ratio possible based on their geometry. The uniformity of voxel properties provide a predictable nature of the octree which is advantageous when traversing the data structure as well.

The predictable size and location of any given node in the tree determined from the root node properties also provides a fast look-up of the deepest node in the tree containing a point in space. This is helpful in providing a starting point for ray traversal which is deeper than the root node, allowing one to potentially avoid some traversal steps in the shallow levels of the tree. Additionally, octrees have non-overlapping nodes which allows for efficient traversal schemes similar to the neighbor linked traversal done in KD-trees. These traversals are conceptually similar to that of the KD-tree's but typically employ some form of Morton encoding to determine which node in the octree the ray should visit next [43]. Other traversal techniques allow the octree to avoid creating and traversing nodes containing empty space which can significantly reduce its memory footprint in cases where internal nodes of the tree aren't required to define some spatial dataset as mentioned above [44]. These methods are often applied in GPU environments due to the limited memory available there. Octrees are often used to store spatial data fields as well and naturally provide a higher resolution of the field near boundaries of volume as the voxels become smaller in that region which can be seen in Figure 2.14.

As mentioned above, octrees are known for having large memory footprints compared to other acceleration data structures, but they can also be used advantageously for a combined purpose if a problem requires the storage of one or more well-resolved data fields near volume boundaries as well as the capability for ray tracing.

2.5.5 Architecture-Based Acceleration

This section focuses on the impact of CPU architecture evolution on ray tracing data structure design and implementation. More specifically, it

emphasizes the advantages of using vectorization-oriented implementations or Single Instruction Multiple Data (SIMD) programming in the field of ray tracing.

When considering the problem of parallelism in computing, programmers focus on one of two areas: *functional parallelism* or *data parallelism*. Functional parallelism describes the method of performing multiple operations in parallel on many processors while data parallelism describes operation on multiple data sets at the same time on a single processor. SIMD is a form of data parallelism in which, as the name indicates, the same set of numerical operations are performed on multiple sets of data in parallel under a single CPU process. Chip-sets with support for SIMD instructions became very popular in the mid-1990s as home PCs became more common and demand for multimedia-related performance increased. In response, many CPU manufacturers at the time such as Intel, IBM, and Motorola began to release products with SIMD instruction sets, the most powerful of which was Intel's Streaming SIMD Extensions (SSE). Over time, CPU clock speed became the larger focus of many manufacturers as dramatic gains in processor speed took precedence in the field. As processor speed increases began to wane, pushing the limit of current cooling technology in the early 2000's, a new shift toward multi-core designs occurred. Currently, as CPU clock speeds remain somewhat steady in multi-core systems, a focus on single-thread performance via SIMD has reemerged. Newer SIMD instruction sets such as Intel's Advanced Vector Extensions (AVX or AVX2) have doubled the width of operable data, allowing for a theoretical doubling of performance in codes relying on SIMD instructions [26].

SIMD execution has found use in many different areas including medical imaging, financial analysis, database management, computer visualization, and physical simulation. As is the case in any problem well-suited to parallel programming, all of these applications perform the same set

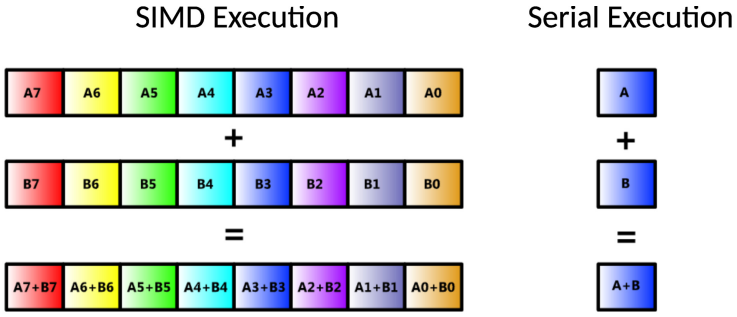


Figure 2.15: Concept of data parallelism using SIMD. Adapted from Intel's documentation on the advanced vector extensions (AVX) instruction set. [30]

of computations many times on similarly structured sets of data. This situation arises quite often in applications related to modeling or and visualization of virtual space. One indicator of a problem which would benefit from parallel programming is the presence of a few common sets of operations done many times or in a recursive manner. Thus, traversal of ray tracing data structures is well suited for SIMD operations as it relies heavily on the performance of a few key operations: ray-node intersections and ray-primitive intersections. The ability to perform intersection checks of many nodes at once or many triangles at once has clear benefit when satisfying geometric queries in simulations or renderings which may require several billion such operations. Several demonstrations of ray-tracing data structures adapted to take advantage of SIMD-enabled optimization on modern CPUs have already been developed.

An early implementation of SIMD used to intersect a ray with many triangles at the leaf nodes of a KD-Tree was performed by Hurley in 2002 [27]. This implementation demonstrated a significant improvement in ray-primitive intersection performance and established many significant observations about the utilization of SIMD commands within ray tracing applications. Despite the fact that the cost of primitive intersection checks was reduced, most of the time spent satisfying the ray query was spent in

traversal of the hierarchy to the leaf nodes. This is true of a well-formed BVH and is verified for DAGMC in Figure 1.1. Noting that the number of triangles in the KD-Tree's leaf nodes were small in comparison to the SIMD registry width, Hurley described two ways in which to further exploit data parallelism of SIMD in ray tracing.

One method is to traverse and intersect multiple rays at the same time. This is refereed to as the N:1 approach. The other is to intersect many nodes with a single ray which is referred to as the 1:N approach. An important characteristic for success of the former method is that the group of rays being intersected has very similar traversal paths through the hierarchy so they may be grouped together in a packet for a narrow traversal path. This property is known as often described as ray coherence. Branching off of Hurley's work, Wald demonstrated that rays can be effectively grouped into packets and traversed in a binary space partitioning tree (a modified KD-Tree) to achieve CPU performance equal to that the high-end graphics hardware of the time [58].

As more realistic physical effects are being applied in rendering applications today (such as light-scattering surfaces, smoke effects, or fog), rendering ray paths become less coherent. This means that the same primary rays will not necessarily follow similar paths through the model or its underlying acceleration hierarchies. Due to this lack of ray coherency, the 1:N approach to data parallelism in which one ray is intersected with many hierarchy elements in a single step has been revisited. Wald wisely observed that taking advantage of SIMD operations in traversal of a KD-Tree is difficult due to the nature of the partition. He goes on to state that the KD-Tree's superior serial performance in comparison to that of serial BVH implementations drove reluctance to move away from the KD-Tree and resulted in the establishment of ray packets, or the 1:N approach [57]. Both Wald and Dammertz [14], concurrently presented implementations of SIMD enabled traversal and primitive intersection on multi-branching

BVHs in 2008 with N:1 approaches. Both implementations showed impressive performance enhancements, ranging anywhere from 3-10 times faster than the baseline ray tracing kernels used for comparison.

Both Wald and Dammertz approached the construction of multi-branching BVHs in the same manner. Each built a standard binary BVH using the adjusted SAH cost analysis in Figure 2.17 with median plane splitting. They then collapsed the tree by directing child nodes to their ancestors to achieve the desired branching ratio. Wald opted to use a more exotic, graphics-oriented, architecture with Intel's Larrabee and was able to apply a branching ratio of 16 to their BVH while Dammertz used a branching ratio of 4 using Intel's Streaming SIMD Extensions (SSE). A higher branching ratio provides higher SIMD utilization and more shallow hierarchies, but Wald conceded that for common CPU-architectures branching ratios between 4 and 8 would be optimal for most common architectures.

AABBs were used in both Wald and Dammertz's implementations. While OBBs have been shown to conform more quickly to the underlying geometry and can generate more shallow trees as seen in Figure 2.16 , AABBs are generally more favorable in SIMD implementations. First, OBBs require more information to be stored. This extra information can limit how many nodes will fit into a single SIMD step and it is often more beneficial to check more AABBs than fewer OBBs despite the tighter fitting to geometric primitives. This is in part because AABBs boxes have faster ray intersection tests without the re-orientation of the ray information to the box coordinates, but also because more nodes can be fit into the SIMD register to be visited at once. Secondly, though OBB hierarchies are more shallow than their axis aligned counterparts', tree depth is of less concern due to the higher n-ary structure of the trees used in these implementations.

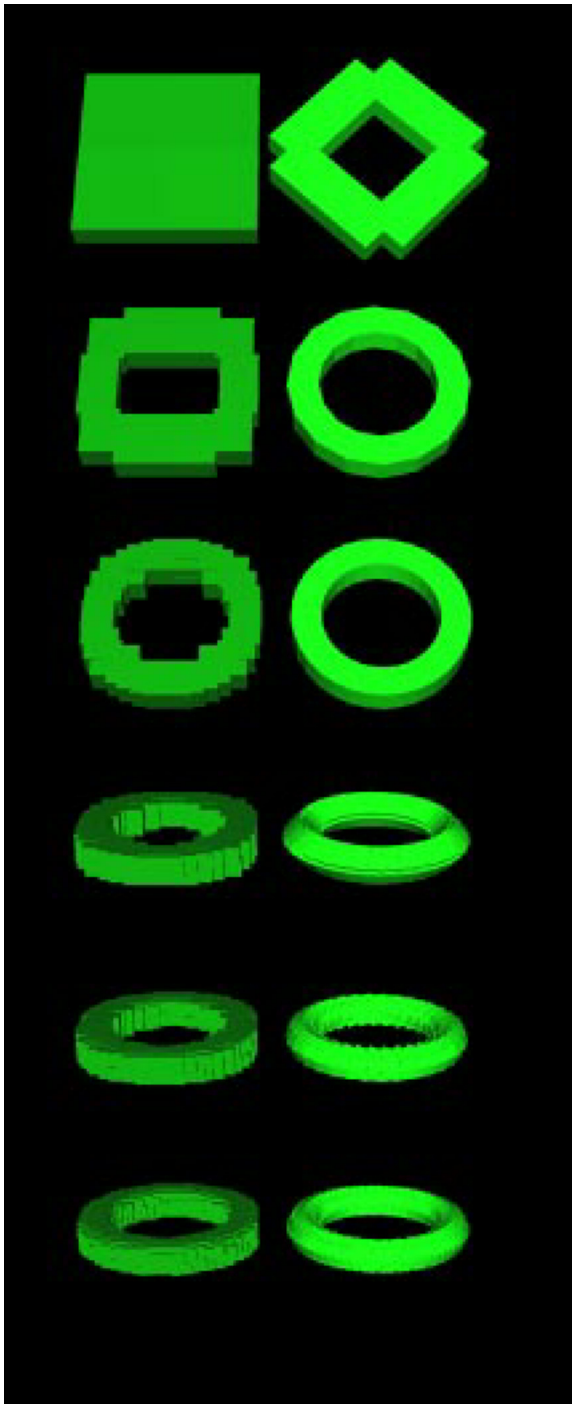


Figure 2.16: A figure replicated from Gottschalk's paper on OBBs exemplifying OBB's rapid conformity to a torus volume (right) in comparison to AABBS (left).

$$C = C_t + \sum_{k=0}^{K} \frac{SA(B_k)}{SA(B)} \frac{|P_k|}{T} C_i \tag{2.9}$$

$K-number\ of\ desired\ children\ per\ interior\ node$
$T-number\ of\ triangles\ in\ SIMD\ register$

Figure 2.17: An adjusted form of the surface area heuristic for an n-ary BVH branching ratio as presented by Wald in [57]. †

For completeness of all ray tracing data structures discussed in this chapter, SIMD implementations of octrees were sought out in literature, but none were found. This is likely due to the fact that SIMD registers on common architectures wide enough to accommodate 8 nodes are not yet common. It is also worth noting that while the KD-Tree is restricted to a binary hierarchy, another variation, the bounding interval hierarchy, might be compatible with higher branching ratios and thus SIMD traversal of the hierarchy [56].

2.5.5.1 Application in Embree

Many of these concepts have been applied in a ray tracing kernel called Embree. Embree is the result of an effort to produce a performant CPU-based ray tracer as a demonstration of the expanding capabilities of modern CPU architectures [60]. In both construction and traversal of its BVHs, Embree takes advantage of many of the latest developments in BVH research and using modern chipset architecture capabilities via vectorization at an implementation level. The combination of these effects leads to a very powerful ray tracing tool in terms of performance, as demonstrated by the many projects which have incorporated Embree as their production ray tracing kernel such as Corona, FluidRay, and Brighter3D [3, 12, 17].

†Some notation has been modified to agree with notation used in Figure 2.11

As a result of its success in other areas, Embree was selected for application in DAGMC to satisfy geometric queries for attached physics kernels. Several critial design features of Embree will be reviewed here and their importance to efficient BVH traversal discussed.

2.5.5.1.1 Memory Alignment For the CPU architectures used in this work, Embree's quad-tree implementation was applied. In order to use this design in a SIMD context, the code constructs must be carefully designed for interpretation by the CPU. Embree defines BVH-related structures such that they occupy aligned memory spaces, meaning that they are both compact in memory usage and can be read into CPU registers in a predictible manner. All bounding volumes and the underlying Cartesian vector types used to define them are specified as structures aligned to 16 bytes in memory. This is important for encoding of node types and leaf sizes, discussed in Section 2.5.5.1.2, as well as the ability for these types to be operated on as either arrays of floats during BVH construction or as native SIMD types during BVH traversal.

2.5.5.1.2 Node Type Encoding Nodes in Embree's BVH are stored compactly using an encoding scheme which contains the memory address of box information as an integer value. Because nodes are byte-aligned, memory addresses will be offset by the same amount as the node alignment. For example, if nodes are memory aligned to 16 bytes, the four least significant bits in the integer representations will be unused. These bits are used to encode information about both leaf nodes and empty nodes. Empty nodes required as part of the data structure but don't contain primitive entities. Figure 2.18 depicts the bits of a leaf node for reference.

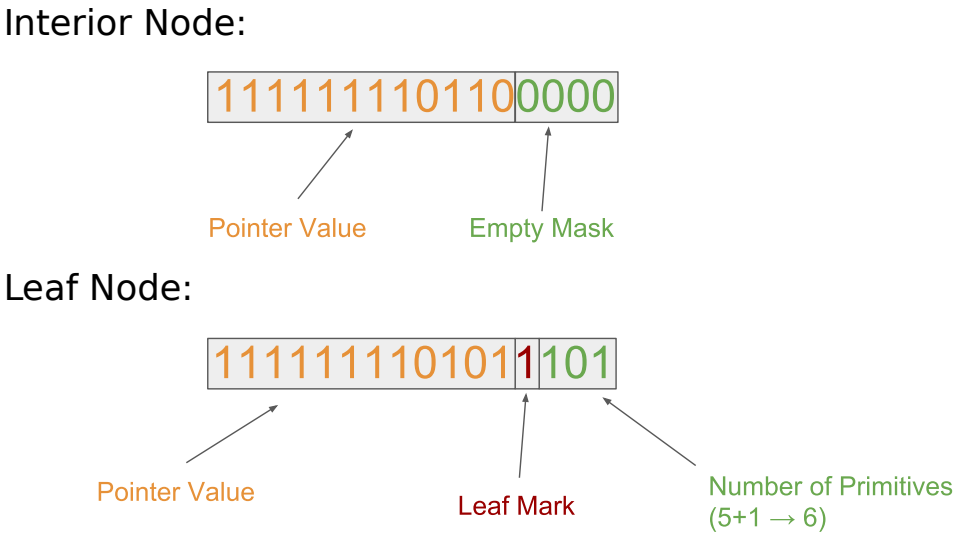


Figure 2.18: Visual representation of leaf encoding using integer-based pointer values. Yellow digits represent bits used to store the address of the first primitive reference in the tree. The red digit indicates the fourth least significant digit which is used to indicate that the stored memory address points to a primitive reference rather than another node reference. The final three green digits indicate the number of primitive references in the leaf node.

This representation reduces memory storage of leaf nodes, which can be considerable given that the number of leaf nodes in a n-ary tree is

$$L = n^h \tag{2.10}$$

L—number of leaf nodes in the tree

n— n-ary of the tree (branching ratio)

h—height of the tree (zero-indexed)

meaning a quad tree with a height of ten will contain over one million leaf nodes. Avoiding storage of the memory address and leaf node size separately reduces memory consumption of the tree and allows the use of fast bit-wise operations to differentiate node types when traversing the tree.

2.5.5.1.3 Memory Prefetching While CPU clock speeds are getting faster each generation, memory latency is often the limiting factor in the performance of many algorithms (see Figure 2.19). To account for this latency, modern CPUs allow compilers and code writers to initiate asynchronous requests for memory access before it's needed for computation. This technique, known as memory prefetching, avoids memory latency by populating low level memory caches with data before it is required by the CPU.

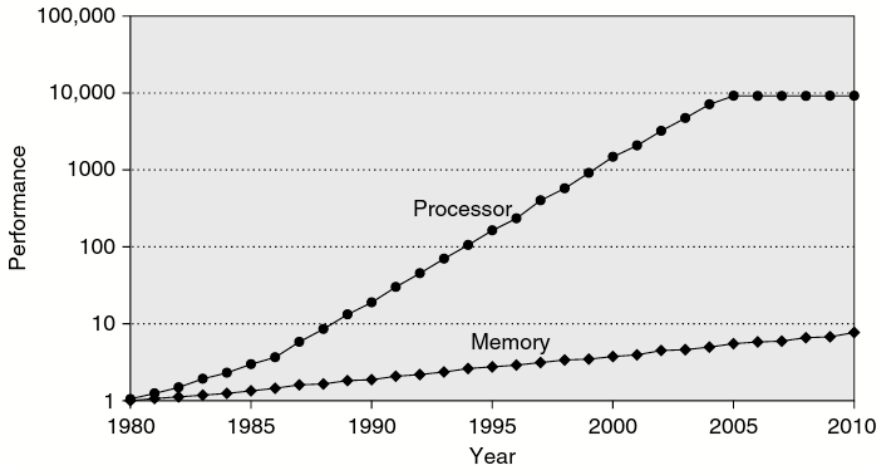


Figure 2.19: Trends in CPU vs. memory performance over time. The processor line shows the increase in memory requests per second on average. The memory line shows the increase in memory accesses per second. [25]

Memory prefetching is applied automatically by compilers in situations

where the compiler can clearly determine the next blocks of memory needed by the CPU. Several common patterns are used by compilers to predict memory access including **Next-Line** and **Next-N-Line** prefetching. These patterns are often applied when looping over large arrays of data. Next-Line prefetching gathers information for the next iteration of the loop during the current iteration. Next-N-Line prefetching groups the loop iterations together to improve the timing between prefetching and computation for the CPU. Proper timing prefetch operations is critical. Memory fetched too early may bump information out of the CPU cache still needed by the current iteration's computational block, causing additional memory latency to re-gather that data. Memory fetched too late defeats the purpose of applying prefetching and can interfere with automatic memory access requests from the CPU. Examples of both Next-Line and Next-N-Line prefetching can be seen in 2.20. Modern compilers are becoming increasingly adept at predicting memory access for these cases and will apply prefetching as part of resulting instruction sets.

```
for(int i = 0; i < N; i++) {
  prefetch(&a[i+1]);
  prefetch(&b[i+1]);
  sum += a[i]*b[i];
}
```

(a) Next-Line Prefetching

```
for(int i = 0; i < N-4; i+= 4) {
  prefetch(&a[i+4]);
  prefetch(&b[i+4]);
  sum += a[i]*b[i];
  sum += a[i+1]*b[i+1];
  sum += a[i+2]*b[i+2];
  sum += a[i+3]*b[i+3];
}
```

(b) Next-N-Line Prefetching.

Figure 2.20: Example of manually implemented memory prefetching for a standard loop performing a vector operation. This type of pattern is often automatically implemented by modern compilers.

Memory prefetching can also be applied manually in situations where the user is aware of the next memory block to be used by the CPU as well.

One example of a user-informed prefetching method is **Pointer-Based** prefetching. In this method, memory locations pointed to by locally accessed data structures are known to be operated on soon by the CPU are prefetched. This method is commonly applied when traversing hierarchical data structures or linked lists. An example of Pointer-Based prefetching can be found in 2.21. The traversal of the BVH in Embree takes advantage of this particular prefetching pattern to reduce memory latency. Because of the indirection involved in traversal of data structures by pointer and the result-based operations at each node in the structure, it is very difficult for compilers to predict memory access patterns and apply prefetching automatically. Application of these Pointer-Based methods are nearly always applied manually.

```
StackArr stack(root_node);
prefetch(&root_node);

while(!stack.empty() {
  node = stack.pop();
  if(node.visit_left(val)) {
    prefetch(&node.left());
    stack.add(node.left());
  }
  if(node.visit_right(val)) {
    prefetch(&node.right());
    stack.add(node.right());
  }
}
```

Figure 2.21: Example of a prefetch pattern using pointers to follow a data structure traversal.

Embree applies Pointer-Based prefetching while traversing its BVH. As nodes intersected by the ray are added to the stack, the bounding box information for the node at the top of the stack is prefetched for intersection in the next step of the BVH traversal.

2.5.5.1.4 Ray-Box Intersection Tests In order to take advantage of compiler optimization and SIMD instructions, ray-box tests in Embree contain a no branching (no if-else statements) version of Kay's slab box intersection test [28]. Removing branching avoids additional memory fetching or re-organizing of CPU registers so that SIMD operations can be efficiently performed in sequence to determine intersection locations with the box along each dimension. Underlying SIMD computation calculates the intersection of a ray with several boxes at once without memory disruption. The distances to intersection along the ray for each box coordinate are then compared to find the near and far intersections with axis-aligned bounding box planes. If the largest of the near-side intersections is smaller than the smallest of the far-side intersections, then the ray strikes the box.

To perform this computation efficiently, some information is pre-computed and added to the ray structure including the inverse of the ray direction. These values are also duplicated into memory-aligned vector types with sizes equal to the n-ary of the BVH, again so that memory doesn't need to be allocated on-the-fly during traversal of the data structure. A bit-mask is used to indicate which boxes are hit and which are not, and the distance to each box intersection is returned so that the closest boxes can be added at the top of the traversal stack.

```
def ray_node_intersection(ray, node):
    # result - mask (0 - miss, 1 - hit)
    # inverse direction - ray.invdir
    # inverse direction times the ray origin - ray.org_invdir
    # tNear/tFar - parameter value of intersections along ray direction

    # compute tNear for N nodes
    tNearX = vecN(&node + ray.nearX) * ray.invdir.x - ray.org.x * ray.invdir.x
    tNearY = vecN(&node + ray.nearY) * ray.invdir.y - ray.org.y * ray.invdir.y
    tNearZ = vecN(&node + ray.nearZ) * ray.invdir.z - ray.org.z * ray.invdir.z

    # compute tFar for N nodes
    tFarX = vecN(&node + ray.farX) * ray.invdir.x - ray.org.x * ray.invdir.x
    tFarY = vecN(&node + ray.farY) * ray.invdir.y - ray.org.y * ray.invdir.y
    tFarZ = vecN(&node + ray.farZ) * ray.invdir.z - ray.org.z * ray.invdir.z
```

```
# compute maximum tnear for all boxes
tNear = max(tNearX, tNearY, tNearZ, ray.tNear)
tFar =  min(tFarX,  tFarY,  tFarZ,  ray.tFar )

# compute a mask indicating which boxes are hit
mask = tNear <= tFar

# set distances to intersection to optimize traversal (nearest nodes first)
dist = tNear

return mask, dist
```

Algorithm 2.2: A non-branching ray-node intersection test for a quad tree implementation with pre-computed near and far box coordinate values.

Further optimization is accomplished by pre-computing the near and far side coordinates based on the signs of the ray's unit direction. In this way, the near and far coordinates of the box are already known. Thus the near and far intersections do not require sorting along each box axis and cand be directly computed. To avoid branching and structure indexing in the ray-box intersection computation, byte offsets values are pre-set on the ray. These offset values are used during intersection tests to gather the correct coordinate values for the child boxes of a node in the hierarchy rather checking Boolean values and adding branches to the test. After computing the near and far side intersections, the maximum near-side and minimum far-side hit distances still need to be found, but skipping the comparison of values for each axis removes a considerable portion of computation from the test. An example of this method is shown in Figure 2.2. It is worth noting that this optimization is not applicable when using OBBs. The unknown orientation of the box axes makes any prediction about which side of the box will be intersected first by the ray impossible. In this case the ray-box intersection must include an additional step to sort the near and far side hits before checking for intersection.

2.5.6 Signed Distance Fields

Signed Distance Fields (SDFs) are commonly derived from implicit surface functions and variations on these functions are known as level-set functions [39]. Both offer a rich and versatile representation of closed manifolds used for modeling, simulation, and rendering. As discussed in Section 2.2.1, Constructive Solid Geometry (CSG) representations seen in native Monte Carlo codes are formed from Boolean combinations of predefined implicit surfaces at their core. While these predefined surfaces do not give the freedom of model creation and manipulation found in many CAD systems, important geometric information required for visualization and simulation can be readily recovered from implicit surfaces.

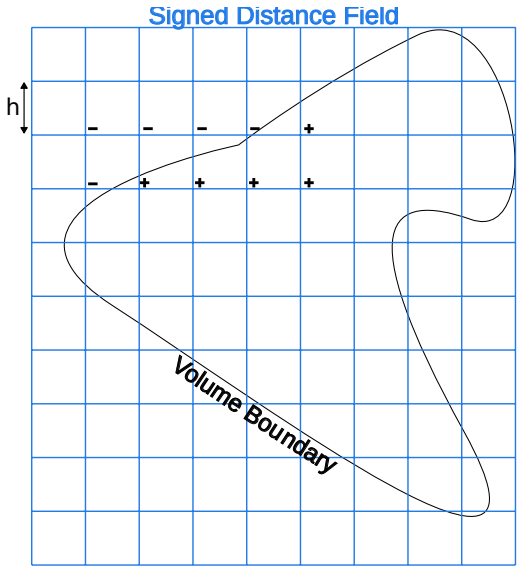


Figure 2.22: 2D visualization of a signed distance field surrounding an arbitrary volume boundary.

Signed distance field generation from implicit surfaces is a particularly valuable property of this geometric representation. A signed distance field, meets the following requirements for any vertex, v, in the total set of locations, V, in the field:

- $|SDF(v)| = min(|\vec{x} - \vec{x_i}|) \, \forall \, v \in V$
- $SDF(v) = 0 \, \forall \, v$ on the surface boundary
- $SDF(v) > 0 \, \forall \, v$ inside the surface boundary, and
- $SDF(v) < 0 \, \forall \, v$ outside the surface boundary

Implicit surfaces can be naturally extended to represent dynamic geometries by including a time dependence in the function, making them powerful tools for populating signed distance fields in simulation and rendering of fluids, smoke, fire, etc. To simulate these phenomenon, the data structure is populated with signed distance values for a given time in the rendering (see Figure 2.22). The signed distance field can then be used to determine point containment queries and approximate nearest surfaces values. It can also trace rays at any time via a method in which the ray length is repeatedly clipped using signed distance values to approach a surface in a process called ray marching [54].

2.6 Monte Carlo Memory Considerations

The bulk of Monte Carlo memory consumption in native calculations is attributed to nuclear cross-section data and tally data structures with the analytic CSG geometry representation taking up a small portion of the overall program memory.

2.6.1 Cross-Section Data

Cross-section data is necessary to represent the physical processes governing particle transport through the virtual geometry. Various formats of this data exist with both continuous and discrete representation of cross section information.

Continuous representations of this data can occupy a considerable amount of space and are more computationally expensive to evaluate than discrete representations, but are also more accurate. Discrete representations of this data consume more memory in simulation, but require less computation when extracting data. The amount of memory occupied by nuclear data is dependent on the number of unique materials used in the problem as well as the material composition in terms of the unique isotopes in the problem. This data can occupy the majority of simulation memory for large problems with many materials.

2.6.2 Tally Data Structures

Results of the Monte Carlo calculation are kept as tallies of particle contributions to physical quantities at various locations in the physical model. Tallies on surfaces or in cells change the memory usage of the simulation insignificantly, unless additional data is needed to calculate derived quantities. Mesh-based tallies on the other hand can dominate the memory usage. In MCNP and many other Monte Carlo codes, mesh tallies are defined by the user as structured grids in either Cartesian, cylindrical, or spherical coordinates. These tallies are often used for better spatial resolution of physical data in the simulation and for coupling to analysis in other engineering domains. Tetrahedral mesh tallies are also supported by several Monte Carlo codes DAGMC interacts with, including MCNP [64]. The size of these tallies vary based on the needs of the user and the trade-offs associated with spatial resolution of solutions and statistical convergence related to mesh element sizes.

2.6.3 Impact on CAD-Based Tessellations

For the Monte Carlo code used in this work, MCNP, the parallelism method applied is a master/slave scheme standard for many applications using the

Message Passing Interface (MPI) [18]. The master process is responsible for initializing the problem, determining the load balance for the number of parallel processes requested by the user, and collecting/accumulating information at the end of the simulation. In MCNP, particle histories are divided into equal segments among the slave processes for evaluation. More information on the method for particle history division and random number sequence assignment was detailed by Deng and Xie [15]. For each slave process, a corresponding set of the nuclear data, tally data structures, and geometry representation are created. Duplication of the geometry representation has a significant implications for the use of CAD-based tessellations.

The tessellations resulting from CAD geometries consume much more memory than corresponding CSG representations. This must be taken into consideration when planning parallel simulations on clusters where the limitation for the number of processes per CPU is often the memory used per process. Because DAGMC's CAD-based particle tracking is an addition to MCNP, it has no influence on the parallelism data model. Given that geometry representations are required to exist in each process of the parallel simulation, memory usage in DAGMC is closely monitored. Some of this memory usage is largely uncontrollable due to the reliance on CUBIT or Trelis' underlying tessellation algorithms, but additional data used to maintain geometric relationships and build acceleration data structures are kept to a minimum if possible.

Model	Tessellation (MB)	Tessellation and MOAB BVH (MB)	Simulation (MB)
FNG	92	213	262
ATR	294	765	865
UWNR	213	446	1250
nTOF	56	88	416
ITER	2592	5547	6679

Table 2.1: A summary of the memory usage for the performance benchmark models shown in Table 1.2 both with and without acceleration data structures.

Table 2.1 shows the memory usage for all of the benchmark models used to assess DAG-MCNP's performance relative to native MCNP geometry representations in Section 1.2. All of the models consume relatively low amounts of memory compared to the storage found on many CPUs in High Performance Computing (HPC) environments, but the addition of acceleration data structures, in this instance MOAB's BVH of OBBs, can more than double the memory occupied for the geometry representation. This effect becomes even more pronounced in larger production models seen in Chapters 4 and 5, though shared memory implementations for higher per-node memory efficiency are on DAGMC's development path.

In addition to the benchmark models from Table 1.2, Table 2.1 includes a model of a demonstration nuclear fusion device currently under construction in southern France. An analysis of ITER using DAGMC was performed at the University of Wisconsin - Madison to calculate a spatial mapping of the dose rate induced by irradiated components of the machine for various cooling times or times after shutdown of the device. This shutdown dose rate calculation was broken into seven spatial source meshes to avoid exceeding the memory limits of the HPC cluster used for this analysis. The values displayed in the table represent the memory usage for one of the seven simulations required to complete the analy-

sis. This model was included as a current representation of the analysis work being performed in DAGMC. It was not included in the comparative benchmark analysis of DAGMC because no native MCNP counterpart of this model exists. This model contains components created in CAD with no analogous MCNP representation, another reason CAD-based tessellations are desirable for MCRT. The simulation memory footprint of the ITER model is much larger in simulation, so much so that the number of processes allocated to a node was memory-limited rather than core-limited during analysis.

2.7 Summary

This discussion of the CAD-based MCRT-t via the DAGMC toolkit, hierarchical spatial data structures, SDF generation, and context for memory use in parallel simulations with MCNP in this chapter provides a basis for the work performed in the remainder of this document. Chapter 3 discusses the implementation and application of SDFs in DAGMC's framework. Next, Chapter 4 presents work on BVHs optimized for use in DAGMC and extensions of those methods to provide robust transport in production simulations. Chapter 5 discusses the effects of splitting heuristics and hierarchy construction surrounding problematic geometric features on the performance of DAGMC simulations.

Chapter 3

Signed Distance Field Preconditioning

This chapter describes the adaptation of a rendering data structure, the Signed Distance Field (SDF), as a tool for accelerating MCRT-t in DAGMC. A model for predicting the data structure's utilization in simulation is also introduced. Finally, demonstrations of its effectiveness for a number of simple problems and production models are shown and discussed.

Signed distance fields are typically represented using a mesh-based structure such as a Cartesian grid as depicted in Figure 2.22 or an octree. Associated with each vertex in the mesh is a signed value whose sign indicates its containment by the surface and whose magnitude represents the distance to the nearest surface of the geometry from its associated location. Signed distance values can be constructed for arbitrary locations within the mesh by interpolating the stored values of the vertices for the mesh element containing the point. These interpolated signed distance values can then be used to quickly provide local geometric information for Monte Carlo queries in simulation.

3.1 Preconditioning Theory

Of the geometric queries that DAGMC supports, *Next Surface*, *Point Containment*, and *Closest Surface* are most commonly called in simulations. At least one ray is fired to satisfy any of these queries in DAGMC with $O(\log N_{triangles})$ complexity using MOAB's BVH, but it is hypothesized that these queries can be accelerated in many cases. For each of the fundamental Monte Carlo geometries outlined below, signed distance values are recovered from the signed distance field via interpolation of the field values and used to avoid more computationally expensive ray fire calls in DAGMC.

3.1.1 Point Containment

Point Containment queries can be preconditioning by examining the interpolated signed distance value for the current particle location. If the point's value is negative (or outside of the SDF), then the point is considered to be outside of the volume. If the point's value is positive, then the point is determined to be inside the volume. Given that there is error associated with each of these interpolated values, the result of this method should only be trusted if the absolute value of the signed distance is greater than the expected error associated with the value. If this is not the case, then a ray must be fired to determine the particle's containment with respect to the volume in question. In effect, this verifies that the location is far enough from the boundary of the volume to make a definitive statement about whether it is inside or outside of the volume based on the sign of the interpolated value. Figure 3.1 graphically describes the outcome of these different cases in two dimensions.

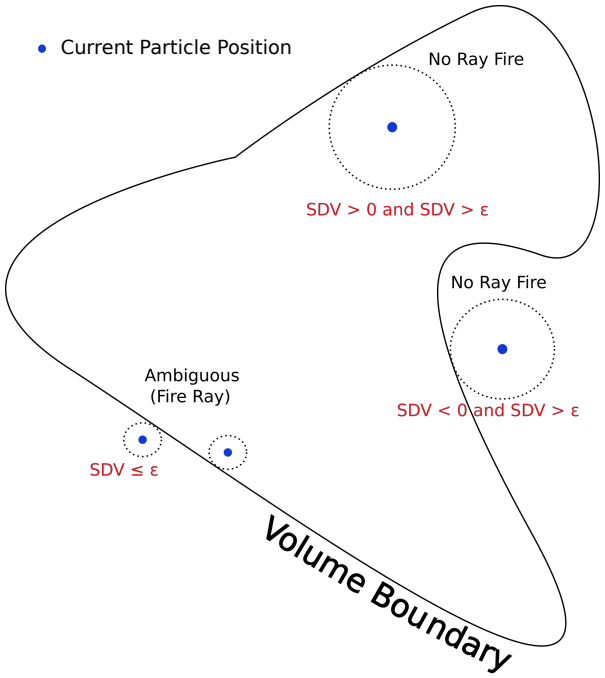


Figure 3.1: Examples of scenarios for *point containment* preconditioning using signed distance values. For locations far from the surface boundary, the sign of interpolated signed distance value can be used to determine point containment. Locations close to the surface with interpolated values less than the interpolation error, ϵ, cannot be used to determine point containment.

3.1.2 Next Surface

Next Surface intersection queries are the most common geometry query in Monte Carlo simulations as demonstrated by data in Table 3.2. These queries are initiated by native Monte Carlo codes to determine if a particle will cross a surface before reaching its next physics event location. Normally in DAGMC at least one ray fired each time this query is called. This can be avoided by using the signed distance field to exclude the possibility of a surface crossing without explicitly determining the

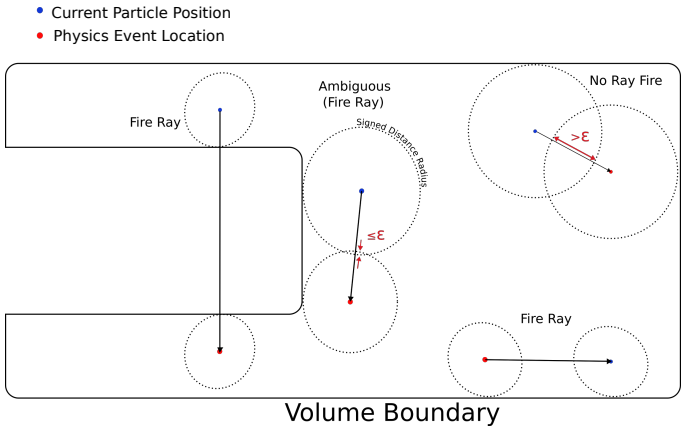


Figure 3.2: Examples of scenarios for next surface intersection preconditioning using signed distance values. Various scenarios are shown in which the signed distance values can be used to avoid a ray fire query by confirming the absence of a surface crossing between the particle's current location and physics event location.

next surface intersection. If the sum of the signed distance values for both the current particle position and the next physics event location is greater than the distance between the two locations, then it can be guaranteed that there is no surface crossing along the path between those two points. The interpolated signed distance value represents the minimum distance to any surface for a given location. The signed distance value of the current location ensures that some portion of the particle's track from the current location to the physics event location is open space. The signed distance value of the physics event location also ensures that a portion of the particle's track is empty space. If the entire track length can be accounted for, then no surface crossing exists between the two locations. Thus the particle can safely advance to the next physics event location without firing a ray to determine the exact distance to the next surface. Figure 3.2 depicts these different cases in

2D space.

For robustness, the error for each interpolation, ϵ should be subtracted from the sum of the signed distance values as a protective measure against invalid surface crossings. If the expanse between the particle's current location and its next physics event location cannot be accounted for by the signed distance values of the two points, then the exact next surface intersection will be found using a ray fire call. It is acknowledged that not all Monte Carlo codes provide the next physics event location along with the particle's current location to their geometry kernels. In this case, preconditioning of these queries in this manner will not be possible.

3.1.3 Closest Surface

Closest Surface queries can be performed in a similar manner to *Point Containment* queries, but they are more dependent on the native code's intent for their use. Some Monte Carlo codes always query for the closest surface intersection in order to determine whether or not the particle will exit the volume before reaching its next physics event location. This information can be interpolated from a signed distance field to the same effect.

In similar fashion to the point containment case, the signed distance value should only be trusted if it is greater than the error associated with the value. Additionally, the error should be subtracted from the value, returning to the code a conservative value for the nearest intersection. If the signed distance value's magnitude is not greater than its error evaluation or if the value is negative, then a ray should be fired to determine the exact location of the nearest boundary crossing for the particle's location. Figure 3.3 depicts these different cases in a 2D example.

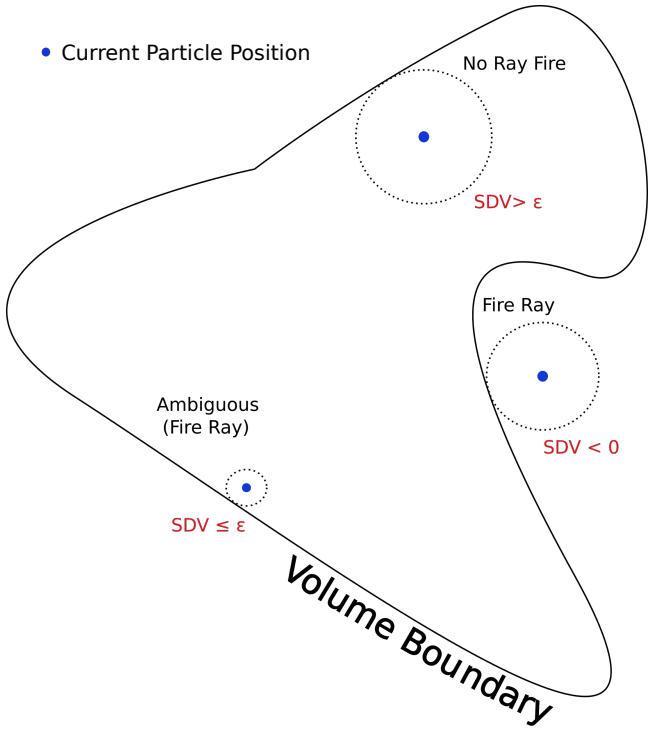


Figure 3.3: Examples of scenarios for closest surface preconditioning using signed distance values. Signed distance values interpolated from the SDF can be used to quickly return an approximate nearest surface location.

Using these methods, signed distance fields can act as a preconditioning tool for the relatively expensive ray fire process to accelerate CAD-based MCRT-t by using an $O(1)$ process to establish that the conditions of a geometric query are such that a more computationally expensive ray fire is necessary before performing the ray tracing operation. It is hypothesized that for some Monte Carlo problems this process can be used to avoid $O(\log N_{triangles})$ ray fire calls to significantly reduce the runtime of the simulation.

3.2 Implementation

3.2.1 Construction

As an initial implementation, one signed distance field is generated for each volume in DAGMC with extents matching the axis-aligned bounding box of the volume. The signed distance field is represented as a uniform structured mesh with a signed distance value at each vertex in the mesh as indicated in Figure 3.4.

MOAB provides an interface for construction of structured meshes which stores explicit vertex coordinates and hex elements. As with any entity stored in MOAB, data can be applied to these vertex elements. These vertex coordinates and hex elements can be accessed using $< i, j, k >$ indexing, where the coordinate $< 0, 0, 0 >$ and $< N_x, N_y, N_z >$ represent the lowest and highest corner of the structured mesh in parameter space for a mesh containing N_x, N_y, N_z elements in the x, y, and z directions respectively. This representation provides a fast path for verification, visualization, and proof of concept in transport test cases for demonstration, but is relatively memory intensive due to the dense nature of the data structure and explicit mesh element information stored in the database. To reduce memory consumption, an implicit version of the structured mesh has been implemented which takes advantage of the uniform step size in each dimension to store only: the number of elements in the mesh, the location of the lowest corner in that mesh, and a flat array of the signed distance values associated with the elements of the mesh. This avoids the storage of vertex coordinates, mesh element connectivity, and all associated handles to those entities. There is an added cost in re-computing vertex coordinates when a signed distance value is interpolated, but this added cost is negligible in comparison to a ray traversal and memory is of greater concern for this spatially dense data structure. This data structure can still interact with MOAB's structured mesh interface to generate an

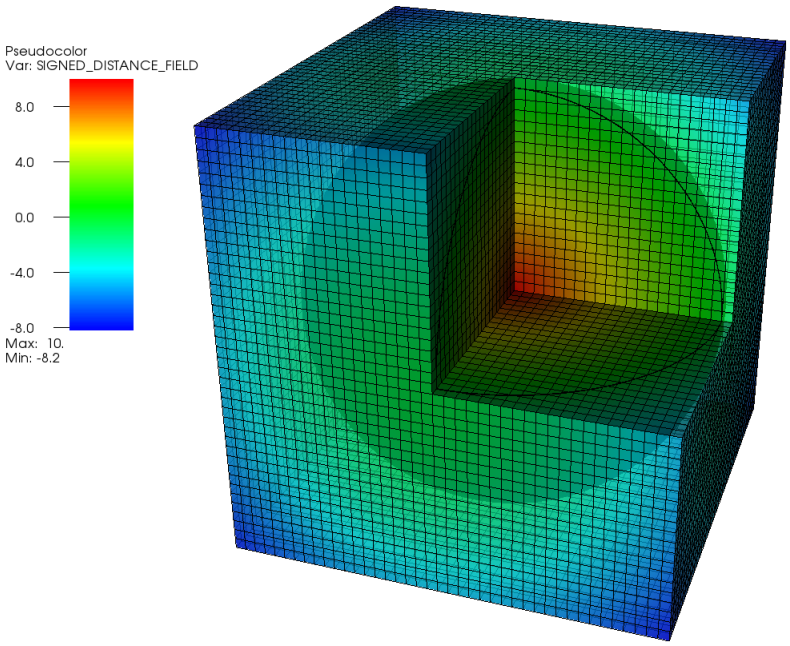


Figure 3.4: A visual of a signed distance field with step size 0.5 cm surrounding the spherical volume of test case with a radius of 10 cm.

explicit mesh for visualization and verification if required.

Signed distance values can be retrieved from the structured mesh by determining which mesh voxel the point lies within. The point's element is accessed by determining an $< i, j, k >$ index using the point's x, y, and z values divided by the structured mesh step size. A tri-linear interpolation of the mesh element's 8 vertex coordinates and their signed distance values are then used to provide the signed distance value for the location of interest.

3.2.2 Population

Signed distance fields are typically generated using an implicit, analytic representation, but a suitable data structure for populating the struc-

tured mesh with signed distance values is already in place in the form of DAGMC's bounding volume hierarchy. It is a more straightforward process to simply use DAGMC's current *Closest Surface* algorithm to generate signed distance values than to create an implicit surface approximation of the triangle mesh. This method also maintains a consistency between the intersections found by the ray tracing kernel and the values stored in the signed distance field.

DAGMC's *Closest Surface* algorithm returns, among other pieces of information, the nearest intersection location and the triangle on which this intersection exists. For each point in the signed distance field mesh, this algorithm is used to determine the magnitude of the distance value. To accomplish this, a ray is constructed from query location to the intersection location. The dot product of this ray vector with the triangle's outward normal vector is used to determine the sign of the distance value. DAGMC maintains enough information to consistently orient triangle normals such that they point outward from the volume they represent. In the rare cases for which the dot product of these vectors is ambiguous, or zero, DAGMC's point containment algorithm is used to disambiguate the value's sign.

3.3 Preconditioning Utilization

In effect, the preconditioner is attempting to check whether or not the particle will actually cross a surface before explicitly searching for the particle's intersection with a surface along its current trajectory. If the result of this preconditioning check is always false and a ray is always fired, then these checks only add to the computational cost of the problem. This will always occur in volumes filled with void, for example, as particles immediately travel from one side of a volume to another. To avoid low-utilization scenarios, the signed distance field should be applied selectively depending on each volume's geometric and material properties for optimal

performance and high utilization of the preconditioning methods. Ideally, this method will only be applied to volumes in which the preconditioning method is able to avoid ray fire calls often or with high utilization. The signed distance field is expected to have the biggest impact in performance when preconditioning *Next Surface* intersection queries, as they are most commonly called in Monte Carlo codes when tracking particles through the geometry (see Table 3.2). As such, this type of query is the focus of utilization measurement for the remainder of this section.

$$\mathrm{U} = \frac{\text{Rays Avoided w/ SDF}}{\text{Number of Geometry Queries}} \tag{3.1}$$

The utilization of the SDF preconditioning method, conceptually defined in Equation (3.1), can be described as the number of ray fire calls related to *Next Surface* intersection queries that are avoided divided by the total number of next surface intersection queries made by the Monte Carlo code. This value can be quantified using this definition using debugging tools, such as Valgrind [41], to determine the number of queries made in DAGMC and the number of rays fired inside of the subroutine. It is expected that, as the utilization of preconditioning methods goes up, the performance of the simulation will also improve.

To understand this utilization more deeply with respect to material parameters, a simple problem was create to demonstrate the performance benefits of this method. The density of a hydrogen-filled sphere of radius 10cm and centered on the origin was varied from 0 to 1 $\frac{g}{cm^3}$ with a 5 MeV neutron point source at the origin. For each density, one simulation was performed without the signed distance field and another with the signed distance field and preconditioning enabled. Figure 3.5 shows the utilization results of this study.

Utilization of the preconditioning method in this study remains high until the hydrogen density falls to 0.1 $\frac{g}{cm^3}$ at which point a distinct knee appears and the utilization falls off quickly. Even at the lowest density

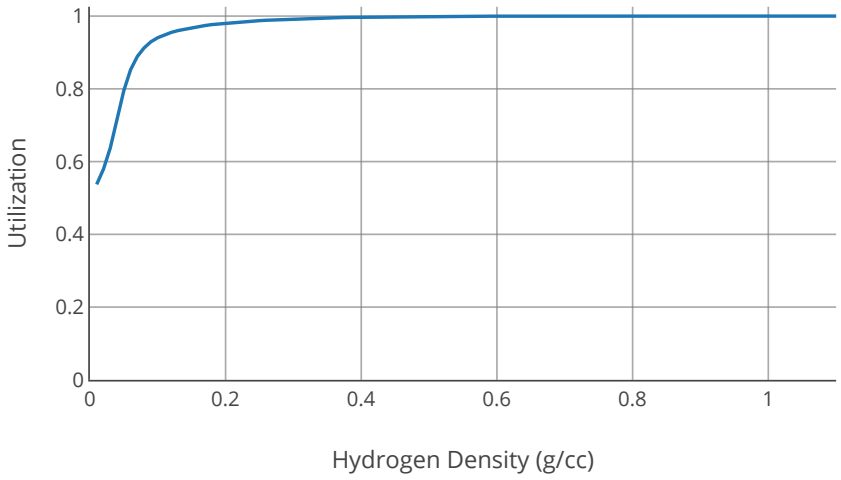


Figure 3.5: Utilization results for a 5 MeV neutron source at the origin of a 10 cm radius sphere. Hydrogen density was varied from 0 to 1 $\frac{g}{cm^3}$.

reached in the study of 0.01 $\frac{g}{cm^3}$, the utilization preconditioning method is 0.54.

Figure 3.6 provides some insight into the impact of the preconditioning method on the simulation. The run times of three implementations converge as the density of the hydrogen is varied. As the hydrogen density approaches 1.0 $\frac{g}{cm^3}$, a factor of 3.5 improvement in runtime is seen in the simulation where the SDF is applied as a preconditioner. The application of the signed distance field allows for significantly improved performance until the density drops below 0.1 $\frac{g}{cm^3}$ in agreement with utilization plot. As the material density decreases, particles quickly leave the geometry after very few collisions. It is difficult to judge the impact on the performance of this simulation for these low density values due to the limited

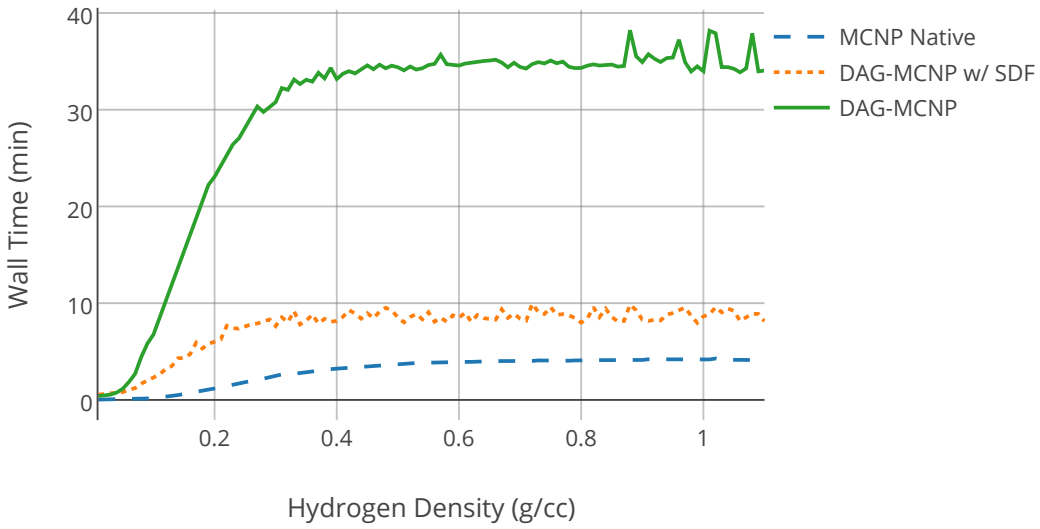


Figure 3.6: Performance results for a 5 MeV neutron source at the origin of a 10 cm radius sphere. Hydrogen density was varied from 0.0 to 1.0 $\frac{g}{cm^3}$. Simulations of 100M histories at each density were performed using native MCNP5, DAG-MCNP5 without the signed distance field, and DAG-MCNP5 with the signed distance field.

size of the geometry and the short lived histories. In order to have more control over a simulation's physical parameters, subsequent experiments were performed using a pseudo Monte Carlo simulation tool.

3.3.1 Utilization Modeling

In order to characterize utilization of a signed distance field as a preconditioner for *Next Surface* intersection queries in DAGMC, a pseudo Monte Carlo simulation tool was developed using DAGMC. This tool was used

to simulate different transport scenarios within a spherical geometry of radius 100 cm. Particles were sourced uniformly inside the volume and scatter isotropically. Particle histories are terminated based on a maximum number of collisions or departure from the problem geometry. Particle distance traveled, d, can be represented by either a fixed distance or by sampling for the standard probability of interaction in a medium with mean free path, λ. The tool allows the value of λ to be set directly, enabling a relation between the signed distance field and this value to be developed with intent for use as a means to characterize appropriate conditions for application of the signed distance field.

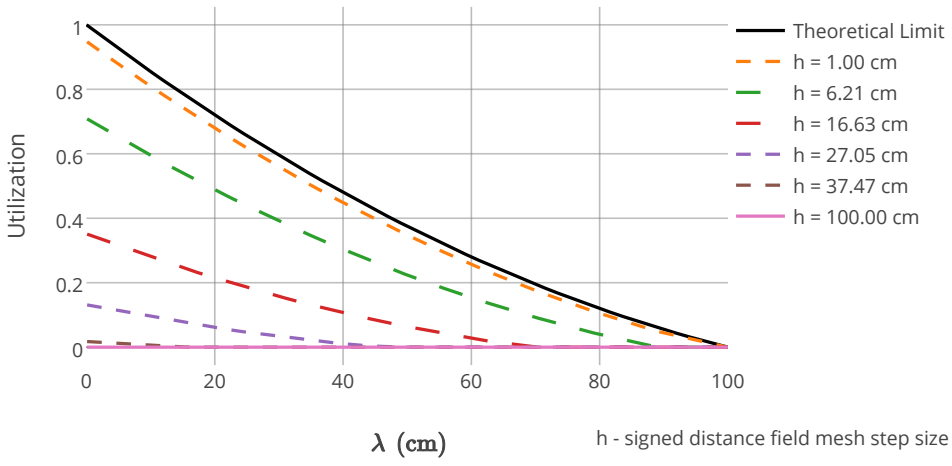


Figure 3.7: Results of the model for the theoretical utilization limit with the results of the simulation for a fixed distance traveled case.

To begin, simulations were performed for particles with a fixed distance traveled for varying distances and signed distance field step sizes.

Run times of the simulation are not shown here as the data structure's utilization is the main focus of this study. The results of the study are shown in Figure 3.7. As the signed distance field mesh step size increases, utilization of the preconditioning method structure decreases due to the increasing error associated with the interpolation of signed distance values. Additionally, utilization is expected to decrease with increasing distance traveled. This decreased utilization is caused by not only the increased distance between the two particles, but also by the increased probability that both locations will be closer to surfaces of the sphere and have smaller signed distance values. A theoretical limit developed by the author for the preconditioning method utilization is also shown in Figure 3.7. The development of this analytic limit will now be discussed.

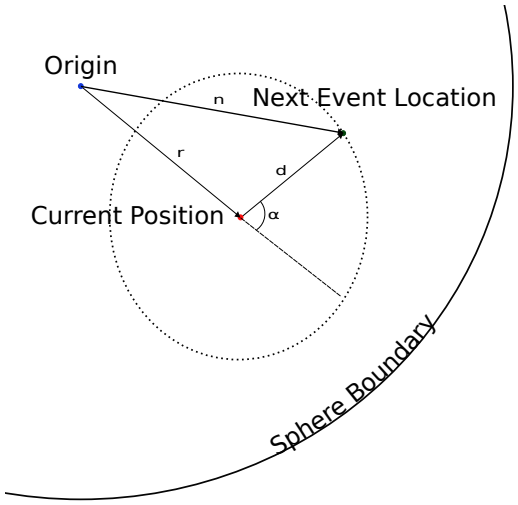


Figure 3.8: Depiction of utilization model variables.

3.3.2 Analytic Model Development

The utilization of the signed distance field as a preconditioner for ray tracing operations can be modeled as an evaluation of the combined probability space for particles with a current position, $\vec{p}$, and a next physics

event location, $\vec{n}$, some distance, d, apart. The fraction of this probability space in which signed distance values can be used to rule out surface crossings for next surface intersections is considered to be the theoretical utilization of the signed distance field. An general form for this probability space can found in Equation (3.2).

$$\int_{V_{sphere}} \int_{V_{track}} p_p(r) p_n(d)\, dV_{sphere} dV_{track} \tag{3.2}$$

In this model, the starting location of particles, $\vec{p}_p(r, \phi, \theta)$, is uniformly distributed, $p_p(r) = 1$, throughout a sphere of radius, R, centered at the origin. The location of the next event, $\vec{p}_n(d, \alpha, \beta)$, where d is the distance traveled by the particle, α is the interior angle between the particle's position vector and the particle's sampled direction vector, and β represents an azimuthal angle for directions traveled with angle of departure, α. Figure 3.8 depicts these variables, r, d, and α graphically.

In order to represent particles traveling a fixed distance, the relationship in Equation (3.3) is applied.

$$p_n(d) = \frac{\delta(d - \lambda)}{d^2} \tag{3.3}$$

The evaluation of this integral provides a representation of all the query space available to the problem

$$A = 8\pi^2 \int_0^R \int_0^\infty \int_0^\pi \delta(d - \lambda)\, r^2 \sin\alpha \, d\alpha dd \, dr \tag{3.4}$$

and represents all geometric query space, labeled A, for a sphere of radius, R and a fixed distance traveled, λ.

In order to understand what fraction of this query space is able to be preconditioned, the condition for avoiding an explicit nearest intersection search along a particle direction in Equation (3.5) will now be applied.

$$SDV(\vec{p}) + SDV(\vec{n}) > |\vec{p} - \vec{n}| \tag{3.5}$$

$$SDV - Signed\,Distance\,Value\,function$$
$$\vec{p} - particle's\,current\,position$$
$$\vec{n} - particle's\,next\,event\,location$$
$$h - mesh\,step\,size$$

This condition establishes that the nearest location to intersection for both points must be greater than the distance between the two points plus any error associated with their signed distance values as previously described in Section 3.1. This condition is true for some fraction of the next surface queries in a Monte Carlo simulation, but not all. The signed distance function of a sphere, shown in Equation (3.6), can be applied in Equation (3.5), to obtain a lower limit, α_{min} for the integral over $d\alpha$ in Equation (3.2).

$$SDV(\vec{x}) = R - |\vec{x}| \tag{3.6}$$

Making these substitutions into the inequality gives

$$\alpha_{min} > \arccos\left(\frac{(2R - r - d)^2 - d^2 - r^2}{2dr}\right) \tag{3.7}$$

This condition on alpha can be interpreted as a minimum interior angle that the particle's trajectory must take relative to the particle's position vector, $\vec{p}$, for a distance traveled, d, for a ray fire to be avoided and the preconditioner to be utilized. The examination of this condition as a function of the distance traveled for various values of r results in some conclusions about how signed distance values are being utilized.

The inequality in Equation (3.7) is undefined until the distance reaches a value $d = R - r$. This is because the angular limit only needs to be applied to areas of the query space in which the distance traveled is large enough

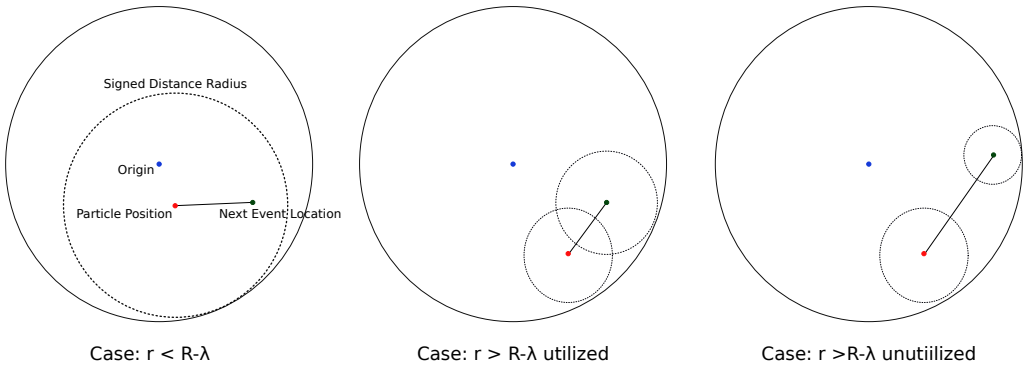


Figure 3.9: Depiction of utilization modeling scenarios. Left: an example of a track for which $d < R - r$. Middle: an example of a track for which $R - r < d < R$ and can be preconditioned. Right: an example of a track for which $R - r < d < R$ and cannot be preconditioned.

to violate the above condition as depicted in Figure 3.9. A violation of this limit may only occur when a particle travels far enough to reach the geometric sphere boundary along the current position vector as if it were moving directly toward the boundary of the sphere.

A plot of the α_{min} limit for various radial starting positions and distances traveled can be found in 3.10. An interesting feature of this figure is the convergence of all the curves on π as d approaches R. The convergence on π indicates that as the distance traveled approaches R the only direction that the particle can move is back toward the origin along the position vector. It also defines a maximum distance a particle can travel in the sphere and still be preconditioned using signed distance values. Intuitively this makes sense as the maximum chord length of a sphere is $2R$, and once a particle travels a distance R the sum of the signed distance values can then be no larger than R. Thus the condition for utilization in Equation (3.5) is always violated for distances traveled greater than R. Hence all curves go to zero at $\lambda = 100cm$ in Figure 3.7. Substitution of the α_{min} condition gives

$$US = 8\pi^2 \int_0^R \int_0^\infty \int_{\alpha_{min}}^\pi \delta(d-\lambda)\, r^2\, d^2 \sin\alpha \, d\alpha dd\, dr \tag{3.8}$$

Evaluating this integral and dividing by all query space gives the following form for the theoretical limit of signed distance field utilization as a preconditioner for ray firing

$$U_{fixed} = \frac{US}{A} = \frac{(1 - H(\lambda - R))(2R - \lambda)(R - \lambda)}{2R^2} \tag{3.9}$$

It can be seen in Figure 3.7 that this utilization limit works well as an upper limit for the simulation results for various signed distance field mesh resolutions. Equation (3.9) is an idealized model and does not account for interpolation error of the signed distance values used in the pseudo simulation. As the step size of the mesh approaches zero, so does the evaluation of the error and the utilization values approach the idealized analytic model for the preconditioner utilization.

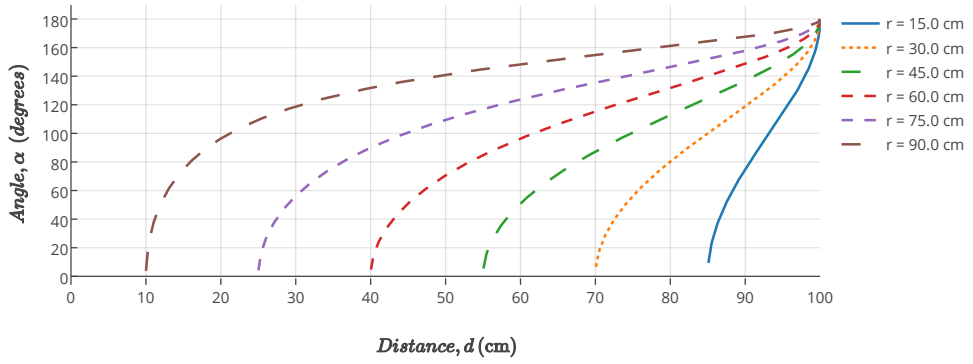


Figure 3.10: Plot of minimum angle of departure restriction for particles with various radial positions between 0 cm and 100 cm and various distanced traveled.

The fixed distance traveled scenario provides a baseline for agreement between the computational results and the model, but a more realistic scenario is required before attempting to apply the model as a predictive utility for application of the SDF in true Monte Carlo codes. The next iteration of the analytic model applies a sampled next physics event distance based on the probability for distance to interaction in a medium with a total cross section, Σ.

$$p = \frac{e^{-\Sigma d}}{d^2} = \frac{e^{-\frac{d}{\lambda}}}{d^2} \tag{3.10}$$

where in this case now λ represents the mean free path of the particle in the medium

In simulation, distances are now sampled as

$$d = -\lambda \ln(c) \tag{3.11}$$

where c is randomly sampled with a uniform distribution between 0 and 1. To find the portion of utilized space for a given R and λ one can apply the same methods from the fixed distance case. Appendix A provides a more detailed description of the model development for sampled distances. The result of the updated distribution for sampled distances is found in Equation (3.13). Applying this distribution to Equation (3.2) gives

$$8\pi^2 \int_0^R \int_0^\infty \int_{\alpha_{min}}^\pi -r^2 \sin\phi \, \lambda c \ln(c)^2 \sin\alpha \, d\alpha d\phi \, dr \tag{3.12}$$

$$U_{sampled} = \frac{\frac{1}{2}\lambda(R-2\lambda)e^{-\frac{R}{\lambda}} + \lambda^2 - \frac{3}{2}R\lambda + R^2}{R^2} \tag{3.13}$$

As seen in Figure 3.11, the updated predictive model describes the idealized utilization of the preconditioning method very well when the SDF step size is small. As the SDF step size is increased, the predictive

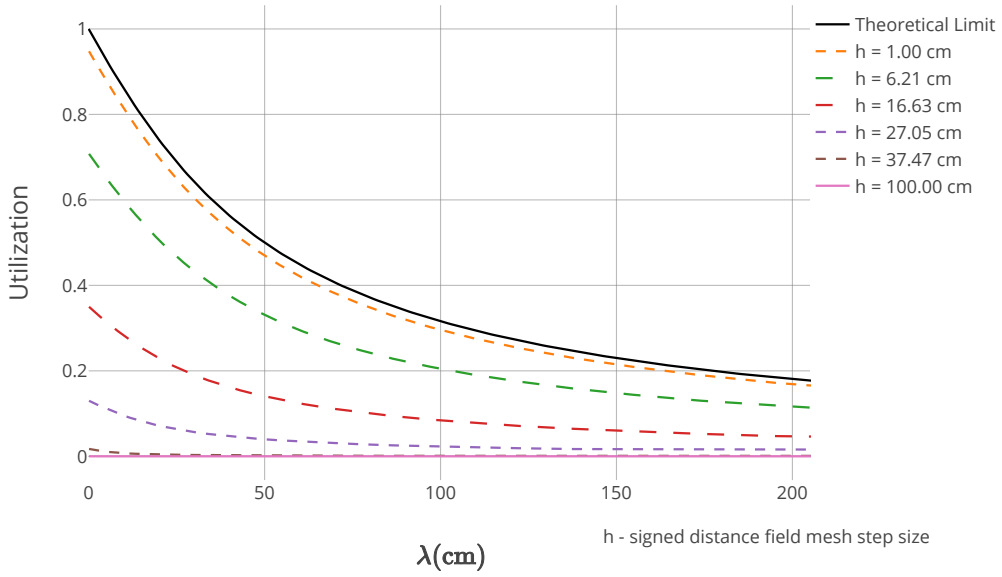


Figure 3.11: Results of the model for the theoretical utilization limit with the results of the simulation for a sampled distance traveled case over many different mesh step sizes.

model becomes increasingly worse at evaluating the correct utilization. For the predictive model to be useful in a production scenario, the relationship of the preconditioning method's utilization to the SDF step size must be understood as well. This relationship can be drawn through the SDF interpolation error, ϵ, which is a function of the mesh step size, h.

3.3.3 Applying Error Evaluation

Thus far, the error associated with the tri-linear interpolation used to recover signed distance values from the signed distance field has been ignored. In order to optimize step size of the uniform mesh making up the signed distance field, it is important to take this error into account

when predicting the utilization of the data structure. The error associated with any linear interpolation is, to first order, dependent on the second order partial derivatives of the function being approximated and the sampling frequency of the points used for interpolation. This relationship is described in two dimensions via the Taylor expansion of the formula for a bi-linear interpolation in Figure 3.3.3. For the case of a signed distance field sampled from a surface boundary, the error can be estimated by using the maximum local curvature of the surface as a conservative estimate of the partial derivative. However, in the case of a discretized mesh, this would be prohibitively computationally expensive to evaluate. Instead, a conservative estimate of the error based on the mesh size is used to ensure invalid geometric information is not returned from signed distance value recovery in the SDF.

$$\epsilon = \frac{1}{2}\Delta x(h - \Delta x)\frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\Delta y(h - \Delta y)\frac{\partial^2 u}{\partial y^2} \tag{3.14}$$

h — mesh interval size

Δx — x distance to interpolation point from data point

Δy — y distance to interpolation point from data point

$u(x, y)$ — sampled function on mesh

ϵ — error

The error estimate used in this work is equal to the largest diagonal of the mesh elements in the uniform SDF grid, $\sqrt{3}h$, and ensures that signed distance values cannot be used improperly to determine that a ray query can be avoided during simulation.

As seen in Figure 3.11, the step size of the mesh can significantly impact

the utilization of the data structure and in turn the overall performance of the simulation. It is important to use a step size which will not limit the effect of the data structure, but the signed distance field is a dense data structure and uses a considerable amount of memory. The memory usage of the signed distance field is inversely proportional to the cube of the mesh step size ($\alpha \frac{1}{h^3}$). Therefore it is important to consider the impact that this error will have on the utilization of the data structure.

$$SDV(\vec{p}) + SDV(\vec{n}) > |\vec{p} - \vec{n}| + 2\varepsilon(h) \tag{3.15}$$

$SDV - \ signed\ distance\ value\ function$

$\vec{p} - \ particle's\ current\ position$

$\vec{n} - \ particle's\ next\ event\ location$

$h - \ mesh\ step\ size$

$\varepsilon(h) - \ error\ evaluation\ for\ signed\ distance\ values$

The expansion of Equation (3.5) with the error term can be seen in (3.15) and results in a new condition for the minimum angle of departure (α_{min}) shown in Equation (3.16)

$$\alpha_{min} = \arccos\left(\frac{(2R - r - d - 2\epsilon)^2 - d^2 - r^2}{2dr}\right) \tag{3.16}$$

This formulation of α_{min} can be used directly in the integration described in Equation (3.8). The final form of the model after the integration is described in Equation (3.17). Again, a more detailed development of this model is provided in Appendix A.

$$U_{sampled} = \frac{(R-\epsilon)(\frac{1}{2}\lambda(R-2\lambda-\epsilon)e^{\frac{-R+\epsilon}{\lambda}} + \lambda^2 - \frac{3}{2}\lambda(R-\epsilon) + (R-\epsilon)^2)}{R^3} \tag{3.17}$$

This model in Equation (3.17) was compared to pseudo-simulations using sampled distances for the next physics event location. The result of this study can be seen in Figure 3.12. The dimensionless parameters n and c as formulated by normalizing the mean free path, λ and the mesh step size, h using the average chord length of the sphere, R_{av}. These dimensionless parameters are useful for describing the utilization in generalized terms of the geometry. This is discussed in more detail in Section 3.3.4.

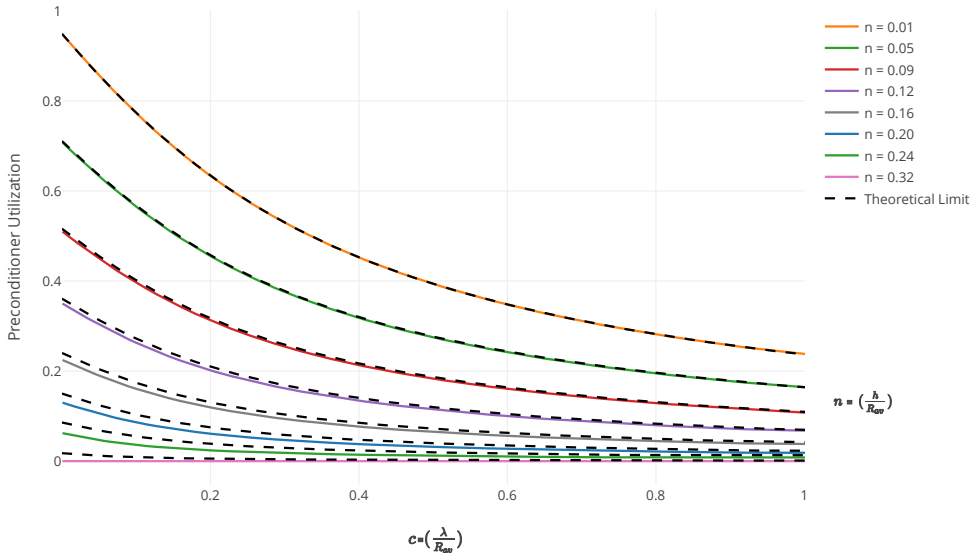


Figure 3.12: Theoretical limits for SDF utilization accounting for error evaluation plotted with simulation results for varying mesh step sizes.

The model presented in Equation (3.17) agrees very well with the

simulated results for small values of n. As the value of n increases, the model begins to over-predict the utilization of the preconditioning method slightly for low relative mean free path values. The utilization values in this regime are so low that they are not concern, however. Knowing that the SDF will also be applied to non-spherical geometries in practice, it is necessary to understand the predictive model's behavior for other geometries as well.

3.3.4 Extension to other geometries

To apply this model to other geometries, the mean free path and mesh step size were normalized using a measure of the free space in a volume compared to its surface area, the average chord length, R_{av} [7]. This value can be defined as seen in Equation (3.18).

$$R_{av} = \frac{4V}{S} \tag{3.18}$$

$$V - Volume$$
$$S - Surface\,Area$$

Normalizing the mean free path to the average chord length is a way of describing the probability that a particle will reach the surface of the volume that is independent of the distance traveled by the particles and the explicit geometry [36]. This in turn allows studies of the utilization to be performed in a [n=(0,1),c=(0,1)] domain for an arbitrary geometry, mean free path, and mesh step size.

This common domain of n and c provides a pathway for comparison of the model in Equation (3.17) to other geometries, which will be important in assessing the utilization of the SDF in arbitrary DAGMC models. In order to test the extension of this analytic model to other geometries,

pseudo simulations were performed for cylinder and cone geometries. The results of the utilization model and simulation results can be found in Figures 3.13 and 3.14.

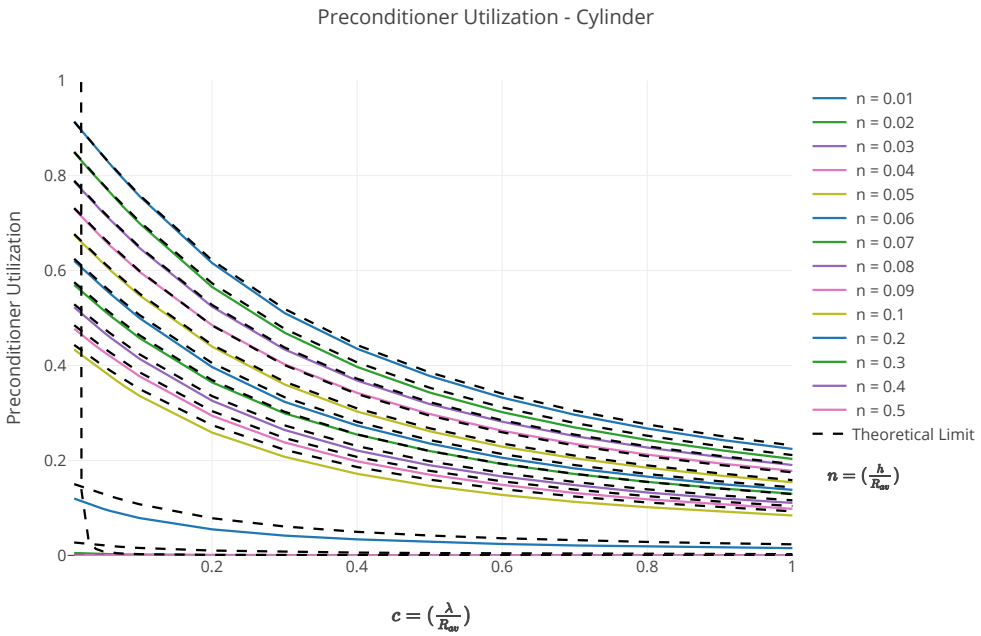


Figure 3.13: Application of the analytic model for preconditioner utilization in a cylinder volume for various mean mean free paths, λ, and mesh step sizes, h.

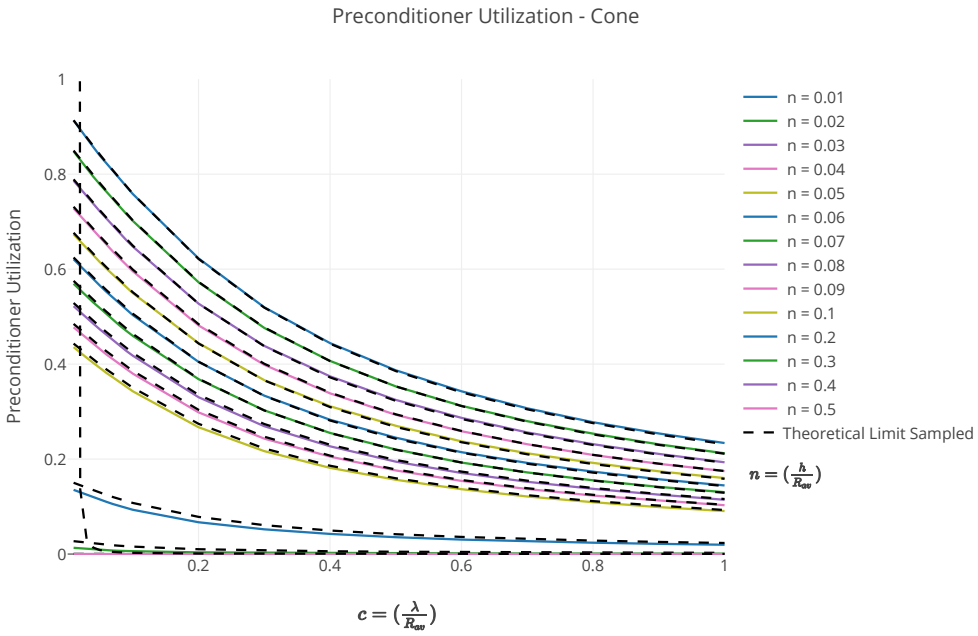


Figure 3.14: Application of the analytic model for preconditioner utilization in a cone volume for various mean mean free paths, λ, and mesh step sizes, h.

For small values of n, the utilization model is much closer to the simulated result. As the value of n increases, however, the model becomes less accurate in predicting the utilization just as in the spherical geometry. The analytic model breaks down for values of n which are outside of the limits set by the α_{min} condition. This is most clearly seen for the $n = 0.5$ trace in Figure 3.14. In the spherical case this occurs when the error evaluation exceeds R, meaning that zero utilization of the data structure is expected. For this condition, any evaluation of the error will exceed the maximum distance a particle can travel and still be utilized.

$$\epsilon(h) > R \tag{3.19}$$

$$\sqrt{3}h = R \tag{3.20}$$

$$h = \frac{R}{\sqrt{3}} \tag{3.21}$$

To extend this limitation to other geometries, the average chord length will again be used as a surrogate for the sphere radius in representing the upper limit of utilization for an arbitrary geometry. For the spherical case, the average chord length is shown in Equation (3.22).

$$R_{av} = \frac{4}{3}R \tag{3.22}$$

Equation (3.22) can be substituted into Equation (3.21) to obtain relationship of the mesh step size to the geometry for an upper limit of h when applying this model in simulation. This equation can then be normalized to place an upper limit on n for a valid use of the utilization model.

$$h < h_{max} = \frac{3}{4\sqrt{3}}R_{av} \tag{3.23}$$

$$n < n_{max} = \frac{\sqrt{3}}{4} \approx 0.433 \tag{3.24}$$

Equation (3.24) provides a value for the maximum allowed normalized step size for which the utilization model from Equation (3.17) can be applied in an arbitrary geometry. While unlikely that a step size this large would be applied in practice, it is important to understand the limitations of this model if it is to be used in an automated process for application of the data structure either on-the-fly or as a preprocessing before simulation.

3.4 Application in DAGMC Simulations

3.4.1 Performance Improvements

The geometry-agnostic version of the utilization model was used to create a *post facto* analysis script for predicting the SDF's utilization in production DAG-MCNP models. This analysis uses information about the average particle track length for each volume as the input mean free path for the utilization model. This information is gathered from Table 126 provided in MCNP output files from previous DAGMC simulations of the geometry[64]. The input for the average chord length of each cell was calculated using DAGMC methods *measure_volume* and *measure_area* for each cell in question. The mesh step size was treated as a variable used to optimize either the memory usage of the data structure or the utilization of the data structure depending on options passed into the script. The output of the script then describes inputs to the utilization model, the predicted utilization of the data structure, and an estimate of data structure's memory usage.

This analysis script was used to examine several models to search for volumes with a significant impact on the simulation. Most of the models used as benchmarks for timing comparisons in Table 1.2 did not contain volumes appropriate for application of the SDF. As a result, other problems were sought out - several of which simulate charged particles just as the in the nTOF problem.

In general, the addition of the data structure has a measurable but limited effect on the performance of the simulations in the production models listed below. One additional factor that the script takes into account is the number of collisions in the volumes it is examining. Despite the fact that the collision density of the transport in a given volume may be high relative to its average chord length, if a small percentage of the transport occurs in that volume, then the benefits of applying the SDF are

diminished. As an estimate of this, the ratio of collisions a given volume to the total number of collisions in the simulation was used as an estimate for the proportion of geometric queries occurring in the volume. Additionally, it has been found through profiling of various simulations that as the collision density increases, the proportion of the runtime spent querying the geometry is small relative to the time spent calling physics subroutines. The SDF preconditioning method was applied in the following models:

ITER

ITER is a demonstration nuclear fusion device currently under construction in southern France. An analysis of DAGMC was performed at the University of Wisconsin - Madison to determine the nuclear heating to shield materials used as primary shielding for fusion neutrons.

NTOF

The neutron Time Of Flight (nTOF) model is a project underway at the Institute for Science and International Security (ISIS) in the United Kingdom. The aim of this project is to determine neutron energies by measuring the time it takes neutrons to travel from one detector to another.

SHINE

The SHINE facility is a device which uses a beam fusion device to produce 14.1-MeV neutrons and irradiate an aqueous uranium solution to produce molybdenum-99 for use in medical imaging.

SNS

The Spallation Neutron Source (SNS), located at Oak Ridge National Laboratory, in which protons are accelerated to high energies (~1 GeV) and directed into a mercury target to produce thermal neutrons for various research applications.

Table 3.1 displays the results of simulations in which the SDF was applied. More detailed information on the parameters of the SDFs and the volumes they were applied to can be found in Appendix A. Improvements in the simulation run times varied from 0-35%. While differences in the run times are significant, they are small in comparison to those demonstrated in Section 3.3 for the hydrogen-filled sphere tests. The results in all simulations were identical regardless of SDF application, however, indicating that the methodology and error evaluation are robust for at least these production scenarios.

Model	**Particle Types**	**Next Surface**	**Closest Surface**	**Point Containment**
SNS	α,p,γ,β,^{3}H,n...	99.1%	< 0.1%	0.8%
nTOF	p,γ,n	99.4%	0.5%	0.1%
ITER	n	9.4 %	88.3%	2.3%
SHINE	n	99.9%	0%	< 0.1%
FNG	n	54.8%	0%	45.2%
ATR	n	96.8	0%	3.1%
UWNR	n	99.9%	0%	< 0.1%

Table 3.2: A summary of the relative number of calls to the various Monte Carlo geometric queries for several DAGMC production models. ‡

Despite apparent limitations of the data structure, application in the medical isotope production model from SHINE (shown in Figure 3.15) is

†This entry represents a secondary simulation of the SHINE model with modified physics cross-sections.

‡The proportion of query calls is dependent on the native physics kernel's tracking code. This is particularly true in the case of charged particle transport.

DAGMC Model	Histories Simulated	Predicted Utilization (%)	Actual Utilization (%)	Timing Ratio
ITER	1M	37.1	40.1	0.86
nTOF	10k	62.9	66.2	0.71
SHINE	500k	60.1	48.2	0.84
SHINE†	500k	60.1	56.5	0.74
SNS	10k	31.0	2.1	1.00

Table 3.1: A summary of SDF application to several production DAGMC models.

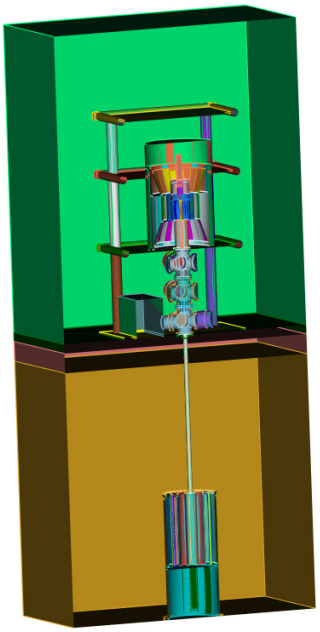


Figure 3.15: A cutaway view of the SHINE model used for demonstration of the SDF data structure.

a somewhat successful demonstration of the preconditioning method's ability to reduce simulation run times. A signed distance field with a 0.1 cm step size was applied to volumes near the bottom of the model which contain both an aqueous solution of uranium sulfate and a surrounding light water moderator. As outlined in Table 3.1, the resulting improvement in performance for a simulation of 500,000 particles was approximately 35% of the total runtime.

For the majority of the production test cases, the preconditioner utilization model is relatively accurate for the widely varying geometries of the DAGMC production models. There are cases in which the model struggles to accurately predict preconditioner usage and ray fire query avoidance, however. This an other observations about the limited effects of the preconditioning method will now be addressed.

3.4.2 Memory Usage

As discussed in Section 3.2, the uniform structured mesh used to store SDFs in DAGMC can be highly memory intensive. Given the memory constraints placed on DAGMC by the context of its application in parallel Monte Carlo simulations (see Section 2.6), additional memory consumption by DAGMC acceleration data structures must be weighed carefully against the benefits in run time reduction. The main concern being that per-process memory usage may prevent additional processes from being added to a node in parallel simulations. Despite the fact that the SDF has been shown to decrease the run time of several production simulations, it is unlikely to be more beneficial than additional processes in parallel simulations.

Model	Without SDF (MB)	With SDF (MB)
ITER	5971	7003
nTOF	598.3	619.2
SHINE	219.3	627.12
SNS	1804	3419

Table 3.3: Memory usage of the SDF for the various production applications.

The memory usage of the SDF varies greatly between the different production models in Table 3.3. For the nTOF model, the absolute memory consumption is very small and the SDF structure adds very little to the simulations overall memory footprint. In other simulations, the memory footprint is greatly affected, increasing usage by up to 1.2 GB. SDF memory usage is largely influenced by the size of the volume it is bounding and the mesh step size needed for high utilization of the preconditioning method. An estimate of the memory used by the SDF is presented as part of the analysis script used to apply the SDF. This estimate should be considered by analysts when creating DAGMC problem input, particularly if the simulation is going to be run in parallel.

3.5 Other Distance Limit Optimizations

DAGMC supports a mode in which the distance to the next physics event is used to limit the length of a ray and optimize the BVH traversal process. Nodes which the ray does not reach are not visited, reducing the average traversal time of rays in the BVH. If the ray does not intersect a surface, DAGMC returns a next surface distance greater than that of the distance to the next physics event. The physics kernel is agnostic to this change, and comparisons of these distances are used as they are normally to determine a particle's next event.

This distance limit optimization is only guaranteed to be robust for fully sealed DAGMC geometries. Because rays are not required to return a surface intersection in this mode, ray queries moving into gaps of the unsealed model will update the particle's position to the next physics event. If the next physics event is outside of the volume, particles will exit the volume without error in the particle tracking algorithm. In this scenario, particles can numerically leave the volume without updating the material in which the particle is interacting in the physics kernel, resulting in inaccurate physics events. This will continue until the particle is killed due to its low statistical contribution to the simulation solution. Occasional *point_containment* checks in the particle tracking algorithm can detect these particles and label them as lost, but this is not guaranteed to happen and the simulation may run without reporting any particle tracking errors. Creating a fully sealed DAGMC model from a general CAD representation is a difficult task for many complex production models, requiring that the model is created in a friendly way such that all volumes can be imprinted and merged successfully without overlaps. Corner cases also exist in which the *make_watertight* algorithm fails to fully seal tessellations of the CAD model [48].

In fully sealed models, however, this optimization is robust and can significantly decrease simulation times without consequence. This opti-

mization has been applied to the same set of production models in Table 3.1 to compare the improvement in performance seen when using the distance limit optimization without SDF preconditioning in comparison to DAG-MCNP simulations with infinite ray lengths.

It is worth nothing that the errant particle pathology associated with the physics distance limit is not present when using SDF preconditioning to accelerate DAGMC simulations. SDF values are generated using known surface locations relative to points in the uniform mesh. As such, information gathered from the SDF during preconditioning of particle queries will not allow particles to cross surfaces so long as the model is well-formed. This pathology is avoided by accounting for the interpolation error associated with signed distance value calculation to avoid incorrectly preconditioned ray queries. Rays fired in simulations where SDF preconditioning is applied still have an effectively infinite length and thus do not invalidate the robust particle tracking process in DAGMC. Particles moving through gaps in unsealed volumes will always be reported by DAGMC.

Model	**Performance Ratio (Distance Optimization / Unmodified)**	**Fully Sealed Model**	**Matching Results**	**No Lost Particles**
ITER	0.23	×	×	×
nTOF	0.46	×	✓	✓
SHINE	0.86	✓	✓	✓
SHINE*	0.51	✓	✓	✓
SNS	0.95	×	✓	✓

Table 3.4: Results from limiting ray lengths in DAGMC using the distance to the next physics event. Performance ratios are measured as simulations using finite ray lengths over those using infinite ray lengths. Matching numerical results, topological watertightness, and lost particle incidence are also reported for each model.

Table 3.4 shows the results of this study. For all simulations, the run time was substantially reduced using this optimization. And for nearly all simulations the results of the two simulations matched exactly with the exception of the ITER simulation. In this case, particles were lost in the unsealed model, causing a change in the results between the two simulations. As stated above, it is also possible that particles were incorrectly tracked in other unsealed models (nTOF and SNS), but had no effect on the final solution of the simulation.

The improvement in performance using the physics event distance to limit ray lengths is substantial for all models except SNS. The reason for this limitation in both this mode of operation for DAGMC and when applying SDF preconditioning to SNS is discussed in Section 3.6.2. For fully sealed (or topologically watertight) models, the physics event mode provides a performance enhancement equal to or better than that of the SDF preconditioning method. For models which are unable to be sealed fully, application of SDF preconditioning may be useful, particularly if a majority of particle interactions are occurring in an unsealed region of the model.

3.6 Performance Mitigating Factors

In the majority of the production models to which SDF geometry query preconditioning was applied, the effect on the overall performance of the simulation is smaller than what was seen in the simple test cases. This is due to a variety of factors, each of which is demonstrated by one of the models in Table 3.1.

3.6.1 Physics Dominant Simulations

The nTOF model was selected as a charged particle transport demonstration. The SDF was expected to be extremely beneficial in this scenario due

to the high collision density of charged particle transport relative to the size of certain volumes in the problem. The computational time spent on the detailed charged particles physics tends to dominate the runtime of the simulation, leaving little room for improvement in the overall simulation runtime by reducing geometric query costs.

A neutral particle example of this occurs in SHINE as well. In order to reduce the amount of time spend in the MCNP kernel for this calculation, group-wise cross-sections were specified for all materials in the problem. This allowed the SDF to have a more significant impact on the simulation runtime, but may not be desirable if continuous energy cross sections are necessary for a higher fidelity result. A separate entry for this simulation is found in Table 3.1.

3.6.2 Utilization Model Assumptions

There are some cases in which the SDF utilization model is inaccurate. The calculation of the average chord length and average track length are assumed to be true over the entirety of the volume in question. In reality, the localized average chord length will vary throughout the volume's domain. In cases where the local value of the average chord length is smaller than predicted, the utilization model will overestimate the SDF utilization. An example of this was found in the SNS test model.

A volume representing the mercury manifold in the SNS model was identified as a good candidate for SDF application. The SDF analysis model estimates the utilization of the SDF data structure under the assumption that the distribution of particle interactions in the volume is uniform. In this case, particle interactions occur mostly in regions of the volume where the local average chord length is smaller than the average chord length of the entire volume. This reduces the effectiveness of the SDF for the step size suggested by the input script. Additionally, an SDF which would provide better utilization would require a prohibitively small mesh step

size in terms of the memory usage of the data structure. The combination of these effects result in low utilization of the applied SDF and in turn the simulation runtime is reduced only marginally.

3.6.3 Surface Mesh Complexity

In Section 3.6.1 limitations of the SDF impact on the nTOF simulations were cited as related to the time spent in physics subroutines, but the faceting of the particular volume which the SDF was applied to may play a role as well. The volume's faceting contained a relatively small number of triangles. This in turn means that the generated BVH for this volume was relatively shallow. The relative improvement in performance when avoiding a ray traversal is directly related to this tree depth. While the complexity of a BVH search is $O(\log N_{tri})$, the cost relative to the $O(1)$ look-up of a signed distance value in the SDF becomes small as N becomes small, limiting the effectiveness of the SDF regardless of the utilization.

3.7 Summary

This chapter presented a novel approach to accelerating CAD-based MCRT through use of signed distance fields to precondition relatively expensive ray fire calls in the DAGMC geometry kernel. An analytic model was developed to predict the utilization of this method based on dimensionless geometric and problem-dependent parameters. The model was shown to be accurate in several production demonstrations. Though the effectiveness of the preconditioning method overall was limited by a variety of factors described in Section 3.6, the improvements in simulation runtime with the same final result were observed. Enhancement of SDF storage and further model development in could increase it's impact on these simulations and allow it to be more broadly applied in future studies.

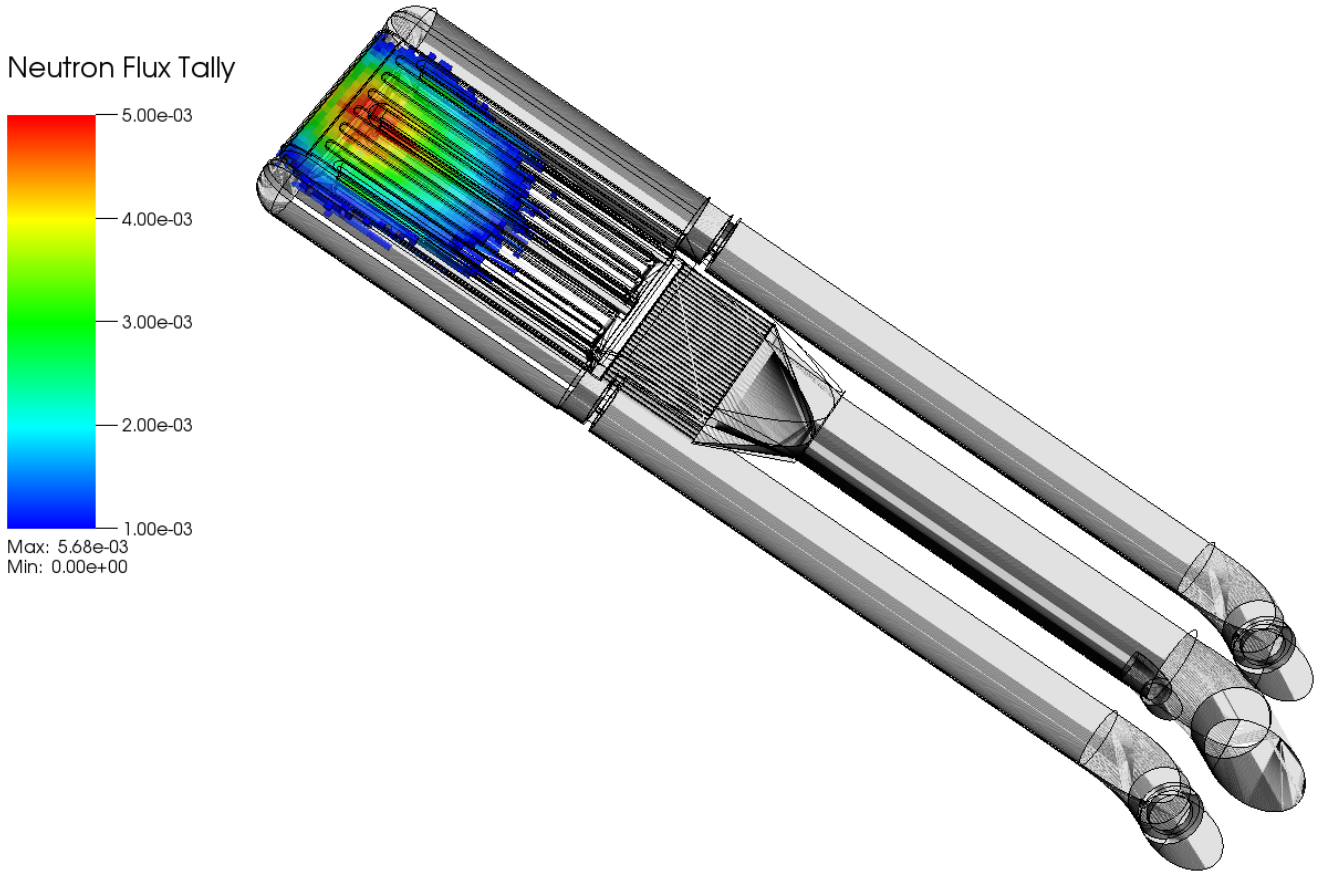


Figure 3.16: Depiction of utilization model limitation due to a small local average chord length in the region of high collision density.

Use of the physics distance limit in DAGMC models which can be made topologically watertight is recommended based on its ability to enhance query performance throughout the model as seen in Section 3.5. For models which cannot be made fully watertight, the SDF preconditioning method provides an alternative use of the physics distance limit to enhance simulation performance under the conditions described in in Section 3.3. In practice it is recommended that well-formed geometries are used in DAGMC, or those with can be made fully watertight, to ensure simulation robustness and high fidelity results. Thus, the SDF preconditioning method is likely to have a small application space in production DAGMC analysis, finding use only in problems where unsealed volumes have high particle collision densities relative to the volume size.

Chapter 4

A Mixed Precision Bounding Volume Hierarchy

4.1 Introduction

As established in Section 1.2, the dominant source of additional runtime in DAGMC is spent in the ray tracing process used to satisfy the set of Monte Carlo geometry queries outlined in Section 2.1. The focus of the work in this chapter is improvements on the performance of the ray tracing process through the application of SIMD-oriented programming for the bounding volume hierarchies in DAGMC. Some preliminary work was performed in this area by replacing the ray tracing kernel used in DAGMC (MOAB's oriented bounding box tree) with a kernel produced by Intel, called Embree. This work is outlined in Section 4.2. All of the simulation and ray fire results presented in this chapter and Chapter 5 were performed using a 3.4 GHz Intel Core i7 processor on a Haswell desktop with AVX2 vectorization instruction sets.

4.2 Linking DAGMC with Intel's Embree

Embree is the result of an effort to produce a performant CPU-based ray tracer as a demonstration of the expanding capabilities of modern CPU architectures [60]. In both construction and traversal of its BVHs, Embree takes advantage of many of the latest developments in BVH research by using modern chipset architecture capabilities via vectorization at an implementation level. Such work was alluded to in Section 2.5.5 of this work. The combination of these effects leads to a very powerful ray tracing tool in terms of performance, as demonstrated by the many projects which have incorporated Embree as their production ray tracing kernel such as Corona, Autodesk, FluidRay, and Brighter3D. As a result of its success in other areas, Embree was selected for application in DAGMC to satisfy geometric queries for MCNP. The resulting combination of Embree and DAGMC will be referred to as EmDAG.

4.2.1 Implementation and Model Transfer

The process of employing Embree as DAGMC's ray tracer begins by establishing an equivalent representation of the MOAB mesh in Embree. In comparison to MOAB, Embree is limited in its ability to represent the underlying topological structure of a model. This topology is necessary and used advantageously during particle tracking in DAGMC by reducing the set of triangles queried to those of the particle's current volume. However, a method was discovered to represent enough of the topology to meet the requirements of DAGMC transport. The highest level representation in Embree is referred to as a scene. Each scene may contain many geometries or triangle surface meshes. Fortunately, this system is enough to create a functional representation of DAGMC geometries in which MOAB volumes are the equivalent of Embree scenes and MOAB surfaces are represented in their respective scenes as surfaces. This mapping is more clearly il-

lustrated in Figure 4.1 This method provides a one-to-one mapping of MOAB volumes and surfaces to their corresponding entities in Embree (geometries and scenes, respectively). The mapping allows all topology-based operations to proceed inside of DAGMC in their usual manner. In this way, the requirement for topological information in DAGMC at the surface and volume level is met. Next, transfer of the primitive mesh data is considered.

Scenes do not share mesh data in the same way volume sets are able to in MOAB, so the triangle connectivity of each surface is reproduced for each scene they belong to. Fortunately, Embree does allow the sharing of vertices between scenes. In order to take advantage of this feature, all of the vertices in the MOAB mesh are provided to the Embree instance as a global vertex buffer. Surface triangles from all scenes can then be defined by a connectivity of vertices from this global pool of points. This method guarantees that each surface can be represented by the same set of stored vertices regardless of scene ownership, giving the exact same representation in each scene. It greatly simplifies particle tracking by guaranteeing that the same surfaces will not numerically overlap as a result of the conversion from double to single precision. Additionally, this method will maintain topological watertightness at the boundaries between surfaces ensuring the same model fidelity as the representation in MOAB.

The computation of triangle normals is critical to the DAGMC particle tracking algorithm established by Smith et. al. in 2011 [48]. In DAGMC, particles on or just outside the surface of a volume are handled by ignoring the near-surface intersection upon being placed in a new volume. This is done to maintain tracking of particles based on their logical position in the model rather than solely their numerical position which can cause ambiguities regarding point containment and cause lost particles or trap particles between surfaces, resulting in infinite or near-infinite histories.

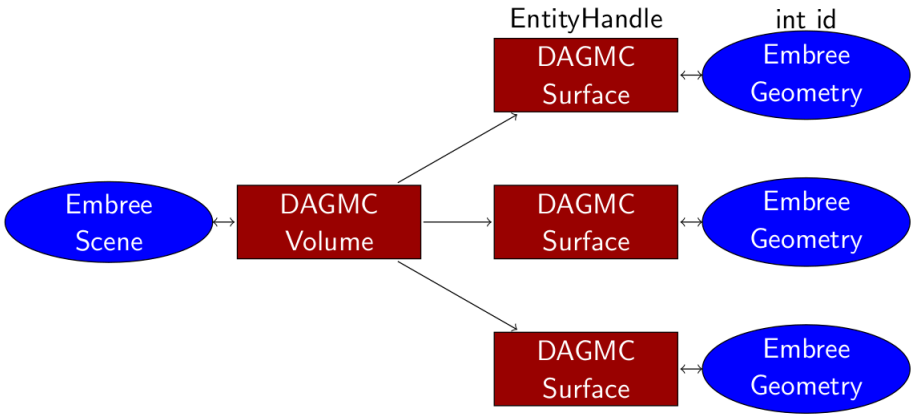


Figure 4.1: Representation of MOAB's topological connections mapped to Embree.

Logical particle tracking is implemented using the convention that triangle normals will always point outward from the center of the volume they belong to. Triangles hit by the ray are ignored if the normal of the triangle opposes the ray direction via a dot product calculation to ensure only exiting ray intersections are considered.

While this has historically been handled inside of DAGMC, this is accomplished in EmDAG via the use of Embree's filter functions. Filter functions allow for a user-defined callback method which allows users to validate a ray hit inside of Embree before returning a final result. Embree will return its most recent intersection with the scene to the filter function (the triangle's unnormalized normal vector included) and allow a method to either accept the hit or instruct Embree to continue tracing the ray path based on the outcome of the filter function. In MOAB, triangle normals are set in a global manner and adjusted using stored information within MOAB based on what volume is being queried at the moment. This is referred to as the surface's sense with respect to that volume. Because surface triangle connectivity is duplicated when transferring mesh entities to Embree, triangle normals are pre-oriented depending its sense with

respect to the volume is being transferred to Embree at load time. This saves steps in gathering this information upon traversal. By matching DAGMC's data model in the areas of topology, watertight representation, and hit acceptance/rejection based on triangle normals, the customized Embree instance can provide DAGMC with all the information needed to perform geometric operations needed for Monte Carlo simulations.

4.2.2 Ray Fire Performance

Using a DAGMC-based ray fire test program, the performance of DAGMC's ray fire ability was compared to that of EmDAG's for three representative models. These models include a simple sphere, a notched sphere, and a high aspect ratio (HAR) cylinder. In each of these tests, the models are tessellated with an increasingly smaller faceting tolerance to vary the number of triangles generated when discretizing the models. The faceting tolerance is defined as the maximum distance between the faceted curve or surface and the geometric curve or surface which it resolves. 600k rays are then fired from the center of the volume isotropically using the same random number seed so that the same set of rays is fired in each ray tracing system.

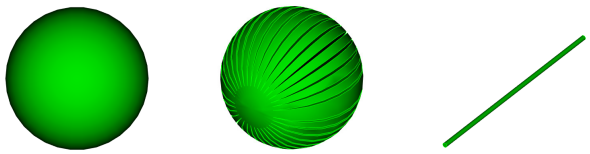


Figure 4.2: CAD representations of the sphere, slotted sphere, and high aspect ratio cylinder (left to right) test models used for ray fire timings of DAGMC and EmDAG.

Each geometry used in these tests presents its own challenges with increasing faceting tolerance. The sphere is a good control case for an in-

creasing number of triangles with a well-behaved tessellation. The number of triangles generated in the spherical case will tend toward a maximum value with decreasing faceting tolerance, but the general nature of the triangulated surface (triangle density, structure, etc.) will remain the same. This is not true of geometries with planar surfaces which may be able to be resolved exactly using some finite number of triangles making the sphere a valuable test model in that regard. In the case of the notched sphere, high-valence regions are generated by the faceting engine as a result of its underlying algorithms for planar surfaces meeting curves surfaces. The high triangle density of high valence regions causes overlaps in bounding volumes which become larger as the faceting tolerance decreases. This results in inefficient hierarchy traversal. Additionally, and perhaps more importantly than the presence of high valence regions, rays being fired with a point of origin at the center of the model causes them to travel either exactly along or very near to the surfaces of the planar slots in the sphere. Such a ray query will visit many internal nodes of the hierarchy during traversal, creating what is referred to as a very wide traversal through the BVH as opposed to a narrow traversal in which fewer branches and fewer nodes of the tree are visited. In this way, the slotted sphere provides a good measure for the performance of a wide traversal through the hierarchy in a situation for which many of the internal nodes are required to be visited. The two sets of timing results can be found in Figure 4.3.

Both MOAB and EmDAG scale relatively well for the HAR cylinder model with decreasing faceting tolerance. Though MOAB is using OBBs and Embree uses AABBs, this indicates that both systems are capable of building efficient BVHs for a model with long, skinny triangles. For both MOAB and Embree, the scaling of the spherical case with increasing triangles is slightly worse than the HAR cylinder most likely because the BVH tree is unavoidably going to become deeper as more and more triangles exist in the model, requiring more traversal steps to reach leaf

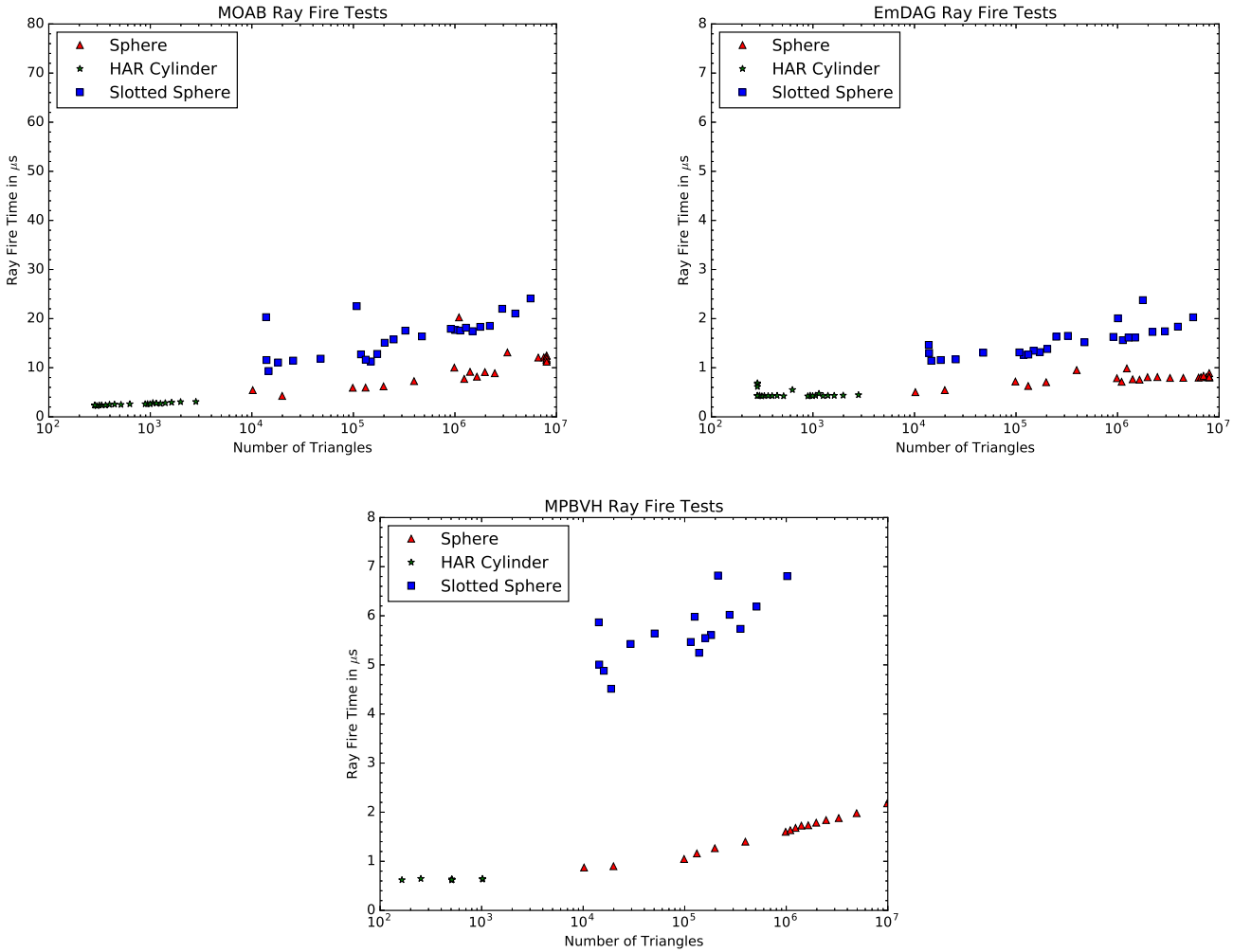


Figure 4.3: Ray fire test results on three representative DAGMC volumes shown in Figure 4.2. Each data point represents the average ray fire time for 600k randomly directed rays from the origin of each volume. Top Left: Results of the tests for MOAB's OBB Tree. Top Right: Results of the tests for DAGMC coupled with Embree, or EmDAG. Bottom: Ray fire test results for DAGMC coupled with the MPBVH. The scale used for the MOAB OBB Tree is 10x greater than in the other two cases due its lower performance.

nodes. Finally, the slotted sphere contains many high valence regions and as expected it scales the worst with decreasing faceting tolerance as the valence of these regions increases. Due to the very similar scaling of each test model, it can be stated that the majority of the discrepancy in the ray fire timings between the two systems occurs in the traversal methods employed by both systems and isn't likely due to a significant difference in the quality of the BVH being built by either system. Changes in the way the BVH is built typically accounts for anywhere from 30-40% difference in ray fire timings whereas the discrepancy seen here between MOAB and Embree is on average **an order of magnitude better** when using Embree. To some degree, this has to do with Embree's freedom of design without the restriction of a ray tracing implementation inside the context of another application. The flexibility of MOAB's core design allows for the robust implementation of an oriented bounding box tree within this context, but comes with the overhead of database calls to retrieve stored information which can be undesirable for a high-performance system and doesn't allow MOAB to take advantage of some implementation optimization available in Embree. The vectorization of Embree's traversal through its BVH contributes greatly to its speed. This can also be considered a part of the design freedom allowed when designing an independent ray tracing system that cannot not be afforded using only MOAB's database interface. MOAB's interface for direct memory access allows for similar freedom in BVH design, however. This will be discussed further in Section 4.3.1.1.

4.2.3 Transport Tests

As an extension of these pure ray fire tests, the effect of an improved ray tracing system in particle transport was studied as well. These tests begin with several simple models and end with the application of EmDAG to one of the models used for DAGMC performance benchmarking in Chapter 1, FNG.

The first transport models to be tested were a single cube and single sphere filled with a dense hydrogen material for high collisionality in the problem resulting in a large number of ray queries in the transport run. Each of these models' principal dimension is 10 cm. The source for these models is a 5 MeV neutron isotropic point source at the center of the volume. One million particles were simulated in each test. All of the test models were generated using a faceting tolerance of 10^{-4}cm. Moving upward in complexity, another set of tests were run using a set of nested cubes and nested spheres. Each of the nested volume models contained three cells: the inner volume, a shell volume, and the graveyard volume. The purpose of these tests was to ensure that particles could in fact be tracked through multiple volumes correctly. The nested cubes model contains an extra volume which consists of the original single cube subtracted from a cube 1cm larger in each dimension. The nested sphere model contains an extra volume consisting of the original sphere from the single volume model subtracted from a sphere 1cm larger in radius. As the purpose of these tests was to test EmDAG's particle tracking between volumes, the dimensions of the offset between the nested volumes is largely irrelevant so long as particles reach all of the volumes in the model.

Test Model	MCNP	DAG-MCNP	EmDAG-MCNP
	time (min)/ ratio to MCNP		
Sphere	2.93 / 1.00	25.13 / 8.58	4.73 / 1.61
Cube	5.03 / 1.00	10.56 / 2.10	5.80 / 1.153
Nested Spheres	4.35 / 1.00	50.82 / 11.68	7.94 / 1.83
Nested Cubes	4.73 / 1.00	9.26 / 1.96	4.35 / 0.92

Table 4.1: Runtime comparison of native MCNP, DAG-MCNP, and EmDAG-MCNP over four contrived transport test problems.

The native MCNP runs were generally the fastest among the test problems with the exception of the nested cubes case in which EmDAG-MCNP

marginally outperformed the native code by 8% (see Figure 4.1). This is likely due to the fact that very few triangles are needed to represent the surfaces of parallel-piped volumes. Exactly two triangles are needed to represent each face of the box. The resulting BVHs of these structures are very simple, often composed entirely of leaf nodes. The fact that these volumes have multiple surfaces is also of importance. MCNP searches linearly through a given volume's surfaces to determine the intersection of a particle with the nearest surface whereas both DAG-MCNP and EmDAG-MCNP perform this search for all surfaces bounding the volume simultaneously by joining surface BVHs into volume BVHs. This process is discussed in detail in Section 4.3.1.4. In the nested cubes model, it is likely that the number of surfaces relative to the number of triangles in their representation is high enough to allow EmDAG-MCNP to overtake MCNP's CSG calculations. This is a good demonstration of how CSG implementations suffer from the lack of a spatial search component when creating volumes from Boolean combination of surfaces as mentioned in Section 2.2.2.

The results of the single-volume test cases for native MCNP differ slightly from the results from the DAGMC-based systems, which match each other exactly. This is not surprising as DAG-MCNP is known to report statistically similar, but not exactly the same, results as native MCNP. Comparisons of DAG-MCNP to native codes are not the concern of this study, only a comparison of the values returned by EmDAG-MCNP in comparison to DAG-MCNP is considered. Differences in the tally results between DAG-MCNP and EmDAG-MCNP are present only in the nested spheres transport model. There is a small difference in the flux tally for the outermost volume as can be seen in Table B.9 of Appendix B. By examining the number of particle tracks in each cell, it was determined via debugging values that this discrepancy is caused by a single particle ending in EmDAG-MCNP near a surface of cell 2 while in DAG-MCNP

the particle crosses into cell 3 before abruptly terminating though it still contributing slightly to the tally in cell 3. This descrepancy is caused by different ray-triangle intersection tests used in MOAB and Embree rather than being result of the double to single floating point conversion of the model that occurs when using EmDAG.

Finally, a production test of EmDAG was conducted on the FNG model using the same volumetric source as in the performance benchmarking tests described earlier. Initially this model failed quickly due to lost particles. This was surprising as the model is expected to have the same watertight fidelity that it does when using DAGMC. In order to allow the run to complete, the number of allowed lost particles was increased to the number of the sources particles being run. The justification for this allowance being that if the lost particle rate is small enough, overall performance and results of the run would still provide a viable comparison of the two systems. In the end, the model lost 255 particles in 100 million histories. While this is concerning in terms of robustness, the lost particle rate per history wasn't considered high enough greatly impact the results from a performance comparison standpoint. A timing comparison of the FNG run using EmDAG-MCNP to the native MCNP model as well as DAG-MCNP is found in Table 4.2.

Implementation	**ctme (min)**	**wall time (min)**	**ratio**	**lost**
MCNP5	209.92	205.99	1.00	0
DAG-MCNP5	1023.04	1023.05	4.99	0
EmDAG-MCNP5	303.49	303.63	1.44	255

Table 4.2: A comparison of transport on the FNG model using a 14.1 MeV volumetric source over 100M histories for native MCNP, DAG-MCNP, and EmDAG-MCNP.

Again, significant gains in performance are seen using EmDAG in comparison to DAGMC and less than a factor of two is found in the ratio of

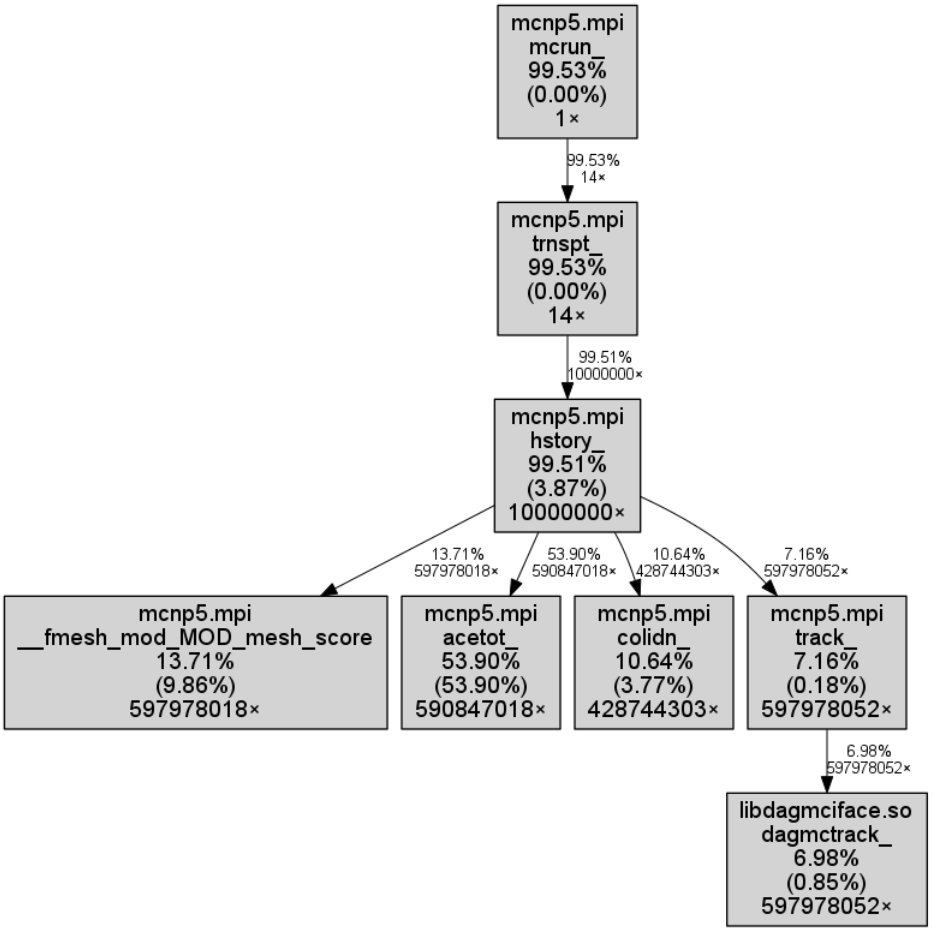


Figure 4.4: Call-graph of the EmDAG run on the FNG model for 1×10^7 histories. Processes taking <=6% of the runtime are filtered in order to simplify the call-graph.

the FNG runtime compared to native DAGMC. Several particles were lost in the EmDAG simulation, however. The cause of these lost particles, which make it impossible for DAGMC to rely on Embree's existing constructs as a robust ray tracing tool, is discussed further in Section 4.2.4.

4.2.4 Limitations

While the implementation of Embree in DAGMC showed a vast improvement in performance relative to DAGMC's current implementation, several problems were encountered during the process. This is not surprising when re-purposing a ray tracing kernel for an unintended application.

One of these problems is the presence of lost particles in a watertight model. The FNG model EmDAG was tested on is a fully sealed model. A fully sealed model is one in which every volume is topologically sealed such that there are no gaps between surfaces or adjacent volumes. As a result, DAGMC is able to robustly track particles through such a model with no lost particles. While the lost particle rate for the EmDAG FNG test relatively low, they in theory should not occur at all as was shown by the DAGMC runs. After a considerable amount of investigation as to the nature of these lost particles, their cause was determined to be a systematic problem not encountered in the nested volume cases due to the simple nature of their geometric topology.

In the DAGMC workflow, a required step for a watertight model is to imprint and merge the surfaces in the geometry representation before faceting the model. Imprinting (shown in in Figure 4.5) is the process by which surfaces and curves that are coincident by proximity in space are made the same. This process is accomplished by splitting entities into their coincident and non-coincident parts. The merging process then topologically combines these coincident parts into single entities such that the single entities are topologically adjacent to all entities adjacent to both of the originals. The result of these steps is non-manifold model with

surfaces shared between neighboring volumes [48].

The imprinting and merging of surfaces allows only one representation of each topological entity to be created upon faceting the model. By using the faceted curves of the model as a reference for where surfaces meet in space, the triangles of a surface are then forced to meet at those curves in a topologically watertight manner via the *make_watertight* algorithm [48]. Topologically watertight in reference to triangle facets refers to shared connectivity of mesh elements between surfaces. This is distinctly different than watertight by proximity or by points of triangles being "close enough" to one another. Topological watertightness of triangles refers to surfaces meshes which share vertex connectivity at their boundary with other surfaces. These triangles then share mesh vertices in MOAB whose coordinates in virtual space are represented using the exact same floating point representation in the database. In this way particles cannot be lost through gaps in surfaces and firing a ray from any position inside a topologically watertight volume should always result in a triangle intersection. Despite the topological watertightness of the triangle meshes used in EmDAG, particles were lost in the transport process.

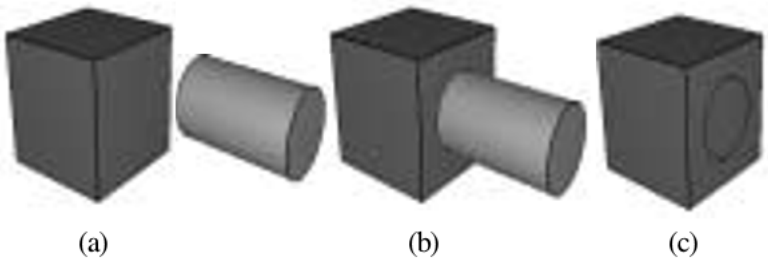


Figure 4.5: Example of two adjacent volumes being imprinted and merged. a) Two volumes, a cube and cylinder are created. b) The cylinder end face is moved such that it is coincident with one of the faces of the cube. c) The imprint operation is performed and the cylinder curve is imprinted onto the cube (cylinder was removed for visibility of imprinted curve). Adapted from [62].

Some detailed debugging of this problem revealed that this occurs in a systematic fashion within the FNG model at intersections of 3 or more volumes. The scenario is that a particle moves onto a triangle edge which is part of the boundary between two surfaces. When this occurs, an intersection with either surface connected to that interface is valid. The particle will then logically move into the volume on the other side of the hit surface. EmDAG handles most of these cases well with the exclusion of scenarios in which a particle will have a zero track length inside one of the volumes. A zero track length in this case meaning that the particles' trajectory is such that it will only glance a volume without having any track length inside of it. An example of this particle tracking pathology can be found in Figure 4.6. In this case, the EmDAG system may be unable to find a hit whereas DAGMC's tracking is robust enough to find the triangle intersection on this volume and move on.

By isolating this particle's history and producing the particle history with locations precise enough to detect the discrepancies between EmDAG and DAGMC, it was found that the position of the particle in EmDAG was

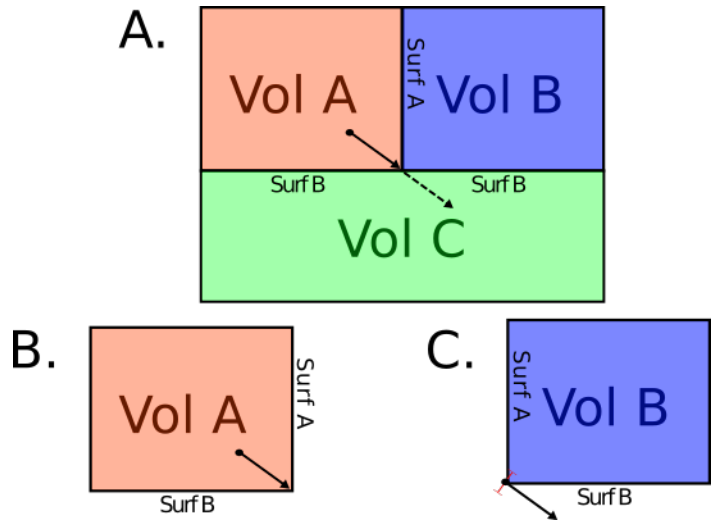


Figure 4.6: **A)** The initial state of the lost particle. The particle's trajectory is such that it intersects with the boundary between surfaces A and B. The correct continuation of the particle into volume C is depicted as a dashed line. **B)** An intersection with surface A is found though either surface A or B are equally valid. The particles position is then updated to its intersection with the boundary of surfaces A and B. The particle then logically moves into volume B. **C)** Upon placement of the particle in volume B the Monte Carlo code requests the distance to next surface intersection. The particles position and direction are converted from double to single precision. The small change in the particle's position places it outside of volume B and the trajectory is such that an intersection is not found. At this point the particle is considered lost.

numerically outside of the new volume after crossing the surface. The particle's position and direction were such the correct triangle hit could not be found in either EmDAG or DAGMC's ray fire systems. The cause of this discrepancy is believed to have to do with the necessary conversion between double and single floating point precision in the EmDAG system.

As previously mentioned, EmDAG uses single floating point representation in its ray tracing kernel while DAGMC uses a double precision representation of the geometry and particle information. In order to ac-

commodate Embree's representation, properties of the particle location and direction are converted to single precision for ray tracing queries in Embree and back to double precision when in DAGMC. When changing the floating point representation, truncation rules based on the computing environment are used to determine the new representation according to IEEE standards for conversion between precision levels [2]. These changes in the particle's location and direction are small, but in the scenario described above it seems that the particle location and/or direction are altered enough throughout the course of its history to cause a failed ray intersection - resulting in a lost particle.

In Brandon Smith's thesis, "Robust Particle Tracking and Advanced Geometry for Monte Carlo Radiation Transport" [48] there is a detailed description of the different pathologies encountered in tracking particles through a surface mesh representation of a geometric model. Briefly mentioned in this chapter is the possibility of a lost particle due to numerical error in the particle's position orthogonal to the particle's trajectory. Lost particles caused by this pathology are not covered however as the double precision floating point representation does not allow the particle position to change enough for this case to occur in practice. EmDAG is vulnerable to this particular pathology due to the constant conversion from single to double precision values between DAGMC and Embree. In order to avoid this problem moving forward, any improvements to DAGMC's ray tracing kernel for particle tracking will need to maintain use of double precision representations for mesh elements for robust coupling of numerical and logical particle positions and directions along with unmodified tally results in simulation.

4.3 Mixed Precision Bounding Volume Hierarchy

A SIMD-oriented version of DAGMC, EmDAG, was implemented and tested in DAGMC. Unfortunately, this system was inherently limited by conversion from single to double precision values. The use of double precision intersections is required for DAGMC to robustly interface with any of the Monte Carlo codes it supports. As a result, reduction of the triangle primitives to single precision is not an option.

This section outlines an approach to using single precision bounding volumes around double precision geometric primitives. Because the majority of the computational work in ray tracing occurs in traversing the BVH as opposed to testing triangles for intersection, it is hypothesized that the majority of the performance benefits seen in the EmDAG implementation can be preserved in the proposed mixed precision system, dubbed the Mixed Precision Bounding Volume Hierarchy (MPBVH).

4.3.1 Implementation and Design

The MPBVH was created by the author as an independent project which draws on design elements from Embree while interfacing with MOAB to optimize CPU-based ray tracing for the application of CAD-based MCRT-t. The details of these features can be found in Embree's documentation [59]. Several such design elements found in both the Embree and the MPBVH include:

- AABBs
- single-precision bounding entities
- bit-encoding of BVH node types
- type-agnostic vectors for support of SIMD instruction sets

- memory pre-fetching of tree node memory during traversal
- pre-computation of near and far box boundaries
- vectorized node intersection

The kernel created by the author is a lightweight version of the Embree kernel focused on static triangle meshes and access to an external mesh database used in engineering analysis. The kernel was implemented outside the context of MOAB allowing it to freely implement many of the performance related concepts found in Embree, but it relies on unique characteristics of MOAB that allow the kernel to access the database's memory directly without duplication of information in memory. The remainder of this section will address some of these features and design alterations.

4.3.1.1 MOAB Direct Access

MOAB provides a rich interface used to create, modify, and query both structured and unstructured mesh representations. It supports many meta-data types as well as the arbitrary grouping of mesh entities into sets to represent boundary conditions, material types, etc. MOAB also allows direct access to memory which is normally protected through its standard interface. This allows for direct look-ups of mesh data and allows other applications access to the memory as well. MOAB's direct access capability can be extremely useful in data transfer or query based operations, such as ray tracing.

The MPBVH uses these methods to populate references to triangles in the BVH without duplication of their coordinate values or connectivity. Access to these locations in memory is handled by a direct access manager object which protects the mesh information from modification during the building and traversal of the BVH. All coordinate and connectivity

information can be accessed through this manager, which protects against accidental modification of values in MOAB's memory space.

Mesh information is retrieved via offset values using indices for triangle and vertex entities. This later allows coordinates values for triangle intersections to be gathered optimally from the database during BVH traversal. Bypassing the MOAB interface in this way requires that the set of entity handles in the database is contiguous. Fortunately, files loaded into MOAB have a contiguous entity handle space regardless of the contiguity at the time the file was written to disk. This guarantees that the set of entity handles representing DAGMC geometries will be contiguous when loading a geometry for simulation. If the mesh data is modified, in particular if entities are deleted, then this condition will be broken and it may not be possible to use the direct access methods robustly without accounting for gaps in the array space. Accounting for these holes will have an unknown affect on the performance of ray tracing queries when triangle coordinates are fetched from MOAB's memory space and will vary depending on the number of holes in the space. MOAB provides tools for collapsing or modifying the entity handle space if this becomes necessary in future applications of this design in DAGMC. Currently, DAGMC does not modify mesh data once the file has been loaded for simulation, so the effect of these gaps and the collapsing of the entity handle space is not explored in this work.

4.3.1.2 Reduced-Precision Ray Tracing

In the SIMD BVH kernel, bounding boxes are calculated in single precision based on double precision triangle coordinates from the MOAB direct access manager. As demonstrated by the limitations of the EmDAG implementation, double precision intersections with triangle primitives are necessary for robust radiation transport in CAD geometries.

Using double precision ray values to intersect with single precision

boxes is possible due to IEEE standards used to truncate double precision values to single precision values, but costly due to the conversion of the ray origin and direction for each box intersection [2]. Instead, a method was applied in which a traversal ray data structure is used represent the ray origin and direction in single precision alongside the double precision ray. This greatly accelerates the traversal process moving through the hierarchy of bounding boxes. Once leaf nodes are reached in the tree, the original double precision values of the origin, direction, and vertex coordinates are used to calculate intersections and return more precise distances to the surface. A visualization of this proposed ray traversal scheme can be found in Figure 4.7.

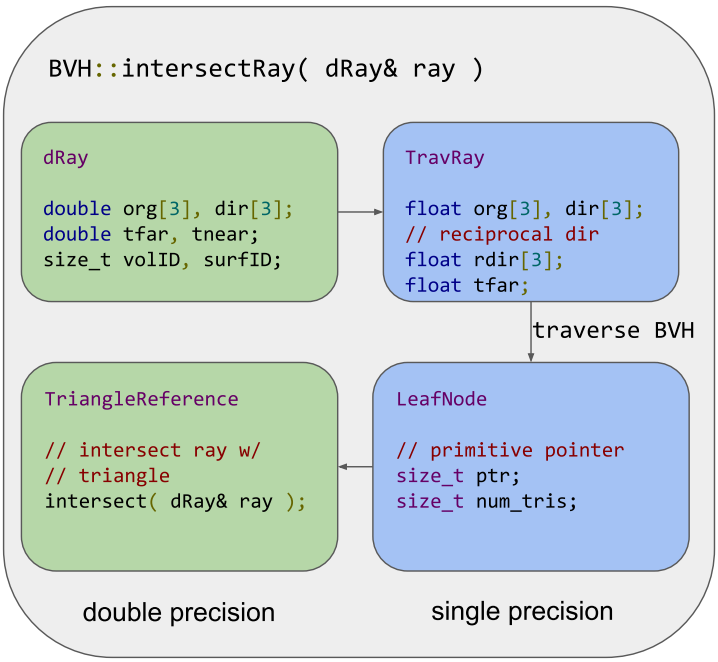


Figure 4.7: The reduced precision scheme used to accelerate ray traversal while returning double precision intersections.

This method removes the need for conversion of single precision intersection distances which caused lost particles in the EmDAG implementa-

tion, but there are other robustness concerns which must be addressed. In order to return the correct intersection distance, the leaf node containing the triangle with the nearest intersection must be visited. Upon converting ray origins and directions from double to single precision, small changes in the ray origin and direction are introduced. This effect is illustrated in Figure 4.8.

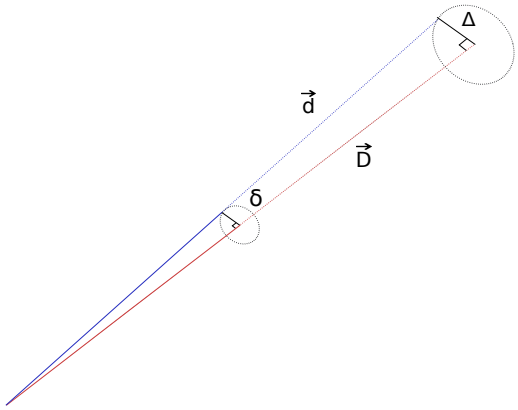


Figure 4.8: A representation of the directional change due to conversion of a ray's origin and direction from double to single precision, where capital letters indicate double precision values.

To ensure that node visits which would occur in double precision will also occur in single precision, the bounding boxes are artificially enlarged by a small amount to ensure that this is true. When determining how much the boxes need to be enlarged, it is important to understand the difference in the box intersection locations caused by this change.

Let the original, double precision, ray be $\vec{R}$ where its origin and direction are

$$\begin{aligned} \vec{O} &=< R_x, R_y, R_z > \\ \vec{D} &=< R_u, R_v, R_w > \end{aligned} \tag{4.1}$$

and the traversal ray, in single precision, be $\vec{r}$ where its origin and direction are

$$\begin{aligned} \vec{o} &= < r_x, r_y, r_z > \\ \vec{d} &= < r_u, r_v, r_w > \end{aligned} \tag{4.2}$$

where the distance between these two vectors at their unit length can be described as

$$\begin{aligned} \Delta = \sqrt{(R_x - r_x)^2 + (R_y - r_y)^2 + (R_z - r_z)^2)} + \\ \sqrt{(R_u - r_u)^2 + (R_v - r_v)^2 + (R_w - r_w)^2} \end{aligned} \tag{4.3}$$

Because the difference between these vectors in each dimension is representative of the truncation of double precision values to single precision, they can be set equal to a value δ. By substituting this value, Δ becomes

$$\begin{aligned} \delta = R_x - r_x = R_y - r_y = R_z - r_z = \\ R_u - r_u = R_v - r_v = R_w - r_w \end{aligned} \tag{4.4}$$

$$\Delta = 2\sqrt{3}\delta \tag{4.5}$$

Extending this to a ray traveling a parametric distance, t, along the respective single and double precision ray directions, the value of Δ

$$\begin{aligned} \Delta = \sqrt{(R_x - r_x)^2 + (R_y - r_y)^2 + (R_z - r_z)^2)} + \\ \sqrt{t^2(R_u - r_u)^2 + t^2(R_v - r_v)^2 + t^2(R_w - r_w)^2} \end{aligned} \tag{4.6}$$

$$\Delta = (1 + t)\sqrt{3}\delta \tag{4.7}$$

Equation (4.7) indicates that boxes will need to be expanded an amount Δ in order to ensure that $\vec{r}$ will intersect any box $\vec{R}$ does. One concern with this expansion is that artificially extending the bounding boxes causes more overlap and raises the average number of nodes visited per traversal of the BVH. If the value of Δ is a large value, then the performance of tracing rays through the data structure may suffer. An examination of the Δ value indicates that this is not the case, however.

As a worst-case consideration, t should be evaluated as the maximum distance a particle might travel through the geometry. This can be evaluated as the longest possible diagonal of the problem's global bounding box using it's largest dimension, l_{max}.

$$t_{max} = \sqrt{3} l_{max} \tag{4.8}$$

Given this value, the truncation will be in the seventh digit on modern CPU systems. This value can be considered to be relative to the overall geometric scale of the problem, represented by t_{max}. Using this analysis, the value of Δ_{max} can be written as

$$\Delta_{max} = (\sqrt{3} + 3 l_{max})\delta \tag{4.9}$$

Another method for working with mixed precision BVHs in double precision geometries was put forth by Vaidyanathan in which the ray's origin is periodically updated during traversal so that the deviation from the double precision intersection is limited [55]. This method is intended for use with geometries in motion and requires additional computation during ray traversal. DAGMC geometries, and MCRT geometries in general, are static, so it is more efficient to account for the discrepancy caused by the reduced precision ray direction during construction of the MPBVH.

To further understand this effect, the value of the box extension in the kernel was varied from 1×10^{-8} to 1×10^{2}. For each box extension value,

the same set of ray fire tests in Section 4.2.2 were run. The average of the ray fire times over all faceting tolerances is used as a representative value in Figure 4.9 to described the performance for each model. Performance of the MPBVH kernel isn't strongly effected below Δ values of 1×10^{-2}. Above this value, the performance degrades rapidly as box overlaps become large and the false positive intersections of the ray with nodes in the tree dominate the run time. When the extension of the boxes becomes large enough, the run times plateau - indicating that every node in the tree is being visited for each ray fire. Below a value of 1×10^{-6}, missed rays begin to appear in the runs. The threshold for this region is accurately predicted using the model in Equation (4.9). The longest dimension for any of the models ranges from 10 to 50 cm in size. Inserting this into Equation (4.9) indicates that missed rays will start occur at values of $\Delta = 3 \times 10^{-6}$. The lower limit of the box extension value and upper limit, the point at which performance begins to suffer, provide a window of several orders of magnitude for this value. This window will shrink as models become larger and the value of Δ_{max} increases, but these are scales rarely seen in radiation transport.

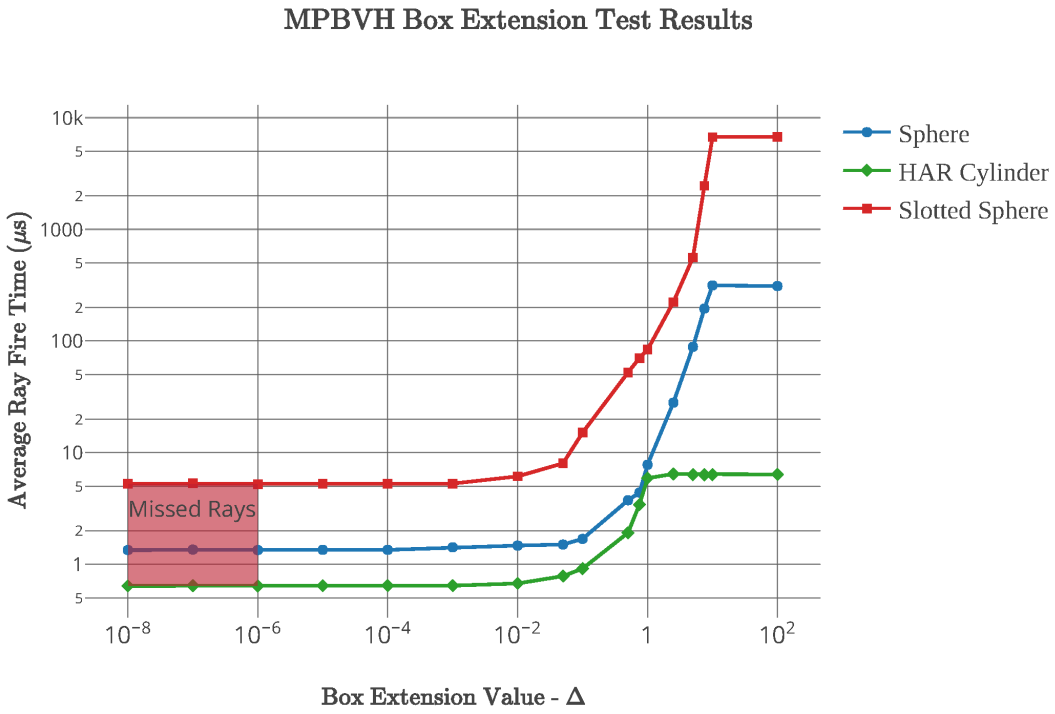


Figure 4.9: A study of ray tracing performance and robustness for various values of the box extension parameter, Δ. For varying faceting tolerances, 600,000 rays were fired at each model. Points in the graph represent the average over all faceting tolerances used.

The biggest geometric problem examined in this work is the ITER model. The largest side of the global bounding box in this model is 4×10^3 cm. Using this as a representative value of l_{max} results in a box expansion value, Δ, of 1.2×10^{-3}. This value is small enough to cause minimal overlap in the bounding boxes and a negligible effect on the BVH performance, as seen in Figure 4.9. For all results discussed in this work Δ was set to 5×10^{-3}.

4.3.1.3 Simplified Surface Area Heuristic

The MPBVH employs a modified version of the Surface Area Heuristic, discussed in Chapter 2, which was developed by the author. This version of the heuristic, dubbed the Simplified Surface Area Heuristic (SSAH), removes estimation of the relative cost between a tree traversal step and a primitive intersection.

This version of the heuristic considers the cost of traversal, C_t, to be small compared to the cost of a triangle primitive intersection, C_i. This assumption is presumably stronger in the MPBVH due to the mixed precision nature of the implementation than for systems in which the hierarchy and primitives are both stored with the same precision. The cost evaluation for a candidate node can be evaluated as seen in Equation (4.13).

$$C = \cancel{C_t} + \frac{SA_L}{SA_P}|P_L|C_i + \frac{SA_R}{SA_P}|P_R|C_i \tag{4.10}$$

$$C_g = \frac{SA_L}{SA_P}|P_L| + \frac{SA_R}{SA_P}|P_R| \tag{4.11}$$

$$C = C_i C_g \tag{4.12}$$

$$\begin{aligned} C^L &= C_i C_g^L \\ C^R &= C_i C_g^R \\ C^L &< C^R \\ C_i C_g^L &< C_i C_g^R \end{aligned}$$

$$C_{simplified} = C_g = \frac{SA_L}{SA_P}|P_L| + \frac{SA_R}{SA_P}|P_R| \tag{4.13}$$

C_t — cost of traversal to child nodes
C_i — cost of primitive intersection check
SA_L — surface area of left child
P_L — primitives contained by the left child
SA_R — surface area of right child
P_R — primitives contained by the right child
SA_P — parent bounding volume
C^L,C^R — left and right child costs, respectively, in a binary tree

Figure 4.10: A form of the simplified surface area heuristic for a binary tree.

By neglecting the traversal cost, the cost of the node can be broken into two components, the cost of the intersection, C_i and the geometric cost, C_g. Because these values are used in a purely relative manner to compare the costs of candidate splits, the constant factor of the primitive intersection can

be ignored during evaluation of the cost. This form of the heuristic is purely a weighted geometric evaluation without estimation of implementation-based value which can vary based on machine architecture, build settings, etc.

4.3.1.4 Surface Root Nodes

In DAGMC, trees exist for each volume and surface in the geometry. Rather than duplicate surface trees for each volume they appear in, one tree is created for each surface. For each volume, its surface trees are joined into a single tree. This avoids duplication of tree information and minimizes the memory footprint of DAGMC. It is critical to DAGMC that the surface intersected is returned as part of the ray's intersection information so that particles can be passed from volume to volume correctly. This information is maintained in the root nodes of surface trees and updated during the BVH traversal process (see Figure 4.11). The same scheme was implemented in the MPBVH, but within the framework of a BVH design for SIMD traversals.

To address surface id tracking during traversal, a specialized BVH node is used to mark the root of surface trees under volume trees. A BVH is created for each set of triangles representing a surface. These surface trees are then joined into a single tree that represents a volume as depicted in Figure 4.11. This avoids duplication of surface trees for each volume. It also allows the return of which surface is intersected, allowing for optimized transport of a particle to the volume on the other side of that surface via the topology represented by the mesh hierarchy described in Section 4.2.1. Embree allows for a similar model in which surfaces are represented as "geometries" rather than surfaces. Those geometries can be grouped into "scenes" rather than surfaces. The creation of a special node to represent the root of a surface tree was implemented to provide this information. This node type is an extended node reference which contains additional

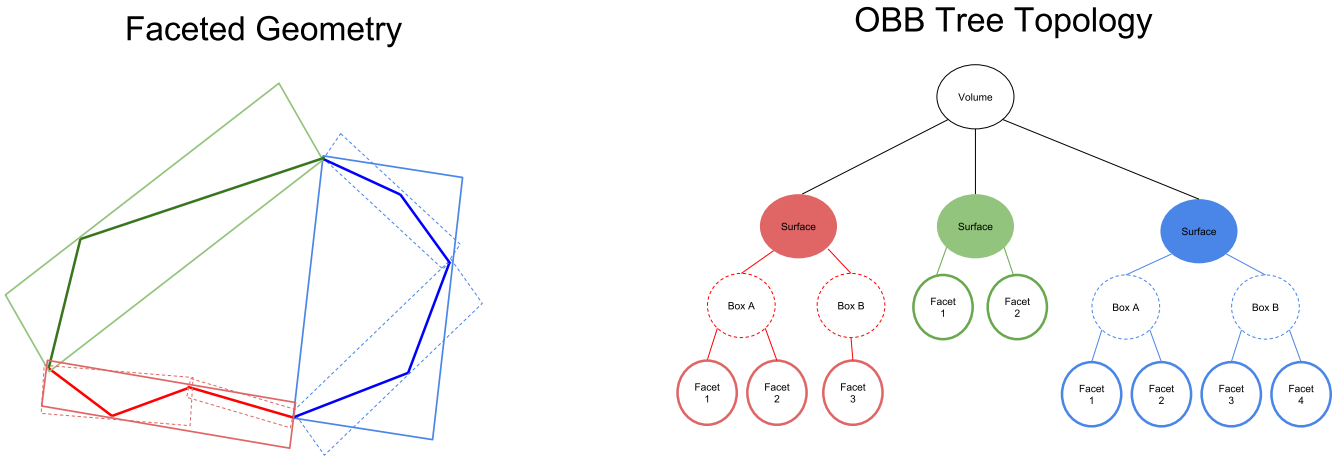


Figure 4.11: Left: Representation of a faceted DAGMC geometry. Right: The corresponding BVH structure for that volume. Surface sub-trees are shared between volumes to avoid duplicate data.

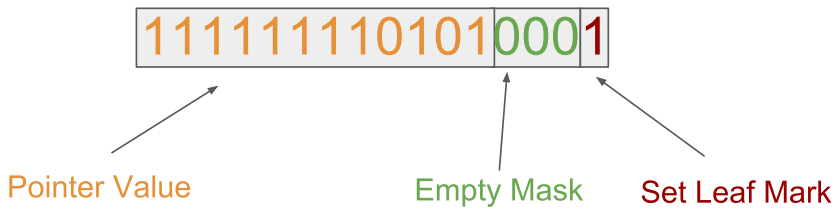


Figure 4.12: Visual representation of set leaf encoding using integer-based pointer values.

information about the surface the tree is constructed around. In particular, this node stores the id of the surface and its sense information relative to the volumes on either side of the surface. This sense information is used to automatically adjust the returned triangle normals for any intersections with that surface, and avoids the duplication of triangle connectivity that was necessary in the EmDAG implementation.

In a similar manner to how leaf nodes are identified during traversal (see Section 2.5.5.1.2), the surface root node reference is identified by toggling the least significant bit in the reference's integer value to recover the location of the node's location in memory. Because only this first bit is altered, the node can be differentiated from leaf nodes containing primitives. Information about the current surface being visited and the sense information is updated on the traversal ray and used to set any hit information on the ray. The node is then pushed onto the stack and the traversal continues relatively uninterrupted. The depth-first traversal of the hierarchy protects against any invalid surface information being set on the ray as the all nodes underneath that surface will be visited before moving on to another surface, at which point the traversal ray information will be updated.

This node type also provides the ability to avoid duplication of triangle connectivity to represent sense-adjusted triangle normals as was required

in the EmDAG system. A MOAB look-up of this sense information after returning ray hits is also a viable option, but it would be relatively expensive in comparison to adjusting triangle normals on the fly.

4.3.2 Robustness Criterion

The the criterion for a robust ray tracing kernel as a replacement for MOAB's OBB tree are as follows:

- every ray query should return the **same triangle id** as MOAB's BVH implementation
- every ray query should return the **same intersection distance** as MOAB's BVH

Many unit tests are used to ensure watertight box intersection using the slab method [28] in single precision, but the mixed precision scheme is best tested using DAGMC models. In MOAB, the Plücker ray triangle intersection method is used to provide watertight intersections with triangle meshses [42]. The same algorithm was applied in the MPBVH to ensure that simulation results using both the MPBVH and MOAB BVH are consistent. Several tests in the kernel fire rays from the center of single-volume DAGMC models with rays directed at the vertices, edges, and center of each triangle in the model. These tests ensure that the same triangle is hit and the same intersection distance is returned in either system. Missed rays and lost particles are also monitored in testing the ray fire performance and in particle transport, though none were found outside of the controlled box extension study in Section 4.3.1.2.

4.3.3 Ray Fire Performance

The same set of ray fire performance tests were run as in Section 4.2.2. As shown in Figure 4.3, the MPBVH shows a significant improvement

compared to the current implementation in MOAB. In addition, the timings are comparable to EmDAG's performance, despite returning double, rather than single, precision intersections. This indicates that attempts at providing accelerated ray tracing for engineering analysis by traversing an expanded, reduced precision BVH were quite successful.

4.4 Simplified Particle Tracking

The MPBVH kernel has been coupled to DAGMC as a numerically based particle tracking tool using a simple algorithm for determining *Next Surface* queries and *Point Containment* queries. The current particle tracking algorithm in DAGMC [48] applies additional logic related to previously hit triangles to avoid lost particles and infinite loops in particle histories. This tracking algorithm uses structures known as the **MBRay** (short for MOAB Ray) and **MBAccumulatorRay**. Both of these structures (depicted in Figure 4.13) are critical to the use of the MPBVH as a robust particle tracking tool.

```
def next_surface(current_volume, point, direction):

    # create a ray with infinite length
    ray = MBRay(current_volume, point,direction,)

    # if the ray orientation is set, ignore
    # hits which opposed the triangle's
    # sense-adjusted normal
    if ray_orientation == 1 :
      set_filter(backface_cull)
    # if the orientation is not set, remove the filter
    else:
      unset_filter()

    # fire the ray
    fire_ray(ray)

    # if the ray missed and overlaps are allowed in the model
    # fire a ray in the opposite direction
```

```
    if ray.surfID == -1 and overlap_thickness != 0.0: {
      small_val = overlap_thickness * 0.01
      ray.direction *= -1
      # apply some small space at the beginning of the ray
      # where hits are ignored
      ray.tnear = small_val
      # set the ray distance to the overlap thickness
      ray.tfar = overlap_thickness

      # unset the ray filter regardless of ray orientation
      unset_filter()

      # fire the ray
      fire_ray(ray)

      # update ray value to zero for this type of hit
      ray.tfar = 0

    # if we have a hit at this point, set the returned
    # surface ID and distance
    if ray.surfID != -1:
      next_surf = ray.surfID
      next_surf_distance = ray.tfar

    # otherwise the particle is lost,
    # set return information accordingly
    else:
      next_surf = 0
      next_surf_distance = ray.tfar

    return next_surf, next_surf_distance

def backface_cull( &mbray ):

    # if the ray direction opposes the hit triangle's
    # normal vector, reject the ray hit
    if dot_product(mbray.dir, mbray.norm) < 0.0:
        ray.surfID = -1

    return
```

Algorithm 4.1: Algorithm used in *Next Surface* queries.

The algorithm in Figure 4.1 avoids many logical checks and the classification of hit locations in the current DAGMC algorithm. It also avoids

mesh database look-ups of adjacent triangles for edge and node hits. This used to be necessary due to the ray history passed into the *Next Surface* query, causing rays to sometimes become stuck in infinite or near-infinite loops. The ray history is an optimization to avoid repeated intersections with the same triangle. By abandoning this construct and relying more on the performance of the underlying ray tracing kernel, the algorithm can be simplified to always return the nearest intersection. Infinite loops in which particles oscillate between volumes on a surface boundary are addressed by returning exiting intersections only.

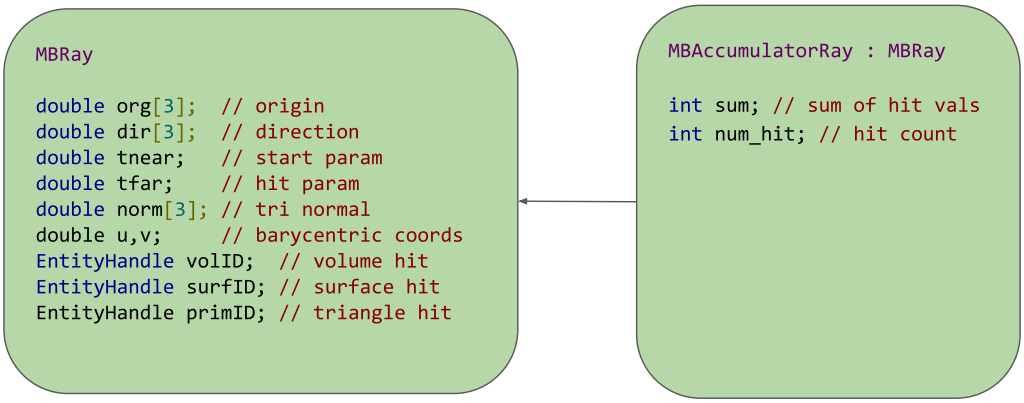


Figure 4.13: Ray structures used to communicate hit information between the MPBVH and DAGMC. The MBAccumulatorRay adds additional attributes to the MBRay for tracking of hit counts and orientations with respect to triangle normals.

The conditions laid out for a robust tracking algorithm by B. Smith are as follows [48]:

a) Particles cannot become lost

b) Infinite loops cannot occur

The algorithm presented here addresses **a)** by using the watertight Plücker intersection test and extending ray boxes to ensure the correct tri-

angle is checked for intersection. Part **b)** is addressed by firing a ray in the opposite direction if the initial ray does not find an intersection. The scenario of concern being that in which the particle is inside a self-intersecting or overlapping volume. To avoid a zero-distance hit when firing in the opposite direction, a small value is also set as the ray's near-side parameter. In traversal, any hits closer than this value will be ignored. The ray's length is also set to the allowed overlap thickness. Any intersections beyond this distance are ignored as well. Thus an exiting intersection in the overlap region can be found and a zero-distance intersection will be returned from the *Next Surface* query.

The MPBVH provides the capability to set callback functions used to filter and validate ray intersections, as demonstrated by the *backface_cull* function above. These filter functions can also be used to accumulate ray hits, which is critical to implementation of a robust point containment algorithm based on the MPBVH, shown in Figure 4.2.

```
def point_contained(volume, point, direction):
    # result values : 0 — outside; 1 — inside

    # start with an outside value
    result = 0

    # create a ray
    ray = MBRay(volume, point, direction)

    # accept ray hits regardless of
    # orientation w.r.t. triangle normal
    unset_filter()

    # fire the ray
    fire_ray(ray)

    # calculate the dot product of
    # ray direction and triangle normal
    dot_prod= dot_product(ray.dir, ray.norm)

    # if a hit is found,
    if not ray.missed and dot_prod not 0.0:
```

```
        if dot_prod < 0.0:
            return INSIDE
        else:
            return OUTSIDE

    # if the ray is tangent, the overlap thickness is
    # non-zero, or it missed -
    # count hits and sum orientations
    if dot_prod is 0.0 or overlap_thickness is not 0.0:
       aray = MBAccumulatorRay aray(volume, point, direction)

       set_filter(count_hits)
       fire_ray(ray)

       # if no hits were found
       # point is outside
       if aray.num_hit == 0:
           return OUTSIDE

       # fire ray in negative direction
       # and reset length
       aray.dir *= -1
       aray.tfar = inf;

       fire_ray(aray)

       # if no hits were found
       # point is outside
       if aray.num_hit == 0:
           return OUTSIDE

       # check the value of the sum
       if 0 > aray.sum:
           return INSIDE
       else:
           return OUTSIDE

def count_hits( &aray ):
    # increment number of hits
    aray.num_hit++
    # add value based on sign of dot product
    aray.sum += sign(dot_product(aray.dir, aray.norm))

    # always reset ray hit values
```

```
aray.continue = True
```

Algorithm 4.2: Algorithm for point containment within a volume.

In this method the ray is intersected with the volume and a hit is evaluated as either entering or exiting based on the dot product calculation with the triangle normal. If the first ray fire finds no intersection and any of the following conditions are true:

- the ray misses the volume
- the ray direction is tangent to the intersected triangle normal
- the overlap tolerance is non-zero

then ray hits are accumulated by firing a **MBAccumulatorRay** in both the positive and negative direction. All hits are registered on the ray as exiting or entering based on the dot product calculation of the direction and triangle normal (see the *count_hits* function). If no hits are found, then the point is outside of the volume. If hits are found, the sign of the summation value indicates whether more exiting or entering hits were found. If more exiting than entering hits are found, then the particle is considered to be inside the volume. Otherwise it is considered to be outside of the volume. This algorithm was applied to all of the simulation results presented in this chapter without lost particles or causing infinite loops.

4.4.1 Simulation Results

4.4.1.1 Simple Test Cases

The MPBVH kernel was applied as the ray tracing kernel for the same set of transport test cases as the EmDAG implementation. Table 4.3 shows the results of tests. The MPBVH implementation performs comparably to

EmDAG for the cube and nested cubes cases, but is significantly slower in the sphere and nested spheres cases. This is expected based on the difference in pure ray fire times from Section 4.3.3

Test Model	**MCNP**	**DAG-MCNP**	**EmDAG-MCNP**	**DAG-MCNP (w/ MPBVH)**
		run time (min)/ ratio to MCNP		
Sphere	2.93 / 1.00	25.13 / 8.58	4.73 / 1.61	9.38 / 3.2
Cube	5.03 / 1.00	10.56 / 2.10	5.80 / 1.15	5.46 / 1.08
Nested Spheres	4.35 / 1.00	50.82 / 11.68	7.94 / 1.82	9.88 / 2.27
Nested Cubes	4.73 / 1.00	9.26 / 1.96	4.35 / 0.92	4.09 / 0.86

Table 4.3: Run time comparison of native MCNP, DAG-MCNP, and DAG-MCNP using the MPBVH over four transport test problems. No lost particles occurred in any of these runs and all results (see Appendix B) match the standard DAGMC implementation exactly.

4.4.2 Production Test Cases

After verifying that no particles were lost in the contrived test cases, the MPBVH implementation of DAGMC was applied to the same FNG model as EmDAG. The results of this simulation can be seen in Table 4.4. In contrast to the EmDAG system, no lost particles were found in the FNG simulation with a volumetric source and all tally results matched the standard DAGMC implementation with reduction in the runtime by nearly a factor of two. Due to the success of the DAGMC MPBVH implementation for the FNG model, this system was applied to several other production models as well. No particles were lost in any of these geometries. The results of the simulations between the unmodified and modified versions of DAG-MCNP were the same for all production models presented in Table 4.4. It is worth noting, however, that the simplified algorithm presented in Section 4.4 was unable to accurately track particles due to the presence unmerged surfaces and unsealed volumes. An identical tracking algorithm

to the one used in the unmodified version of DAGMC was used instead to obtain the same numerical result.

Test Model	MCNP	DAG-MCNP	DAG-MCNP MPBVH
	run time (min)/ ratio to MCNP		
FNG	2.49 / 1.00	9.83 / 3.95	6.48 / 2.60
ATR	3.18 / 1.00	36.16 / 11.37	8.54 / 2.68
UWNR	270 / 1.00	1452.14 / 5.38	505.83 / 1.48
ITER	N/A / N/A	55.46 / N/A	12.64 / N/A

Table 4.4: Runtime comparison native MCNP, DAG-MCNP, and DAG-MCNP using the MPBVH over four transport test problems. No lost particles occurred in any of these runs and all results match the standard DAGMC implementation exactly.

A simple analysis of memory usage for each of the production models was also conducted. While not the focus of this work, it is expected that the single precision values and compact node form of the MPBVH node result in a lower memory footprint in DAGMC's acceleration data structures. The results of this study shown in Table 4.5 verify that this is the case. One might expect that the memory savings is closer to a factor of two for the conversion from double to single precision values, but there are competing factors at play. The use of AABBs rather than OBBs results in deeper trees for these models due to the AABBs limited ability to conform to arbitrary shapes. While still preferable due to the fast computation of ray-box intersections, the use of AABBs results in more total nodes in the trees, and may account for the higher than initially expected memory usage of the MPBVH. Regardless, the MPBVH consistently has a lower memory footprint by 20-25%, resulting in a significant memory savings of 1.2 GB in simulations of the ITER model.

MPBVH Memory Usage	
Test Model	**Ratio to unmodified DAGMC**
FNG	0.71
ATR	0.78
UWNR	0.84
ITER	0.69

Table 4.5: Memory usage of the BVH structures using both MOAB's OBB tree and the MPBVH.

4.5 Limitations and Future Work

As discussed in Section 4.3.1.2, the box extension value may need to be varied depending on the size of the model. The value currently applied in the kernel is suitable for the majority of the models used in DAGMC. The study on this value indicates that it could be increased significantly to retain robustness in models of a larger geometric scale without detriment to the performance of the simulation.

In the future, this extension value could be set on a volume-by-volume basis as intersection distances in DAGMC should never exceed the maximum chord length of a volume before being considered lost. This would limit the performance degradation for models of a large global scale with small, local volumes.

Though this work does not address such applications, time-dependent simulations could update this extension value on-the-fly when updating bounding boxes, though the additional cost of ensuring that all parent bounding boxes are updated appropriately is unclear [55].

4.6 Summary

The MPBVH implementation provides a means of exploiting the performance capabilities of CPUs for single precision values while providing the double precision intersection values required for robust engineering analysis tools such as DAGMC. The demonstration of this capability is enabled by the direct access methods available in MOAB, but the methods described in this work could in theory be applied to other spatial databases with contiguous memory designs.

For the purposes of CAD-based MCRT-t, the MPBVH kernel's design, inspired largely by Embree, allows for a simple yet robust tracking algorithm in DAGMC. These methods have been demonstrated on production models used in verification and analysis applications of DAGMC. The combination of this algorithm and the SIMD-oriented traversal of the hierarchy in single precision provides a reduction in DAGMC runtime by factors of 2-5 depending on the model with no change in the final result.

Chapter 5

High Valence Vertices

High valence (HV) vertices are a mesh feature which cause significant degradation in DAGMC ray tracing performance. The valence of a vertex in a mesh is defined as the number of edges connected to that vertex. *High* valence vertices are defined as vertices connected to an unusually large number of edges. This region, known as a HV region, will typically take on a fan-like shape as seen in Figure 5.1. The geometric origins of HV regions are typically a planar surface intersected with some form of curved boundary condition.

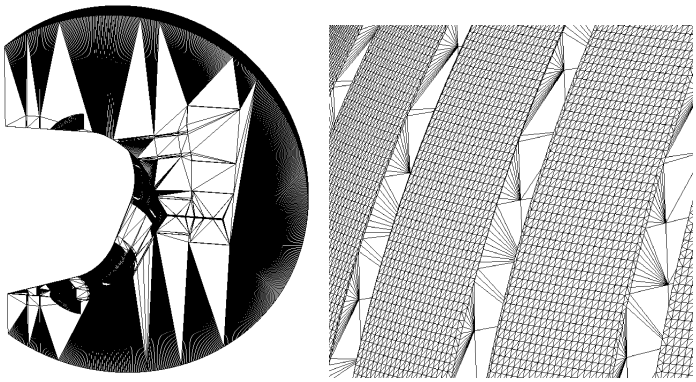


Figure 5.1: Examples of HV vertices in production models.

These regions are commonly generated in the faceting algorithms used to produce DAGMC meshes. This faceting scheme (which comes from ACIS libraries underlying the CUBIT/Trelis geometry engine) is designed to produce the smallest number of triangles possible to represent the model within the representation tolerance specified in DAGMC's surface mesh generation preprocessing. This restriction is favorable to the rasterization process commonly used to display models interactively in CAD GUIs. Fewer triangles are better for the purpose particle tracking in DAGMC as well as long as the geometry is accurately represented. Even the ideal ray tracing acceleration structure queries for a given triangle mesh scale as $O(\log(N))$, and the size of models being analyzed using the toolkit provides motivation to keep memory footprints as low as possible. However, even with fewer triangles undesirable configurations can impede performance as is shown by a set of tests conducted on models generated by this faceting scheme.

5.1 Previous Work

A study conducted by Steve Jackson in 2010 on the performance of the MOAB OBB tree revealed a steep degradation in performance with a decreasing faceting tolerance or an increasing number of triangles [52]. Using a DAGMC-based ray fire test program, the performance of DAGMC's ray fire ability was evaluated for four models. These models include a simple sphere, a notched or slotted sphere, and an outer volume of an ITER model, shown in Figure 5.2.

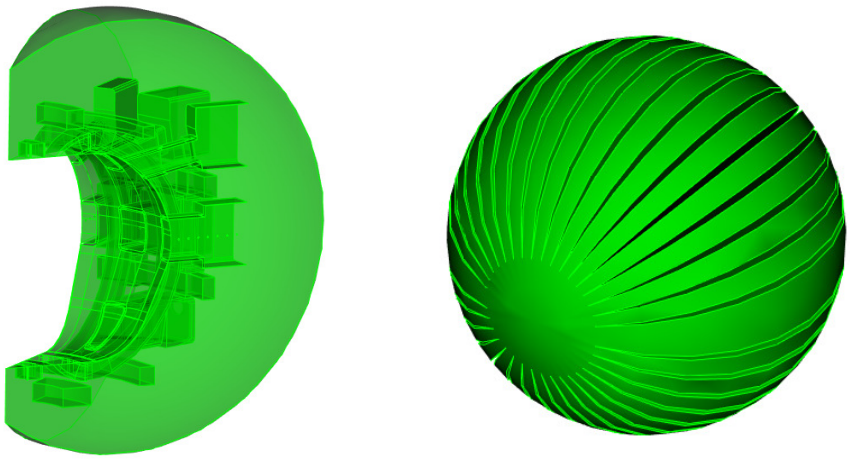


Figure 5.2: Images of the slotted sphere and ITER volume used to perform DAGMC ray fire performance tests with increasing number of triangles or, equivalently, decreasing faceting tolerance.

In each of these tests, the models are tessellated with an increasingly smaller faceting tolerance with the faceting tolerance being defined as the maximum distance between the faceted curve or surface and the geometric curve or surface it resolves. By this definition, the number of triangles needed to represent a model scales inversely with the value of the faceting tolerance. An increase in the number of triangles leads to a more complex nature of the surface mesh in terms of BVH construction and traversal.

The first three models are identical to the ones used to measure ray fire times in Chapter 4. Finally, the faceting of a volume from an ITER model is used as a production demonstration of the effect of HV regions on DAGMC performance (see Figure 5.1).

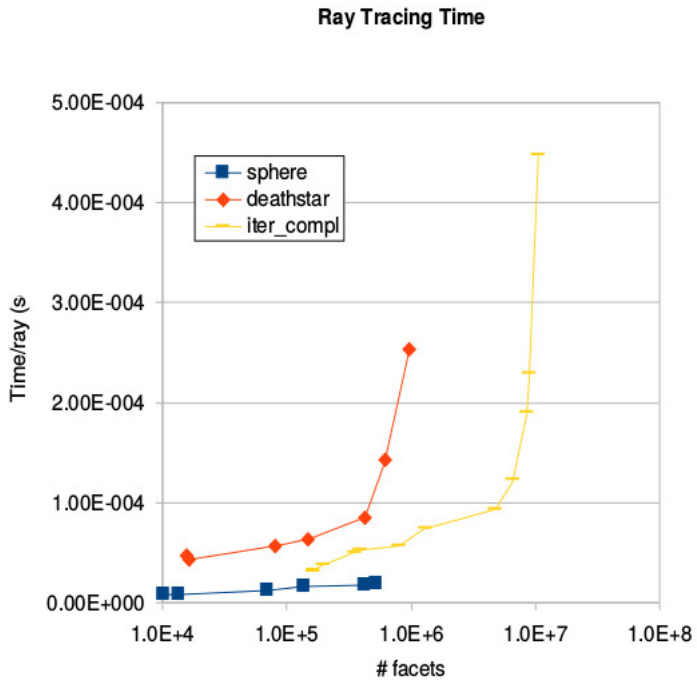


Figure 5.3: Results of MOAB ray tracing performance tests with decreasing faceting tolerance performed by Steve Jackson in June of 2010. Data points represent average time spent in firing a ray for random rays originating at the center of each model[52]. The "deathstar" model is the same as the slotted sphere model used in tests performed by the author.

While the sphere model scales well with a decreasing faceting tolerance, the ITER volume and slotted sphere both have a pronounced increase in average ray fire time with decreasing faceting tolerance. Knowing that both of the latter models contain HV regions, it was postulated at the time that these regions had a significant effect on the scaling.

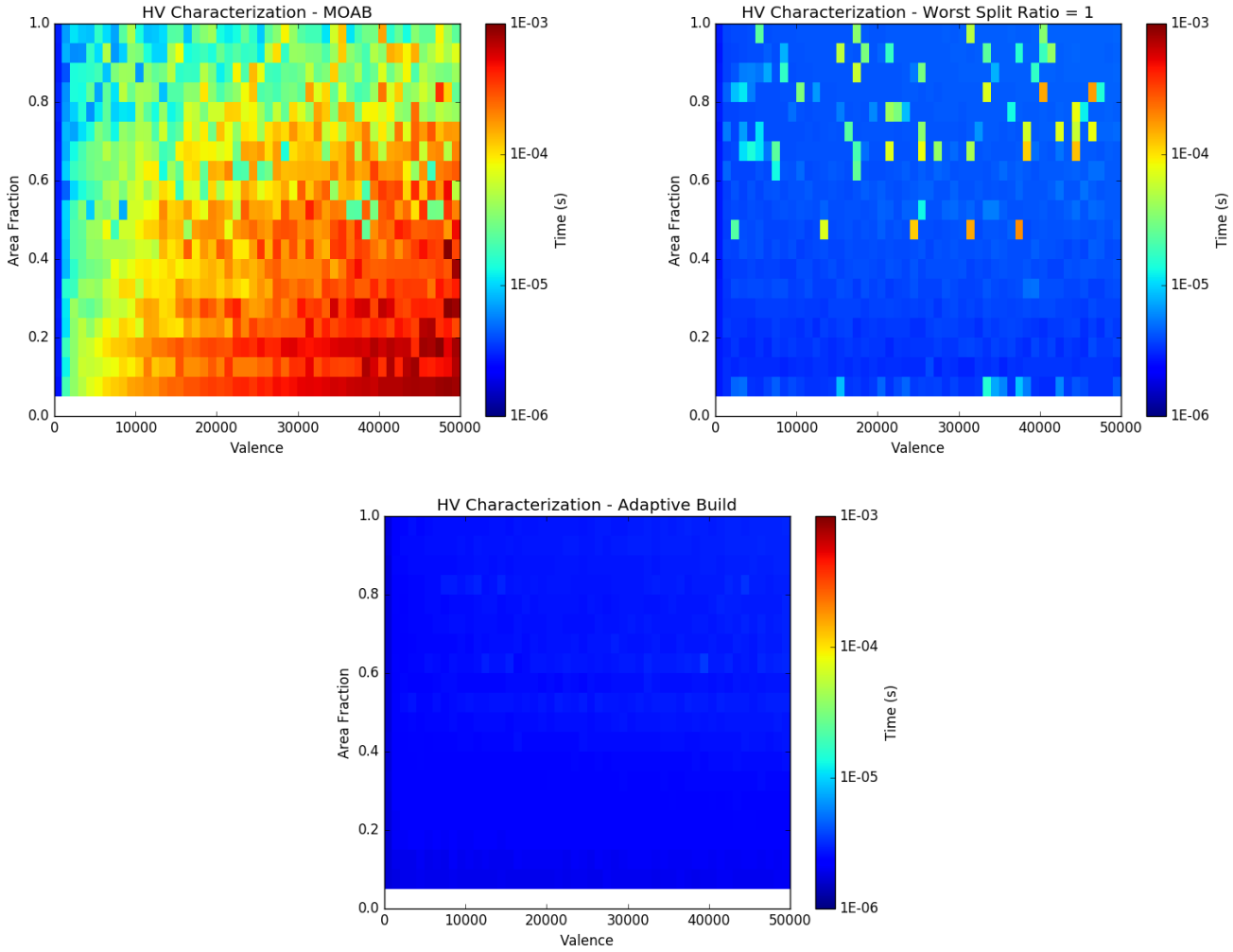


Figure 5.4: HV characterization study results for all MOAB OBB tree implementations. Top Left: Unmodified MOAB results. Top Right: Manual modification of MOAB OBB tree's build settings. Bottom: Results with adaptive construction for HV regions.

5.2 High Valence Characterization Test Framework

In order to isolate a HV region, a test model was manually generated in MOAB with an artificial HV region (shown in Figure 5.5). This mesh is a modified cube mesh centered on the origin in which one of the two-triangle surfaces has been replaced by a more complicated planar surface of triangles including an interior HV region within the face. The HV region was generated by inserting vertices along the diagonal of the interior box and connecting them to the opposing corners of the box. This mesh is generated using two input parameters: the valence of the corner vertices in the interior region and the relative size of the interior region. A parametric study was then performed by varying these two parameters in order to characterize the performance impediment and determine its root cause.

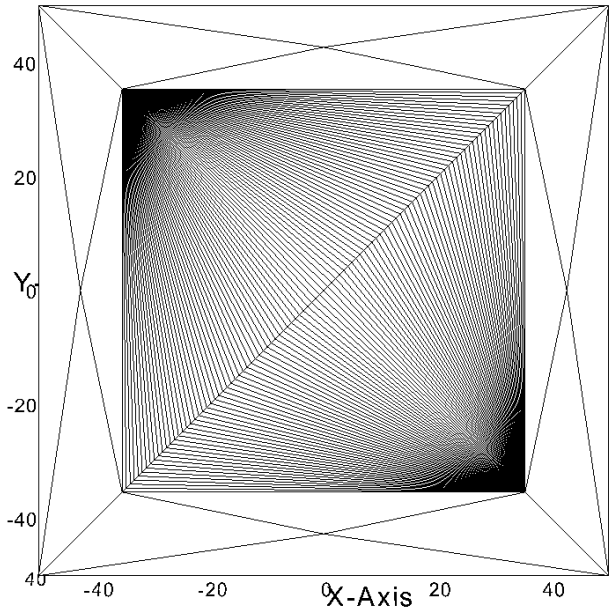


Figure 5.5: Side-on view of the modified cube mesh used to study the HV vertex problem.

DAGMC's ray fire test program was used to construct MOAB's BVH and perform ray queries on this model. This program was used to fire rays with random direction from the origin of the HV test model while biasing the ray directions such that they are always incident upon the modified surface containing the HV region. A parameter sweep was performed by varying the percentage of the surface covered by the HV region as well as the valence of the region. Each test shown in this section varies the valence of the corner vertices from 2 to 50,000 and the relative area of the HV region from 0 to 1. Results of this study provided insights into performance pathologies for various BVH implementations.

5.3 MOAB's BVH

From what is known about the construction of MOAB's OBB-BVH, it was expected that the average ray fire time would be most strongly correlated to the relative area of the HV region, but would also increase with increasing valence of the corner vertices. The initial results of this study, shown in Figure 5.4, meet only one of the expectations however. The ray fire times become far worse with increase in valence, but for a constant valence, the smaller relative area models show a longer average ray fire time than the models with larger relative areas. This suggests that the presence of a HV region in a surface is detrimental to performance regardless of the likelihood that a ray intersects with a triangle in that region, which is counter intuitive. In order to investigate this matter further, a visualization tool for MOAB's BVH was developed to improve the author's intuition about this mesh pathology.

5.3.1 Visualization and Diagnosis

Using the same visitor pattern employed to traverse rays through the hierarchy, each OBB in the HV model tree was converted into a hexahedral element and saved in a VTK mesh format. Each hexahedron representing an OBB was tagged with its depth in the tree as well as the triangle entities it contains, if it is a leaf node. These mesh files can then be used to visualize the hierarchy level-by-level and can be superimposed on the geometric mesh. An example of this OBB visualization for the HV characterization model is shown in Figure 5.6.

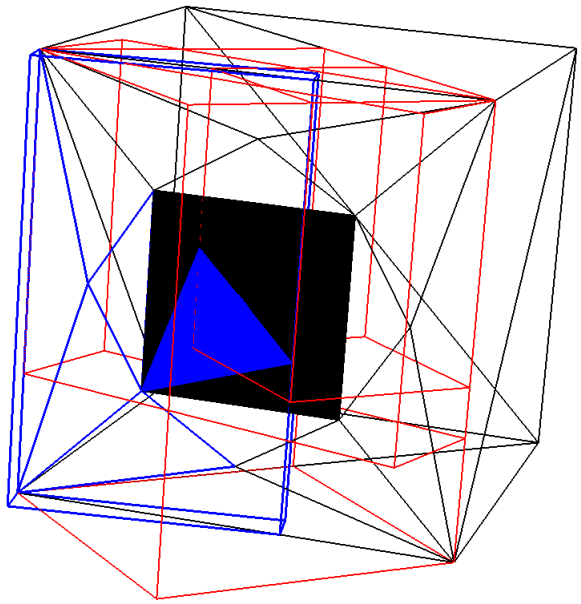


Figure 5.6: View of an OBB containing many HV region triangles as well as other surface triangles. Blue box and triangles: leaf node bounding box and associated triangles. Several thousand triangles are represented in the solid blue region shown here. Red boxes: Other representative OBBs at the same depth in that tree.

Because there are more entities to partition in the HV region, it is expected the deepest levels of the BVH will contain only OBB's bounding triangles in that region. It was expected that leaf nodes of the BVH might contain many triangles of the HV region, causing performance degradation of ray traversal in that many triangles must be checked for intersection. These types of poorly formed leaf nodes shift the complexity of the BVH traversal back toward a linear search - which is sub-optimal. This feature of MOAB's OBB tree were observed, but visualization of the BVH provided the ability to observe another characteristic as well. Many leaf nodes containing large numbers of triangles in the HV region also contained one or two large triangles outside of that region. The inclusion of these large

triangles significantly increases the probability that a hierarchy traversal will visit that leaf node and thus all of the triangles contained by that leaf node. This artificial increase in the HV regions cross-section greatly exacerbates the already poorly created nodes in the tree.

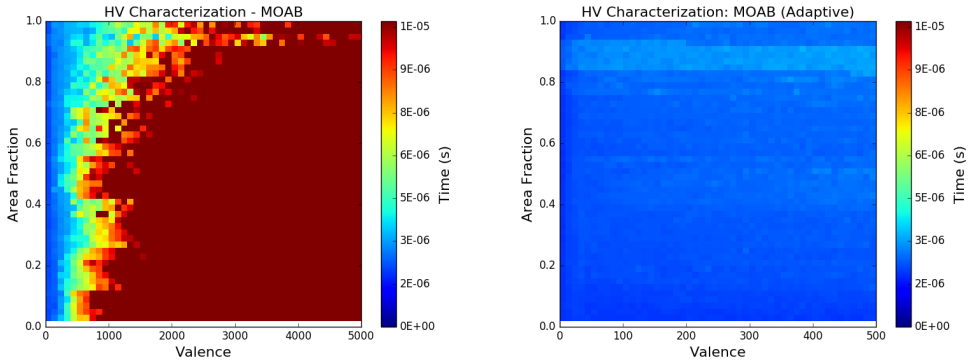


Figure 5.7: A small scale version of the HV study in Figure 5.4 using valences representative of those seen in DAGMC models using a linear scale.

Due to the nature of the entity ratio heuristic used to divide nodes in the tree, as the HV region becomes smaller more triangle centroids are shifted onto one side of the median splitting planes used to divide the nodes which contain portions of that region. If enough triangle centroids are on this side of the plane, then the cost of the entity ratio evaluation will exceed the preset upper limit of the cost (0.95 in MOAB) and the construction process will declare that node a leaf node. The settings governing this split process can be altered in MOAB, and changing the worst split ratio to 1.0 significantly improves the average ray fire time in the HV characterization study, as seen in Figure 5.4. While the values used for this study are helpful in understanding the large-scale effects on MOAB's BVH algorithms, a small-scale version of this study was also performed using a valence value more representative of those seen in typically DAGMC models. The results of this study exhibit the same trends

as the large scale and provide information on what the true performance degradation might be in DAGMC simulations.

As expected, the use of this setting largely removes the degradation in performance caused by the HV region by forcing the continued splitting of entities in the tree where large leaf nodes would have been created before. This demonstrates that altering this setting works well for this mesh feature, but it would be detrimental to the memory footprint of the overall hierarchy in the general case, causing portions of the OBB tree to be deeper than necessary.

5.3.2 Adaptive BVH Construction

One of the benefits to having a BVH tool which is part of a mesh database like MOAB is the ability to query for more information about the mesh when constructing hierarchies like the BVH. This information has been used to detect HV regions in the mesh and adapt BVH construction to improve the hierarchy quality in these areas.

5.3.2.1 High Valence Detection

A simple algorithm is used to determine whether or not the entities inside a leaf node are part of a HV region. Any vertex connected to more than a user-defined proportion, α, of the entities in a given leaf node will be considered a HV region, indicating that an alternate build method or build settings should be applied.

```
def detect_hv_region(triangles, alpha):
    assert(a <= 1.0 and a > 0.0)
    connectivity = get_connectivity(triangles)

    for vertex in connectivity:
        # get all entities of dim 2 adjacent to the vertex
        adj_entities = get_adjacencies(vert, to_triangles)

        # determine the number of non-adjacent entities
        overlap = triangles - adj_entities
```

```
    if size(overlap)/size(triangles) < 1.0 - alpha:
        return true
return false
```

Algorithm 5.1: Algorithm for detecting HV regions.

5.3.2.2 Implementation

First, MOAB's BVH constructor was modified to tag poorly formed leaf nodes in the tree. This option will apply a tag to any leaf nodes which contain more entities then specified in the settings for the tree. These nodes are then revisited for further refinement later in the build process.

For handling of these poorly formed leaf nodes, a new class was created in MOAB named the *BVHRefiner*. This class visits each leaf node tagged by the build process to determine if further refinement is appropriate based on the available set of mesh features it is capable of adapting to. For each mesh feature added to the BVH Refine class, a detection method and build method is required. This class applies these detection methods and altered build methods to resolve these leaf nodes into improved regions of the tree. The intent of this design was to support the detection and adaptation to other pathological mesh features in the future.

In the HV case, the refiner class was instructed to search for any vertex connected to more than 80% of the entities in that particular leaf node. If this condition is met, the leaf node is declared part of a HV region and an altered build method is applied. For the HV case, this build method has been established by the characterization tests as the standard build algorithm with the worst case splitting ratio set to the maximum value of 1.0.

5.3.2.3 Application to the HV test model

This adaptive build method was applied to the HV test model as before. The results of this test can be seen in Figure 5.4. Using the adaptive method, the average ray fire time is consistent across both valence and relative area of the HV region and shows none of the stochastic behavior seen when manually altering the tree's global build settings.

5.3.2.4 Application to Production Models

This method was applied to several DAGMC production models in order to determine its effect on simulation run times during transport.

Model	HV regions	Run Time reduction	Build time increase	HV Leaf Visit Relative Frequency
FNG	514	24.2%	18.5%	3.83 %
ATR	755	< 3%	22.9%	0.07 %
UWNR	496	< 2%	< 5%	3.87 %
ITER	3522	29.4%	5.2%	7.34 %

Table 5.1: Results of run time reduction in several DAGMC production models when applying the BVH refinement method.

As seen in Table 5.1, the ATR and UWNR problem run time is decreased only marginally, but in the ITER and FNG models the run time is reduced by $\approx \frac{1}{4}$. Given the outcome of this study it might seem that the high valence regions are visited more often in the ITER and FNG models, but information gathered during simulation using MOAB's mesh tagging interface suggests otherwise. A sweep of the high valence parameter, α, for these models is shown in Figure 5.8 which reflects the performance impact shown in Table 5.1. It also describes the relationship between the simulation run time and the high valence parameter value in the ITER and FNG models. Even for a relatively high α parameter of 0.9, a significant reduction in the run time is seen in these models. As the value decreases,

the run time is futher reduced to nearly 30% of the maximum run time in the study. The large change at a high value of α indicates that even coarse detection of high valence regions has a measurable impact on performance.

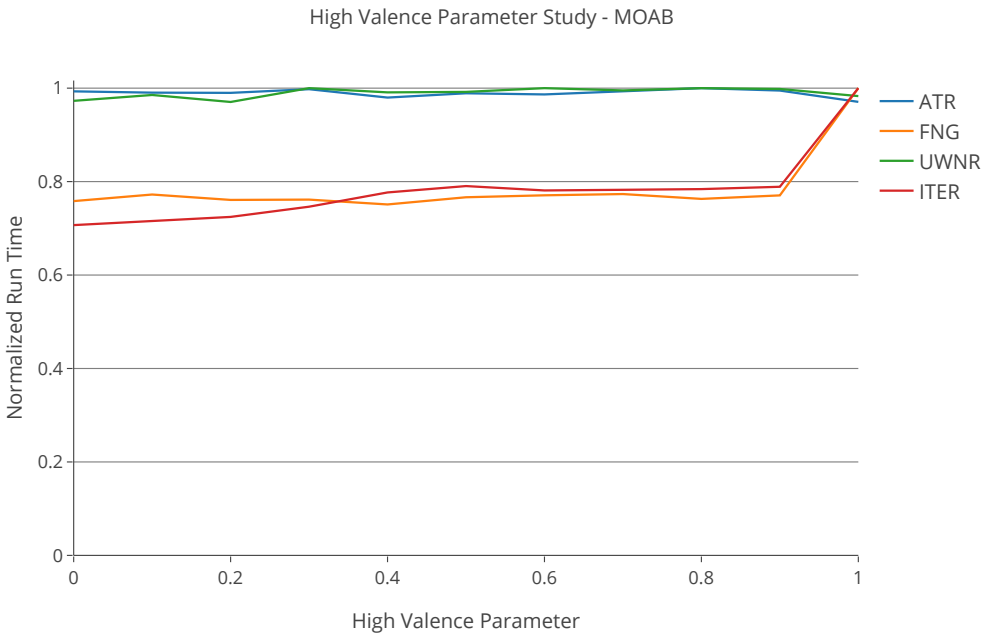


Figure 5.8: Variation in run time for different HV detection parameters on a representative set of DAGMC models using the adaptive build method in MOAB.

The information on the frequency of HV leaf visits in Table 5.1 was collected on regions of the triangle mesh identified as high valence during BVH construction as well as simulation. Leaf nodes in MOAB's OBB Tree determined to be part of high valence regions with an HV parameter, α, equal to 0.5 were tagged. A counter was incremented each time one of these tagged HV leaf nodes was visited in simulation, along with counters for the number of total nodes visited and the total number of leaf nodes visited. The adaptive building method was not applied to these simulations to

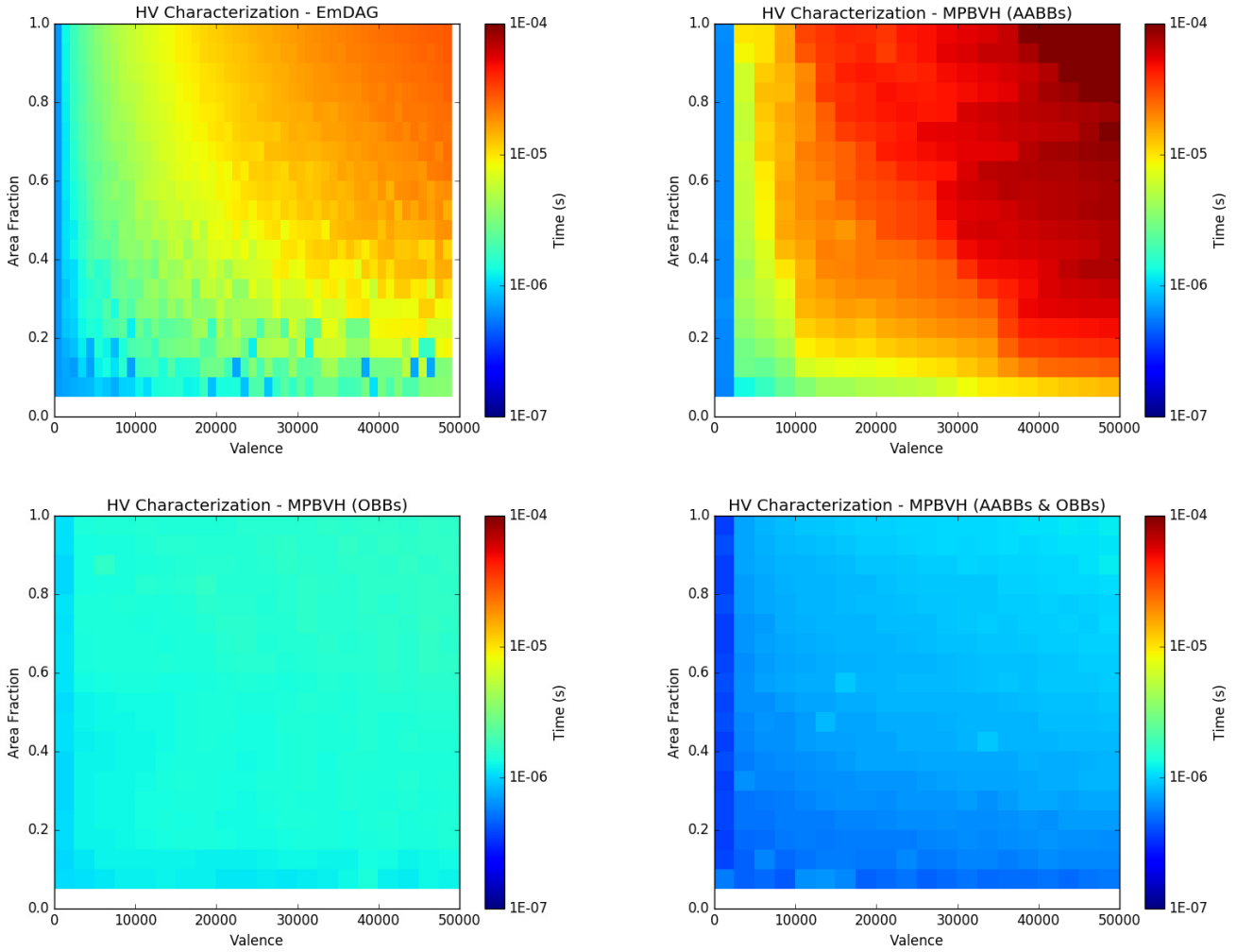


Figure 5.9: HV characterization results for all SIMD-enabled ray tracing kernels. Top Left: Results of the HV study for EmDAG. Top Right: Results of the HV study using the MPBVH with AABBs. Bottom Left: Results using the MPBVH with OBBS. Bottom Right: Results for the MPBVH with an adaptive build method which applies OBBs in HV regions.

avoid biasing the values for leaf node visits. This lends some insight into the impact of HV regions on DAGMC simulation performance. It is interesting to note that, in all of these models, HV leafs are visited rather infrequently, yet they still have a significant impact on the performance of the simulation. This isn't surprising given the severity of the ray fire performance degredation seen in Figure 5.4.

5.3.3 Embree's Ray Tracing Kernel

For all of the tests and simulations performed in Chapter 4, EmDAG was much faster than DAGMC. Using the same ray fire test program in EmDAG, the HV characterization study was performed with somewhat surprising results.

5.3.4 High Valence Characterization

The same test described in Section 5.3 was performed using the ray fire test program compile with using EmDAG to fire rays. A different behavior was expected when using Embree to construct the underlying hierarchies due to its use of the SAH. Figure 5.9 contains the results of this test run. Using the Embree kernel, the performance degrades in direct proportion to both valence and the relative area of the HV region. This data presents the behavior originally expected of MOAB's BVH. Unlike MOAB however, Embree's use of the surface area heuristic prevents the artificial increase in probability of visiting triangles in the HV region. Larger triangles adjacent to this region are more readily separated from the leaf due to the additional cost they add by increasing a bounding box's surface area and in turn its evaluated cost. Separation of triangles in the HV region itself is still difficult when restricted to axis-aligned candidate split planes.

While this data provides evidence that HV regions are problematic for production-level ray tracing kernels, it is difficult to employ any adaptive

tree construction techniques. Without access to the underlying mesh representation of the triangle surface, it is difficult to detect HV regions as implemented in MOAB's OBB tree. This is both a strength and weakness of Embree's design in that it can build highly efficient data structures using only primitive point and connectivity data, but it has no infrastructure in place to interrogate the mesh more deeply. Methods for detecting and adapting to these HV regions are addressed using the MPBVH, a system with similar characteristics but with more access to the mesh interface. Due to the similarity in design and use of AABBs, the MPBVH will act as something of a surrogate for the EMDAG in terms of its response to HV regions.

5.4 Mixed Precision Bounding Volume Hierarchy

5.4.1 High Valence Characterization

The MPBVH kernel created by the author has a similar trend to Embree's kernel where ray fire times increase proportionally to both the valence and size of the HV region as seen in Figure 5.9. The author's BVH applies a modified form of the surface area heuristic as described in Section 4.3.1.3. Near HV regions this heuristic is able to separate large triangles from the HV region in the same way as the unmodified surface area heuristic from Section 2.5.1, but it fails to create a well-structured tree for triangles that are part of the HV area.

5.4.2 Visualization and Diagnosis

Both Embree and the MOAB SIMD BVH have the shared characteristic of maximum leaf size of eight primitives due to the way the leaves are

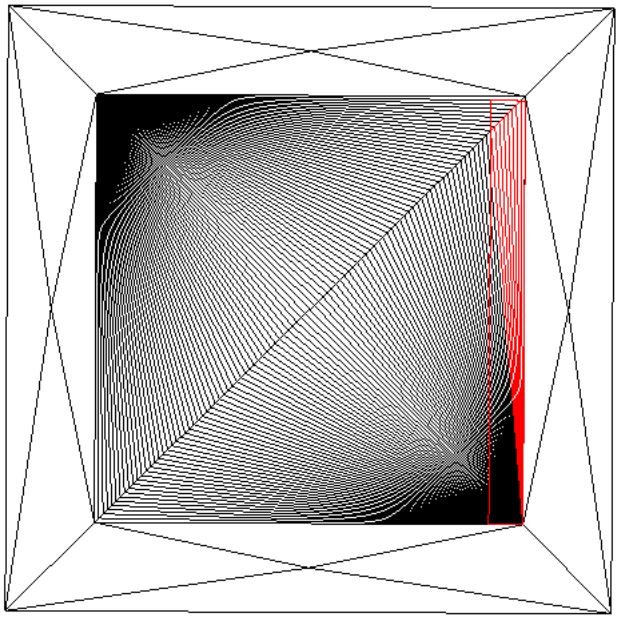


Figure 5.10: MPBVH Axis-Aligned leaf node for the HV test model with area fraction 0.5 and a valence of 100.

encoded. This removes the possibility that large numbers of triangles in leaf nodes as the cause of performance degradation, which was the case for MOAB's OBB Tree. For Embree and the MPBVH, the largest cause of performance degradation is overlapping regions of the leaf nodes.

Because both of these implementations use AABBs, bounding of high aspect ratio, off-axis triangles results in bounding boxes with a considerable amount of empty space in them. An example of such a leaf can be seen in Figure 5.11. In HV regions, this results in the same space being occupied by a high number of leaf bounding boxes. If a ray is fired into these overlapping leaves, the end result is a large number of leaf nodes, and in turn, triangles, visited.

To alleviate the effect of the overlapping AABBs in this region, OBBs

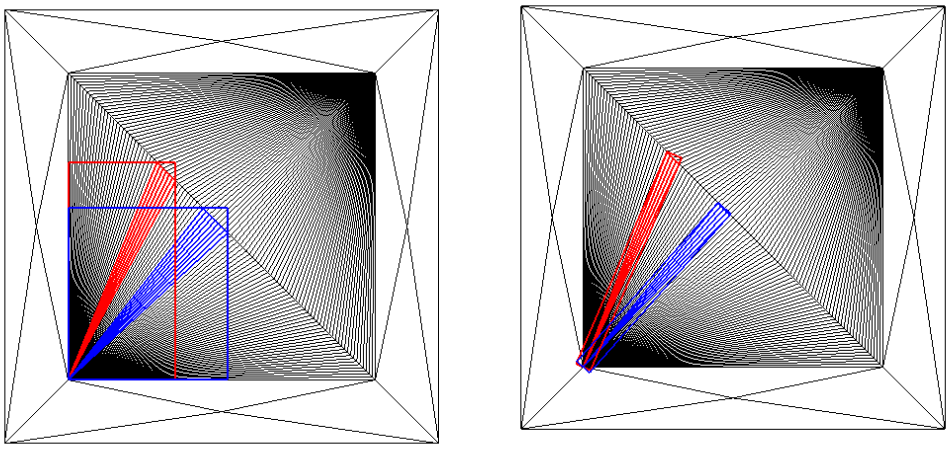


Figure 5.11: Visualization of both axis-aligned and oriented boxes surrounding low aspect ratio, off-axis triangles.

were implemented as part of the MPBVH. Figure 5.9 shows the results of the HV characterization study using only OBBs. Because the simplified SAH heuristic is able to separate large triangles exterior to the HV region into different leaf nodes and the maximum leaf node size is predetermined, scaling very similar to the MOAB OBB implementation after HV refinement is applied can be seen, though the overall speed is somewhat improved due to the single precision implementation.

5.4.3 Adaptive BVH Construction

5.4.3.1 OBB Implementation

The same co-variance method used in MOAB to construct OBBs was applied in the MPBVH [61], but the storage technique used differs in order to accommodate the SIMD programming involved in the BVH traversal. OBB nodes are stored as scaled, affine transformations of the global problem space and the box's lower left corner is stored as a scaled reference point

in the oriented coordinate system. When a ray is intersected with a node it is transformed and scaled at the same time, but separately, for each of the four OBBs the node represents. A reciprocal direction is then calculated with respect to the oriented axes of each box. After this transformation, intersection values and distances are then returned in the same way they are calculated for a node of AABBs [60].

As discussed in Chapter 2, OBB nodes contain additional information in the form of the affine space transformation and are more computationally expensive when computing ray transformations. Some of this computational cost is avoided by incorporating the spatial scaling of the node into both the affine space and reference point to avoid the additional computational cost of scaling the parametric values of the ray intersection later in the calculation.

5.4.3.2 OBB Ray Fire Testing

Tests of the raw ray fire speed using only OBBs were performed using the MPBVH, which can be seen in Figure 5.12. These tests were conducted primarily as verification that the exclusive use of OBBs results in a slower ray fire times.

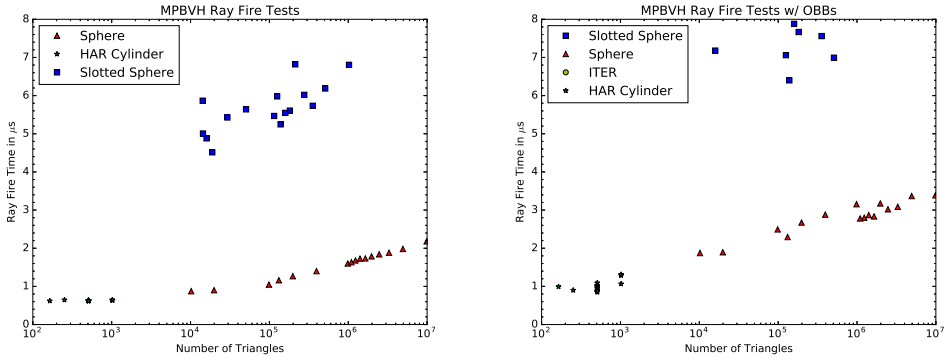


Figure 5.12: Ray fire tests on three representative DAGMC volumes. Each data point represents the average ray fire time for 600k randomly directed rays from the origin of each volume. Left: MPBVH with AABBs only. Right: MPBVH with OBBs only.

On average, the ray fire times using OBBs only were about two times slower than those using only AABBS. The combination of the MPBVH leaf examination in HV areas with the slower ray fire times of OBBs in the common case leads to the conclusion that OBBs can be beneficial to ray fire times, but only if they are used in regions of the model where sibling AABBs contain large overlaps for these pathological mesh features.

5.4.3.3 A Mixed AABB/OBB BVH

The solution to HV regions when using the SIMD BVH kernel proposed by the author is to apply OBBs only in regions determined to be HV using the same detection method described in Section 5.3.2.1. Large leaf nodes being split into smaller nodes will apply oriented bounding boxes to contain entities if the set of triangles is determined to be a HV region using the same method discussed in Section 5.3.2.1 for MOAB's OBB tree. The results using the mixed tree with OBBs applied in HV regions are shown in Figure 5.9. A small-scale of this study is also provided in Figure 5.13 for valences more commonly seen in DAGMC models. The

mixed tree implementation shows some slight degradation in speed with both area fraction and valence, but overall timing is significantly better, with the worst case timing still beating out the best case for the OBB-only characterization test. This indicates that the AABBs are accelerating intersections higher up in the tree while the OBBs resolve the HV region with minimal overlap, resulting in an overall improvement in the run times for these regions.

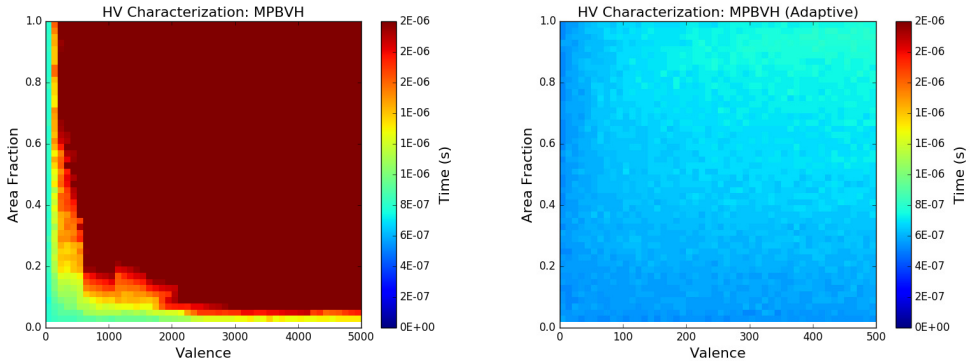


Figure 5.13: A small scale version of the HV study in Figure 5.9 using valences representative of those seen in DAGMC models using a linear scale.

To support a tree containing both AABBs and OBBs, additional encoding of interior nodes is required so that the appropriate node intersection methods are applied during traversal. Axis-Aligned and Oriented nodes are identified using the two remaining bit configurations available for node definitions by setting the appropriate bits in the node reference objects. These node masks are stripped from the node reference's integer value to retrieve the pointer to the node objects themselves with little added overhead in the traversal process.

5.4.3.4 Application to Production Models

The Mixed MPBVH was applied to the same set of production transport tests as the AABB MPBVH in Section 4.4.2. Timing results of these simulations can be found in Figure 5.14.

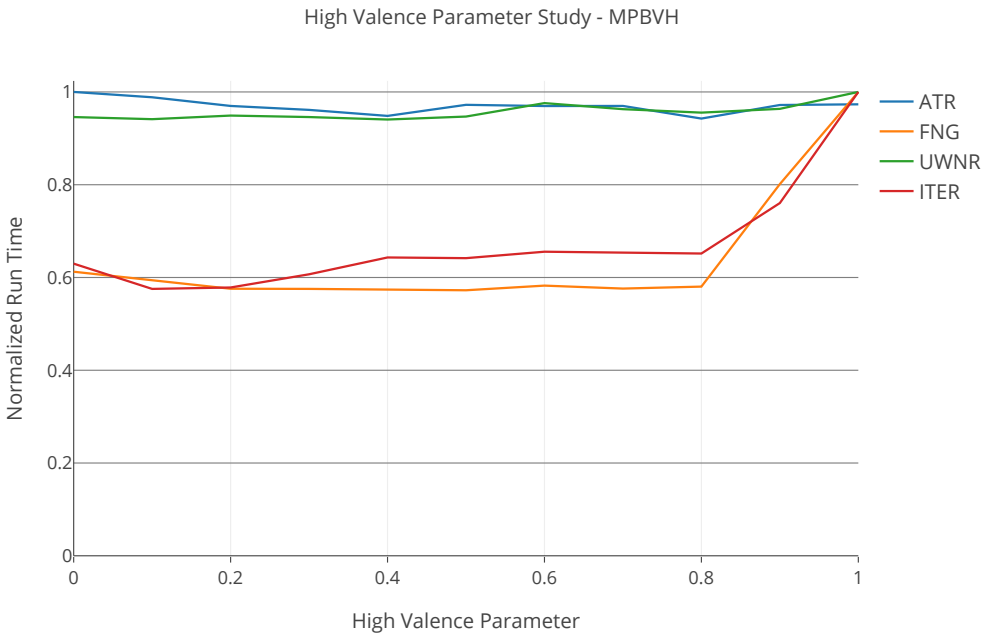


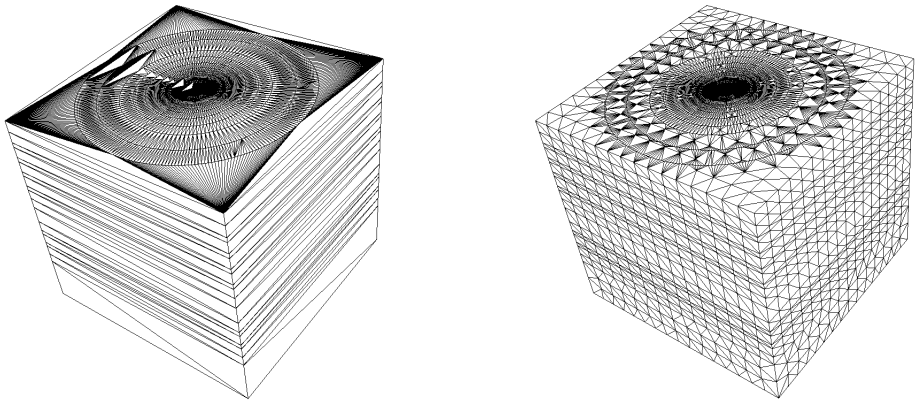
Figure 5.14: Variation in run time for different HV detection parameters on a representative set of DAGMC models using the MPBVH.

The application of OBBs in HV regions makes little difference in the simulation run times of ATR or UWNR, but in the FNG model and ITER model the run times are reduced by up to 40% with little additional build times in the BVH and minimal increase in the memory footprint by replacing some of the AABBs with OBBs. This reflects the behavior seen in results of Table 5.1 for simulations using MOAB's OBB tree. A study on the HV leaf visit frequency was not performed here due to a limited

ability to mark HV leaf nodes in the MPBVH system.

5.4.3.5 Alternative Surface Meshing Methods

In addition to the faceting tolerance discussed in Section 2.4, other parameters can be applied to the tessellation process in CUBIT/Trelis. One such parameter is the length tolerance. The length tolerance places an upper limit on the edge length for any triangle in the resulting surface mesh. Passing this parameter to the tessellation algorithm along with a faceting tolerance tends to create more triangles with an aspect ratio ≈ 1. It also tends to generate more triangles, particularly when the length tolerance specified is smaller than the characteristic length of surfaces in the model. Limited edge lengths also have the effect of reducing the area of HV regions as seen in Figure 5.15 where both the faceting tolerance and length tolerance were applied to the FNG model. In most cases, the valence of the HV regions are reduced as well. This is a by-product of the tessellation algorithm's avoidance of high-aspect ratio triangles. Given that the performance degradation of ray fire times in the MPBVH scales with both the valence and HV area, applying a length tolerance during tessellation may be another pathway to improving performance in these models.



(a) Faceting Tolerance Only (b) Faceting and Length Tolerance

Figure 5.15: The effect of applying only the faceting tolerance (a) vs. the application of both a faceting tolerance and length tolerance (b) in the FNG model. HV regions are separated and made smaller when using both tesellation parameters.

The same set of production test models from Table 5.1 were faceted using both faceting and length tolerances. The corresponding simulations were then performed using DAGMC with the MPBVH kernel with BVH refinement disabled, meaning it will apply AABBs only during BVH construction. There are competing factors at play in this analysis. Because the models will contain many more triangles, the BVH will inherently become deeper, resulting in a higher number of nodes visited per traversal. Conversely, the overall number of HV regions will increase, but their area and valence will be significantly reduced.

Table 5.2 shows the results of this study. A length tolerance of 10 was applied to all production models with the exception of the ITER model where 25 was used. For all models, results using facet tolerance only tessellations were not numerically the same as those also applying the edge length tolerance as well. The same numerical answer for different tessellations is not expected and results for the models were statistically similar in all corresponding simulations. In agreement with the BVH

Model	Triangles (M)		Memory Usage (MB)		Timing Ratio (Faceting Tol. / Faceting & Length Tol.)
	Faceting Tolerance	Faceting & Length Tolerance	Faceting Tolerance	Faceting & Length Tolerance	
FNG	1.17	1.76	152	463	1.32
ATR	4.95	30.00	593	2785	0.83
UWNR	3.34	10.10	376	1770	1.01
ITER	39.63	42.39	3841	6654	1.31

Table 5.2: The results of production model simulations when applying both a faceting tolerance and length tolerance during model tessellation to reduce HV region severity.

refinement study, the greatest effect on performance was seen in the FNG and ITER simulations where the run times were reduced by $\approx 30\%$. In contrast, the ATR simulation became almost 20% worse. This is likely caused by the dramatic increase in the number of triangles for this model which could be mitigated by increasing the specified length tolerance. It is difficult to know a priori what faceting tolerance is optimal when tessellating models, but this study shows that an alternative meshing scheme may be another solution to mitigating the effect of HV regions on simulation performance.

When using BVH refinement to address HV region performance, the acceleration data structure is altered to better suit the tessellation. Whereas application of the length tolerance alters the tessellation to better suit the data structure by producing triangles more amenable to use of AABBs and reducing HV region severity. Each approach has its merits. BVH refinement requires no modification of the input tessellation, but it does require a way to gather more information about the mesh structure. Alternate tessellation schemes can produce more friendly to the BVH data structure, but may require additional memory - a notable constraint in the context of this work (see Section 2.6). In the future, local re-tessellation of these regions, either as a preprocessing step or during BVH construction, could provide reduced HV severity similar to the length tolerance faceting without a dramatic increase in memory requirements for simulation.

5.5 Summary

This chapter introduced performance degradation caused by a tessellation feature seen in engineering models for CAD-based MCRT-t. A test model was then developed to characterize and study the underlying cause of this degradation for several ray tracing systems. In MOAB's OBB tree, the main cause was leaf nodes containing large numbers of triangles. The

cause of poor ray tracing performance in this system was a combination of the tree's design and default splitting heuristic. A simple HV detection algorithm was presented and applied in a solution where build settings of the splitting heuristic are adjusted for portions of the tree bounding HV regions. This method was then verified to remove performance degradation of ray tracing performance in these regions. A similar process was then repeated for both EmDAG and MPBVH.

In the MPBVH, and presumably in EmDAG, the exclusive use of AABB nodes results in large overlaps between child boxes. Many negative triangle intersections result from these overlaps, a similar effect to containing many triangles in a single leaf node. The solution proposed and verified by the author is the use of OBBs only in HV regions by applying the same HV detection method as was used in the MOAB OBB tree. It is important to note, that this was only possible in the MPBVH due to it's application alongside MOAB, a fully formed mesh database. The use of mesh-specific concepts such as element adjacencies makes this detection method simple and efficient. This work was not duplicated in Embree due to the difficulty involved in obtaining the same set of information and application of a mixed BVH for ray tracing on triangle elements.

Finally, a study of the adaptive build method to production models is presented for understanding of the HV parameter's effect on simulation run times. Little impact was found for some models while a significant reduction was seen in others. It was also was found that these run times are significantly reduced despite the fact that a very small number of rays are intersected with triangles in HV regions as shown in Table 5.1.

5.6 Future Work

For the models studied in this chapter, any HV detection parameter below 1.0 was shown to be effective in reducing simulation run times. This

parameter could be studied and optimized using a broad range of test cases to determine the best value for use in the general case.

Alternative methods to reducing the HV performance pathology could be explored as well. Here, BVH data structures were modified to better suit the HV mesh feature. Conversely, the HV regions could also be modified to better suit the BVH. A re-meshing of the HV region to contain a more uniform tessellation would allow a BVH of AABBs to maintain traversal performance without modification to the BVH build, though this would presumably come at the cost of more triangles in the model. Re-meshing of HV regions was not explored in this work due to the concerns for the additional memory cost outlined in Section 2.6.

Chapter 6

Conclusion

6.1 Impact on CAD-Based MCRT

This work provides a set of methods and implementations which significantly improves the performance of DAGMC's particle tracking capability. Performance improvements varied from factors of 1.1 to 9.54 for all tests presented, without change in the results compared to the unmodified DAG-MCNP code.

The SDF implementation in DAGMC showed promising results for contrived test cases, but at most provided a 30% improvement in performance for production models. The opportunity space for this data structure is limited, but the model used to inform it's application is accurate for the majority of cases, though some improvements could be made for locally small average chord length values.

The Mixed Precision Bounding Volume Hierarchy (MPBVH) provides a robust method for returning higher precision intersections suitable for engineering analysis while exploiting CPU SIMD instructions for reduced precision bounding entities. The implementation provided by the author demonstrates performance comparable to entirely single precision systems used for rendering with a significant demonstrated impact on transport

performance.

The use of a mixed AABB/OBB tree to improve performance for models containing severe high valence regions was demonstrated. This effort also demonstrated the value using of an associated mesh database to inform construction of spatial data structures for these performance-degrading mesh features.

Model	Ratio to unmodified DAG-MCNP
Signed Distance Field	
ITER	1.16
nTOF	1.41
SHINE	1.56
SHINE*	1.35
SNS	1.00
Mixed Precision BVH	
FNG	2.09
ATR	4.56
UWNR	3.13
ITER	9.54

Table 6.1: A summary of the best-case performance improvements for modificed versions of DAG-MCNP in a series of production test cases.

The overall impact of this work on CAD-based MCRT-t is summarized in Table 6.1. Coputational savings of 2x at a minimum when applying the MPBVH were observed, with greater benefits of >4x seen in some test cases. For access to all open source code, raw data, and results presented here please refer to FigShare archive **DOI_HERE**.

6.2 Broader Impacts

Broader impacts of this work include a performant CPU ray tracer suitable for engineering analysis work. The MPBVH is coupled with MOAB, a mesh database currently in use for engineering analysis purposes at Argonne

National Lab. This tool can be built using standard GNU compilers and is relatively lightweight.

Work on adaptive BVH construction around HV regions demonstrates the importance of spatial query capabilities which are closely linked to a mesh framework in order to be able to detect problematic mesh features and adjust build settings.

The use of a mixed AABB/OBB to improve performance for models containing high valence regions suggests that for engineering analysis, or at least for the surface meshses generated by Cubit/Trelis for use in DAGMC, that AABBs can cause large amounts of overlap in certain regions of the model, resulting in several orders of magnitude degradation in ray query times.

6.3 Suggested Future Work

Some improvement of the SDF model is suggested in which additional information about the geometric properties of the target volume is used to inform decisions about SDF application. In particular, information on the variation of the average chord length could be used to avoid application of the data structure where its predicted utilization is over estimated. Spatially varying information on the collision density of particles, could also be used to better inform the predictive tool for analysis. Another extension of the SDF predictive model could be its use to inform algorithms for domain subdivision in codes where Woodcock delta tracking is applied [31] [66].

There are several directions work on the MPBVH could take. Slightly more exotic architectures used in high performance computing clusters provide wider registers than the AVX2 instructions used in this work. This creates the possibility of creating tree branching ratio of 8 or 16 for even more shallow trees and lower memory footprints. Higher branching ratios

also allow the MPBVH to extend naturally to other hierarchies like the octree which has other benefits in particle tracking and data field storage.

As suggested in Chapter 4, box extension values could be applied on a volume-by-volume basis for MCRT to avoid performance degradation in models with a large global scale, but regions with small components. This is application-specific, but could be integrated into the interface of the MPBVH to be set according to different use-cases.

Mesh features similar to the high valence region could be sought out using a variety of meshing schemes, geometries, and a program designed to highlight regions of models in which ray fire times are much lower than average to automatically highlight these areas for characterization and analysis as was done for the high valence region. Local re-meshing of high valence regions could also be explored to allow used of AABB-only hierarchies without performance detriment and minimal modifications to input mesh geometries.

Appendix A

Signed Distance Field Model Development and Data

A.1 Detailed Signed Distance Field Model Formulation

A.1.1 Fixed Distance Model Development

The utilization of the signed distance field as a preconditioner for ray tracing operations can be modeled as an evaluation of the combined probability space for particles with a current position, $\vec{p}$, and a next physics event location, $\vec{n}$, after traveling a distance, d. The fraction of this probability space in which signed distance values can be used to rule out surface crossings for next surface intersections is then considered to be the theoretical utilization of the signed distance field. An initial form for this probability space can found in Equation A.1.

$$\int_{V_{sphere}} \int_{V_{track}} p_p(r) p_n(d)\, dV_{sphere} dV_{track} \tag{A.1}$$

In this model, the starting location of particles, $\vec{p}_p(r, \phi, \theta)$, is uniformly

distributed, $p_p(r) = 1$, throughout a sphere of radius, R, centered at the origin. The location of the next event, $\vec{p}_n(d, \alpha, \beta)$, where d is the distance traveled by the particle, α is the interior angle between the particle's *position* vector and the particle's sampled direction vector, and β represents an azimuthal angle for directions traveled with angle of departure, α. Figure 3.8 depicts these variables, r, d, and α more clearly.

The outer integral in Equation A.1 represents all possible particle positions within the geometric sphere and expands to

$$\int_0^R \int_0^{2\pi} \int_0^{\pi} \int_{V_{track}} r^2 \sin\phi \, d\phi d\theta dr \, p_n(d) dV_{track} \tag{A.2}$$

The inner integral over V_{track} then expands to

$$\int_0^R \int_0^{2\pi} \int_0^{\pi} \int_0^{\infty} \int_0^{2\pi} \int_0^{\pi} r^2 \sin\phi \, p_n(d) d^2 \sin\alpha \, d\alpha d\beta dd \, d\phi d\theta dr \tag{A.3}$$

Integration of ϕ, θ, and β can now be performed with the knowledge that they are symmetric with respect to the problem and integration of $p_n(d)$ does not rely on them.

$$8\pi^2 \int_0^R \int_0^{\infty} \int_0^{\pi} p_n(d) \, r^2 \, d^2 \sin\alpha \, d\alpha dd \, dr \tag{A.4}$$

In order to represent particles traveling a fixed distance, the relationship in Equation A.5 is applied.

$$p_n(d) = \frac{\delta(d - \lambda)}{d^2} \tag{A.5}$$

The evaluation of this integral then gives a representation of all the query space available to the problem

$$A = 8\pi^2 \int_0^R \int_0^{\infty} \int_0^{\pi} \delta(d - \lambda) \, r^2 \, \sin\alpha \, d\alpha dd \, dr \tag{A.6}$$

and represents all geometric query space, labeled A, for a sphere of radius, R and a fixed distance traveled, λ. The condition for preconditioner utilization without error consideration is as follows

$$SDV(\vec{p}) + SDV(\vec{n}) > |\vec{p} - \vec{n}| \tag{A.7}$$

$$\begin{aligned}
&SDV - \ signed\, distance\, value\, function\\
&\vec{p} - \ particle's\, current\, position\\
&\vec{n} - \ particle's\, next\, event\, location\\
&h - \ mesh\, step\, size
\end{aligned}$$

To apply this within the spherical geometry, the signed distance function of a sphere with radius, R, from Equation (3.6) is applied

$$R - |\vec{p}| + R - |\vec{n}| > |\vec{p} - \vec{n}| \tag{A.8}$$

The right hand side of this inequality can be described as the distance traveled, d, and the magnitude of $\vec{p}$ can be represented by the variable r.

$$R - r + R - |\vec{n}(d, \alpha, \beta)| > d \tag{A.9}$$

Reducing the next event location, $\vec{n}(d, \alpha, \beta)$, into an expression in terms of r, d, and α requires further examination of the problem. Because the coordinates of n depend on the current particle position, the magnitude of n with respect to the geometry origin must be obtained to get a correct form for the signed distance value. Again, Figure 3.8 depicts the value of n graphically for reference. The magnitude of n can then be described using the law of cosines as

$$|n(d, \alpha, \beta)| = \sqrt{r^2 + d^2 - 2rd\cos\pi - \alpha} \tag{A.10}$$

inserting this into the inequality gives

$$R - r + R - \sqrt{r^2 + d^2 + 2rd\cos\alpha} > d \tag{A.11}$$

The inequality has now been reduced to the three variables seen in Equation A.6: r, d, and α. This can be applied to construct limits of integration representing boundaries of space in which the SDF can be utilized. As described in Chapter 3, α_{min} can be used as a limit on the integral over $d\alpha$. It is also mentioned that α_{min} is undefined until $d > R-r$ as shown in Equations A.12 and A.13.

$$d < R - r : \alpha_{min} = 0 \tag{A.12}$$

$$\alpha_{min} > \arccos\left(\frac{(2R - r - d)^2 - d^2 - r^2}{2dr}\right) \tag{A.13}$$

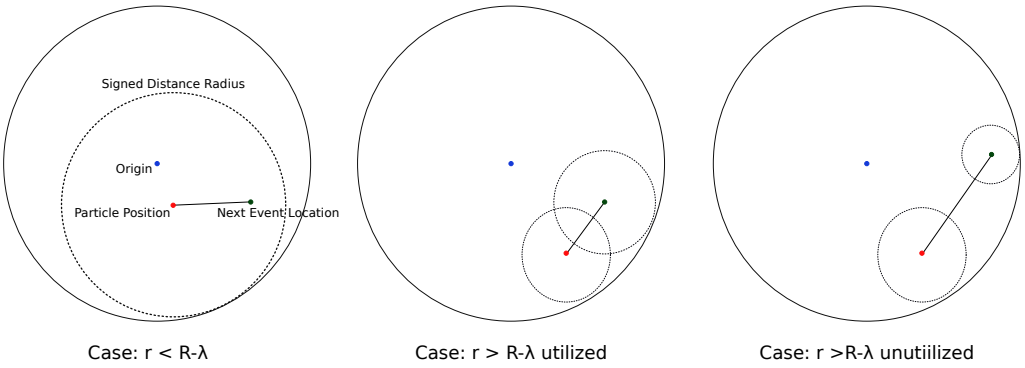


Figure A.1: Depiction of modeling cases. Left: an example of a track for which $d < R-r$. Middle: an example of a track for which $R-r < d < R$ and can be preconditioned. Right: an example of a track for which $R-r < d < R$ and cannot be preconditioned.

In order to account for the fact that the form of α_{min} is undefined until $d = R - r$, a Heaviside function is applied before applying it as a limit

on the particle's angle of departure from the position vector. Similarly, because the α_{min} condition is undefined after $d = R$ a Heaviside function is used to limit the condition to π for any distances traveled larger than R.

$$\begin{aligned}\alpha_{min} =&(H(d-(R-r))-H(d-R))\arccos\left(\frac{(2R-r-d)^2-d^2-r^2}{2dr}\right)\\ &+\pi H(d-R)\end{aligned} \tag{A.14}$$

By inserting this condition as a lower limit of the $d\alpha$ integration, Equation A.15 will give all utilized space, US, in the query space of the simulation.

$$US = 8\pi^2\int_0^R\int_0^\infty\int_{\alpha_{min}}^\pi \delta(d-\lambda)\,r^2\,d^2\sin\alpha\,d\alpha dd\,dr \tag{A.15}$$

Evaluating and simplifying this fully formed integral gives us the model presented in Chapter 3.

$$U_{theoretical} = \frac{US}{A} = \frac{(1-H(\lambda-R))(2R-\lambda)(R-\lambda)}{2R^2} \tag{A.16}$$

A.1.2 Sampled Distance Model Development

The sampled distance probability distribution presented in Chapter 3 is as follows:

$$p = \frac{e^{-\Sigma d}}{d^2} = \frac{e^{-\frac{d}{\lambda}}}{d^2} \tag{A.17}$$

with distances sampled using the function

$$d = -\lambda ln(c) \tag{A.18}$$

where c is randomly sampled with a uniform distribution between 0 and 1

An examination of the change of variables in the general form for the utilized space from Equation (3.2) gives

$$\frac{dp}{dd} = -\frac{p}{\lambda} \tag{A.19}$$

$$d = 0 \rightarrow c = 1 \tag{A.20}$$

$$d = \infty \rightarrow c = 0 \tag{A.21}$$

Resulting in an integral with the following form for the sampled distance model

$$\int_0^R \int_0^{2\pi} \int_0^{\pi} \int_1^0 \int_0^{2\pi} \int_0^{\pi} -r^2 \sin\phi \, \lambda c \ln(c)^2 \sin\alpha \, d\alpha d\beta dc \, d\phi d\theta dr \tag{A.22}$$

As in the fixed distance case, the condition for α_{min} is a piece-wise function based on the distance traveled. The expression for d changes slightly for this case however, given that the integral is now being performed over the variable c.

$$d < R - r : \alpha_{min} = 0 \tag{A.23}$$

$$d > R - r : \alpha_{min} = \arccos\left(\frac{(2R - r - d)^2 - d^2 - r^2}{2dr}\right) \tag{A.24}$$

Now that the distance traveled is being used to construct these two regions in the model, this integral must be separated into two pieces, one with limits of 0 to $R - r$ and another with limits $R - r$ to R. Based on the distance sampling distribution, these values become

$$d_{min} = R - r \rightarrow c_{max} = e^{(-\frac{(R-r)}{\lambda})} \tag{A.25}$$

$$d_{max} = R \rightarrow c_{min} = e^{(-\frac{R}{\lambda})} \tag{A.26}$$

and the resulting integral becomes

$$\int_0^R \int_0^{2\pi} \int_0^{\pi} \int_1^{c_{max}} \int_0^{2\pi} \int_0^{\pi} -r^2 sin(\phi)\, \lambda c ln(c)^2 sin(\alpha)\, d\alpha d\beta dc\, d\phi d\theta dr + \tag{A.27}$$

$$\int_0^R \int_0^{2\pi} \int_0^{\pi} \int_{c_{max}}^{c_{min}} \int_0^{2\pi} \int_{alpha_{min}}^{\pi} -r^2 \sin\phi\, \lambda c ln(c)^2 \sin\alpha\, d\alpha d\beta dc\, d\phi d\theta dr \tag{A.28}$$

The evaluation of this integral gives the final form of the analytic preconditioner limit from Equation (3.13)

$$U_{sampled} = \frac{\frac{1}{2}\lambda(R - 2\lambda)e^{-\frac{R}{\lambda}} + \lambda^2 - \frac{3}{2}R\lambda + R^2}{R^2} \tag{A.29}$$

A.1.3 Inclusion of Error in Model Development

As provided in Chapter 3, the condition for avoiding a ray fire call when including error associated with signed distance value interpolation is:

$$SDV(\vec{p}) + SDV(\vec{n}) > |\vec{p} - \vec{n}| + 2\varepsilon(h) \tag{A.30}$$

$SDV -$ signed distance value function

$\vec{p} -$ particle's current position

$\vec{n} -$ particle's next event location

$h -$ mesh step size

$\varepsilon(h) -$ error evaluation for signed distance values

The limits of the α_{min} condition need to be adjusted yet again to account for the error that will be subtracted from the signed distance values. This becomes a relatively straightforward process in which the error is also subtracted from the arguments to the Heaviside functions in Equation A.14. Accounting for interpolation error reduces the maximum possible distance the particle can travel and still be preconditioned. It also reduces the value of d_{min} where α_{min} becomes non-zero.

$$\begin{aligned}\alpha_{min} = &\arccos\left(\frac{(2R-r-d-2\epsilon)^2-d^2-r^2}{2dr}\right)[H(d-(R-r-\epsilon))-H(d-(R-\epsilon))]\\ &+\pi H(d-(R-\epsilon))\end{aligned} \tag{A.31}$$

After these adjustments to the α_{min} condition, the integral can be evaluated and simplified to give the form found in Equation (3.17):

$$US_{sampled} = \frac{(R-\epsilon)(\frac{1}{2}\lambda(R-2\lambda-\epsilon)e^{\frac{-R+\epsilon}{\lambda}}+\lambda^2-\frac{3}{2}\lambda(R-\epsilon)+(R-\epsilon)^2)}{R^3} \tag{A.32}$$

Appendix B

Modified DAGMC Simulation Results

This appendix contains all results for simulations performed using modified versions of DAGMC. Because a numerical comparison, as opposed to a statistical comparison, is being made between these different implementations, error values are not included in the tables of this appendix. The results of native MCNP simulations have been provided when possible for reference. MCNP results are expected to be statistically similar but not the same as DAG-MCNP results. All implementations of DAG-MCNP (EmDAG, SDF, unmodified DAGMC, etc.) are expected to report the same numerical tally results. Additional CDF plots for differing mesh tally results are included when necessary.

B.1 EmDAG-MCNP

The results in this section were generated using the EmDAG implementaion presented in Section 4.2. These include several contrived test cases in Section B.1.1 along with a production model test of FNG with the EmDAG system in Section B.1.2.

B.1.1 Simple Test Cases

B.1.1.1 Single Cube

Value	MCNP	DAG-MCNP	DAG-MCNP with MPBVH
Flux	5.61567E-03	5.61567E-03	5.61567E-03
Energy	2.98787E-03	2.98787E-03	2.98787E-03

Table B.1: Single hydrogen-filled cube tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.1.1.2 Nested Cubes

Value	MCNP	DAG-MCNP	EmDAG-MCNP
Cell 1 Tallies			
Flux	1.38556E-02	1.38556E-02	1.38556E-02
Energy	1.05974E-02	1.05974E-02	1.05974E-02
Cell 2 Tallies			
Flux	1.41609E-03	1.41609E-03	1.91644E-04
Energy	4.81169E-04	4.81169E-04	4.81169E-04
Cell 3 Tallies			
Flux	1.92847E-04	1.92847E-04	1.92847E-04
Energy	1.10361E-04	1.10361E-04	1.10361E-04

Table B.2: Nested hydrogen-filled cube tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.1.1.3 Single Sphere

Value	MCNP	DAG-MCNP	DAG-MCNP with MPBVH
Flux	4.98069E-03	4.98639E-03	4.98639E-03
Energy	3.17629E-03	3.18079E-03	3.18079E-03

Table B.3: Single hydrogen-filled sphere tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.1.1.4 Nested Spheres

Value	MCNP	DAG-MCNP	EmDAG-MCNP
Cell 1 Tallies			
Flux	5.25725E-03	5.25734E-03	5.25734E-03
Energy	3.17869E-03	3.17873E-03	3.17873E-03
Cell 2 Tallies			
Flux	1.91645E-04	1.91644E-04	1.91644E-04
Energy	5.22131E-05	5.22137E-05	5.22137E-05
Cell 3 Tallies			
Flux	1.18371E-05	1.18**376**E-05	1.18**410**E-05
Energy	4.96282E-06	4.96285E-06	4.96285E-06

Table B.4: Nested Spheres Tally Results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.1.2 Production Test Cases

B.1.2.1 FNG

Value	MCNP	DAG-MCNP	EmDAG-MCNP
Neutron Flux			
Average	7.228758E-05	7.321**88**E-05	7.321**69**E-05
Maximum	1.457240E-04	1.485**8**80E-04	1.485**8**40E-04
Minimum	1.997170E-05	1.938**4**40E-05	1.938**5**0E-05

Table B.5: Summary of the flux tally results for a global volumetric source in the FNG model for MCNP, DAG-MCNP, and Embree coupled with DAGMC, or EmDAG-MCNP. Units of flux are in cm^{-2}.

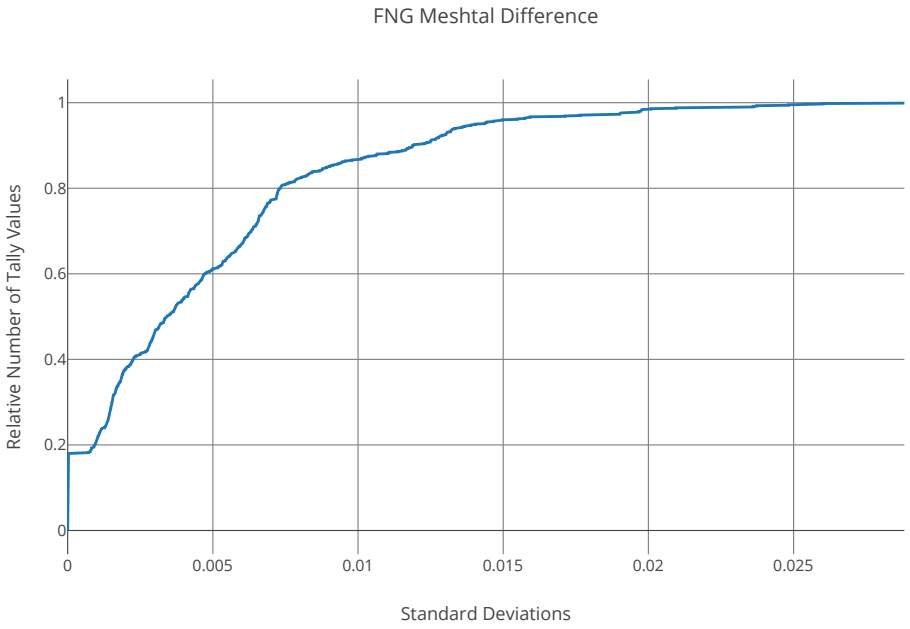


Figure B.1: A cumulative distribution function for the differing tally values between DAG-MCNP and EmDAG-MCNP for the FNG model. Zero entries for the unmodified DAG-MCNP simulation are set to zero here.

B.2 MPBVH

The results in this section were generated using DAG-MCNP relying on the MPBVH as its ray tracing kernel as presented in Section 4.3. These include several contrived test cases in Section B.2.1 along with several production model demonstrations in Section B.2.2.

B.2.1 Simple Test Cases

B.2.1.1 Single Cube

Value	MCNP	DAG-MCNP	DAG-MCNP with MPBVH
Flux	5.61567E-03	5.61567E-03	5.61567E-03
Energy	2.98787E-03	2.98787E-03	2.98787E-03

Table B.6: Single hydrogen-filled cube tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.2.1.2 Nested Cubes

Value	MCNP	DAG-MCNP	EmDAG-MCNP
Cell 1 Tallies			
Flux	1.38556E-02	1.38556E-02	1.38556E-02
Energy	1.05974E-02	1.05974E-02	1.05974E-02
Cell 2 Tallies			
Flux	1.41609E-03	1.41609E-03	1.91644E-04
Energy	4.81169E-04	4.81169E-04	4.81169E-04
Cell 3 Tallies			
Flux	1.92847E-04	1.92847E-04	1.92847E-04
Energy	1.10361E-04	1.10361E-04	1.10361E-04

Table B.7: Nested hydrogen-filled cube tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.2.1.3 Single Sphere

Value	MCNP	DAG-MCNP	DAG-MCNP with MPBVH
Flux	4.98069E-03	4.98639E-03	4.98639E-03
Energy	3.17629E-03	3.18079E-03	3.18079E-03

Table B.8: Single hydrogen-filled sphere tally results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.2.1.4 Nested Spheres

Value	MCNP	DAG-MCNP	DAG-MCNP with MPBVH
Cell 1 Tallies			
Flux	5.25725E-03	5.25734E-03	5.25734E-03
Energy	3.17869E-03	3.17873E-03	3.17873E-03
Cell 2 Tallies			
Flux	1.91645E-04	1.91644E-04	1.91644E-04
Energy	5.22131E-05	5.22137E-05	5.22137E-05
Cell 3 Tallies			
Flux	1.18371E-05	1.18376E-05	1.18376E-05
Energy	4.96282E-06	4.96285E-06	4.96285E-06

Table B.9: Nested Spheres Tally Results. Flux tally units are cm^{-2}. Energy tally units are MeV/g.

B.2.2 Production Tests

B.2.2.1 FNG

Value	MCNP	DAG-MCNP	DAG-MCNP-MPBVH
Average	7.227982E-05	7.230336E-05	7.230336E-05
Maximum	1.471770E-04	1.471210E-04	1.471210E-04
Minimum	1.918080E-05	1.877310E-05	1.877310E-05

Table B.10: Summary of the flux tally results for a global volumetric source in the FNG model for MCNP, DAG-MCNP, and DAG-MCNP using the MPBVH.

B.2.2.2 ATR

Value	MCNP	DAG-MCNP	DAG-MCNP-MPBVH
k_{coll}	0.99218	0.98456	0.98456
k_{abs}	0.98764	0.99448	0.99448
k_{track}	0.98899	0.98283	0.98283

Table B.11: Results of an eigenvalue simulation for the Advanced Test Reactor at Idaho National Laboratory over 100 cycles.

B.2.2.3 UWNR

Value	MCNP	DAG-MCNP	DAG-MCNP-MPBVH
k_{coll}	1.02050	1.02084	1.02084
k_{abs}	1.02044	1.02072	1.02072
k_{track}	1.02041	1.02063	1.02063

Table B.12: Results of an eigenvalue simulation for the University of Wisconsin Nuclear Reactor over 75 cycles.

B.2.2.4 ITER

Value	DAG-MCNP	DAG-MCNP with MPBVH
Neutron Flux		
Average	7.222176E+12	7.222176E+12
Maximum	1.410510E+14	1.410510E+14
Minimum	0.000000E+00	0.000000E+00

Table B.13: Result comparison for a mesh-based flux tally applied to the ITER neutronics model for 1×10^6 histories. Units are in cm^{-2}

B.3 Signed Distance Field

The results in this section are the result of signed distance field application in DAGMC production models, seen in Section 3.4.

B.3.1 Simple Test Case

B.3.1.1 Hydrogen-Filled Sphere

Tally Number	DAG-MCNP	DAG-MCNP with SDF
6	4.18872E+03	4.18872E+03
4	7.01007E+03	7.01007E+03

B.3.2 Production Tests

B.3.2.1 ITER

Value	DAG-MCNP	DAG-MCNP with SDF
Neutron Flux Tally		
Average	2.123795E-09	2.123795E-09
Maximum	3.170170E-06	3.170170E-06
Minimum	0.000000E+00	0.000000E+00

Table B.14: Result comparison for a mesh-based flux tally applied to the ITER neutronics model for 1×10^7 histories. Units are in cm^{-2}.

B.3.2.2 nTOF

Value	DAG-MCNP	DAG-MCNP with SDF
Neutron Flux Tally		
Average	7.730000E+00	7.730000E+00
Maximum	3.250500E+01	3.250500E+01
Minimum	1.704500E+01	1.704500E+01
Proton Flux Tally		
Average	1.539597E-04	1.539597E-04
Maximum	1.045990E+00	1.045990E+00
Minimum	0.000000E+00	0.000000E+00
Photon Flux Tally		
Average	3.255857E-03	3.255857E-03
Maximum	5.343100E-01	5.343100E-01
Minimum	0.000000E+00	0.000000E+00

Table B.15: A result comparison between DAG-MCNP with and without the SDF field applied for 10K histories in the the neutron Time-Of-Flight model. Units are in cm^{-2}.

B.3.2.3 SHINE

Value	DAG-MCNP	DAG-MCNP with SDF
Neutron Flux Tally		
Average	2.486558E+10	2.486558E+10
Maximum	2.330400E+12	2.330400E+12
Minimum	0.000000E+00	0.000000E+00

Table B.16: A result comparison for the SHINE model with and without SDF application for 10k particle histories. Units are in cm^{-2}.

B.3.2.4 SNS

Tally Number	DAG-MCNP	DAG-MCNP with SDF
6	1.5927E-01	1.5927E-01
24	1.8279E-01	1.8279E-01
46	1.1338E+00	1.1338E+00
76	4.3240E-01	4.3240E-01
106	6.7470E-02	6.7470E-02
126	1.4468E+00	1.4468E+00
146	6.5084E-02	6.5084E-02
166	1.6627E+00	1.6627E+00
206	1.1284E+01	1.1284E+01
114	4.0555E+00	4.0555E+00
174	4.0451E+00	4.0451E+00
14	3.7261E-02	3.7261E-02
26	4.2035E-01	4.2035E-01
56	1.7891E-01	1.7891E-01
86	1.7698E-01	1.7698E-01
116	2.6279E-01	2.6279E-01
136	9.6872E-02	9.6872E-02
156	2.5599E-01	2.5599E-01
176	1.0053E-01	1.0053E-01
216	8.3326E+00	8.3326E+00
134	1.9720E+00	1.9720E+00
214	6.0137E+02	6.0137E+02
16	1.0738E-01	1.0738E-01
36	4.1034E-01	4.1034E-01
66	5.4782E-01	5.4782E-01
104	2.9490E+03	2.9490E+03
124	8.8337E+02	8.8337E+02
144	6.8868E+02	6.8868E+02
164	7.3002E+02	7.3002E+02
204	7.9032E+01	7.9032E+01
226	3.1626E-01	3.1626E-01
154	3.4720E+00	3.4720E+00

Table B.17: Result comparison for a variety of atom displacement tallies in the Spallation Neutron Source model in DAG-MCNP both with and without the SDF for 10K histories.

references

[1] Trelis user's guide. https://www.csimsoft.com/help/trelishelp.htm. Accessed: 2018-05-09.

[2] 2008. IEEE standard for floating-point arithmetic. *IEEE Std 754-2008* 1–70.

[3] 2018. Corona renderer. https://corona-renderer.com.

[4] ABAQUS. 2018. Abaqus (finite element analysis & multiphysics simulation). http://www.feasol.com/solutions/engineering-simulation/abaqus/.

[5] ACIS. 2018. 3-d acis modeling. https://www.spatial.com/products/3d-acis-modeling.

[6] Agostinelli, S., J. Allison, et al. 2003. Geant4: a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506(3):250 – 303.

[7] Bardsley, J. N., and A. Dubi. 1981. The average transport path length in scattering media. *SIAM Journal on Applied Mathematics* 40(1):71–77.

[8] Bentley, Jon Louis. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18(9):509–517.

[9] Bittner, Jiří, Michal Hapala, and Vlastimil Havran. 2013. Fast insertion-based optimization of bounding volume hierarchies. *Computer Graphics Forum* 32(1):85–100.

[10] Blacker, Tedd, William Bohnhoff, Tony Edwards, James Hipp, Randy Lober, Scott Mitchel, Gregory Sjaardema, Timothy Tautges, Tammy Wilson, and William Oakes. 1994. Cubit mesh generation environment.

[11] Bohlen, T.T., F. Cerutti, M.P.W. Chin, A. Fassó, A. Ferrari, P.G. Ortega, A. Mairani, P.R. Sala, G. Smirnov, and V. Vlachoudis. 2014. The fluka code: Developments and challenges for high energy and medical applications. *Nuclear Data Sheets* 120:211 – 214.

[12] Brighter3D. 2018. Brighter3d: Sketchup rendering plugin. `https://www.brighter3d.com`.

[13] CASCADE, Open. 2018. Open cascade technology, 3d modeling & numerical simulation. `https://www.opencascade.com/`.

[14] Dammertz, H., J. Hanika, and A. Keller. 2008. Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. *Computer Graphics Forum* 27(4):1225–1233.

[15] Deng, Li, and Zhong-Sheng Xie. 1999. Parallelization of mcnp monte carlo neutron and photon transport code in parallel virtual machine and message passing interface. *Journal of Nuclear Science and Technology* 36(7):626–629.

[16] Ernst, Manfred, and Gunther Greiner. 2007. Early split clipping for bounding volume hierarchies. In *2007 IEEE symposium on interactive ray tracing*. Institute of Electrical & Electronics Engineers (IEEE).

[17] FluidRay. 2018. Fluidray: A physically based renderer. `https://www.fluidray.com`.

[18] Forum, Message P. 1994. Mpi: A message-passing interface standard. Tech. Rep., Knoxville, TN, USA.

[19] Glassner, Andrew S., ed. 1989. *An introduction to ray tracing*. London, UK, UK: Academic Press Ltd.

[20] Goldsmith, J., and J. Salmon. 1987. Automatic creation of object hierarchies for ray tracing. *Computer Graphics and Applications, IEEE* 7(5):14–20.

[21] Goorley, T., M. James, T. Booth, F. Brown, J. Bull, L.J. Cox, J. Durkee, J. Elson, M. Fensin, R.A. Forster, J. Hendricks, H.G. Hughes, R. Johns, B. Kiedrowski, R. Martz, S. Mashnik, G. McKinney, D. Pelowitz, R. Prael, J. Sweezy, L. Waters, T. Wilcox, and T. Zukaitis. 2016. Features of {MCNP6}. *Annals of Nuclear Energy* 87, Part 2:772 – 783.

[22] Gottschalk, Stefan, Ming C Lin, and Dinesh Manocha. 1996. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques*, 171–180. ACM.

[23] GroÃŸe, D., U. Fischer, K. Kondo, D. Leichtle, P. Pereslavtsev, and A. Serikov. 2013. Status of the mccad geometry conversion tool and related visualization capabilities for 3d fusion neutronics calculations. *Fusion Engineering and Design* 88(9):2210 – 2214. Proceedings of the 27th Symposium On Fusion Technology (SOFT-27); LiÃ¨ge, Belgium, September 24-28, 2012.

[24] Havran, Vlastimil. 2000. Heuristic ray shooting algorithms. Ph.D. thesis, Czech Technical University.

[25] Hennessy, John L., and David A. Patterson. 2012. *Computer architecture: A quantitative approach*. 5th ed. Morgan Kaufmann.

[26] Hughes, Christopher J. 2015. Single-instruction multiple-data execution. *Synthesis Lectures on Computer Architecture* 10(1):1–121.

[27] Hurley, Jim, Er Kapustin, Er Reshetov, and Alexei Soupikov. 2002. Fast ray tracing for modern general purpose cpu. In *In proceedings of graphicon*, 2002.

[28] Kay, Timothy L., and James T. Kajiya. 1986. Ray Tracing Complex Scenes. *SIGGRAPH Computer Graphics* 20:269.

[29] Kessel, C.E., J.P. Blanchard, A. Davis, L. El-Guebaly, et al. 2017. Overview of the fusion nuclear science facility, a credible break-in step on the path to fusion energy. *Fusion Engineering and Design*.

[30] Lamont, Chris. 2011. *Introduction to intel advanced vector extensions*. https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions.

[31] Leppänen, Jaakko. 2010. Performance of woodcock delta-tracking in lattice physics applications using the serpent monte carlo reactor physics burnup calculation code. *Annals of Nuclear Energy* 37(5):715 – 722.

[32] Leppänen, Jaakko, Maria Pusa, Tuomas Viitanen, Ville Valtavirta, and Toni Kaltiaisenaho. 2015. The serpent monte carlo code: Status, development and applications in 2013. *Annals of Nuclear Energy* 82:142 – 150. Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms.

[33] Lewis, Elmer E., and Warren F. Miller Jr. 1993. *Computational methods of neutron transport*. Amer Nuclear Society.

[34] MacDonald, Kellogg S., J. David ; Booth. 1990. Heuristics for ray tracing using space subdivision. *The Visual Computer* 6(3):153–166.

[35] Malouch, Fadhel, Brun, Emeric, Diop, Cheikh, Hugot, François-Xavier, Jouanne, Cédric, Lee, Yi-Kang, Malvagi, Fausto, Mancusi, Davide, Mazzolo, Alain, Petit, Odile, Trama, Jean-Christophe, Visonneau, Thierry, and Zoia, Andrea. 2017. Recent developments in the tripoli-4 monte-carlo code for shielding and radiation protection applications. *EPJ Web Conf.* 153:06007.

[36] Mazzolo, Alain. 2014. Cauchys formulas for random walks in bounded domains. *Journal of Mathematical Physics* 55(8):083308.

[37] Metropolis, Nicholas, and S. Ulam. 1949. The monte carlo method. *Journal of the American Statistical Association* 44(247):335–341.

[38] O'Rourke, Joseph. 1985. Finding minimal enclosing boxes. *International Journal of Computer & Information Sciences* 14(3):183–199.

[39] Osher, Stanley, and Ronald P. Fedkiw. 2003. *Level set methods and dynamic implicit surfaces*. Applied mathematical science, New York, N.Y.: Springer.

[40] Pandya, Tara M., Seth R. Johnson, Thomas M. Evans, Gregory G. Davidson, Steven P. Hamilton, and Andrew T. Godfrey. 2016. Implementation, capabilities, and benchmarking of shift, a massively parallel monte carlo radiation transport code. *Journal of Computational Physics* 308:239 – 272.

[41] Peña, Antonio J., and Pavan Balaji. 2016. A data-oriented profiler to assist in data partitioning and distribution for heterogeneous memory in hpc. *Parallel Computing* 51:46 – 55. Special Issue on Parallel Programming Models and SystemsSoftware for High-End Computing.

[42] Platis, Nikos, and Theoharis Theoharis. 2003. Fast Ray-Tetrahedron Intersection Using Plucker Coordinates. *Journal of Graphics Tools* 8(4): 37–48.

[43] Revelles, J., C. Ureña, and M. Lastra. 2000. An efficient parametric algorithm for octree traversal. In *Journal of wscg*, 212–219.

[44] Samet, Hanan. 1989. Implementing ray tracing with octrees and neighbor finding. *Computers and Graphics* 13(4):445 – 460.

[45] Sethian, J. A. 1996. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences of the United States of America* 93(4):1591–1595.

[46] Sigg, Christian. 2006. Representation and rendering of implicit surfaces.

[47] Smith, B.M., T.J. Tautges, and P.P.H. Wilson. 2010. Sealing Faceted Surfaces to Achieve Watertight CAD Models.

[48] Smith, Brandon M. 2011. Robust Tracking and Advanced Geometry for Monte Carlo Radiation Transport. PhD Nuclear Engineering and Engineering Physics, University of Wisconsin-Madison, Madison, WI, United States.

[49] 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.

[50] Talamo, Alberto, Yousry Gohar, and J. LeppÃ¤nen. 2018. Serpent validation and optimization with mesh adaptive search on stereolithography geometry models. *Annals of Nuclear Energy* 115:619 – 632.

[51] Tautges, T. J., R. Meyers, K. Merkley, C. Stimpson, and C. Ernst. 2004. MOAB: A Mesh-Oriented Database. SAND2004-1592, Sandia National Laboratories. Report.

[52] Tautges, Timothy J., P. P. H. Wilson, Jason Kraftcheck, Brandon M. Smith, and Douglass L. Henderson. 2009. Acceleration Techniques for Direct Use of CAD-Based Geometries in Monte Carlo Radiation Transport. In *International Conference on Mathematics, Computational Methods & Reactor Physics (M&C 2009)*. Saratoga Springs, NY: American Nuclear Society.

[53] TEAM, X-5 MONTE CARLO. 1987. Mcnp – a general monte carlo n-particle transport code, version 5. Tech. Rep. LA-UR-03-1987, Los Alamos National Laboratory.

[54] Tomczak, L. J. 2012. GPU ray marching of distance fields. Supervised by Associate Professor Jeppe Revall Frisvad, jrf@imm.dtu.dk, and Associate Professor Jakob Andreas Bærentzen, jab@imm.dtu.dk, DTU Informatics.

[55] Vaidyanathan, K., T. Akenine-Möller, and M. Salvi. 2016. Watertight ray traversal with reduced precision. In *Proceedings of high performance graphics*, 33–40. HPG '16, Aire-la-Ville, Switzerland, Switzerland: Eurographics Association.

[56] Wächter, Carsten, and Alexander Keller. 2006. Instant ray tracing: The bounding interval hierarchy. *Eurographics*.

[57] Wald, I., C. Benthin, and S. Boulos. 2008. Getting rid of packets - efficient simd single-ray traversal using multi-branching bvhs -. In *Interactive ray tracing, 2008. rt 2008. IEEE symposium*, 49–57.

[58] Wald, Ingo, Philipp Slusallek, Carsten Benthin, and Markus Wagner. 2001. Interactive rendering with coherent ray tracing. *Computer Graphics Forum* 20(3):153–165.

[59] Wald, Ingo, Sven Woop, and Carsten Benthin. 2018. Embree: High performance ray tracing kernels. `https://github.com/embree/embree`.

[60] Wald, Ingo, Sven Woop, Carsten Benthin, Gregory S. Johnson, and Manfred Ernst. 2014. Embree: A kernel framework for efficient cpu ray tracing. *ACM Trans. Graph.* 33(4):143:1–143:8.

[61] Weghorst, Hank, Gary Hooper, and Donald P. Greenberg. 1984. Improved computational methods for ray tracing. *ACM Trans. Graph.* 3(1):52–69.

[62] White, David R., and Sunil Saigal. 2002. Improved imprint and merge for conformal meshing. In *International meshing roundtable*.

[63] Wu, Yican, Jing Song, Huaqing Zheng, Guangyao Sun, Lijuan Hao, Pengcheng Long, and Liqin Hu. 2015. Cad-based monte carlo program for integrated simulation of nuclear system supermc. *Annals of Nuclear Energy* 82:161 – 168. Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013, SNA + MC 2013. Pluri- and Trans-disciplinarity, Towards New Modeling and Numerical Simulation Paradigms.

[64] X-5 Monte Carlo Team. 2004. Mcnp - a general monte carlo n-particle transport code, version 5 - volume III: developers guide. Tech. Rep. LA-CP-03-0284, Los Alamos National Laboratory.

[65] Yu, Shengpeng, and Yican Wu. 2013. Mcam 5.2: Advanced interface program for multiple nuclear analysis codes. *Transactions of the American Nuclear Society* 109:724–725.

[66] Yue, Yonghao, Kei Iwasaki, Bing - Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. 2011. Toward optimal space partitioning for unbiased, adaptive free path sampling of inhomogeneous participating media. *Computer Graphics Forum* 30: 1911–1919. https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.02049.x.