Robust Tracking and Advanced Geometry for Monte Carlo Radiation Transport

B.M. Smith

March 2011

UWFDM-1391

Ph.D. thesis.

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Robust Tracking and Advanced Geometry for Monte Carlo Radiation Transport

B.M. Smith

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

March 2011

UWFDM-1391

Ph.D. thesis.

ROBUST TRACKING AND ADVANCED GEOMETRY FOR MONTE CARLO RADIATION TRANSPORT

by

Brandon M. Smith

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

2011

Abstract

A set of improved geometric capabilities are developed for the Direct Accelerated Geometry for Monte Carlo (DAGMC) library to increase its ease of use and accuracy. The improvements are watertight faceting, robust particle tracking, automatic creation of nonsolid space, and overlap tolerance. Before being sealed, adjacent faceted surfaces do not have the same discretization along shared curves. Sealing together surfaces to create a watertight faceting prevents leakage of particles between surfaces. The tracking algorithm is made robust by ensuring numerical consistency and avoiding geometric tolerances. Monte Carlo simulation requires all space to be defined, whether it be vacuum, air, coolant, or a solid material. The implicit creation of nonsolid space reduces human effort otherwise required to explicitly create nonsolid space in a CAD program. CAD models often contain small gaps and overlaps between adjacent volumes due to imprecise modeling, file translation, or intentional deformation. Although gaps are filled by the implicit creation of nonsolid space, overlaps cause geometric queries to become unreliable. The particle tracking algorithm and point inclusion test are modified to tolerate small overlaps of adjacent volumes. Overlap-tolerant particle tracking eliminates manual repair of CAD models and enables analysis of meshed finite element models undergoing structural deformation. These improvements are implemented in a coupling of DAGMC with the Monte Carlo N-Particle (MCNP) code, known as DAG-MCNP. The elimination of both manual CAD repair and lost particles are demonstrated with CAD models used in production calculations.

Acknowledgments

I am grateful for the guidance, patience, and support of my advisor, Prof. Paul Wilson. He invested a great deal of time in my professional development and research skills. Prof. Tim Tautges was generous with his expertise in computer geometry and mesh generation. Together, Profs. Wilson and Tautges helped me identify research objectives. Their oversight ensured that this work remained focused and meaningful. I am thankful for Prof. Mohamed Sawan's guidance through several fusion neutronics analyses. Jason Kraftcheck assisted with the design and implementation of this work in the Mesh-Oriented Database. Much of DAGMC, then known as MCNPX/CGM, originated with the Ph.D. work of Mengkuo Wang. In addition to those already mentioned, Timothy Bohm, Douglass Henderson, Laila El-Guebaly, Ahmad Ibrahim, Brian Kiedrowski, and Rachel Slaybaugh made substantial contributions to the DAGMC project. Daniel Villa, Tyler Tallman, and Jeffrey Smith at Sandia National Laboratories (SNL) worked especially hard to conduct the structural simulations that I used to investigate the impact-induced reactivity change of space reactors. For the space reactor work, substantial contributions and guidance were provided by the SNL team including Ron Lipinski, Tracy Radel, and Ross Radel.

This work was supported, in part, by SNL and the US ITER Project through Sandia contracts 579323 and 866756. The US ITER Project Office, Oak Ridge National Laboratory, is managed and operated by UT-Battelle, LLC for the United States Department of Energy under contract DE-AC05-00OR22725. This work was also supported by the SNL Lab Directed Research and Development Program.

To my family and especially my wife, Nicole—thank you for your support and encouragement. I could not have accomplished this without you.

Table of Contents

Page

Page

List of Tables

List of Figures

Figure Page

List of Algorithms

Nomenclature

ATR Advanced Test Reactor

BREP Boundary Representation

CAD Computer-Assisted Design

CGM Common Geometry Module

CSG Constructive Solid Geometry

DAGMC Direct Accelerated Geometry for Monte Carlo

DAG-MCNP Direct Accelerated Geometry - Monte Carlo N-Particle

FWS First Wall and Shield

IGES International Graphics Exchange Specification

ITER International Thermonuclear Experimental Reactor

LANL Los Alamos National Laboratory

LLNL Lawrence Livermore National Laboratory

MCNP Monte Carlo N-Particle

MOAB Mesh Oriented datABase

OBB Oriented Bounding Box

PIP Permuted Inner Product

PIT Point Inclusion Test

RTI Ray-Triangle Intersection

SNL Sandia National Laboratories

SPH Smoothed Particle Hydrodynamics

STEP Standard for the Exchange of Product model data

STL Steriolithography

TBM Test Blanket Module

Chapter 1

Introduction

Radiation transport methods determine particle flux or derived quantities across space, angle, energy, and time. Together, the space, angle, energy, and time domains are known as *phase space*. The behavior of neutral particles is described by the linear Boltzmann, or radiation transport equation [1]. In the deterministic method the transport equation is solved by discretizing phase space. Time and memory constraints may limit the number of energy groups, angular resolution, and mesh density. Error is determined by the discretization of the domain. The discrete solution approaches the continuous solution as the number of energy groups, angular resolution, and mesh density increases. Although time-dependent solutions are possible, they are not significant to this work.

An alternative approach of solving the transport equation used in this work is to simulate the interactions of individual particles across phase space. The stochastic or Monte Carlo approach solves the transport equation by simulating many particle histories [2]. The Monte Carlo method was developed at Los Alamos National Laboratory (LANL) during World War II by Fermi, von Neumann, Ulam, Metropolis, and Richtmyer [3]. As the number of histories approaches infinity, the mean response of the statistical system approaches the continuous solution. Statistical error falls inversely proportional to the square root of the number of histories. Pseudo-random numbers are used to determine interaction probabilities and outcomes.

The strengths and weaknesses of both methods complement each other. Discretization of phase space is not required by the Monte Carlo method. This implies that only statistical error is incurred beyond the error of input parameters such as cross sections and geometry. A measure of statistical

error accompanies every Monte Carlo result. Deterministic methods incur error due to the discretization of phase space. The magnitude of deterministic error can be explored by investigating the resolution of the discretization.

It is difficult for the Monte Carlo method to determine particle flux through a radiation shield because statistical error increases as particle population decreases. Deterministic methods have no such difficulty with shielding. On the contrary, Monte Carlo methods excel at calculating results through void or low-density material, due to the continuous treatment of position and angle. It is difficult for deterministic methods to model particles streaming through vacuum, causing ray effects if the angular approximation is too coarse.

Deterministic approaches inherently calculate solutions over the entire domain. It is challenging for Monte Carlo simulations to determine global solutions with constant statistical error. Because statistical error of the Monte Carlo method depends on the particle population, tallies with narrow phase space are prone to high statistical error. The memory and runtime of deterministic approaches increase with the mesh density, angular approximation, and number of energy groups. The runtime of the Monte Carlo method increases with the number of histories simulated.

1.1 Monte Carlo Method

In the Monte Carlo method, histories are initiated according to a source distribution. Pseudo-random numbers are generated to sample probability distributions that describe source position, direction, energy, number of mean free paths to the next collision, and collision outcomes. The material cross section and number of mean free paths to the next collision is used to calculate the distance to collision. The next event is the closest of either a collision or surface crossing. Surface crossings change the volume, or *cell*, in which a particle is traveling. If the volume contains a material with different cross section, the distance to collision is adjusted. Collisions change the particle's energy and direction of travel. After a collision occurs, the number of mean free paths until the next collision is resampled. If the particle travels into a region of interest, or *tally*, information representing the particle will be added to the tally. For example, to determine particle flux the track length will be summed and normalized by both the number of histories and volume

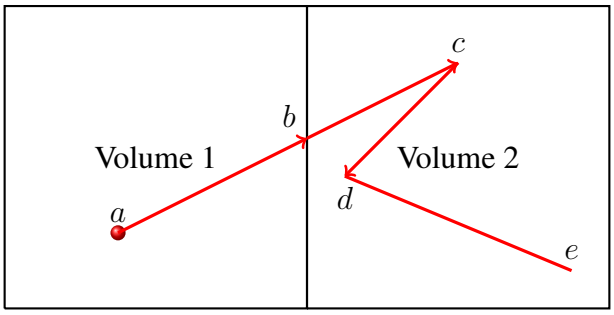


Figure 1.1: Particle track through two volumes, in two dimensions.

of the region of interest. A History ends when the particle travels outside the geometry or is absorbed in a collision. Additional histories are simulated until statistical error falls below a desired threshold [1].

A history is shown in Figure 1.1. The particle is born in volume 1 at a. The distance to the surface at b is less than the distance to collision, so the particle streams through volume 1. At b the particle crosses the surface and enters volume 2. The particle undergoes a collision at c and d before reaching e. At e a collision occurs in which the particle is absorbed.

1.2 CAD Geometry Support

The traditional, or *native* approach to specify geometry for Monte Carlo simulation is to use Boolean constructors of half spaces defined by quadratic or planar surfaces[1], known as constructive solid geometry (CSG). Each surface and volume is manually entered into a text file. While native geometry is robust and efficient for simple models, improvements in computational power now enable the use of computer-assisted design (CAD).

The benefits of CAD geometry are increased efficiency and accuracy. Complex models with hundreds of components can be created manually but with low human efficiency. Such work is tedious and error-prone. Instead, it is desirable to use CAD models to define geometry. In addition to reducing human error, CAD models provide a common domain with other analyses such as heat

[1] MCNP also supports fourth-order elliptical tori [3].

transfer, fluid modeling, stress analysis, and electromagnetism. Shared geometric domains form a common basis for solution coupling between simulations. Finally, CAD geometry can represent free-form surfaces. Native representations are typically limited to planar and quadratic surfaces.

Monte Carlo codes perform geometric operations as listed in Table 1.1. During initialization surfaces and volumes are measured for tally normalization. The point inclusion test determines inside which volume a particle starts. The next surface, or *ray tracing* operation determines the distance and surface at which a particle exits a volume. Ray tracing is performed during each particle advance and requires significant computational effort. The next volume test determines which volume is on the other side of a surface. Used every time a particle crosses a surface, this function can be implemented as a lookup table. The closest surface function is used for charged particle transport. It limits the step size as charged particles are affected by different materials in adjacent volumes [4]. The surface normal function is used for source distributions, tallying, and reflective surfaces.

Although the geometric operations in Table 1.1 can be computed on both native and CAD geometry, CAD geometry is not inherently suited for Monte Carlo radiation transport. Undefined nonsolid space and geometric overlaps are challenging to Monte Carlo codes.

Nonsolid Space All space must be defined such that particles are always inside exactly one volume. This implies that both solid objects and nonsolid space need to be explicitly modeled. CAD models do not typically include space occupied by vacuum, gas, or liquid. For example, a reactor model must include coolant volumes in addition to fuel and structural materials. Representation of nonsolid space is a challenge for the use of CAD models in Monte Carlo simulations.

Overlaps Because particles must always be inside exactly one volume, it is implied that volumes cannot overlap. Small overlaps among adjacent volumes are caused by imprecise modeling, file translation, and intentional deformation. Draftsmen may create CAD models for prototype visualization instead of more rigorous purposes such as meshing or Monte Carlo simulation. Often CAD models are shared by translation through neutral file formats, such

Table 1.1: Geometric operations for Monte Carlo radiation transport.

Geometric Operation	**Description**
Measure	Given a volume or surface, determine the volume of the volume or the area of the surface.
Point Inclusion	Given a volume, particle location, and possibly direction, determine if the point is inside, outside, or on the boundary of the volume.
Next Surface	Given a volume, particle location, direction, and possibly previous surface, determine the next surface of the volume that the particle intersects. Determine the distance to intersection.
Next Volume	Given a volume and surface, determine the adjacent volume on the other side of the surface.
Closest Surface	Given a volume and particle location, determine the distance to the closest surface of the volume in any direction.
Surface Normal	Given a surface and particle location, determine the unit normal vector of the surface at a point closest to the particle location.

as the Standard for the Exchange of Product model data (STEP) or the International Graphics Exchange Specification (IGES). Neutral formats must approximate the internal representation used by solid modeling engines, incurring translation error.

There are two approaches for performing geometric operations on CAD geometry. One approach is to translate CAD geometry into the CSG that Monte Carlo codes natively use. Translation avoids altering the Monte Carlo code. The richness of the translation is limited to the geometric primitives supported by the Monte Carlo code. Free-form surfaces must be approximated as piecewise first and second order surfaces. This limitation is important if the CAD model contains significant higher-order surfaces. Piecewise surface approximations have a one-to-many instead of one-to-one mapping from CAD to CSG.

For computational efficiency, some translation approaches simplify complicated volumes into several convex volumes. Convex volumes are preferred because the closest of all intersections is accepted as the next intersection. On the contrary, continuous surfaces that bound concave volumes contain intersections that do not lie on the boundary of the volume, making selection of the next intersection more complicated. Volumes with few bounding surfaces are preferred because MCNP's native geometry implementation must search every surface of a volume for intersections each time a particle advances. Lack of correspondence between CAD entities and CSG entities makes model manipulation difficult. For example, determining particle flux across a CAD surface may require finding several CSG surfaces over which to compute the corresponding tally.

Several translation approaches have been created for the Monte Carlo N-Particle (MCNP) code developed at Los Alamos National Laboratory [3]. MCNP is a widely-used code that supports coupled neutron, photon, and electron transport.

Moritz Developed by White Rock Science [5], Moritz can import STEP-formatted CAD models containing planar, quadratic, and tori surfaces that MCNP supports.

MCAM The Monte Carlo Automatic Modeling (MCAM) system [6, 7, 8] from the Chinese Academy of Sciences is currently at version 4.7. It supports bi-directional translation between CAD and native Monte Carlo geometry, including MCNP. Capable of geometry creation and repair, it uses the ACIS modeling engine. MCAM's *Fixer* is used to fix small gaps and overlaps between adjacent volumes. It has a graphical interface through which the user can heal, decompose, and reconstruct geometry. The heal function can detect and correct a large percentage of defects in the model [9]. Volumes are reconstructed to remove gaps. Decomposition is used to simplify complicated volumes into convex volumes. Volume importance, materials, and tallies can be specified through the graphical interface.

TopAct The Translation Optimization for Part-wise Adaptive Combinatorial Transport (TopAct) is a translator developed by Raytheon and used by Lawrence Livermore National Laboratory (LLNL) [10, 11]. It translates CAD files in Parasolid, STEP, and IGES formats to CSG used in MCNP and TART [12]. Materials can be assigned in Pro/ENGINEER, the TopAct graphical interface, or a text file. Geometric defects are detected by flooding the geometry with particles. The graphical interface is used to suggest the location of defects, inferred by lost particles.

McCad McCad is a bi-directional translator with graphical interface developed at Forschungszentrum Karlsruhe (FZK) in Germany [13, 14]. Models are preprocessed in McCad to remove small features and overlaps. Nonsolid space is automatically defined during conversion to CSG. Like other translation approaches, McCad increases the number of volumes. In one example the number of volumes increased from 916 to 6025, including 3736 void volumes [13]. McCad uses the Open CASCADE solid modeling engine and STEP file format [15].

GEOMIT A team from the Japanese Atomic Energy Agency has created the GEOMIT translator [16]. Using the Parasolid modeling engine, nonsolid space is created by subtracting volumes from a grid of cubes in the solid model. When converted to MCNP's native CSG, many cubes are preferred over a single, complex void volume. As with other translators, the geometry must be simplified to conform to MCNP's available surface definitions.

The other approach for using CAD geometry in Monte Carlo simulations is the direct approach. The geometric operations listed in Table 1.1 are computed directly on the CAD geometry. This approach requires changing the Monte Carlo code's geometry routines. Ideally, the CAD model can be utilized without simplification or preprocessing. Direct computation on the CAD model is slower than on native geometry because ray tracing is more expensive on complicated surface representations.

MCNP-BRL A linkage between MCNP and BRL-CAD was developed at Oak Ridge National Laboratory [17]. BRL-CAD is a CSG solid modeling system originated by the U.S. Army's Ballistic Research Laboratory in 1979, and released open source in 2004. Geometric operations are handled by a BRL-CAD API. The user provides a CAD file, MCNP data cards, and a file with attributes of the CAD model. The `rtcheck` function of BRL-CAD allows users to detect geometry overlaps using ray tracing [18]. The BRL-CAD engine uses CSG instead of boundary representation (BREP). Translation from BREP to CSG is possible using faceted solids through BRL-CAD's bag of triangles (BOT) primitive [19].

OiNC A collaboration at Sellafield and Serco in Britain have created a direct geometry system for the MCBEND [20] and MONK [21] Monte Carlo codes [22]. The OiNC system can import both IGES and stereolithography (STL) models. It is possible to use Imported CAD geometry in the same model as native CSG. This hybrid method combines the efficiency of CSG representation and with the flexibility of BREP. OiNC can use natively input CSG and convert portions of IGES models to CSG. Combinatorial logic is used to convert complicated solids into Boolean expressions of simple objects. Hierarchical bounding boxes are used to accelerate searching among solids. Octree acceleration is used for ray tracing on STL solids.

DAGMC An implementation of the direct approach was developed at the University of Wisconsin [23]. Known as DAGMC (Direct Accelerated Geometry, Monte Carlo) or DAG-MCNP when coupled with MCNP, this effort does not depend on a specific solid modeling engine. Through the Common Geometry Module (CGM) [24], DAGMC has used the ACIS and CATIA solid modeling engines, with support for Open CASCADE underway. Instead of

developing a unique graphical interface, it uses existing CAD design tools for model manipulation and preparation. The DAGMC library, including mesh geometry and tally manipulation are implemented using the Mesh Oriented datABase (MOAB) [25], hosted at Argonne National Laboratory (ANL). Mesh results are viewed with the VisIt visualization tool from LLNL [26]. DAGMC, CGM, and MOAB are open source libraries available on the web from world-readable repositories at `http://trac.mcs.anl.gov/projects/ITAPS`.

Although several CAD-Monte Carlo linkages are under development, the work of this thesis is implemented in DAGMC. In the author's view, the direct approach is preferred because it avoids the creation of a separate translated model, provides a richer surface representation, and inherently supports mesh geometry. DAGMC is mature and has been used for numerous fission and fusion applications [27, 28, 29, 30]. CGM provides a consistent interface for various solid modeling engines. It adds features that are not always found in solid modeling engines such as virtual geometry and non-manifold surfaces. MOAB provides a flexible framework for features added as part of this work.

1.3 DAGMC

The analysis procedure and implementation of DAGMC will be discussed. Such background is necessary to understand the shortcomings that motivate the work of this thesis, and provides context for Chapters 2-5.

1.3.1 Analysis Procedure

Performing radiation transport with DAG-MCNP is comparable to fluid, heat transfer, structural, or electromagnetic modeling. The user assigns boundary conditions to a well-posed CAD model. Solution parameters are entered before solving the simulation. Results are viewed and interpreted. The following steps are used for a typical analysis.

Create Solid Model A solid model is created in any solid modeling engine that can export a neutral file format (STEP or IGES). ACIS output is preferred because CGM is usually built

with the ACIS solid modeling engine. Adjacent volumes cannot overlap. This step does not require familiarity with radiation transport.

Preprocess Geometry for DAGMC Geometry is preprocessed in the CUBIT Geometry and Mesh Generation Toolkit [31]. If insignificant to the radiation transport simulation, small details can be removed. Boundary conditions are added for reflecting and absorbing surfaces that limit the extent of the model. Groups are created for material and tally specification. All space must be defined in a volume, including fluids and vacuum. Coincident surfaces must be imprinted and merged such that only one surface exists between adjacent volumes. CUBIT can display unmerged surfaces and automatically check for overlapping surfaces and volumes. It is common for overlaps to occur among small details in the model.

Create MCNP Input File A standard MCNP input file is prepared including only data cards. Cell and surface cards are instead described by the CAD model. The input file specifies the particle source, isotopic composition of materials, runtime, and physics parameters. If desired, mesh tallies are defined in the input file.

Run DAG-MCNP DAG-MCNP requires both the MCNP input file and CAD file. Only the geometry functions inside MCNP have been modified, so DAG-MCNP executes much like MCNP.

Visualize Results If mesh tallies are specified, they can be viewed using VisIt. VisIt can open MCNP5's meshtal files through the ITAPS-MOAB MCNP5 reader. Mesh results can be interpolated for multiphysics coupling with existing MOAB tools.

1.3.2 Implementation

The DAGMC library is implemented as a MOAB tool in C++. MCNP, written in FORTRAN, calls DAGMC library functions through a C interface. Although typically implemented with MCNP, coupling to other Monte Carlo codes has been accomplished through DAGMC's generic interface. Installation is targeted for the Linux operating system with GNU compilers.

During initialization, CGM calls the solid modeling engine to generate a faceted representation of curves and surfaces creating a facet-based model (FBM). Volumes are not meshed. Instead, volumes are represented by parent-child relationships to faceted surfaces. Faceted surfaces are linked by parent-child relationships to faceted curves. The resulting FBM is stored in a MOAB instance, accessible to DAGMC. Faceted surfaces are used to accelerate geometric operations in DAGMC. The accuracy of the faceted approximation can be specified by the user, although the default 10 μm has been found to be sufficient in most cases. The Facets of each surface are contained in MOAB's geometry set for each surface. All DAGMC geometry functions are optimized to use the faceted approximation.

Five different volume representations are used in DAG-MCNP. Two volume notions, the ID and index, are used by the MCNP and must correspond to volumes of the CAD model. The MOAB geometry handle, oriented bounding box (OBB) handle, and CGM geometry handle are specific to DAGMC. MCNP IDs and indices are both integers, while handles are integer pointers to MOAB data locations. The extra volume notions are required for CAD-specific geometry operations.

CGM Geometry Handle If DAGMC is initialized with a solid modeling engine and geometry is provided as a CAD file, then a list of surfaces and volumes available through CGM is created.

MOAB Geometry Handle Each volume and surface is given a MOAB geometry set handle. In addition to containing entities, MOAB's volume and surface set handles have parent-child relationships and store generic data. MOAB geometry handles are tagged with entity IDs and OBB handles. MOAB surface handles are tagged with MOAB volume handles to determine forward and reverse senses.

MOAB OBB Handle OBB trees provide an efficient method to check for ray intersections [32, 33]. The solid modeling engine facets each surface into a set of triangles, or *facets*. A tree of hierarchical bounding boxes is constructed from the triangles of each surface. Bounding boxes are arranged in a hierarchy such that successfully intersected large parent boxes lead to intersection checks of smaller, child boxes. In this manner the intersected facets can be

found through a tree of progressively smaller bounding boxes. Bounding boxes of surfaces are collected to represent volume bounding boxes.

MCNP ID The MCNP IDs are identical to the CAD volume IDs. Solid modeling engines number each volume as it is created. Because volumes are created and deleted during CAD manipulation, MCNP IDs are not continuous. In a native MCNP input file, MCNP IDs are user-specified cell numbers assigned to each volume in the list of cell cards, and need not be continuous. Preserving the correspondence between CAD IDs and MCNP IDs is an advantage of direct approaches over translation approaches.

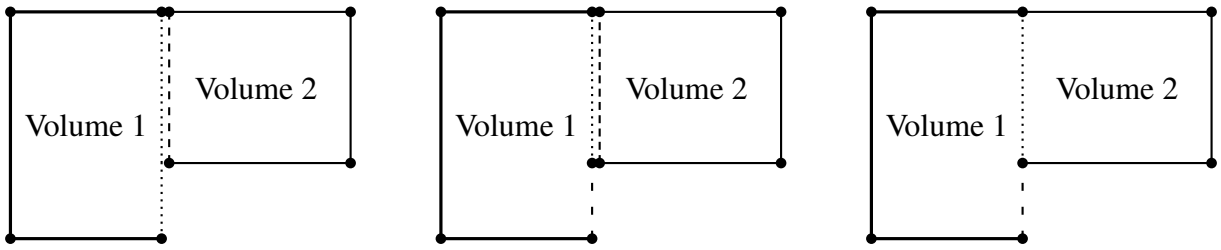
MCNP Index A continuous list of MCNP indices is created from the MOAB geometry handles. In native MCNP, the same list of indices is created by renumbering MCNP IDs into a continuous list in the order they are given in the input file.

Calls to the geometric operations listed in Table 1.1 are intercepted in the physics code. These calls use DAGMC functions instead of their native geometry functions. DAGMC uses a MOAB instance to answer geometric queries about the CAD model. Most computational effort is required by the `point_inclusion` and `next_surface` functions.

1.4 Statement of Problem

The original version of DAGMC was a significant improvement over manual creation of native geometry for Monte Carlo radiation transport. Foremost, accuracy improved because the error-prone task of manually creating native geometry is avoided. Accuracy also improved because CAD models provide a richer surface representation than is typical of native geometry. Quality control improved because material properties, boundary conditions, and tallies are assigned while manipulating the model in a 3D CAD interface. DAGMC can use the same material sets in the CAD model as defined for other physics simulations, reducing the chance of human error due to material assignment.

Although the accuracy of the simulation increased, the original version of DAGMC did not always increase the human efficiency of creating complex models. For complex models the tedious,



(a) Before imprint. (b) After imprint. (c) After merge.

Figure 1.2: Two adjacent volumes are imprinted and merged to create a non-manifold model.

time-consuming, and error-prone task of manual geometry creation was replaced, in part, by the tedious and time-consuming tasks of imprinting and merging coincident surfaces. Imprinting and merging is automated in CUBIT, but does not always succeed due to modeling errors and tolerance problems caused by file translation. Such errors are too small to visually detect, but can be revealed using CUBIT's automated overlap detection.

Imprinting is the process of making surfaces and curves that are partially coincident in space be fully coincident. This is accomplished by splitting the original entities into coincident and non-coincident parts. This process can be viewed as the opposite of regularization, a common operation in most CAD systems. Following this, entities coincident in space and with like topology are merged, such that only one entity remains. This merging step modifies the topology of the geometric model, such that the remaining merged entity is adjacent to all entities bounded by the entities that were merged. Surfaces bounding two volumes have a sense (either forward or reverse) with respect to each of those volumes. By convention, a forward sense implies the surface normal points out of the volume. The result of this process is a non-manifold model—some surfaces shared between neighboring volumes, and vertices and curves bounding multiple volumes.

Figure 1.2 shows the imprint and merge process for two adjacent volumes. The pair of adjacent surfaces are partially coincident, although they are shown apart to clarify this example. Imprinting splits the surface from volume 1 into a surface that is fully coincident and a surface that is not coincident, as shown in Figure 1.2b. The fully coincident pair of surfaces are merged, leaving a single surface between volumes 1 and 2.

Non-manifold models are preferred for Monte Carlo radiation transport because they encourage efficient particle tracking. When traversing adjacent volumes, particles must cross either two coincident unmerged surfaces (manifold case) or a single merged surface (non-manifold case). The manifold case requires an additional ray tracing step to determine the next surface, although the surfaces are coincident. Unmerged surfaces increase the computational expense of tracking particles across neighboring volumes.

The computational expense of tracking particles across unmerged surfaces could be justified by avoiding the tedious process of manually merging surfaces that fail to merge automatically. However, tracking across coincident unmerged surfaces is problematic because the `next_surface` function cannot reliably find intersections at zero or negative distance along the particle trajectory. Merged surfaces would not be necessary if the tracking algorithm was tolerant of overlaps, which cause the next surface intersections to occur at negative distances. The development of an overlap-tolerant tracking algorithm would simplify the preprocessing step of DAGMC analysis.

The original version of DAGMC required that all space be explicitly defined in the solid model. It is typical for solid models to include only manufactured components, and not gas or liquid volumes. Nonsolid space, known as *complement*, can be created in CAD programs by using Boolean subtraction on explicit volumes. It is common for this operation to fail due to the complexity of the complement. Even if creation of the complement succeeds, imprinting and merging the surfaces of the complement is a difficult task. Instead of investing weeks to manually create native geometry, one could spend days creating complement, then imprinting and merging surfaces. In this situation, the original version of DAGMC replaced one tedious task with an equally-tedious alternative. By avoiding the explicit creation of nonsolid space as discussed in Chapter 4, an implicit complement would simplify the preprocessing step of DAGMC analysis.

The accuracy of DAGMC is limited by lost particles. Lost particles remove statistical information from the simulation, and may influence results because particles are not uniformly lost throughout phase space. Particles become lost when the tracking algorithm cannot determine where the next intersection occurs. As discussed in Chapter 2, adjacent faceted surfaces do not share the same boundary. Particles can leak between surfaces, becoming lost. Instead, surfaces

of the FBM should be sealed together to prevent leakage. By forming watertight FBMs, particles would no longer leak from the geometry.

Another cause of lost particles is DAGMC's tracking algorithm. As described in Chapter 3, several cases exist which are known to challenge the tracking algorithm. Problems are due to both faulty logic and numerical imprecision. Examples include collisions near surfaces, tangent intersections, and oscillation between surfaces. Robustly handling such cases will decrease the number of lost particles, increasing the accuracy of DAGMC.

The largest remaining loss of human efficiency is due to small overlaps in the CAD model, as addressed in Chapter 5. Small overlaps between adjacent volumes should not significantly affect results of the simulation. However, overlaps cause particles to become lost in the original version of DAGMC. Overlaps can be removed in a CAD program such as CUBIT. In some cases, overlaps can be eliminated by training draftsmen to built CAD models more precisely. In new applications such as the mesh-based simulation of deformed reactors, overlaps cannot be avoided. The ability for DAGMC to tolerate overlaps will significantly improve human efficiency by avoiding manual CAD repair.

1.5 Statement of Thesis

This thesis will increase the ease of use and accuracy of Monte Carlo radiation transport by sealing faceted surfaces to become watertight, developing a robust tracking algorithm, implicitly creating nonsolid space, and tolerating geometric overlaps. These features will be implemented in the DAGMC library, without sacrificing computational efficiency. The elimination of both manual CAD repair and lost particles will be demonstrated with CAD models used in production calculations.

1.6 Overview of Thesis Structure

One chapter is dedicated to each of the four topics. Each topic will be motivated before previous efforts are reviewed. An algorithm will be presented with implementation details, followed

by testing and analysis. Although each chapter is self-contained, their order is not arbitrary. First, lost particles are eliminated by watertight faceting and robust tracking. Next, manual CAD repair is eliminated by the implicit complement and overlap-tolerant tracking. These improvements are shown graphically in Figure 1.3. The overlap tolerance chapter is concluded with a detailed explanation of deformed space reactor analysis. The deformed space reactor application was enabled by the implicit complement, robust tracking, and overlap tolerance features added to DAGMC.

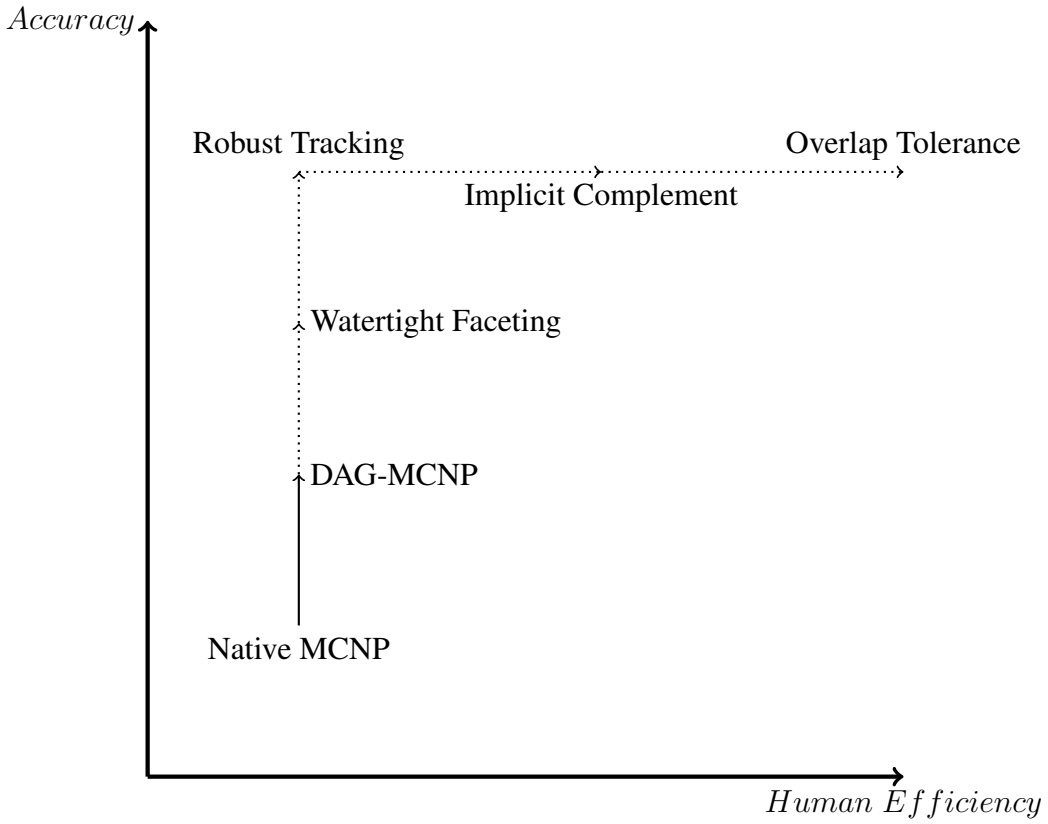


Figure 1.3: New features improve the accuracy and human efficiency of DAG-MCNP.

Chapter 2

Watertight Faceting

2.1 Motivation

The fundamental geometric computation in Monte Carlo radiation transport simulations is the determination of the next surface intersection. Given the numerical position of a particle and its direction of travel, the distance to the next surface intersection is found. The process of determining the next surface intersection, known as *ray tracing*, is fundamental to computer graphics. The computer graphics community is most concerned with the computational speed of ray tracing, or how fast frames of an animation can be rendered. Speed is of interest to Monte Carlo radiation transport, however accuracy and consistency are also important.

To increase the efficiency of ray tracing and other geometric queries, the CAD model is discretized in the form of a facet-based model (FBM) [33]. A FBM represents geometric vertices, curves, and surfaces by collections of points, edges, and triangles. In addition to accelerating geometric queries, FBMs serve a variety of uses in computational simulation. They are often the means of transferring geometric models between applications, often in the form of stereolithography (STL) files [34]. FBMs are also used as the basis for generating the three-dimensional discretization, or mesh, for a given geometric model [35, 36].

FBMs promote efficient ray tracing. Intersections are found among faceted surfaces using a series of ray-triangle tests. This avoids costly iteration during the root-finding procedure when intersecting continuous, high-order surfaces of the solid model. Instead of performing a linear test of a ray with every triangle of a surface, the triangles can be arranged in a hierarchical tree.

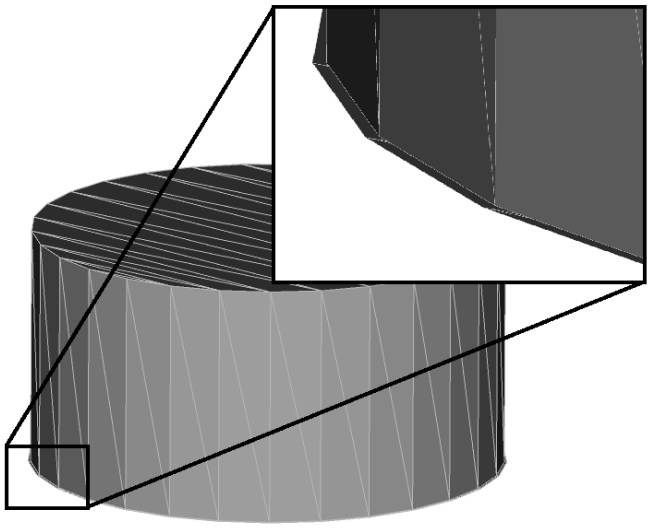


Figure 2.1: A gap exists between surfaces of the cylinder because each surface is faceted independent of bounding curves.

By avoiding groups of triangles outside the vicinity of the ray, the number of ray-triangle tests is reduced to $O(\log n)$ where n is the number of triangles.

One of the reasons FBMs are used so widely is that they are provided by virtually all geometric modeling systems. Modeling engines such as ACIS [37] and Open CASCADE [38] all provide application programming interface functions for getting the triangular facets for each surface and facet edges for each curve. Typically, facet edges and triangles are guaranteed to be within a facet tolerance of the geometric entities they resolve, and this facet tolerance is input to the modeling engine. However, most FBMs provided by geometric modeling systems suffer from a fundamental flaw which prevents their usage as-is for other applications: the FBMs are not watertight. That is, each surface and curve in the geometric model is faceted independently, with neighboring surfaces not sharing facet points with the curve where they meet nor with each other. While there may be points in each of the facetings that are coincident along the shared curve, this is not always the case, and for non-manifold models, this is almost never true. For example, Figure 2.1 shows a model where facetings of two surfaces of a cylinder are far from coincident at their bounding curve.

Flaws must be fixed before FBMs can be used for ray tracing and other applications mentioned above. Occurring as gaps and overlaps, flaws cause ray tracing to fail by missing intersections between adjacent surfaces. Particles leak through volume boundaries and become lost. In addition to problems with ray tracing, flawed faceting may cause ambiguous results from point inclusion tests (PITs). PITs are used to determine in which volume a particle starts and require a watertight volume represented by a closed, orientable shell. As a consequence of faceting each surface independently, faceted surfaces do not form a closed shell.

There has been a great deal of previous work on the subject of making faceted models watertight. These efforts fall roughly into two groups: one that views the faceted model similar to a point cloud that is the basis of deriving a closed faceted model, and another group that focuses on fixing localized problems in the model, e.g. by filling and closing holes in a faceted surface. None of the efforts reviewed in this work makes use of the topological information often available from geometric models, and few work on models having non-manifold topological features. Furthermore, these approaches vary greatly in their robustness, and few come with guarantees about what kind of FBMs can be made watertight. Implementations of these methods are also not available as open-source software, for use with other geometric modeling packages. A provably reliable solution for fixing FBMs, in an open-source implementation that could be applied in other modeling environments, is a key capability in developing other applications on FBMs.

The contribution of this chapter is the demonstration of an automatic algorithm for sealing facet-based models. However, the algorithm is not intended to fix point cloud data or repair large holes. It is demonstrated how using topological information that accompanies a certain class of geometric models simplifies this process, and prove that, under certain weakly-restrictive conditions, the method is guaranteed to succeed. Robustness is demonstrated using non-manifold geometric models from a variety of applications, having thousands of surfaces and volumes. The algorithm is freely available as open-source software under the LGPL license at `http://trac.mcs.anl.gov-/projects/fathom/wiki/MeshKit`.

2.1.1 Statement of Problem

The faceting of adjacent surfaces along curves does not match. There is no one-to-one correspondence between facet edges of adjacent surfaces. Instead of being watertight, gaps and overlaps exist along curves. This algorithm will *fix faceting flaws between surfaces*. As the output of the watertightness algorithm, both curves and facet skin will be represented by a single set of edges. The skin of adjacent faceted surfaces is merged topologically to create sets of pseudo two-manifolds. That is, the neighborhood at every point on the boundary of a volume will be topologically equivalent to a two-dimensional disk.

The remainder of this chapter is structured as follows. The balance of this section discusses previous work in fixing FBMs, and the nomenclature and functions used for accessing FBMs and geometric models. The algorithm for *sealing* FBMs, along with a proof of its reliability, is given in Section 2.3. Implementation of the algorithm is described in Section 2.4, with results given in Section 2.5. Conclusions are described in Section 2.6.

2.2 Theory

There are two approaches to creating a watertight FBM. One approach is to facet entities constrained to their bounding, lower-dimensional faceted entities. The resulting model is guaranteed to be watertight because the faceted curves are the boundary of the surface faceting. An alternative approach is to independently facet each surface of the solid model, then fix the flaws to obtain a watertight faceted model. It is likely to be less complex to fix an existing faceted model than to write a robust algorithm that will facet surfaces constrained to their faceted curves. This approach is pursued in the following sections of this chapter.

Previous work in converting flawed FBMs into watertight faceted models can be grouped in two categories. Volume-based approaches may guarantee watertightness of a solid bounded by triangles, but at the cost of reconstructing the entire volume using an intermediate representation. Mesh-based approaches fix faceting errors with small perturbations in the vicinity of the error, but offer no guarantee of watertightness for the overall volume [39]. In a recent survey [40], Ju

concludes that mesh-based methods have been most successful at repairing CAD models where errors are localized, while volume-based methods are most successful at reconstructing polyhedra from poor quality data, with loss of detail.

Volume-based methods [41, 42, 43, 44] reconstruct the model based on lower-level data derived from the faceting. Because they discard the original facets, these methods do not preserve the details of the original geometric topology, at least in cases where those details are not also geometrically distinct. Thus, surfaces imprinted on another surface will not be preserved, since they are not geometrically distinct. Although volume-based methods are the only option in the case of range-scanned data, they are not applicable to the problems described here, and are not considered further in this work.

Watertight volumes, known as regularized sets, are rigid solids that are subsets of three-dimensional Euclidean space which are bounded, closed, regular, and semi-analytic. The boundary of a regularized set is the closure of its interior [45]. Also, the boundary of a regularized set is a two-manifold, meaning that the neighborhood of every point in the set is homeomorphic to a two-dimensional disk [46]. Two-manifolds can be further qualified such that every edge belongs to exactly two facets, every point is surrounded by a single cycle of edges and facets, and facets may not intersect each other except at common edges or points [47]. Non-manifold surfaces can be viewed as two surfaces which are geometrically coincident but not topologically coincident. The boundaries of adjacent volumes are known as pseudo two-manifolds [46, 47]. Pseudo two-manifolds are orientable and can be treated as two-manifolds, allowing merged surfaces to form watertight volume boundaries.

Bohn and Wozny [46] create a watertight shell by using topology to identify local gaps in the faceting, which they close based in part on heuristics. The algorithm does not move or merge any points. Their method begins by identifying edges that belong to a single facet, known as free edges. Free edges are organized into loops, where each loop bounds an area that must be filled in with facets. This filling is done using an *ear clipping* algorithm, where two consecutive edges in the loop are used to form a triangle, with the third edge replacing the two edges in the loop. Edge

pairs with the smallest angles are favored in this approach. The algorithm concludes by using edge flips to improve geometric quality of the facets.

Barequet and Sharir also approach this problem [48] by idenitfying free edges, but split those edges such that the resulting edges are more consistent in edge length. A distance- and orientation-based heuristic is then used to determine matching groups of edges; gaps between these groups are filled by constructing new facets. In a subsequent effort [49], points within a certain distance tolerance are moved to eliminate gaps between triangulated surfaces. Edges are considered in pairs, rather than by groups forming arcs or loops. Free edges that remain after this process are identified and assembled into loops, and triangulated using [50]. In ambiguous situations, the user is prompted for guidance. This approach is likely to decrease the speed of geometric computations on the watertight model due to excessive creation of new triangles.

Sheng and Meier [51] use a series of increasing distance tolerances, up to a maximum tolerance, to merge points then edge pairs whose endpoints have been merged. This ensures that the best candidates will be paired with each other. One drawback of this approach is that it requires facet edges of similar lengths to close gaps in the faceting; this is often not the case in typical facetings returned from modeling engines. Geuziec et. al [52] also note that Sheng and Meier's algorithm may create edges adjacent to more than two triangles, when the same edge is merged more than once. They avoid this simply by preventing merges that will create edges adjacent to three triangles.

Borodin et al. improves on the merge-only approach by introducing a point-edge contraction, or t-joint operator [53, 54]. In this operation a free edge is split if a corresponding point is too far from either end but close enough to the edge. Point-edge contraction allows model repair to proceed using a smaller tolerance than if implemented with point-point contraction alone. Kahlesz et al. [55] use a similar approach.

An approach by Busaryev et al. [56] represents surface boundaries as strings of intersecting balls. Boundaries of adjacent surfaces are joined by combining balls using a tolerance. Although the watertight output is produced using Voronoi diagram and Delaunay triangulation techniques, the *repair* phase of combining balls resembles point-point contraction.

The advances in mesh-based watertightness are well summarized in an algorithm by Chong et al. [57]. First point-point contraction is performed by proximity. Next point-edge contraction is used, then large holes are triangulated. Finally skewed elements are removed by edge swapping or point-point contraction. The last step is important because prior point-point and point-edge contraction likely create facets with poor aspect ratios.

These mesh-based algorithms attempt to restore missing information about where facets meet to close gaps, using three methods:

1. Perform point-point contraction by proximity.

2. Perform point-edge contraction by proximity.

3. Triangulate a patch across gaps that minimizes area or another heuristic quantity.

However, much of the information they try to restore based on spatial proximity and searching was known at one point, but was lost or thrown away. For example, point-point and point-edge contractions are usually motivated by sealing neighboring geometric surfaces at shared curves. Knowing these topological relationships greatly narrows the search for matching free facet edges. In the case where the geometric model is still available, new points in the faceting can even be projected to the geometric entities the faceting is supposed to resolve. *Methods for repair of FBMs are greatly improved by taking advantage of geometric model information that previously has not been used. Furthermore, this information is often available when the original faceted model is created, but has not been used because of limitations in file formats used to transfer FBMs to applications.* The sealing algorithm described in this chapter uses geometric model information to improve the robustness of obtaining watertight FBMs.

2.3 Algorithm

Although watertight FBMs are generically useful, the requirements listed below will yield an algorithm optimized for DAGMC. Supporting non-manifold models with minimal creation of new triangles will preserve simulation efficiency. Minimizing deformation of the input model will

ensure that sealed FBMs are as geometrically accurate as possible. Previous methods to obtain watertight faceting have in some cases relied on user guidance. This type of guidance is impractical for the large models encountered in our work. Therefore, automation is critical to practical use of this algorithm.

The requirements of the proposed algorithm are:

- Seal faceted surfaces along curves to create a watertight model.
- To preserve human efficiency, the algorithm must be automatic.
- New facets must be owned by exactly one surface.
- Support non-manifold surfaces.
- Fast enough to use as a preprocessing module.
- Deformation of input model should be minimized, if possible.
- Creation of new triangles should be minimized, if possible.

2.3.1 Assumptions

In this work, the geometric model is in the form of a boundary representation (BREP). This model consists of geometric vertices, curves, surfaces, and volumes. Note that non-manifold topology is allowed; in particular, many models have multiple volumes, with surfaces shared between two of those volumes, and curves and vertices shared by multiple volumes. The collection of geometric model entities forms a cell complex; that is, the intersection of two entities of like dimension is either empty or corresponds to one or more entities of lower dimension that are also in the cell complex. Each vertex, curve, and surface has a corresponding faceting represented by a set, with each of these sets containing points and possibly edges and triangles, depending on the topological dimension. Faceted volume sets are empty, since volumes have no facets, but are still represented in the model, to get adjacency relations with other surfaces. Assume that the faceting for each geometric curve with a single vertex has at least three facet edges. Curve facetings not satisfying this

assumption would not be useful for graphics purposes, and in practice this assumption is satisfied in most observed facetings. Each faceted entity itself is also a $d-$dimensional cell complex; however, this cell complex does *not* share points with the facetings of other model entities, even those bounding the model entity in question. This is also typical of faceted models from most modeling engines.

There are two tolerances that are important to the algorithm described in this chapter. First, the *merge tolerance* is the distance below which two entities are considered spatially coincident; denote this as ϵ_g, recognizing that in effect this also serves as the geometric tolerance for those model entities. This tolerance is used in various places during the merging process described earlier. The facet tolerance ϵ_f is the maximum distance between a faceted curve or surface and the geometric curve or surface it resolves.

Several assumptions exist about the faceting, based on guarantees provided by the modeling engines constructing them. First, all facet points are within ϵ_g of the corresponding model entities. While this is typical of faceted models from most modeling engines, it could also be achieved by projecting facet points to the corresponding model entity, using a function provided by virtually all modeling engines. Second, all facet edges and triangles are within ϵ_f of the corresponding model entities. Most modeling engines providing facetings take ϵ_f as an input parameter, though in practice this input is disregarded if it is much larger or smaller than the default value used by the modeling engine. Also assume that $\epsilon_f \gg \epsilon_g$; since both these parameters can be changed through user input, this does limit the variations allowed in one parameter after the other has been chosen. Finally, all points on the boundary of a given faceting are within ϵ_g of *some* model entity that bounds the corresponding model entity, though *which* bounding model entity is not known on input. While not stated explicitly by the modeling engines, in practice this has been found to be the case for both ACIS and Open CASCADE. Each faceted curve and surface is non-degenerate (all points of $d-$dimensional facets, $d > 0$, are distinct), and is oriented and non-inverted (normal of facet is consistent with that of the underlying model entity in the neighborhood of the facet[1]).

[1] By convention, the surface normal vector points outward from the volume.

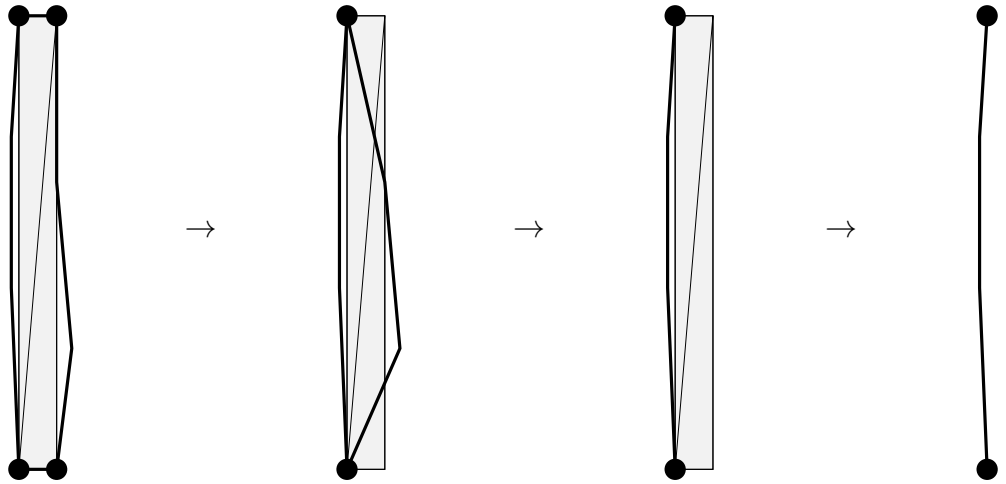
Although cases have not been encountered where this assumption is invalid, invalidities of this type could be fixed using preprocessing similar to that discussed in Section 2.3.2.

The *local feature size* (LFS) [58] at point x is the radius of the smallest closed disk centered at x that intersects two vertices, two non-adjacent curves, or a curve and a vertex that is not adjacent to the curve. Assume that LFS $\gg \epsilon_f$. Cases where this assumption is not valid have been observed; preprocessing is used to fix these cases, as described in Section 2.3.2.

The input to this algorithm is a set of vertices, curves, surfaces, and volumes with appropriate topological relationships. Surfaces and curves are represented by facets and edge elements that resolve their geometric entities to within ϵ_f. The boundary of each faceted surface is not the same as the surface's adjacent faceted curves. The endpoints of adjacent faceted curves are the same such that facet edges can be joined to form loops. The boundaries of volumes are represented by the facets of their bounding surfaces. The facet tolerance is selected based on the feature size and required accuracy of the faceted representation. The input model is faceted, implying that ϵ_f has previously been selected.

2.3.2 Preprocessing

On input, the geometric model may not satisfy the local feature size assumption; that is, the model may contain features that are smaller than the facet tolerance input to the sealing algorithm. For example, the 40° section of the ITER benchmark model, discussed in Section 2.5, has multiple curves and surfaces of near-zero length and area, respectively. These features must be removed before the sealing process can start. Curves with length less than the facet tolerance are removed by merging their endpoints together as shown in Figure 2.2b; curves that average less than the facet tolerance apart from each other are merged together (Figure 2.2c). Surfaces can be removed if all of their bounding curves occur in merged pairs (Figure 2.2d). These surfaces are approximately one-dimensional and are the result of imprinting neighboring volumes in the solid model. Volumes are removed if all of their surfaces have been removed, though in practice this does not occur for any of the models tested for this work.



(a) Before preprocessing. (b) Remove small curves. (c) Merge curves. (d) Remove surface.

Figure 2.2: Preprocessing of an approximately one-dimensional surface.

One additional defect was identified for several faceted models used in this work, where skin points were distinct but within geometric tolerance (ϵ_g) of each other. This results in exactly two extra skin edges per defect. To remedy this, skin points within ϵ_g of each other are merged. The search for these defects is only within each surface's skin edges, rather than all geometrically proximate facet points. Therefore, locating the defect is relatively inexpensive, scaling linearly with the number of skin edges.

After preprocessing is completed, the FBM includes topological relations along with specification of the geometric and faceting tolerances ϵ_g and ϵ_f, respectively.

2.3.3 Sealing

Given the input described in Section 2.3.2, the goal is to seal facets in the model together, such that the facetings of the surfaces, curves, and vertices satisfy the requirements of a cell complex. That is, the intersections of points, edges, and triangles should correspond to other points and edges in the faceting.

In this section the sealing algorithm is presented in its entirety; proof of its success is discussed in Section 2.3.4. The general structure of the algorithm is to work surface by surface, sealing the skin of the surface's faceting to the faceting of bounding curves. The algorithm takes advantage of the known topological relations between surfaces and their bounding curves and vertices, and the correspondence between a vertex, curve or surface and its faceting.

The sealing algorithm is described in two parts. First, the higher-level algorithm for sealing a surface faceting with that of bounding curves is given in Algorithm 2.1 and shown in Figure 2.3. This part of the algorithm has five main parts:

1. Skin each faceted surface to resolve its bounding edges, or *skin edges* (Figure 2.3b). The collection of bounding edge is assembled into bounding loops using topology.

2. The endpoints of faceted curves are faceted vertices. Match vertices to points on the skin of the surface faceting using proximity (Figure 2.3c). The closest skin point is merged to the vertex.

3. Separate the loops of edges on the skin of the surface faceting into arcs using the vertices as separators (Figure 2.3d);

4. Associate arcs with corresponding curves that bound the surface (Figure 2.3e). This step uses a function $d_{oriented}$ that computes the average distance between two ordered lists of facet edges. This distance function is computed as the summed distance between parameterized points in both edge sequences normalized for edge length, *as the edges are traversed in their respective orders.* This direction-aware measure will compute a significant distance between two sequences of edges that are exactly spatially coincident but ordered in opposite directions. This algorithm is necessary for distinguishing a number of pathological cases observed in the facetings of various real geometric models.

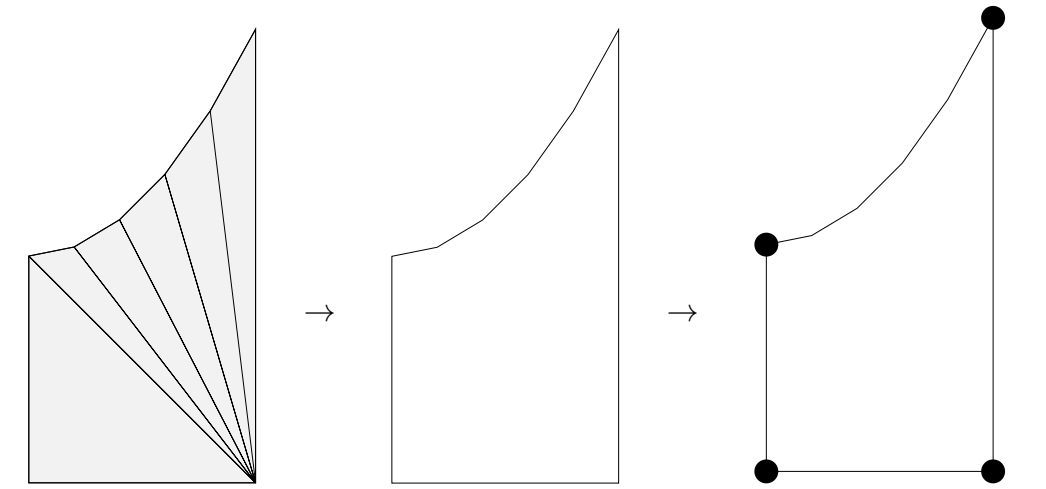
5. For each arc-curve pair, determine if the curve has already been replaced by an arc. If not, replace the curve with the arc. Otherwise, seal the edges of the arc to those of a corresponding curve that bounds the surface; this step requires an algorithm `seal_pair`, described in Algorithm 2.2.

   The skin of each adjacent surface must be sealed to the curve. By replacing the curve with the first arc that it is to be sealed with, the number of constraining arcs is reduced by one. Each constraining arc can increase the number of facet edges (and adjacent triangles) due to point-edge contraction. Reducing the number of constraining arcs for each curve allows the sealed FBM to have fewer triangles.

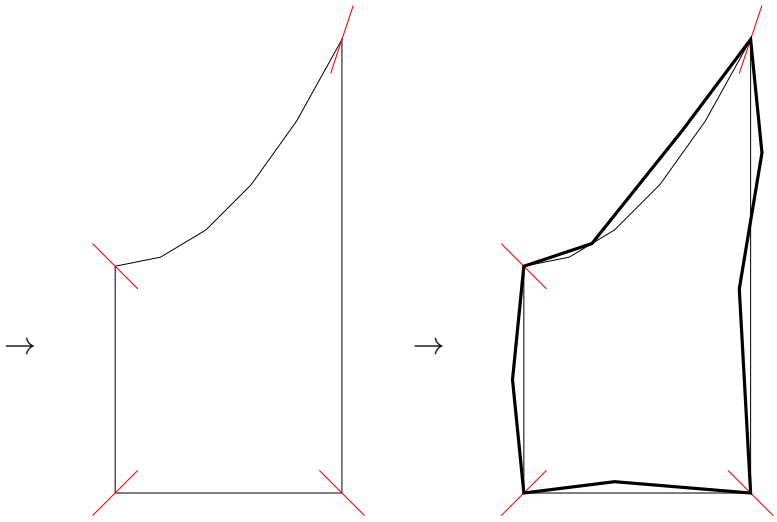
In the second part of the algorithm, shown in Algorithm 2.2 and Figure 2.4, the task is to seal two sequences of facet edges. These sequences are ordered, such that they share the facet point(s) at their start point and, if distinct, their end point. A function $d(.,.)$ returns the distance between the indicated entities. Overall, this algorithm works by considering points p_{curve} and p_{arc} from the curve and arc sequences, respectively, that share a point, $p_{current}$. Three situations exist based on the distances between the current point $p_{current}$, next arc point p_{arc}, and next curve point p_{curve}. Advancement continues along the sealed edge until the arc is sealed to the curve. Arc-curve pairs

```
seal_surface( &surface )

// Initialize data.
triangles  = get_contained_triangles( surface )
curves     = get_bounding_curves( surface )
vertices   = get_endpoints( curves )

// Resolve the 1D bounding loops of the surface.
skin_edges = find_skin_edges( triangles )
skin_loops = assemble_into_loops( skin_edges )

// Merge curve endpoints, or vertices, to closest skin points.
for all vertices
  closest_skin_point = find_closest_point( vertex, skin_loops )
  merge( vertex, closest_skin_point )

// Cut skin loops into skin arcs, using vertices as separators.
skin_arcs = cut_loops( vertices, skin_loops )

// Pair each skin arc with a curve.
arc_pairs< skin_arc, curve >
for all skin_arcs
  curve = min_d_oriented( skin_arc, curves )
  arc_pairs.push_back( skin_arc, curve )

// Seal or replace each pair.
for all arc_pairs
  if( arc_pair.curve has not been replaced by skin_arc )
    curve = skin_arc
  else
    seal_pair( &arc_pair.curve, &arc_pair.skin_arc, &triangles )

return success
```

Algorithm 2.1: Surface sealing algorithm.



(a) Input faceted surface. (b) Skin the surface. (c) Match skin points to vertices.



(d) Separate skin arcs (e) Associate arcs with curves.

Figure 2.3: Procedure to associate skin arcs with faceted curves.

are sealed using the `seal_pair` algorithm until no pairs remain; then the next surface is sealed using the `seal_surface` algorithm.

point-point contraction at front of edge If the distance from the current point to the next point, arc or curve, is less than facet tolerance, perform point-point contraction with the next point to the current point. During the merge operation the current point is chosen as the entity to keep. The current point is unchanged. Either the arc point or curve point is advanced, depending on which has been merged away. Triangles adjacent to both merged points are removed because they become degenerate.

point-point contraction at back of edge If the distance between the curve point and arc point is less than facet tolerance, merge the arc point to the curve point. The current point is advanced to the curve point. Both the curve point and arc point are advanced.

vertex-edge contraction If the first two situations do not occur, perform a point-edge contraction. Depending on which entity is next, either insert the arc point into the next curve edge or insert the curve point into the next arc edge. Advance the current point to the next point. Either the arc point or curve point is advanced, depending on which was inserted into the opposite edge. Triangles adjacent to the edge are split when a new point is inserted into the edge. This ensures that surfaces which have already been sealed remain conformal to the curve as subsequent surfaces are sealed.

Figure 2.4 shows the sequence of operations used to seal a faceted surface to a curve. First the arc point is merged to the current point (Figure 2.4b). Next the curve point is inserted into the opposite arc edge using point-edge contraction (Figure 2.4c). Finally, the arc point is merged to the current point using point-point contraction (Figure 2.4d).

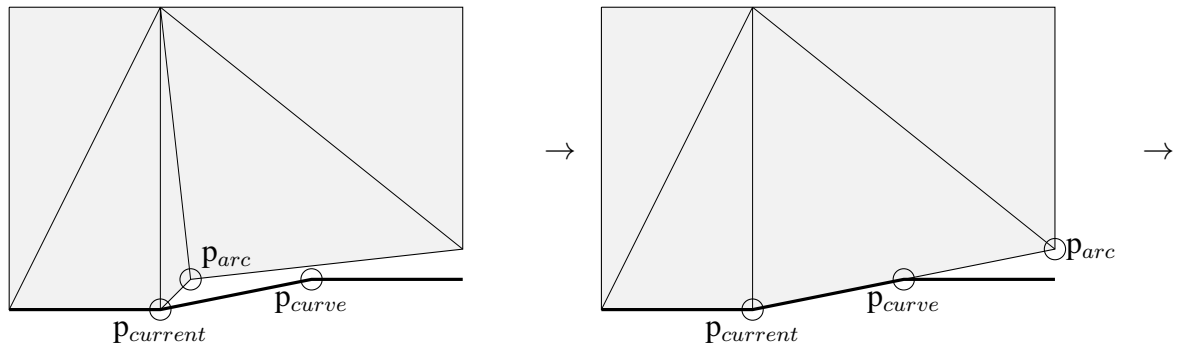
2.3.4 Proof Approach

The proof of reliability of this algorithm is developed in four steps:

Can seal vertices to points on surface skin: By definition, the vertex facet point p is located on the geometric model, and the facet edges in the boundary of the faceted surface are within ϵ_f of

```
seal_pair( &curve, &arc, &triangles )

// Initialize data.
p_current = curve.front()
p_curve   = curve.front()
p_arc     = arc.front()

until arc is sealed
  // The next point depends on proximity to the current point.
  if( d( p_curve, p_current ) < d( p_arc, p_current ) )
    arc_is_next = true
    p_next = p_arc
  else
    arc_is_next = false
    p_next = p_curve

  // Perform point-point contraction at front of edge.
  if( d( p_current, p_next ) <= facet_tol )
    merge p_next to p_current
    p_current is unchanged
    if( arc_is_next ) p_arc++
    else              p_curve++

  // Perform point-point contraction at back of edge.
  else if( d( p_curve, p_arc ) <= facet_tol )
    merge p_arc to p_curve
    p_current = p_curve
    p_arc++
    p_curve++

  // Perform point-edge contraction.
  else
    insert p_next into opposite_edge
    p_current = p_next
    if( arc_is_next ) p_arc++
    else              p_curve++

  update_adjacent_triangles( &triangles )

return success
```

Algorithm 2.2: Algorithm to seal boundary of faceted surface to faceted curve.



(a) Unsealed curve.

(b) Point-point contraction to $p_{current}$.

(c) Point-edge contraction.

(d) Point-point contraction to p_{curve}.

Figure 2.4: Example of sealing boundary of faceted surface to faceted curve.

the geometric curves on the boundary of the geometric surface. Because the faceted surface is a cell complex and so is the geometric model itself, each vertex point p will therefore be within ϵ_f of the boundary of the faceted surface as well. p will be closest to only one point of the boundary of the faceted surface because LFS $\gg \epsilon_f$. p can therefore always be sealed to the boundary of the faceted surface, possibly after splitting one of the edges to insert a point closest to p.

Can separate boundary of faceted surface into arcs: The boundary of the faceted surface is a cell complex, with points corresponding to geometric vertices of the corresponding geometric surface sealed to the faceted surface. Because the geometric model is also a cell complex, these points will separate the boundary of the faceted surface into arcs of facet edges with each arc bounded by one or two vertex facet points p.

Arcs corresponds to faceted curves that bound the surface: The endpoints of each arc are vertex facet points. The faceted surface and geometric model are cell complexes. Each arc will correspond to one faceted curves, and faceted curves are distinct because of the preprocessing done before execution of the sealing algorithm.

Can seal arcs to faceted curves: The facet points on the arc and corresponding faceted curve are within ϵ_g of the geometric model; the facet edges on the arc and curve are within ϵ_f of the geometric model; and the arc and corresponding faceted curve are ordered consistently with each other. Therefore, similar to sealing the vertices to the boundary of the faceted surface, the arc and faceted curve can be sealed together. This likely requires splitting some of the edges of the arc or corresponding faceted curve.

When all surface facetings have been sealed to the facetings of their bounding curves, the entire model has been sealed.

2.4 Implementation

This algorithm is implemented in C++ using the Mesh-Oriented datABase (MOAB) [25]. FBMs are usually created from CAD models. MOAB calls the Common Geometry Module (CGM) [24] to facet the CAD model. CGM provides a uniform interface for several different solid modeling engines including ACIS, CATIA, and Open CASCADE. CGM calls the solid modeling engine to perform faceting, thereby avoiding the complexity of implementing faceting algorithms within CGM. The resulting FBM is stored in a MOAB instance.

Of the volume notions discussed in Section 1.3.2, at the time of sealing only the CGM geometry handles and MOAB geometry set handles exist. The sealing algorithm uses only MOAB geometry handles. MOAB represents the geometric model entities as entity sets in the mesh, with the entity sets containing the facets defining the entity and topological relations between entities represented using parent/child relations between the sets. Volume sets are empty, since there is no three-dimensional mesh. The sequences of facet edges used in the algorithm described earlier are represented only as ordered lists of facet points. Facet edges are not represented explicitly, since they are being modified frequently as sealing proceeds. Spatial searches are accelerated using MOAB's k-d tree decomposition.

The sealed FBM is represented in MOAB using the same geometric entity sets as used for the sealing algorithm's input. This representation can be output to various mesh formats, or can be left in MOAB for subsequent operations (e.g. DAGMC or facet coarsening and refinement to generate finite element meshes).

2.5 Testing and Analysis

The test models displayed in Figure 2.5 and listed in Table 2.1 were chosen from previous radiation transport simulations. Models vary in complexity, with the most intricate having more than 29 million triangular facets. Before being sealed, five test models were already watertight by proximity. Although not topologically the same, skin points of adjacent faceted surfaces were within the solid modeling engine's absolute tolerance of each other. These models were created

Table 2.1: Geometric entity count and number of triangular facets [millions] as a function of ε_f.

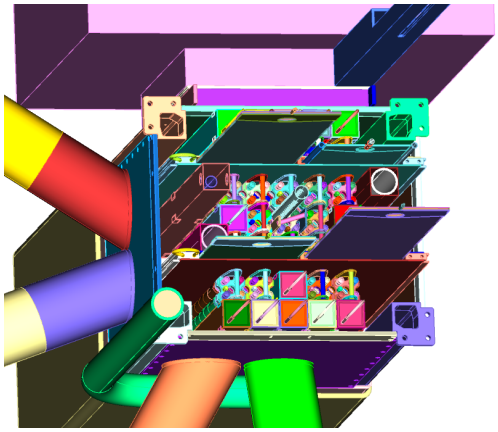
Model	Geometric Entity			Facet Tolerance [μm]				
	Volumes	Surfaces	Curves	1000	100	10	1	0.1
UW Nuclear Reactor	2820	30237	65078	2.62	2.62	2.98	8.56	29.1
Advanced Test Reactor	2132	11827	22402	0.44	0.45	0.84	2.44	7.65
40° ITER Benchmark	902	9834	20485	0.32	0.78	2.07	8.76	16.3
ITER Test Blanket Module	71	4870	13625	0.07	0.08	0.12	0.38	1.57
ITER Module 4	155	4155	10255	0.29	0.29	0.34	1.07	2.89
ITER Module 13	146	2407	5553	0.28	0.29	0.50	2.54	8.65
FNG Fusion Benchmark	1162	4291	5134	0.11	0.11	0.14	0.46	1.14
ARIES First Wall	3	358	743	0.17	0.87	1.21	1.55	2.45
High Average Power Laser	15	139	272	0.15	0.47	0.53	0.61	0.88
Z-Pinch Fusion Reactor	24	95	143	0.05	0.29	0.99	1.17	1.53

using the ACIS solid modeling engine. Five test models did not originate in ACIS and were not watertight by proximity, due to file format translation, non-manifold surfaces, and imprecise modeling.

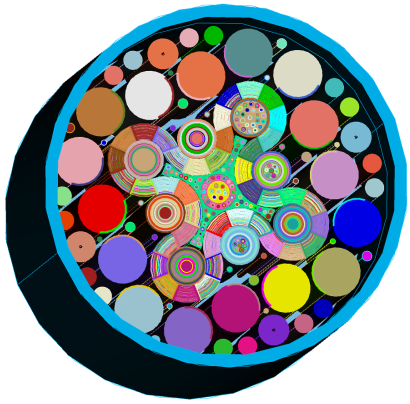
Figures 2.6 and 2.7 show faceted surfaces before and after sealing. Figure 2.6 examines three surfaces at the base of the ITER test blanket module. The sealed surfaces have more facets due to point-edge contraction. Figure 2.7 focuses on three surfaces surrounding a port in the 40° ITER benchmark model. Similarly, more facets are found in the sealed surfaces.

2.5.1 Sealing Success

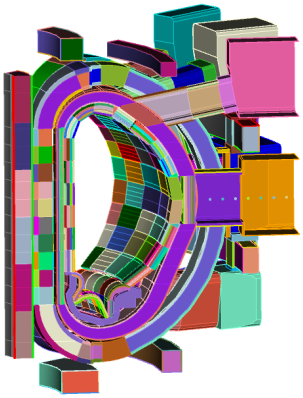
Table 2.2 shows the number of surfaces that fail to conform to their bounding curves after sealing, for various models and facet tolerances. Note that for facet tolerances of 100 μm to 0.1 μm all but the ITER benchmark model were watertight. This includes the default facet tolerance of 10 μm. The ITER benchmark failed to completely seal because it contained many surfaces with features smaller than the facet tolerance, created by imprinting. Failures occur if the facet tolerance becomes larger than the feature size, as in the 100 μm group.



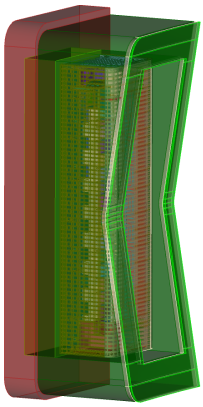
(a) UW Nuclear Reactor



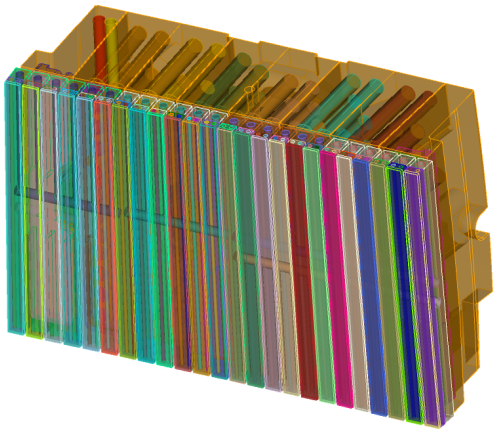
(b) Advanced Test Reactor



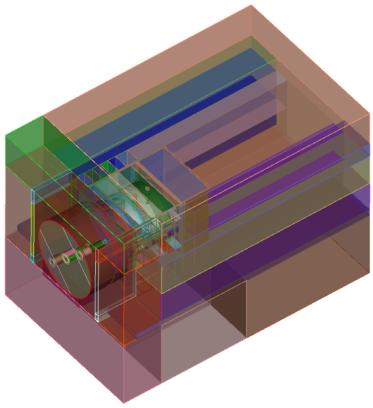
(c) 40° ITER Benchmark



(d) ITER Test Blanket Module



(e) ITER Module 13



(f) FNG Benchmark

Figure 2.5: Detailed CAD models used to test the sealing algorithm.

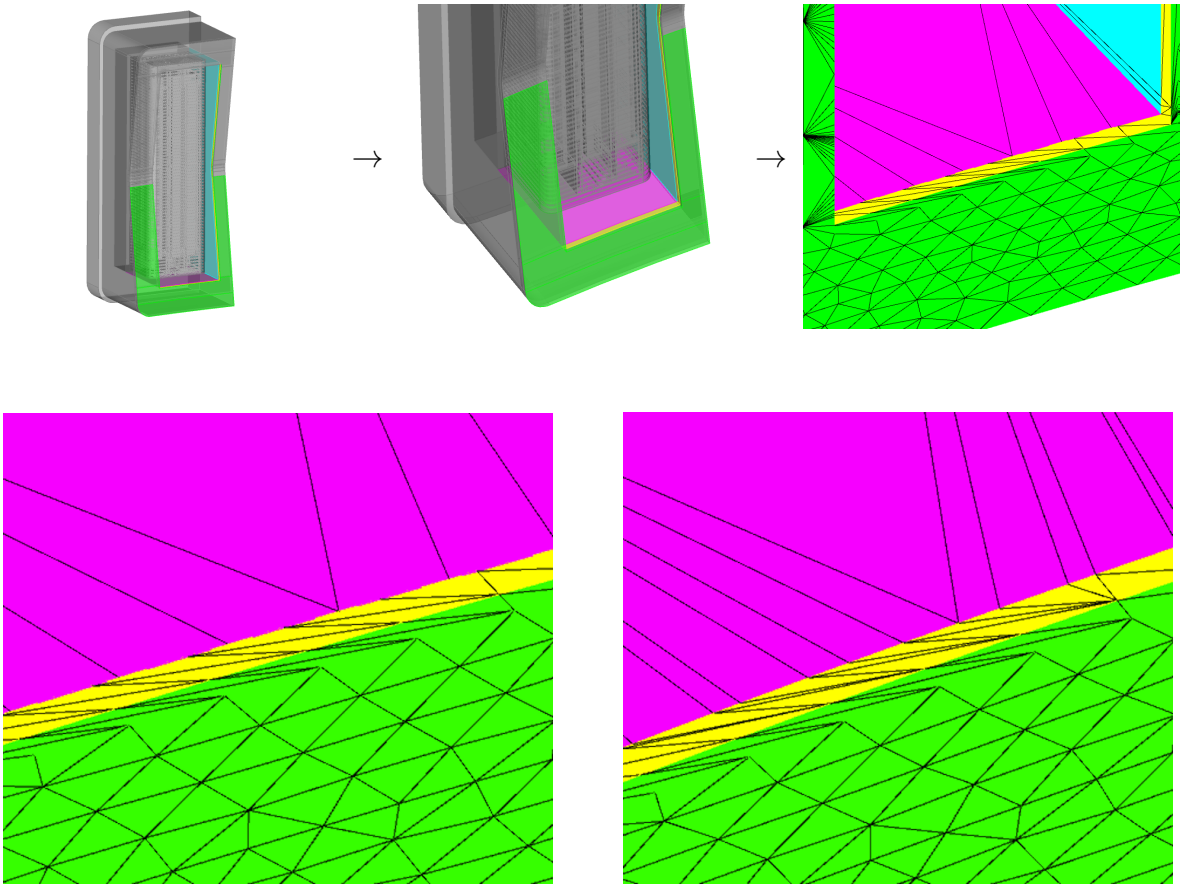


Figure 2.6: ITER test blanket module before (left) and after (right) sealing.

Table 2.2: Number of surface sealing failures as a function of ε_f.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	1019	0	0	0	0
Advanced Test Reactor	88	0	0	0	0
40° ITER Benchmark	18	9	0	18	191
ITER Test Blanket Module	0	0	0	0	0
ITER Module 4	0	0	0	0	0
ITER Module 13	2	0	0	0	0
FNG Fusion Benchmark	63	0	0	0	0
ARIES First Wall	1	0	0	0	0
High Average Power Laser	0	0	0	0	0
Z-Pinch Fusion Reactor	3	0	0	0	0

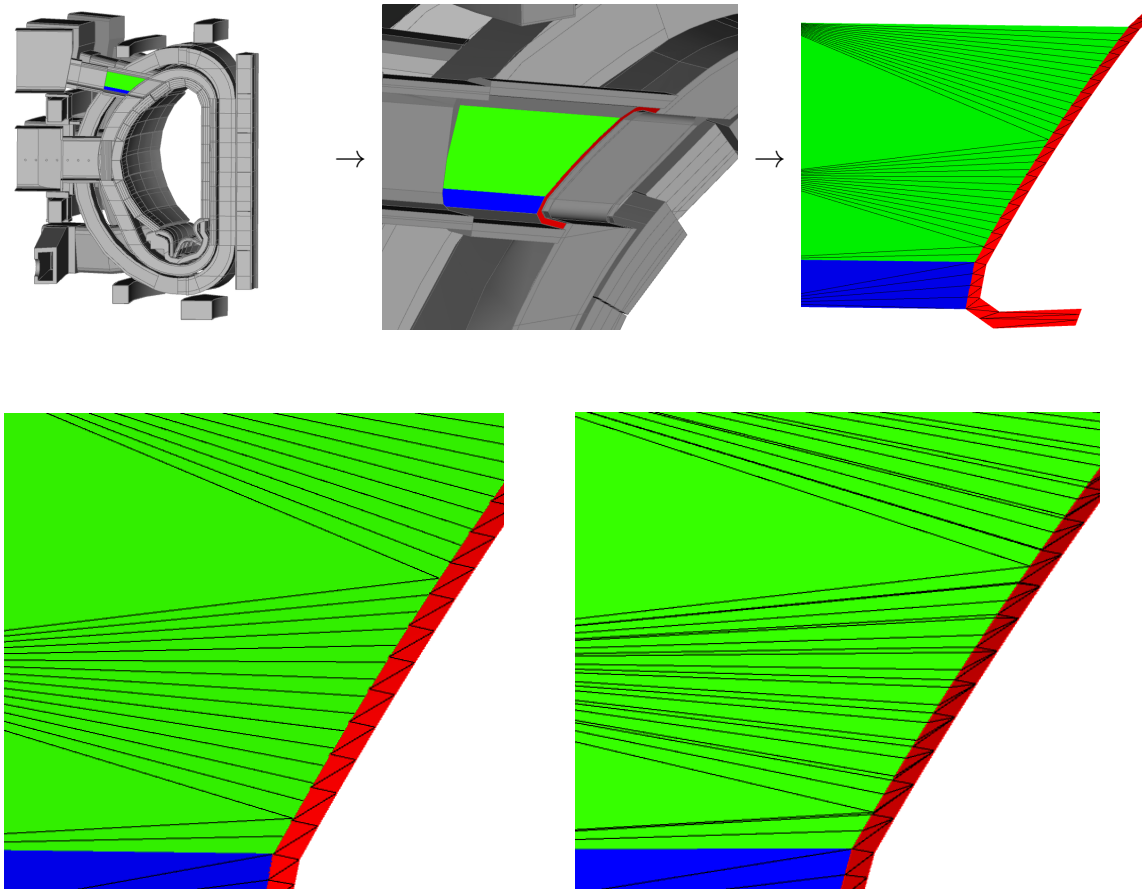


Figure 2.7: 40° ITER benchmark model before (left) and after (right) sealing.

Table 2.3: The change ratio $[sealed/unsealed]$ in the number of facets due to sealing as a function of ε_f.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	0.71	0.99	1.00	1.00	1.01
Advanced Test Reactor	0.64	1.00	1.00	1.00	1.00
40° ITER Benchmark	1.00	1.01	1.02	1.02	1.03
ITER Test Blanket Module	0.90	1.00	1.01	1.01	1.01
ITER Module 4	0.65	0.98	1.00	1.01	1.01
ITER Module 13	0.78	1.00	1.00	1.00	1.00
FNG Fusion Benchmark	0.60	1.00	1.00	1.00	1.00
ARIES First Wall	1.00	1.00	1.00	1.00	1.00
High Average Power Laser	1.00	1.00	1.00	1.00	1.00
Z-Pinch Fusion Reactor	0.87	1.00	1.00	1.00	1.00

2.5.2 Triangle Count

The speed of computations performed on the FBM is affected by the number of triangles in the model. The number of triangles increases due to point-edge contraction (since triangle facets connected to the edge are split with the edge), but decreases due to point-point contraction of adjacent skin points of the same surface. For the default facet tolerance of 10 μm, the change in the number of triangles ranged from 0% to 2%. Across the entire test suite of ten models and five facet tolerances, the change in the number of triangles ranged from -40% to 3%. A decrease in the number of triangles was common for the 1000 μm facet tolerance, since at that tolerance many facet points on the same surface become merged.

During sealing, each surface imposes an additional constraint on adjacent curves. If the faceted curve itself were to be sealed, it too would impose an additional constraint on the sealing process. By replacing the faceted curve with a corresponding skin arc as suggested in Algorithm 2.1, the number of constraints on the edge is reduced by one. This decreases the number of additional triangles due to vertex-edge contraction.

Table 2.4: Number of surfaces containing inverted facets after sealing as a function of ε_f.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	272	0	1	0	13
Advanced Test Reactor	30	0	0	0	0
40° ITER Benchmark	7	7	4	0	10
ITER Test Blanket Module	0	0	0	0	0
ITER Module 4	0	0	0	0	0
ITER Module 13	2	0	0	0	0
FNG Fusion Benchmark	16	0	0	0	0
ARIES First Wall	0	0	0	0	0
High Average Power Laser	0	0	0	0	0
Z-Pinch Fusion Reactor	2	1	0	0	0

2.5.3 Inverted Facets

When sealing, facets become inverted if a point is moved across an edge of the same facet. This occurs with long, narrow facets. Each inverted facet is removed along with adjacent facets to create a polygon. The polygon is refaceted using the ear clipping algorithm as implemented in [59]. It has a time complexity of $O(n^2)$, with n being the number of points bounding the polygon. It is possible that the new facets are still inverted. If so, the polygon is iteratively expanded and refaceted until the new facets are no longer inverted. The polygon can only be expanded to the skin of the surface, so that the new facets are constrained to the skin. One can think of the surface skin as being owned by curves instead of the surface. Inverted facets that cannot be fixed by this method are a rare occurrence if the input model conforms to the input assumptions, as suggested by Table 2.4. Although the ear clipping algorithm is $O(n^2)$, n is typically small.

2.5.4 Timing

The implementation was executed on an Intel Xeon 3 GHz processor with 64 bit Linux. The sealing algorithm is intended to be used as a preprocessing step for applications that use FBMs, such as DAGMC. Timing results are shown in Table 2.5. Results do not include file reading and writing, because as part of an existing application, the use of this algorithm will not incur additional

Table 2.5: The time [seconds] to seal each model as a function of ε_f, on one core of an Intel Xeon 3.00 GHz CPU.

Model	Facet Tolerance [μm]				
	1000	100	10	1	0.1
UW Nuclear Reactor	136	65	64	156	587
Advanced Test Reactor	93	16	27	76	235
40° ITER Benchmark	6	12	38	71	236
ITER Test Blanket Module	15	9	9	14	30
ITER Module 4	10	8	8	23	67
ITER Module 13	6	5	6	19	67
FNG Fusion Benchmark	7	4	4	9	29
ARIES First Wall	1	3	5	13	36
High Average Power Laser	1	1	2	5	25
Z-Pinch Fusion Reactor	1	1	2	4	12

runtime due to file reading and writing. In general, sealing required less than one minute for most models and facet tolerances.

2.5.5 Lost Particles

This algorithm was developed, in part, to seal faceted CAD models for Monte Carlo radiation transport. One of the causes of lost particles is leakage between surfaces. Figure 2.8 shows the lost particle fraction for each model before and after sealing. The default facet tolerance of 10 μm was used for all models. The 40° ITER benchmark, test blanket module, module 4, and module 13 models lost significantly fewer particles after sealing. Sealing did not significantly affect the UW nuclear reactor, advanced test reactor, FNG fusion benchmark, ARIES first wall, high average power laser, and z-pinch reactor models. This reflects the input models, of which half were already watertight by point proximity.

Sealing did not eliminate lost particles. The first three lost particles of each sealed model were investigated. In each case the particles became lost because of a specific defect in the particle tracking algorithm unrelated to watertightness known as the *discard distance tolerance*. This and other weaknesses of the tracking algorithm are addressed in Chapter 3.

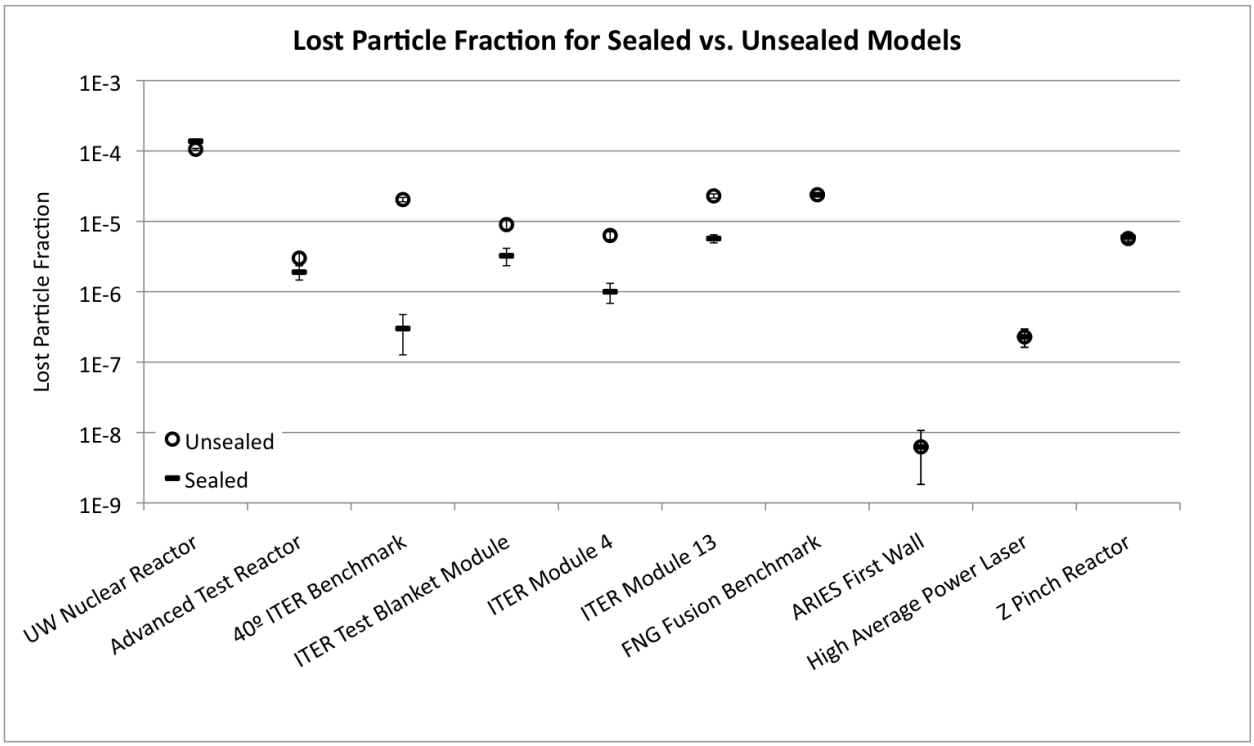


Figure 2.8: Lost particle fraction before and after sealing each model. Error bars indicate one standard deviation.

One consequence of sealing is the creation of triangles with high aspect ratios due to edge-point contraction. The faceting of a cylindrical surface produces long, narrow triangles. For example, the advanced test reactor model contains triangles 100 cm high by $\sim$0.001 cm wide. This may create numerical problems for some applications, including DAGMC's original PIT. As discussed in Section 3.2.3, one motivation for replacing the original PIT was lack of robustness in these situations.

2.6 Conclusion

A tool was developed to make existing faceted models watertight without human intervention [60]. Surfaces were sealed to their bounding curves using point-point and point-edge contraction. Because sealing progresses one surface at a time, this algorithm naturally supports non-manifold geometry. Ten CAD models were tested over a four decade range of facet tolerances. Models were successfully sealed if the input assumptions of the algorithm were met. It is assumed that the facet tolerance is less than the feature size and greater than the merge tolerance. On average, sealed models contained more triangles than unsealed models as a result of point-edge contraction. Sealing can create inverted facets, with most occurring when input assumptions are not met. Timing results show that the algorithm is fast enough to be used in preprocessing for applications that use FBMs. DAGMC was used to test models before and after sealing for lost particles. One cause of lost particles, leakage between unsealed surfaces, was eliminated through application of the sealing algorithm. The causes of remaining lost particles were determined to be logical and numerical errors in the tracking algorithm; this is the topic of Chapter 3.

Chapter 3

Robust Tracking

3.1 Motivation

As shown in Figure 2.8, lost particles persist despite watertight faceting. The first three lost particles of each model listed in Figure 2.8 were investigated. In all cases the lost particles were due to a defect in DAGMC's original tracking algorithm, known as the *discard distance tolerance*. The discard distance tolerance is the distance below which intersections are rejected in the particle tracking algorithm. Due to numerical precision the current surface may appear to be at a small positive distance along the particle trajectory instead of zero distance. Intersections at distances less than the discard distance tolerance are discarded. Use of a discard distance tolerance is a common strategy in ray tracing to avoid hitting the same surface that the particle is currently on. A valid intersection may occur at a distance less than the discard distance tolerance if the particle is located near a corner, for example. In this situation a valid intersection is discarded and the particle will become lost. Most lost particles are due to the discard distance tolerance.

Special cases exist that are known to be challenging for particle tracking algorithms, due to numerical and logical errors. Numerical errors are caused by finite precision arithmetic. Logical errors are caused by faulty algorithms. The algorithm presented in this chapter is robust against these situations. Although the special cases shown in Figure 3.1 are described in terms of *surfaces*, DAGMC represents surfaces as sets of triangular *facets*. For DAGMC, these special cases can be described in terms of facets instead of surfaces.

Two notions of particle location exist: logical position and numerical position. The logical position is the volume that the particle is inside, and occasionally the surface that the particle is

on. The numerical position is described by the particle's coordinates. The logical and numerical position may become inconsistent due to numerical error when the particle advances. Consistency can be checked by performing a point inclusion test (PIT). If consistent, the numerical position is inside the volume described by the logical position.

Figure 3.1 shows several cases that are challenging for the particle tracking algorithm. In Figures 3.1a- 3.1d the square represents the boundary of a two-dimensional volume. The particle's numerical position is indicated along with its trajectory. In Figures 3.1e-3.1f the particle trajectory is toward an edge adjacent to two triangular surfaces. Each failure mode is described as *numerical* or *logical*.

Behind or on a surface, leaving a volume (numerical) When entering a volume, the particle's logical position is on the surface where it has entered the volume. The particle's numerical position may be on the boundary, inside the current volume, or outside the current volume due to numerical error. A `next_surface` computation is performed to determine where the particle will intersect the boundary and leave the volume. As shown in Figure 3.1a if the numerical location is outside, the ray will likely intersect the surface that the particle is logically on. To prevent the particle from becoming lost, the current surface should not be returned as the exit intersection.

Ahead of a surface, leaving a volume (numerical) A `next_surface` computation is performed to determine how far a particle must travel before exiting the volume. The exit computation occurs when a particle enters a new volume or when a collision occurs, changing the particle's direction. Consider the case of a particle undergoing a collision close to a surface. The particle changes direction, and the distance to exit must again be determined. The particle's logical position is inside the current volume. Due to numerical error the particle's numerical position may be outside the volume, as shown in Figure 3.1b. To prevent the particle from becoming lost, an exit intersection must be found so that the particle can leave the volume.

Tangential trajectory (numerical) If a particle travels nearly tangent to a surface, most of the particle's trajectory will be in the direction parallel to the surface. For a particle at position

(x_1, y_1) traveling in direction (v_x, v_y), the next position at distance d is (x_2, y_2).

$$x_2 = x_1 + v_x * d \quad [3.1]$$

$$y_2 = y_1 + v_y * d \quad [3.2]$$

If $y_1 = y_1 + v_y * d$ due to numerical precision, the particle will not cross the surface. Instead it will *skip* along it in very small increments of $v_x * d$ as shown in Figure 3.1c. Without protection against this scenario, the particle will become stuck in a pseudo-infinite loop until it reaches the end of the surface.

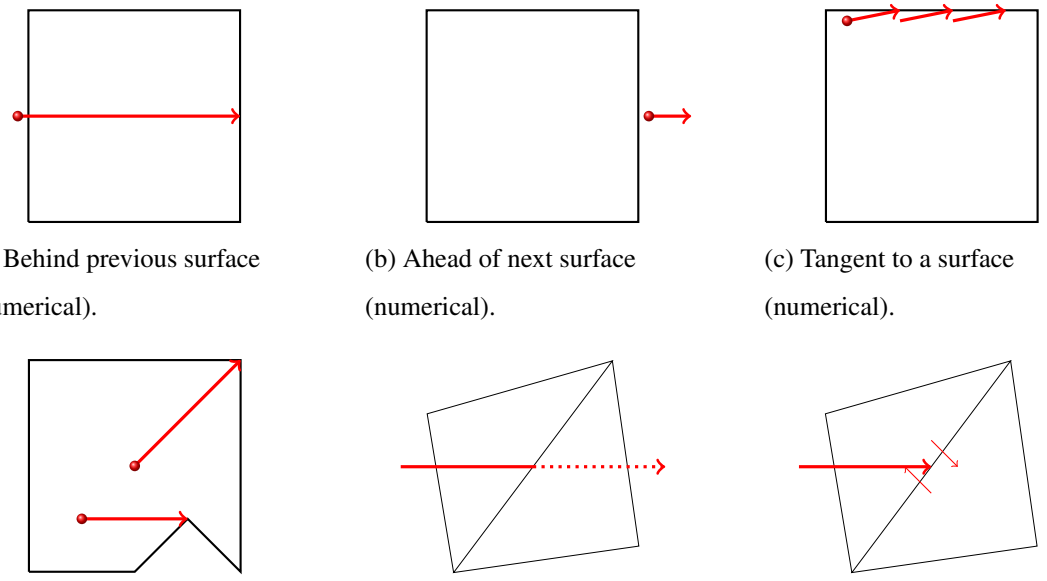
Intersect an edge or point (logical) A particle may intersect a point or edge between surfaces such that it does not have a corresponding entry or exit intersection. If not handled properly, a particle could become lost if a glancing intersection is interpreted as one boundary crossing instead or zero or two. Discussed in Section 3.3.3, both glancing and piercing intersections are shown in Figure 3.1d.

Leak between surfaces (numerical) Although adjacent surfaces share an edge, a particle may not intersect either surface due to numerical precision. This occurs because the intersection test is not numerically identical along the edge adjacent to both surfaces. The particle leaks between adjacent surfaces, unable to find an exit intersection as shown in Figure 3.1e.

Oscillation between surfaces (logical) A particle may intersect an edge or point between two or more surfaces. Subsequent intersections may find an adjacent surface, at zero distance. Without advancing any distance the particle will oscillate in an infinite loop between adjacent surfaces, as shown in Figure 3.1f.

3.1.1 Statement of Problem

This chapter presents an algorithm that *tracks particles through a faceted CAD model while ensuring that particles do not become lost or enter an infinite loop*. The original DAGMC algorithm has defects that prevent particles from crossing surfaces closer than the discard distance tolerance.



(a) Behind previous surface (numerical).

(b) Ahead of next surface (numerical).

(c) Tangent to a surface (numerical).

(d) Intersects edge or point (logical).

(e) Leak between surfaces (numerical).

(f) Oscillate between surfaces (logical).

Figure 3.1: Particle tracking failure modes when determining the exit intersection from the current volume.

Other special cases exist that are challenging for Monte Carlo particle tracking. The proposed algorithm solves these problems while reducing computational effort.

This chapter will discuss previous work and the development of the robust tracking algorithm including intersection of edges and points, enumeration of intersection cases, and zero-distance advance. It concludes with implementation details and analysis of test results.

3.2 Theory

After briefly presenting terminology, previous work will be reviewed in particle tracking, ray-triangle intersection, and point inclusion tests. At the end of this section DAGMC's original tracking algorithm will be listed.

A *ray-triangle intersection (RTI)* is an intersection of the particle trajectory with a triangle contained in the boundary of the volume. An RTI includes the distance to intersection, surface containing the triangle, and triangle itself. An *exit intersection* is a single RTI passed from the DAGMC library to the physics code. Several RTIs are pruned to determine a single exit intersection. The particle leaves the current volume by passing through the boundary at the exit intersection. The surface or triangle *normal vector* defines the outward direction at each point on the boundary of a volume.

3.2.1 Particle Tracking

MCNP [3] tracks particles through constructive solid geometry (CSG) robustly by using the surface normal vector and previous surface. By comparing the ray direction with the surface normal, RTIs can be categorized as *forward* or *reverse*. Let α be the cosine of the angle between the ray direction and the surface normal. Forward RTIs have $0 < \alpha < 1$ while reverse RTIs have $-1 < \alpha < 0$. A ray cannot intersect a planar surface more than once. The triangular facets that DAGMC uses to represent surfaces are planar, suggesting that a ray should not intersect the same facet more than once.

Tolerances are offered as a tool to increase robustness [61, 62]. For example, a small amount of track length can be temporarily added to each particle advance. Extra track length will move the

particle beyond the next surface, avoiding the case where it falls short due to numerical precision. Track length must later be removed to avoid biasing the physics simulation. An alternative approach is to add a tolerance to surfaces, giving two-dimensional surfaces a thickness. Particles are considered to be on a surface if their numerical position is within tolerance of it. Instead of adding track length or thickening boundaries, a different approach is to search a small distance behind the particle for intersections [63]. In this scheme the closest intersection with correct orientation is chosen as the exit intersection, including those slightly behind the particle's numerical location.

3.2.2 Ray-Triangle Intersection

Although simulations that use continuous surface representations determine intersections on a per-surface basis, DAGMC is optimized to determine ray intersections on a per-triangle basis. Each triangle is contained in exactly one surface, allowing DAGMC to resolve intersections on a per-surface basis if desired by the physics code. A robust ray-triangle intersection test is required for a robust tracking algorithm. Algorithm speed is important because the ray-triangle intersection test is frequently used. The inputs to the test are triangle point coordinates, ray origin, and direction. If an intersection occurs, the ray-triangle test returns the distance to intersection. The triangle normal vector can be stored to reduce computational cost but increase memory requirements. At present, DAGMC simulations are memory-limited due to large mesh tallies. Further increase of the memory requirement is to be avoided.

The ability to detect edge and point intersections is a desirable trait of the ray-triangle intersection test to be used in DAGMC. However, unless due to source specification the intersection of an edge or point is improbable. DAGMC's original tracking algorithm did not handle edge and point intersections. This is a logical deficiency in which the particle has a glancing intersection with an edge between two triangles. Although precluded by the discard distance tolerance, the particle could become lost because it has an entry point into a volume without a corresponding exit point. A similar situation occurs when a particle intersects a point adjacent to three or more triangles.

Another weakness of DAGMC's original ray-triangle intersection test is leakage between adjacent triangles. Particles can leak between adjacent triangles if the same numerical computation is

not performed with the ray and shared edge on both triangles. Due to numerical precision the ray may appear to be outside both triangles when it actually intersects at least one of them. A robust test would be edge-stable—that is, the same numerical test should be used to test the ray against the edge of both adjacent triangles.

Several ray-triangle intersection tests will be surveyed in search of a replacement for the original DAGMC test. These include the projection test, Möller test, Plücker test, and signed volume test. This topic will end with a discussion of tolerances and increased numerical precision.

Projection Test One method of determining ray-triangle intersection is to project the ray and triangle to two dimensions [59]. The direction of the largest component of the triangle's normal vector is selected as the coordinate to dismiss. Selecting the largest component reduces numerical error by keeping the two-dimensional projection of the triangle as large as possible. The ray is intersected with the plane that the projected triangle lies inside to determine the possible point of intersection. Next, the signed area is calculated for each of the three sides using the ray-plane intersection point. If each of the signed areas are positive, the ray intersects the triangle. If one signed area is zero, the ray intersects an edge. If two signed areas are zero, the ray intersects a point.

Möller Test The original version of DAGMC uses the Möller test [64]. It translates the triangle to the origin then transforms it such that the ray and plane of the triangle are perpendicular. The coordinates of intersection are computed as a weighted sum of the triangle's points x_0, x_1, and x_2, with weights u, v, and w. The sum of the weights, or *barycentric coordinates*, is unity if the point is located on the triangle. It is typical to explicitly use u and v to describe the intersection location with w being implicit.

Plücker Test In the Plücker test, each edge of the triangle is tested against the ray to determine its relative orientation [65]. The canonical ordering of edges is used to give each edge a corresponding representation as a ray. Plücker coordinates are calculated from a point P and a direction L.

$$\pi_r = \{L : L \times P\} = \{U_r : V_r\} \qquad [3.3]$$

Plücker coordinates are compared to each other using the permuted inner product (PIP). As described by [65], the PIP for ray r and s is:

$$\pi_r \odot \pi_s = U_r \cdot V_s + U_s \cdot V_r \quad [3.4]$$

The PIP is the sum of two scalar triple products. The sign of the PIP created by a ray with each triangle edge is used for all tests.

$$\text{If } \pi_r \odot \pi_s > 0 \quad \text{then } s \text{ goes counterclockwise around } r \quad [3.5]$$

$$\text{If } \pi_r \odot \pi_s < 0 \quad \text{then } s \text{ goes clockwise around } r \quad [3.6]$$

$$\text{If } \pi_r \odot \pi_s = 0 \quad \text{then } s \text{ intersects or is parallel to } r \quad [3.7]$$

The ray is tested against each edge of the triangle to determine if intersection occurs. For edges e and ray r:

$$\text{If } \pi_r \odot \pi_{ei} \geq 0 \; \forall \; i \quad \text{and } \exists \; j : \pi_r \odot \pi_{ej} \neq 0 \text{ then } r \text{ intersects (enters) triangle} \quad [3.8]$$

$$\text{If } \pi_r \odot \pi_{ei} \leq 0 \; \forall \; i \quad \text{and } \exists \; j : \pi_r \odot \pi_{ej} \neq 0 \text{ then } r \text{ intersects (leaves) triangle} \quad [3.9]$$

$$\text{If } \pi_r \odot \pi_{ei} = 0 \; \forall \; i \quad \text{then } r \text{ is coplanar with triangle} \quad [3.10]$$

If an intersection occurs, the barycentric coordinates can be found by scaling the PIPs. Determining the distance to intersection is trivial once the coordinates of intersection are known. For each edge test, only the edge and ray is required. This allows adjacent triangles to use the same edge test for shared edges in such a way as to ensure consistency. Particles cannot leak between adjacent triangles because the edge test is the same for adjacent triangles. Edge and point intersections are trivial to detect due to the computation of barycentric coordinates. Triangles coplanar with the ray are similarly identified. Platis and Theoharis show that the Plücker test can be 25% faster than the Möller test if the result of each edge comparison is saved for use in adjacent triangles [65]. In their comparison, both tests determine intersection coordinates and distances.

Signed Volume Test Segura and Feito propose a test that represents the ray as a line segment [66]. The line segment is tested against each edge of the triangle in a signed distance computation.

A tetrahedron is created with the ray segment endpoints and triangle edge endpoints. The signed volume is a scalar triple product, evaluated as the determinant of a 3×3 matrix. The signed volume of a tetrahedron with points A, B, C, and D is:

$$[DABC] = \frac{1}{6}\begin{vmatrix} x_a - x_d & y_a - y_d & z_a - z_d \\ x_b - x_d & y_b - y_d & z_b - z_d \\ x_c - x_d & y_c - y_d & z_c - z_d \end{vmatrix} \quad [3.11]$$

The ray will intersect the triangle if the signed volume calculation for each edge is nonnegative. Signed volumes of zero imply that the ray intersects the edge. Two signed volumes of zero imply that the ray intersects a point. The signed volume test is shown to be 9% faster than the Möller test assuming the distance to intersection is determined. The time comparison does not include computing the endpoints of the ray segment. Also absent from test results is the proportion of intersections that occur. Ray-triangle intersection tests perform differently depending on the proportion of tests that result in an intersection. One advantage of the Plücker and signed volume tests is that the same calculation can be performed for both triangles adjacent to an edge in the faceting. This removes the possibility of missing both triangles due to numerical error.

Add tolerance to triangle size If ray-triangle intersection tests do not perform the same numerical calculation when testing the ray against the shared edge of adjacent triangles, it is possible for the ray to leak between both triangles due to numerical error. Leakage could be prevented by adding a tolerance to expand the size of each triangle. By enlarging the triangle, false positive intersections will be found instead of false negatives. If false positives are found, a scheme is needed to determine which intersections are valid. Identifying false positives is unlikely to be more successful than the original problem of identifying edge and point intersections. It is computationally expensive to add an absolute tolerance to the size of each triangle. Although less computationally expensive, adding a relative tolerance is less reliable due to the large variation in triangle size compared with numerical precision.

A different approach is to test if the ray passes within a tolerance, ϵ, of points and edges [67]. If a ray passes within ϵ of an edge or point, an edge/point intersection is identified. Due to the

added thickness of ϵ, edges are cylindrical and points are spherical in shape. The computational cost of three ray-cylinder intersections and three ray-sphere intersections is significant compared to the cost of the ray-triangle intersection.

Increase arithmetic precision Increased precision can be used to reduce numerical error that may allow rays to leak between adjacent triangles. Compared with double precision, quadruple precision is about 10x slower. Quadruple precision reduces but does not eliminate numerical error. Instead of quadruple precision, arbitrary precision arithmetic could be used. The GNU Multiple Precision Library is mature, as indicated by its use in Maple and Sage [68]. It can use increased precision beyond quadruple but is 10-1000x slower than double precision. Again, arbitrary precision would reduce numerical error but not eliminate it.

3.2.3 Point Inclusion Test

In DAGMC, point inclusion tests (PITs) are performed on volumes to determine inside which volume source particles start. Although PITs are computed on the FBM, they does not require the full geometric topology because the result is simply *inside* or *outside*, without reference to lower-dimensional topology. Instead, polyhedra are a sufficient geometric domain for PITs. From [69], "a *polyhedron* is a closed, piecewise planar surface that bounds a solid in three-dimensional space. A polyhedral region is the solid bounded by a polyhedron. The faces of a polyhedron are its planar pieces, assumed to be finite in number. The loops of a face are the closed, non-self-intersecting polygonal curves that bound it." The face normals of a polyhedron are oriented outward. The loops of a face are oriented counterclockwise with respect to the normal.

One source of lost particles in the original version of DAGMC was inconsistency between the PIT and the tracking algorithm. For example, the PIT may determine that a particle was inside a volume but the tracking algorithm was unable to find an exit intersection from the volume. This occurs due to numerical error, tolerances, and faulty logic of the PIT. Several different PITs have been devised and will be surveyed as potential replacements for the original test used in DAGMC.

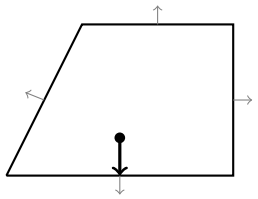


Figure 3.2: The direction of the normal at the closest location on the boundary is the same as the vector from the test point to the closest location, therefore the test point is inside.

Enlarged Orientation Method In two dimensions, a planar polygon is represented by a set of ordered edges. The closest location on the polygon boundary to the test point is determined, including the entire length of each boundary edge. If the test point is closest to the interior of an edge, the orientation of the edge can be used to determine if the test point is inside or outside the polygon, as shown in Figure 3.2. If the test point is closest to the endpoint between two polygon edges, then both edges must be examined. The test point is inside the polygon if the endpoint is at a concavity, or outside if the endpoint is at a convexity with respect to the boundary of the polygon. If the closest location is an edge endpoint that is neither a concavity or convexity with respect to the polygon, the orientation of either adjacent edge can be used to classify the test point [70].

This method is named after the concept of enlarging a circle around the test point. The circle, centered at the test point, is grown until it contacts the boundary of the polygon. When contact occurs, the entire circle is either inside or outside of the polygon. Comparison of the vector from the test point to the contact point and the boundary normal at the contact point indicates if the test point is inside or outside.

Generalizing to three dimensions, the closest facet is found instead of the closest edge. It it possible that the test point is closest to an edge (two facets) or point (three or more facets). If the test point is closest to an edge or point, then the test point is inside the polyhedron if the edge or point is at a concavity with respect to the polygon. The test point is outside if the edge or point is at a convexity with respect to the polygon. If the faces adjacent to the closest edge/point are coplanar, then either adjacent face can be used to classify the test point [71].

A variant of this test is used in the original version of DAGMC. As the test point becomes farther from the polyhedron, with greater frequency the closest location on the boundary becomes a special case—either a point or edge instead of a triangle interior. As implemented, the numerical computation to determine concavity or convexity uses a tolerance, making this implementation less robust, especially for triangles with large aspect ratios.

The concavity or convexity of adjacent faces is determined using facet normal vectors. An elegant approach to this calculation using a synthetic normal composed of adjacent face normals is presented in [72]. The closest point is located on a face, edge (two faces), or point (multiple faces). N_i is the normal of face i and θ_i is the angle between edges of the face adjacent to the point. The synthetic normal at closest point p is given as:

$$\text{1. Face:} \quad n_p = N \tag{3.12}$$

$$\text{2. Edge:} \quad n_p = \frac{N_1 + N_2}{|N_1 + N_2|} \tag{3.13}$$

$$\text{3. Vertex:} \quad n_p = \frac{\sum_{i=1}^{m} \theta_i N_i}{|\sum_{i=1}^{m} \theta_i N_i|} \tag{3.14}$$

In work by Khamayseh and Kuprat [72], a similiar method was demonstrated that does not require the closest location in all directions on the boundary of the polyhedron. Instead of the closest point in all directions (i.e. *global*), the closest point in any direction (i.e. *local*) is sufficient. This is advantageous because it avoids a global search and includes the trivial case where the ray is outside and does not intersect the boundary. The global test is computationally expensive for special cases such as finding the closest point on a sphere when the test point is near the center of the sphere. The local approach is more efficient, but requires watertight polyhedra. If the model is not watertight, the closest facet may be found with the wrong orientation due to leakage between facets. The global approach is robust against faceted models with small gaps. Another consequence of the local approach is that the synthetic normal is no longer sufficient for cases in which the closest location on the boundary is an edge or point of the faceting. This is because the surface at the locally closest point is not guaranteed to be moving away from the test point. In the local approach, if the closest location is a faceting edge or point a more complicated test must be used as described by Khamayseh and Kuprat [72]. A variant of the local approach is

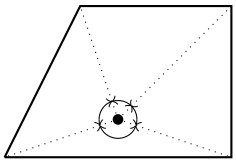


Figure 3.3: The sum of angles projected onto a circle is 2π, therefore the point is inside.

introduced in Section 3.3.5 as part of the robust tracking algorithm. Instead of using the absolute first intersection, it uses the first *piercing* intersection.

Winding Number Method In two dimensions, a planar polygon is represented by a set of ordered edges. An angle is made at the test point with the endpoints of each edge. The angle is summed for every edge in the polygon and may be negative depending on the orientation of the edge. Summing the angles around the test point is analogous to projecting the edges of the polygon onto a unit circle centered at the test point. The summation of the angles will be 0 if the point is outside, π if the point is on the boundary, and 2π if the point is inside, as shown in Figure 3.3 [69]. Note that because the result must be either 0, π, or 2π up to $\pi/2$ error in the summed angle will still yield the correct result. The polygon does not need to be connected (it could have separate regions) or simply connected (it could have holes).

This method generalizes to three dimensions for polyhedral volumes. Instead of projecting each edge of a polygon onto a unit circle, each facet of a polyhedron is projected onto a unit sphere. Facets are oriented such that their normal points outward. The area of the summed projection is 0 if the point is outside, 2π if the point is on the boundary, and 4π if the point is inside the polyhedron [69]. This method does not require preprocessing of the polyhedron, as every facet will need to be projected. The time complexity is $O(n)$ where n is the number of faces in the polyhedron. The winding number method is slow for volumes with many facets.

Ray Intersection Parity Method In the ray intersection parity method, a ray originating at the test point is fired in any direction. The number of boundary intersections is counted. If the ray intersects the boundary an even number of times, the test point is outside the volume, as shown in

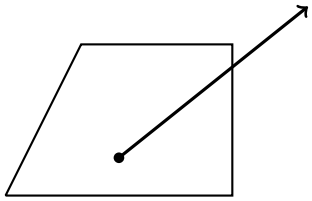


Figure 3.4: The boundary is intersected one time, therefore the point is inside.

Figure 3.4. If the ray intersects the boundary an odd number of times, the test point is inside the volume. The ray intersection parity method becomes more costly as the complexity of the volume boundary increases. Care must be taken if the test point is on a boundary. Depending on the ray tracing algorithm, the on-boundary case may not consistently be identified. If the ray passes through a facet point or edge, depending on the ray-triangle intersection test, it could intersect none or multiple facets. A common strategy to avoid edge/point intersection is to fire another ray. This approach depends on the ability to identify that an edge/point intersection has occurred. To properly consider edge/point intersections, the neighborhood in the vicinity of the intersection must be examined. The ray intersection parity method fails for facet-based models that are not watertight [70].

3.2.4 Original Tracking Algorithm

There are several weaknesses of the original tracking algorithm, listed in Algorithm 3.1. Lost particles occur because the correct RTI was not identified, or the correct RTI was erroneously discarded. The original algorithm does not check orientation to ensure that the particle will exit the volume upon intersection. All RTIs within an *add distance tolerance* were saved for future disambiguation. The add distance tolerance defaults to 10 μm and the discard distance tolerance defaults to 0.01 μm. To avoid intersecting the previous triangle again, the original algorithm discards all RTIs less than the discard distance tolerance if the last event was a surface crossing. Instead of identifying the previous intersection as occurring at zero distance, the discard distance tolerance is used—it is unlikely that the previous triangle is found at exactly zero distance due to numerical precision. All particles within the discard distance tolerance of the correct exit intersection are

lost. In addition to discarding intersections closer than the discard distance tolerance, the original algorithm failed for the *ahead of a surface, leaving a volume (numerical)* case, as depicted in Figure 3.1b. Other special cases are superseded by the use of the discard distance tolerance.

In addition to the discard distance tolerance, RTIs are discarded inside the `ray_intersect_facets` function. An oriented bounding box tree [32] is used to accelerate ray tracing. A distance limit is kept at which all RTIs must be less than. In a process known as *pruning*, triangles are discarded in an entire branch of the tree due to the distance limit. Inside `ray_intersect_facets` pruning occurs in the `ray_box_intersect` and ray-triangle intersection (Section 3.2.2) functions. Initially the distance limit is the distance to the next collision, known as the physics limit. Only the closest RTI is kept beyond the add distance tolerance. In effect, this reduces the distance limit to the distance of the closest RTI at greater distance than the add distance tolerance.

3.3 Algorithm

This section will present the tracking algorithm, discussing each component in detail. After listing assumptions, the algorithm is pieced together by combining the ray-triangle intersection test, edge/point post processing, point inclusion test, and zero-distance advance. The proposed algorithm is analyzed using enumerated intersection cases.

The geometric model created by the initialization routine is in the form of a boundary representation. The solid model consists of geometric volumes, surfaces, curves, and vertices. Geometric entities are represented by meshed entities, of which DAGMC utilizes only surfaces. Volumes do not contain three-dimensional mesh but are represented by bounding surfaces. Surfaces are represented as sets of mesh faces. Mesh faces are triangles specified by three points. Individual mesh faces and their geometric surface share the same orientation. The boundary of each volume is a manifold. However, surfaces may be shared by two volumes, creating a non-manifold model.

```
next_surface( prev_surf, prev_dist, ray_dir, ray_point,
              volume, physics_limit,
              &next_dist, &next_surf )

// Set distance limit of intersection search.
nonneg_dist_limit = physics_limit

// Return all RTIs less than ADD_DIST_TOL and up to 1 RTI greater than
// ADD_DIST_TOL if less than physics_limit.
// Each intersection has a surface, distance, and facet.
call ray_intersect_facets( volume, ray_point, ray_dir,
                           ADD_DIST_TOL, nonneg_dist_limit,
                           &surfs, &dists, &facets )

// Avoid RTIs on the previous surface.
for all dists
  if( dist < DISCARD_DIST_TOL && 0 != prev_surf )
    dists.erase(  dist  )
    surfs.erase(  surf  )
    facets.erase( facet )

if( dists.empty() )
  // If using physics_limit, assume a collision occurs before an exit intersection.
  if( NULL != physics_limit )
    next_dist = physics_limit+1
    next_surf = 0
    return SUCCESS

  // Particle is lost if no RTIs exist.
  else
    next_dist = HUGE_VAL
    next_surf = 0
    return FAILURE

// Return the closest RTI.
min_idx = get_minimum_index( dists )
next_dist = dists[idx]
next_surf = surfs[idx]
return SUCCESS
```

Algorithm 3.1: Original tracking algorithm.

3.3.1 Assumptions

The collection of entities in the facet-based model (FBM) forms a cell complex. In the form of a boundary representation, volumes are regularized sets enclosed by one or more[1] pseudo two-manifolds. Pseudo two-manifolds are resolved into surfaces composed of planar facets. Each faceted surface is non-degenerate (all points of $d-$dimensional facets, $d > 0$, are distinct), and is oriented and non-inverted (normal of facet is consistent with that of the underlying model entity in the neighborhood of the facet). Each surface is adjacent to two volumes, including explicit and implicit[2] representations.

3.3.2 Plücker Ray-Triangle Test

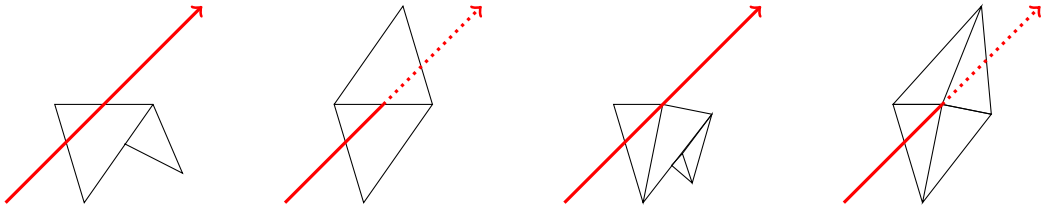
Four ray-triangle intersection tests were discussed in Section 3.2.2 as candidates for use in the tracking algorithm. The projection test and the Möller test do not consistently determine intersections near edges and points among adjacent triangles. The signed volume test can compare edges consistently but truncates the ray to a line segment for each signed volume computation. The Plücker test was selected because intersections near edges and points are treated consistently without adding a tolerance. Although more robust, the Plücker test is more computationally expensive than the Möller test which is used in DAGMC's original tracking algorithm.

3.3.3 Edge/Point Post Processing

Particles become lost when an exit intersection cannot be found. An edge or point intersection may be an exit without a subsequent exit intersection of the next volume. An example of such an intersection in two dimensions is shown in Figure 3.1d. As discussed in Section 3.3.5, intersections will be used for the point inclusion test. The PIT requires that RTIs be categorized as *entering* or *leaving* by comparing the particle trajectory with the surface normal at the intersection point.

In addition, edge and point intersections are categorized as *glancing* or *piercing*. As shown in Figures 3.5b and 3.5d, rays pass through the boundary in piercing intersections. Intersections

[1] The boundary may contain more than one pseudo two-manifold due to inner voids.
[2] Implicit volume representations are discussed in Chapter 4.



(a) Glancing edge. (b) Piercing edge. (c) Glancing point. (d) Piercing point.

Figure 3.5: Edge and point intersections are categorized as glancing or piercing.

through the interior of triangles are always piercing intersections. Glancing intersections, as shown in Figures 3.5a and 3.5c intersect the boundary at a single point without passing through it. Triangles in the neighborhood of a piercing intersection have normals that consistently identify the ray as entering or leaving the polyhedron. Glancing intersections will not be used for particle tracking or point inclusion testing.

To determine if an intersection glancing or piercing, the angle between the particle trajectory and triangle normal is computed for each triangle in the neighborhood of the intersection. Let α be the cosine of the angle between the surface normal and ray direction, as shown in Figure 3.6. In this figure the boundary of the volume is a square in two dimensions, the ray directions are thick, red arrows and the surface normals are short, gray arrows.

If α has the same sign (or is zero) for every triangle adjacent to the intersection, then the intersection is piercing. Otherwise it is glancing.

3.3.4 Ray-Triangle Intersection Enumeration

In Section 3.1 several situations are discussed that challenge particle tracking algorithms. It is useful to further enumerate the ray-triangle intersection outcomes as a function of numerical position and the angle between the ray direction and surface normal.

Table 3.1 presents the possible outcomes of intersecting a ray with triangles of the volume boundary as a function of numerical position and α. The logical position is inside the volume. Possible numerical positions are inside, outside, and on the boundary of the volume. The $\alpha = 0$

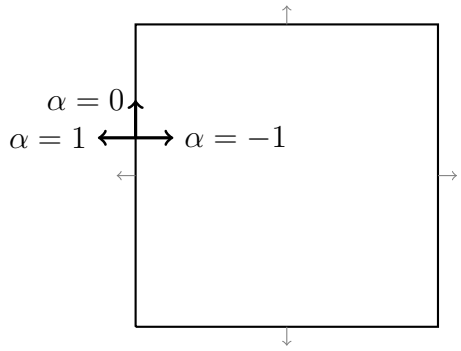
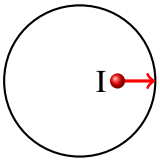
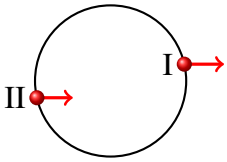
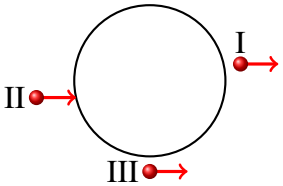


Figure 3.6: α is the cosine of the angle between the particle direction vector and surface normal vector.

case is not possible because coplanar rays are not considered to be intersections. However, a coplanar ray will eventually be detected as an edge or point intersection of an adjacent triangle that is not coplanar. As explained in Section 3.3.3, only piercing intersections are considered. Representative values of α are -1, 1, and null. The null case implies that no intersection occurs. Without loss of generality the volume is assumed to be a sphere, shown as a circle in two dimensions. The base of the ray is the numerical position of the particle. *Of interest is the first intersection with the ray.*

The desired result for each case is listed in Table 3.1. Case *inside-I* is typical of a numerical position inside the volume, perhaps due to collision. The intersection is the exit intersection. Cases *inside-II* and *inside-III* cannot occur because the numerical position is inside. Case *on boundary-I* is rare because it is unlikely that a collision would place the numerical location on the boundary. This a valid exit intersection, at zero distance. Case *on boundary-II* is typical upon a particle entering a volume. This intersection is not an exit intersection because it has the wrong orientation. Case *on boundary-III* cannot occur because an intersection is always found. Also shown in Figure 3.1b, the *outside-I* case is rare because it is unlikely that numerical error after a collision would place the numerical location outside of the volume. In DAGMC's original algorithm the exit intersection would not be detected because it is behind the numerical position in the direction of the particle trajectory. Case *outside-II*, also explained in Figure 3.1a, is likely to occur when a particle logically enters the volume. The numerical position is outside due to numerical error. This intersection is not the exit intersection because is has the wrong orientation.

Table 3.1: Desired result for ray-triangle intersections as a function of numerical position and α. The logical position is *inside* the volume.

		Inside 	**On Boundary** 	**Outside** 
I)	$\alpha > 0$	Typical—Return this intersection	Rare—Return this intersection	Rare—Exit is behind numerical position
II)	$\alpha < 0$	Cannot Occur	Typical—Do not return this intersection (previous exit)	Typical—Do not return this intersection (previous exit)
III)	null	Cannot Occur	Cannot Occur	Rare—Exit is beside numerical position

Finally, case *outside-III* is rare but may occur due to numerical error. It prevents an exit intersection from being found because there is no intersection along the particle trajectory.

3.3.5 Point Inclusion Test

Ray Intersection Orientation Method A point inclusion test was developed that uses the orientation of the closest piercing intersection along the particle trajectory as shown in Figure 3.7 and Algorithm 3.2. Using α as described in Section 3.3.3, intersections can be categorized as *entering* or *leaving*. If $\alpha > 0$ the intersection is leaving, and if $\alpha < 0$ the intersection is entering. If the closest intersection is leaving, the numerical position of the particle is inside the polyhedron. If the closest intersection is entering, the numerical position is outside.

Edge and point intersections are identified as glancing or piercing before utilized in the PIT. Only piercing intersections are used for point inclusion testing. This ensures the normal of the intersected triangle will provide a sign of α that is representative of all triangles in the neighborhood of the intersection.

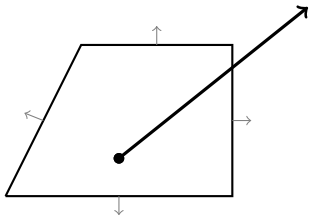


Figure 3.7: The first piercing intersection along the ray is *leaving*, therefore the point is inside.

```
point_inclusion( point, direction, volume, prev_facets, &result )

// If a direction does not exist, use a random direction.
if( NULL == direction ) direction = get_random_direction()

// Set distance limits of the intersection search.
nonneg_dist_limit = HUGE_VAL
neg_dist_limit    = 0

// Return the closest piercing intersection.
// Coplanar triangles ( 0 == alpha ) are not piercing intersections.
// Each intersection has a surface, distance, and facet.
call ray_intersect_facets( volume, point, direction, prev_facets,
                           neg_dist_limit, nonneg_dist_limit,
                           &surf, &dist, &facet )

// No intersection is the trivial case.
if( NULL == facet )
  result = outside
  return SUCCESS

// Get the orientation of the intersection.
normal = get_normal( volume, facet, surf )
alpha  = get_dot_product( normal, direction )

// Determine entering/leaving.
if( 0 > alpha ) result = inside
if( 0 < alpha ) result = outside

return SUCCESS
```

Algorithm 3.2: Point inclusion test.

It is possible that the ray will be coplanar with triangles. Coplanar rays are detected by the Plücker test and not interpreted to be intersections. Although coplanar triangles are not intersections, their existence implies that point or edge intersections of adjacent triangles will occur. The adjacent point or edge intersection will be found and treated as such.

Is it beneficial for both the tracking algorithm and PIT to be based on ray tracing. A point will be inside a volume if and only if its closest intersection is *leaving*. This guarantees that the tracking algorithm can use the same numerical computation to determine the same exit intersection from the volume. This synergy would not be possible if an alternative PIT was used. As a benefit, the PIT will have about the same computational cost as the tracking algorithm. A single implementation of the ray-intersection code will be shared for both purposes.

3.3.6 Zero-Distance Advance

It has been shown that the PIT and tracking algorithm rely on the same numerical computation. This guarantees that the logical position determined by the PIT will be consistent with the numerical exit position determined by the tracking algorithm. Unfortunately, the logical position and numerical position cannot be guaranteed to remain consistent—the subsequent ray intersection calculation is not the same because the numerical position is different. Another cause is of inconsistency results from saving particle positions in phase space for future use. For example, the numerical position, logical position, direction, energy, and weight of particles crossing a surface may be recorded for use as a surface source. Later, the surface source is used to start particles on the same surface but in a different simulation. The limited precision of the file format may cause the numerical position and logical position to become inconsistent.

Independent of the cause, a zero-distance advance can be used to reestablish consistency. Suppose the numerical and logical position of the particle are inconsistent. An intersection is determined to occur at a small negative distance. However, the physics code cannot accept a negative track length—unphysical particle tracking may result in undefined behavior. Instead, DAGMC reports an intersection occurs at zero distance to the physics code. Although the numerical position

does not change, the particle changes logical position due to crossing the surface of intersection. Track length tallies are unchanged and the logical and numerical position become consistent.

3.3.7 Tracking Algorithm

Components described in Sections 3.3.1–3.3.6 will now be assembled to describe the tracking algorithm as shown in Algorithm 3.3. First it is determined if the particle is streaming or reflecting. The previous surface crossing of streaming particles is known and the direction must be the same. A particle has been reflected if the previous surface crossing is a reflecting surface. A list of previously intersected facets is maintained because streaming particles cannot cross the same facet twice. If the previous surface was reflective, only the last intersected facet is preserved. Otherwise, if the particle is not streaming the list of previously intersected facets is cleared.

Distance limits are used to constrain the intersection search. The nonnegative distance limit is the farthest acceptable RTI distance along the forward trajectory of the particle. The negative distance limit is the farthest acceptable RTI distance along the reverse trajectory of the particle. The nonnegative distance limit is initialized to the distance of the next collision. The negative distance limit is initialized to a value larger than numerical precision.

The `ray_intersect_facets` function is called which returns RTIs on the boundary of the volume, represented by a polyhedron. For optimization, pruning occurs within the function by orientation, previous facet exclusion, glancing vs. piercing, and distance limits. RTIs are discarded unless they are piercing exit intersections, absent from the list of previously intersected facets. In addition, RTIs must be closer than the distance limits which are continuously decreased as closer RTIs are discovered. The closest RTI at nonnegative distance is returned, and if closer, one RTI at negative distance.

An RTI at negative distance implies the numerical position of the particle may be inconsistent with the logical position. A PIT is performed using the next volume across the RTI at negative distance. Previous facets are used inside the PIT to logically avoid previous intersections. An *inside* result confirms that the point is inside the next volume, and the numerical and logical position are inconsistent. A zero-distance advance is performed to reestablish consistency. If the point is inside

```
next_surface( prev_surf, prev_dist, ray_dir, ray_point,
              volume, physics_limit,
              &next_dist, &next_surf, &prev_facets)

// Determine if the previous surface was reflective.
if( reflecting )
  last_facet = prev_facets.back
  prev_facets.clear()
  prev_facets.push_back( last_facet )

// Determine if the particle is streaming.
else if( !streaming )
  prev_facets.clear()

// Set distance limits of the intersection search.
nonneg_dist_limit =  HUGE_VAL
neg_dist_limit    = -NUM_PRECISION
if( NULL != physics_limit )               nonneg_dist_limit =  physics_limit
if( nonneg_dist_limit < -neg_dist_limit ) nonneg_dist_limit = -neg_dist_limit

// Return the closest intersection in the positive direction and if closer, one in
// the negative direction less than neg_dist_limit. Only return piercing exit intersections.
// Each intersection has a surface, distance, and facet.
call ray_intersect_facets( volume, ray_point, ray_dir, prev_facets,
                           neg_dist_limit, nonneg_dist_limit,
                           &surfs, &dists, &facets )

// Is the RTI at negative distance inside the next volume?
if( NULL != facets[0] )
    next_vol = get_next_volume( surfs[0], volume )
    call point_inclusion( ray_point, ray_dir, next_vol, prev_facets, &result )
    if(INSIDE == result)
        next_dist = 0
        next_surf = surfs[0]
        prev_facets.push_back( facets[0] )
        return SUCCESS

// Return the RTI at positive distance if it exists.
if( NULL != facets[1] )
    next_dist = dists[1]
    next_surf = surfs[1]
    prev_facets.push_back( facets[1] )
    return SUCCESS

// If using physics_limit, assume a collision occurs before an exit intersection.
if( NULL != physics_limit )
  next_dist = physics_limit+1
  next_surf = 0
  return SUCCESS

// Otherwise the particle is lost.
next_dist = HUGE_VAL
next_surf = 0
return FAILURE
```

Algorithm 3.3: Robust tracking algorithm.

the next volume, the RTI is returned as the exit intersection. To conserve track length, negative distances are reported to the physics code as zero.

If the RTI at negative distance is not returned, the RTI at nonnegative distance is returned. If an RTI at nonnegative distance does not exist, it is assumed that an intersection did not exist within the physics limit. If a physics limit is not used, the particle will become lost. The function returns without an intersection and a collision occurs instead of a surface crossing.

3.3.8 Proof Approach

The goal of the algorithm is to track particles through a faceted CAD model while ensuring that particles do not become lost or enter an infinite loop.

Particles cannot become lost: Particles become lost when an exit intersection cannot be found from the current volume. Volume boundaries of the FBM are assumed to be oriented, watertight, pseudo 2-manifolds that bound a solid region. The Plücker test guarantees that the set of all RTIs with the faceted boundary of the volume can be found. RTIs must occur with either the interior or boundary edge/point of a triangle. That is, rays traveling in the same plane as the triangle are not determined to intersect. Glancing intersections are rejected, guaranteeing that the remaining RTIs pierce the boundary of the volume.

Given the set of all piercing RTIs, the algorithm is guaranteed to find an exit intersection. Ray-triangle intersections were enumerated in Section 3.3.4. Cases *inside-II*, *inside-III*, and *on boundary-III* are not addressed in the tracking algorithm because they cannot occur. If within the nonnegative distance limit, case *inside-I* is returned to the tracking algorithm as the RTI at nonnegative distance. Case *on boundary-I* is handled the same, although being zero distance from the boundary implies that it will always be returned to the tracking algorithm as the RTI at nonnegative distance. Cases *on boundary-II* and *outside-II* are never returned to the tracking algorithm because α is negative. Instead the next intersection will be returned, having correct orientation. The tracking algorithm only returns exit intersections—not entrances. Case *outside-I* is discovered by using the negative distance limit to search backward along the trajectory. The tracking algorithm uses a PIT to ensure the numerical location is outside of the logical volume, then proceeds with

a zero-distance advance. Case *outside-III* is not handled by the tracking algorithm but has never been observed to occur in a simulation. The likelihood that numerical error moves the numerical position perpendicular to the particle trajectory is small.

Infinite loops cannot occur: In the original version of DAGMC, oscillation occurred if the same RTI was found in consecutive calls to `next_surface`. This cannot occur because sequential RTIs must always be exit intersections. Oscillation is impossible because a facet cannot be oriented outward for both adjacent volumes. The original version of DAGMC did not enforce orientation.

Due to the negative distance limit, a new type of oscillation is now possible. Exit intersections at negative distance can cause oscillation due to a zero-distance advance. This is prevented by avoiding reuse of previously-intersected facets along a streaming path. The number of stored facets must be greater than or equal to the number of intersections within the negative distance limit, which represents numerical precision. The facet tolerance is much greater than numerical precision ($\epsilon_f \gg numerical\ precision$). In Chapter 2 it is assumed that the local feature size (LFS) $\gg \epsilon_f$. Together, $LFS \gg \epsilon_f \gg numerical\ precision$. The local feature size at point x is the radius of the smallest closed sphere centered at x that intersects two disjoint features. For a corner in the faceting (a *feature*), storing two previous facets is sufficient. It is prudent to store more facets to avoid oscillation if the LFS assumption is violated.

3.4 Implementation

The tracking algorithm and PIT test are implemented in MOAB's DAGMC library. Oriented bounding boxes are used to organizes triangles in a bounding volume hierarchy. The `ray_intersect_facets` function performs the tree traversal, subject to the constrains imposed by the calling function. For tracking, these constraints include orientation, distance, previous facets, and glancing vs. piercing. When the tree traversal encounters a leaf node the Plücker test is used to test each triangle of the leaf against the ray. The Plücker test and edge/point post processing is implemented below the `ray_intersect_facets` function, as indicated in Figure 3.8. When comparing the ray with each triangle edge, the Plücker test orders edge endpoints by MOAB handle.

This ensures the same numerical operation is performed for the shared edge between two adjacent triangles.

The only tolerance in the tracking algorithm and PIT is the negative distance limit. It is the distance to search behind the particle for RTIs, and compensates for numerical imprecision associated with the particle advance. Because performance is not significantly affected by this value, it defaults to a generous 10 μm. For robustness, all previous facets along a streaming path are stored. In practice this does not significantly decrease the tracking rate because iterating through the array of previously intersected facets is cheaper than searching the OBB tree, containing at least several thousand facets. Few facets are intersected along a streaming path compared to the number of facets in each volume of the FBM.

3.5 Testing and Analysis

The purpose of the robust tracking algorithm is to increase accuracy by eliminating lost particles. The DAG-MCNP implementation was analyzed using: 1) a test suite developed for DAGMC, 2) the 40° ITER benchmark model, and 3) ten CAD models featured in Chapter 2. The detail and expense of each test varies. The entire output file is compared in the DAGMC test suite. Tracking rate, tallies, collisions, random numbers, and lost particles are compared using the 40° ITER benchmark model. The CAD models used in Chapter 2 are only tested for lost particles. The test suite and benchmark model were tested in a few minutes; the testing of ten intricate CAD models required a month. Unless specified otherwise, tests did not utilize the physics limit so that particles could become lost.

3.5.1 Test Suite

The DAGMC test suite includes tests for a critical sphere, beam at the center of a plane, beam along a surface between two cubes, beam along a curve adjacent to four cubes, beam grazing a cylinder, fissile spheres in a vacuum, thin spherical shells, beams onto a disk, beams along a sphere, beam grazing a surface, beam traveling down a narrow vent, array of plutonium cylinders, three uranium cylinders, and wedges converging to an axis. The differences between the original

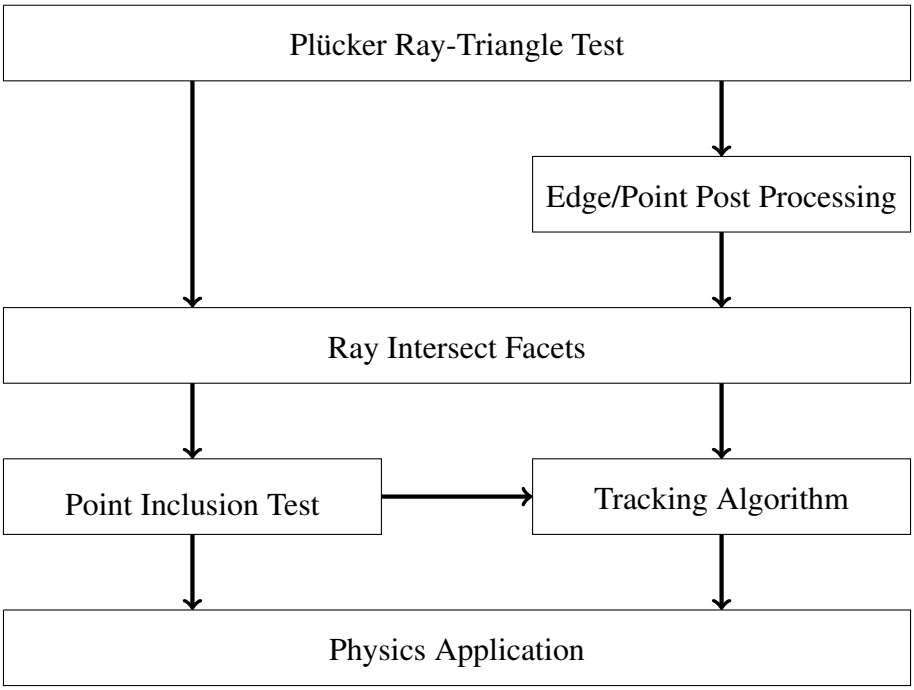


Figure 3.8: Flow of information from the ray-triangle test to the physics application.

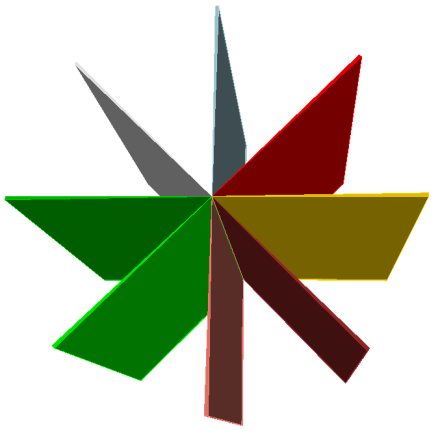


Figure 3.9: Wedges converge at an axis.

and robust tracking algorithm were not significant, except for the lack of lost particles. The original algorithm lost three particles for the *beam traveling down a narrow vent* and 29 particles for *wedges converging at an axis*, shown in Figure 3.9. These simulations used 5 million particles. The robust algorithm did not lose any particles.

3.5.2 ITER Benchmark Model

The robust tracking algorithm was compared with the original algorithm to investigate tracking rate and accuracy, as shown in Table 3.2. The 40° ITER benchmark model was selected due to its complexity and prominence in many analyses. In all cases the model was sealed as described in Chapter 2. The model was tested with and without materials to investigate streaming. The physics limit accelerates ray tracing by limiting the surface intersection search to less than the distance to the next collision. The tracking rate is the number of source particles simulated per minute. The 40° ITER benchmark model contains 13 tallies. The random number and collision counts are measures of the similarity between cases. Although different particle trajectories could result in the same random number and collision counts, it is unlikely. In this comparison they are used as a checksum.

The results in Table 3.2 were computed on one core of an Intel Core2 2.66 GHz CPU using 100k particle histories. All coincident surfaces were merged, implying the model does not have

Table 3.2: Particle tracking of the original and robust versions of DAG-MCNP is compared using the 40° ITER benchmark model.

Case	Executable	Physics Limit	Tracking Rate [part./min.]	Tallies	Collisions [#]	Random Numbers [#]	Lost Particles [#]
			with materials				
1	original	no	9766	*baseline*	12701310	185958957	0
2	original	yes	10299	identical	12701310	185958957	0
3	robust	no	9976	identical	12701310	185958957	0
4	robust	yes	11339	identical	12701310	185958957	0
			without materials				
5	original	no	74674	*baseline*	0	754270	2
6	robust	no	81225	identical	0	754270	0

overlaps. Cases 1-4 used materials, while cases 5-6 compared performance in voided models. The tallies, random number count, collision count, and lost particle count were identical in the first four cases. The last two cases had the same collision and random number count. However, this agreement is not meaningful because voided models will not have any collisions, and random numbers are used only to spawn source particles. The tallies in the last two cases were identical, although the original executable lost two particles. The robust executable is 2-10% faster than the original, mostly due to its implementation. Distance limits are enforced more rigorously during the ray-intersection search. This could be back-ported to the original implementation, likely showing a similiar performance increase.

For the first time, DAGMC can use the physics limit to accelerate tracking without masking lost particles. Use of the physics limit with the original algorithm prevents the discovery of lost particles. If the tracking algorithm fails to find an exit intersection within the physics limit, it may mistakenly assume that a collision occurs before a surface crossing. As intended, the physics limit increases the tracking rate without altering results. Comparing cases 1 and 4, the physics limit allows the robust algorithm to be 16% faster than the original algorithm without the physics limit. As suggested in Section 3.5.3, the robust algorithm does not lose particles. Therefore, the robust algorithm with the physics limit cannot mask lost particles. Use of the physics limit is appropriate only when using watertight geometry with the robust tracking algorithm. Cases with the original algorithm using the physics limit were included for tracking rate comparison only.

3.5.3 Lost Particles

In Section 2.5.5 lost particle fractions were compared before and after facet-based models were sealed using the original tracking algorithm. Although the number of lost particles decreased, they did not disappear. The same sealed models were again tested to determine the fraction of lost particles using the robust algorithm, as listed in Table 3.3. Each model was assessed using one core of an Intel Xeon 3.00 GHz CPU for 30 days, resulting in no lost particles. For comparison, the number of lost particles using the original algorithm with sealed models from Section 2.5.5 is extrapolated for the same number of particles simulated.

Table 3.3: The number of lost particles for watertight models using the original and robust tracking algorithms with $\epsilon_f = 10\mu m$. Error range indicates one standard deviation.

Model	Particles Simulated	Particles Lost	
	[millions]	Original Algorithm	Robust Algorithm
UW Nuclear Reactor	41	5649 ± 178	0
Advanced Test Reactor	74	141 ± 32	0
40° ITER Benchmark	225	67 ± 39	0
ITER Test Blanket Module	205	665 ± 184	0
ITER Module 4	59	59 ± 19	0
ITER Module 13	79	450 ± 60	0
FNG Fusion Benchmark	1310	31273 ± 989	0
ARIES First Wall	4070	25 ± 18	0
High Average Power Laser	286	65 ± 19	0
Z-Pinch Fusion Reactor	409	2454 ± 317	0

Although no particles became lost using the robust algorithm with sealed models, lost particles are still possible for unsealed models. The unsealed 40° ITER benchmark model was tested using the robust tracking algorithm, resulting in a lost particle fraction of 1.699 x 10^{-5} ($\pm$ 1.369 x 10^{-6}).

3.6 Conclusion

A robust algorithm was developed to track particles through facet-based models without becoming lost. The algorithm relies on logical position when possible to reduce the error associated with numerical computation. When numerical computations are required, they are performed consistently, such as the ray intersections that are used for both point inclusion and tracking. The algorithm was tested on DAGMC's test suite and sealed CAD models introduced in Chapter 2. A combined 300 computer-days resulted in no lost particles out of more than 6 billion particles simulated. Testing of the 40° ITER benchmark model demonstrates that the increase in robustness was accomplished without a decrease in the particle tracking rate.

Chapter 4

Implicit Complement

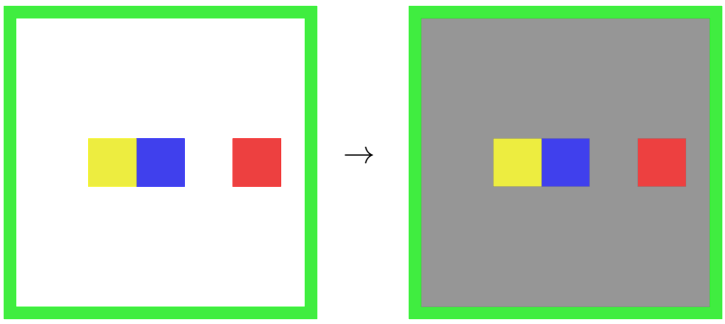
4.1 Motivation

All space must be defined for Monte Carlo radiation transport. CAD models typically define solid objects, but not surrounding space. The nonsolid space, or *complement*, represents all negative volume not modeled as solid objects. Examples include air in a room or coolant in a fission reactor. The complement can be manually created in a CAD program by subtracting solid objects from an encompassing volume. The result is an intricate volume, typically containing more surfaces than any other volume. Unfortunately, Boolean operations such as subtraction often fail when manipulating intricate models.

Once the complement is created, coincident surfaces of adjacent volumes must be imprinted and merged. Merged surfaces are topologically adjacent to two volumes, while unmerged surfaces are adjacent to one volume. Although most surfaces ($\sim$95%) merge automatically, some do not. Problematic surfaces appear to match, yet small defects prevent merging because entities do not have the same lower-dimensional topology. These errors result from file conversion or imprecise draftsmanship. Manually merging the remaining surfaces of the complement requires significant human effort.

4.1.1 Statement of Problem

Although explicit creation of the complement is time-consuming, many steps are repetitive. All unmerged surfaces of the model eventually become surfaces of the complement. In this manner it is possible to *build an implicit representation of nonsolid space*, consisting of all unmerged



(a) Without complement. (b) With complement.

Figure 4.1: The implicit complement is used to define nonsolid space between explicit volumes.

surfaces. Through use of an implicit complement, both Boolean operations associated with explicit complement creation and merging of complement surfaces can be avoided.

4.2 Algorithm

4.2.1 Assumptions

The collection of geometric model entities forms a cell complex. In the form of a boundary representation, volumes are regularized sets enclosed by one or more pseudo two-manifolds. Each surface is adjacent to one or two explicit volumes. Merged surfaces are adjacent to two explicit volumes, while unmerged surfaces are adjacent to a single explicit volume.

4.2.2 Implicit Complement Creation

As listed in Algorithm 4.1, each surface is queried to determine the number of adjacent volumes. If a surface is adjacent to exactly one volume, it is added to the boundary of the implicit complement. Graphically this is shown in Figure 4.1. The resulting volume will be a valid solid because its boundary will be composed of two-manifolds that bound explicit volumes. Regions of the implicit complement may not be connected, and will likely have interior voids.

```
build_implicit_complement( surfaces )

// Create volume notions for the implicit complement.
impl_compl_vol   = create_geometry_handle()
impl_compl_obb   = create_OBB_root()
impl_compl_index = greatest_existing_index + 1
impl_compl_id    = greatest_existing_id    + 1

// Collect surfaces that are not merged.
for all surfaces
  parent_volumes = get_parent_volumes( surface )
  if( 1 == parent_volumes.size() )
    add_parent_child( &impl_compl_vol, &surface )
    surface_obb = get_obb_handle( surface )
    impl_compl_surf_obbs.push_back( surface_obb )

// Build the OBB tree for the complement volume.
join_trees( &impl_compl_obb, impl_compl_surf_obbs )

return SUCCESS
```

Algorithm 4.1: Creation of the implicit complement volume.

4.3 Implementation

Unmerged surfaces and their OBB trees are assigned to the implicit complement MOAB geometry handle and MOAB OBB handle, respectively. Although the algorithm is simple, the volume notions in DAGMC must be adjusted to include the implicit complement.

CGM Geometry Handle The implicit complement does not exist in the input CAD model. Unlike the other four volume notions, a CGM geometry handle is not created. Lack of an explicitly defined solid for the implicit complement is not important for DAGMC.

MOAB Geometry Handle Nominally, a MOAB geometry set handle is assigned for each CGM geometry handle after the FBM is created. The implicit complement does not have a CGM geometry handle. Therefore, a MOAB geometry handle for the implicit complement is added as an extra step after the last pre-existing MOAB geometry handle. The MOAB geometry handle is assigned parent-child relations for all unmerged surface geometry handles as described in Algorithm 4.1.

MOAB OBB Handle The implicit complement OBB handle is assigned all unmerged surface OBB handles as described in Algorithm 4.1. The MOAB geometry handle is tagged with the handle of the implicit complement OBB tree.

MCNP Index An index of one greater than the maximum index is assigned to the implicit complement.

MCNP ID An ID of one greater than the maximum ID is set on the implicit complement geometry handle.

Group names defined in CUBIT are parsed to assign tallies, material IDs, densities, and boundary conditions. In certain situations, the implicit complement may need a material ID and density. For example, fuel rods in a fission reactor are surrounded by coolant. Usually volumes are added to a group named `mat_<mat_id>_rho_<rho>` for which `mat_id` is the material ID and `rho` is the

material density. For the implicit complement this is not possible because there is no explicit volume to add to a group. Instead, a group is named `mat_<mat_id>_rho_<rho>_comp`. The implicit complement will be assigned the material ID and density specified in this group name.

In some situations, it is desirable to apply reflective boundary conditions to surfaces that are part of the implicit complement. These surfaces must be part of an explicit volume in order to be read during initialization. To create a reflective boundary of the implicit complement, an extra volume is constructed containing the reflecting surface. The other sides of the extra volume must be outside the geometry, where no particles will travel.

The volume of each volume is calculated for tally normalization. This is not possible because the implicit complement has an extra shell that encompasses all outside space. Although the volume of the implicit complement is actually infinite, it is assigned an arbitrary volume of 1 cm^3 therefore tallies must be normalized manually.

Use of the implicit complement alters the analysis procedure from Section 1.3.1. Before implementation of the implicit complement, unmerged surfaces could exist only on the outer boundary of the entire model. All other surfaces were merged with a surface from the adjacent explicit volume. Most useful for complex models, CUBIT could be employed to visualize unmerged surfaces. The analyst could then identify coincident unmerged surface pairs that must be merged. However, the implicit complement permits the existence of unmerged surfaces within the interior of the model. Instead of simply displaying unmerged surfaces, CUBIT must now check for overlapping surfaces and volumes.

4.4 Testing and Analysis

The implicit complement was tested by deleting the explicit complement from existing CAD models, forcing construction of an implicit complement. It was expected that the output files of each would be identical. Sealed models were tested with 100,000 particles.

4.4.1 Cubes

A simple model was needed in which each step of transport can be checked during code development. For this exercise, a CAD model of three identical cubes was created, as shown in Figure 4.2a. A 14.1 MeV point source was located in the center of the middle cube. A graveyard[1] surrounds the cubes as a large, hollow cube with inner length of 100 cm. The complement surrounds the cubes and extends to the graveyard. The cubes are filled with water at standard density, and scored with track length tallies. Both simulations produced the same response, as listed in Table 4.1. No particles were lost.

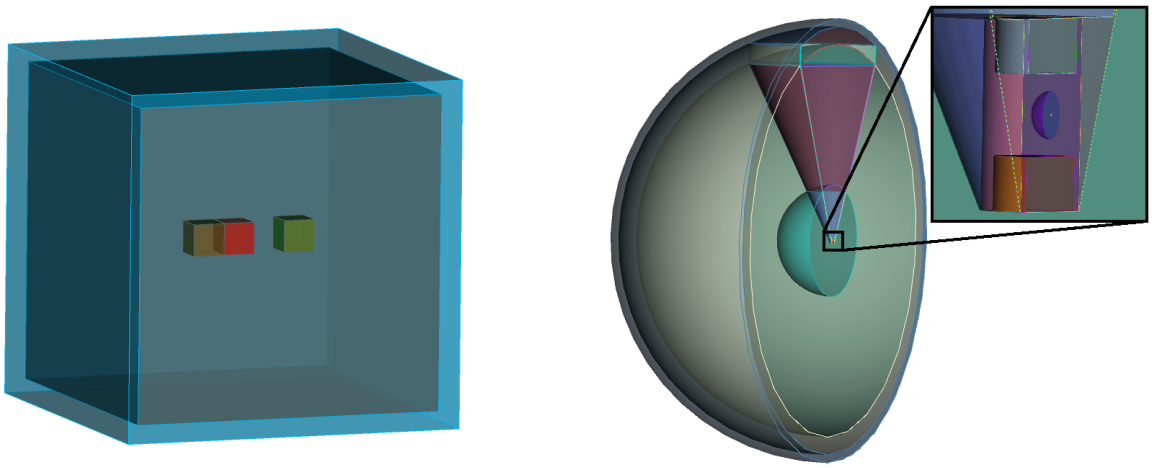
4.4.2 Z-Pinch Fusion Reactor

Next a z-pinch fusion reactor model was tested, as displayed in Figure 4.2b. This featured a sphere of foam fluoride-lithium-beryllium salt (FLiBe) surrounding the target instead of a liquid FLiBe breeder. A reflecting boundary divided the model in half for symmetry. The target was modeled at its actual dimensions, with the complement surrounding the breeding sphere and extending to the pressure vessel. Tallies were identical as shown in Table 4.1. Random number and collision counts differed by $\sim$0.001%. No particles were lost.

4.4.3 ITER Module 13

The first wall and shield (FWS) module 13 for the International Thermonuclear Experimental Reactor (ITER) was used as the last test case, as shown in Figure 2.5e. An intricate complement surrounds the solid components of the first wall and shield. Reflecting surfaces bound the outside of the complement. The complement was filled with homogenized shield material, making this the first test involving a non-void complement. Track length tallies scored nuclear heating in each material. All ten tallies, the random number count, and the collision count were identical as summarized in Table 4.1. No particles were lost.

[1] When particles reach the outside world, or *graveyard*, their histories are terminated.



(a) Cubes

(b) Z-Pinch Fusion Reactor

Figure 4.2: Models used for implicit complement testing.

Table 4.1: Comparison of DAG-MCNP simulation results using explicit vs. implicit nonsolid space.

Model	Tallies	Random Number Count	Collision Count
Cubes	identical	identical	identical
Z-Pinch Fusion Reactor	identical	within 0.001%	within 0.001%
ITER Module 13	identical	identical	identical

Unexpectedly, the explicit and implicit complement do not always produce identical results. Merged surfaces differ within a tolerance, and are responsible for the discrepancies shown in Table 4.1. Although only a single surface representation exists for the implicit complement, one of two surface representations exist for the explicit complement. When two surfaces are merged, one surface is kept while the other is discarded. Merging is performed automatically, making the selection of the final surface ambiguous.

4.5 Conclusion

With implementation of the implicit complement, it is possible to perform DAGMC simulations without defining nonsolid space in a CAD program. The complement is now automatically assembled as a collection of unmerged surfaces. Except for existing in the CAD model, the implicit complement has all of the DAGMC volume representations as would an explicit volume. It was demonstrated that DAGMC performs similarly for explicit and implicit complements—in most cases exactly the same. The largest variation between explicit and implicit complement simulations was $\sim$0.001%, as determined with the random number and collision counts of the z-pinch fusion reactor model. Significant productivity increases have resulted from eliminating the Boolean operations and merging associated with creating the complement. This feature has since been used in the ITER test blanket module, 40° ITER benchmark, ITER FWS module 4, and several fission reactor models.

Chapter 5

Overlap Tolerance

5.1 Motivation

In a properly defined CAD model, a point that is not on a volume boundary is contained in at most one volume. This reflects physical intuition because two solid objects cannot intersect. Monte Carlo codes expect particles to be located in exactly one volume during each segment of their trajectory. Although difficult to detect when visualizing geometry, overlapping volumes are common in CAD models. The original version of DAGMC cannot track particles across overlaps. Manually removing overlaps in a CAD program adds days to the analysis time for intricate models. Repairing CAD models is time-consuming and tedious, lowering the human efficiency of DAGMC analysis.

Overlaps occur due to imprecise draftsmanship, file translation, and intentional deformation of finite element models. Engineering analysis of CAD models demands more precision than required for prototyping. If designers are not accustomed to creating accurate models, defects such as overlapping volumes may result. Training draftsmen to build models without overlaps somewhat improves the situation. For example, significant improvement was made during iterations of the ITER FWS module 13 design. Although early versions had overlapping volumes, later models were free of defects.

Overlapping volumes may be caused by file translation. Although DAGMC predominantly uses the ACIS solid modeling engine, CAD models may originate in other formats. Translation through a neutral file format such as STEP or IGES changes the model and introduces overlaps.

When collaborating with other institutions, translation cannot be avoided if collaborators prefer modeling engines unsupported by DAGMC's solid model interface, CGM.

Contact of adjacent volumes may be intentional as a result of deformation. In a collaboration with Sandia National Laboratories (SNL), space reactor finite element models were deformed to simulate impact onto a concrete pad after launch failure. DAG-MCNP was then used to investigate the criticality of the deformed mesh model. Small overlaps are caused by imprecise contact calculation of adjacent mesh elements in the impact simulation. While tolerances may be tightened in the impact simulation, overlaps are difficult to eliminate. Overlaps are prevalent because contact exists between every adjacent mesh block in the deformation. All particles become lost if using DAGMC's original tracking algorithm.

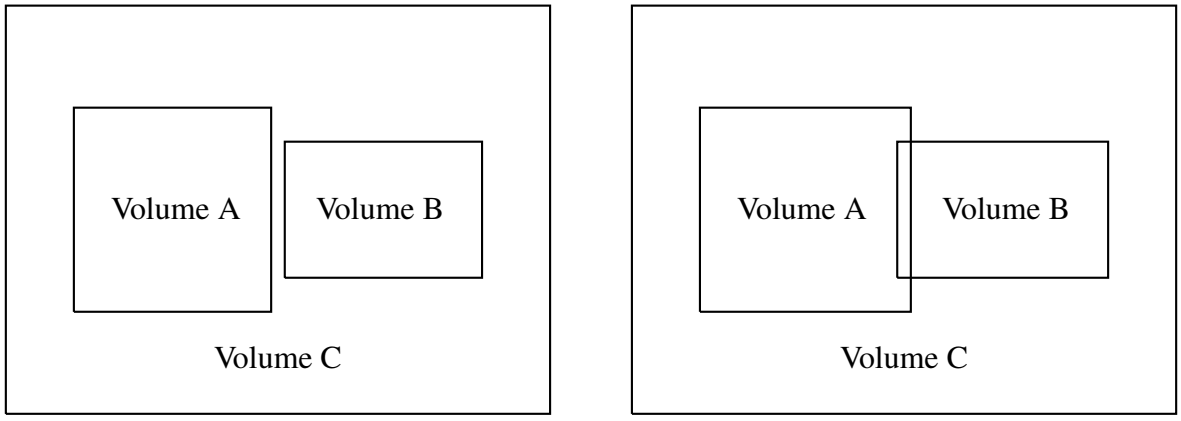
5.1.1 Statement of Problem

The purpose of this chapter is to present an algorithm that *tracks particles through a faceted CAD model with overlaps*. The original algorithm fails because particles cannot find exit intersections between volumes which are overlapping, causing particles to become lost. Small overlaps are prevalent in CAD models and do not significantly affect tally results. Without manual CAD repair, models containing overlaps cannot be analyzed using the original algorithm. A new algorithm was devised to solve this problem, reducing manual labor required to analyze CAD models.

The remainder of this chapter is structured as follows. Overlaps and self intersection will be explained before reviewing previous work and the behavior of point inclusion tests (PITs) for self-intersecting volumes. A tracking algorithm and PIT that are tolerant overlaps will be presented. After discussing the implementation, DAGMC will be tested using the 40° ITER benchmark model and deformed space reactor models.

5.2 Theory

First, some terminology is described for tracking particles through overlapping volumes. Volumes *overlap* if their intersection is not either empty or corresponds to one or more entities of lower dimension (i.e. surfaces, curves, or vertices) of the facet-based model (FBM). In Section 5.3.1 it is



(a) No overlap.

(b) Volumes A and B overlap.

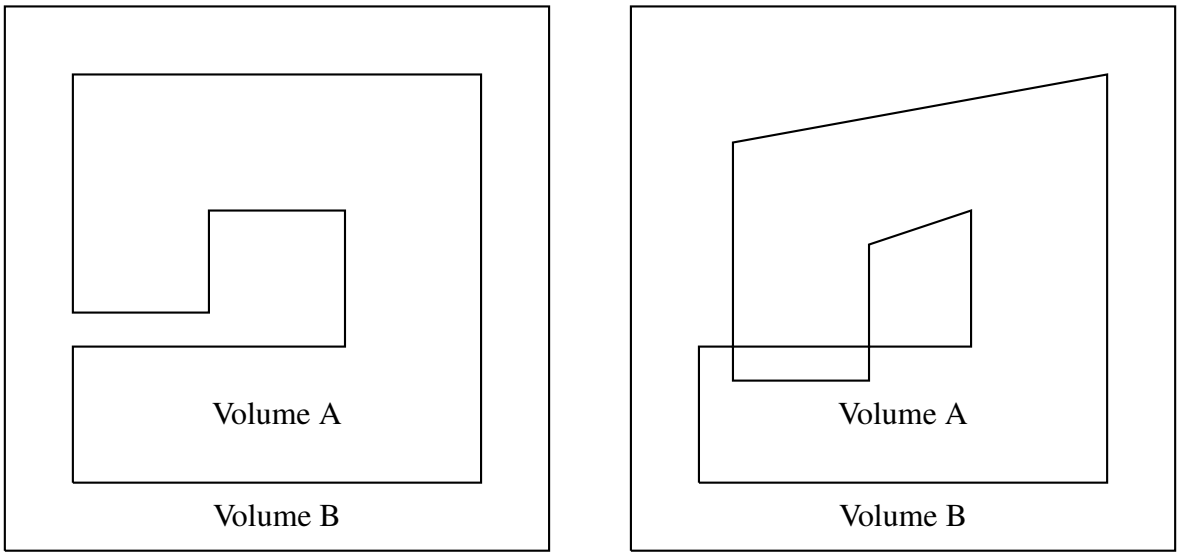
Figure 5.1: Volume C becomes self-intersecting because volumes A and B overlap.

assumed that overlaps are too small to significantly influence results of the simulation. An *overlap thickness* can be determined for all points on the boundary of an overlap. It is the maximum distance across the overlap at the point being queried. A *self intersection* occurs when the boundary of a polyhedron intersects itself. Implicit volumes are self-intersecting if adjacent explicit volumes overlap. As detailed in Chapter 4, explicit volumes typically represent solid material, while implicit volumes represent void or fluid.

5.2.1 Overlaps and Self Intersection

Two or more volumes may overlap each other, or a volume may overlap with itself. In Figure 5.1a volumes A and B do not overlap. After imprecise draftsmanship, file translation, or intentional deformation, volumes A and B overlap as shown in Figure 5.1b. Figure 5.2 shows an example of how a volume may overlap with itself.

Overlaps cause self-intersecting volumes, for which the original tracking algorithm becomes unreliable. Figure 5.1 depicts how the overlap of volumes A and B leads to self intersection of volume C. The volume that contains self intersection is adjacent to overlapping volumes. If a volume overlaps with itself, it will be self-intersecting, as demonstrated in Figure 5.2. For efficient ray tracing, DAGMC searches for exit intersections only in the current volume—not against the entire



(a) No Overlap.

(b) Volume A and overlaps with itself.

Figure 5.2: Volumes A and B become self-intersecting because volume A overlaps itself.

geometric model. Overlaps are not detected when particles travel through overlapping volumes. Instead, overlaps are detected when searching for exit intersections from self-intersecting volumes. When tracking, overlaps occur if the exit intersection exists at a negative distance along the particle trajectory.

Self intersections often occur in the implicit complement volume, although self intersection is possible with explicit volumes also. For example, when modeling fuel rods, the gap between cladding and fuel may be represented as an explicit volume. Due to deformation of the finite element model, the gap between the fuel and clad will close. The fuel and clad volumes slightly overlap each other with the gap volume containing self intersections. To limit special cases, in this work a general approach is pursued in which both implicit and explicit volumes are treated the same.

It is useful to review how DAGMC tracks particles through geometric models. The tracking algorithm in DAGMC searches for the intersection at which a particle leaves the current volume, known as the exit intersection. Each surface is adjacent to exactly two volumes, including both implicit and explicit representations. When a particle leaves the current volume, the next volume is determined using surface adjacency information. The tracking algorithm then searches for the exit intersection of the next volume. Only exit intersections are found because the exit location out of the current volume is identical to the entrance location into the next volume.

Self-intersecting volumes present a challenge because the exit intersection may occur behind the particle's numerical location, as demonstrated with Figure 5.3. In this example a volume has three internal voids, two of which overlap causing a self intersection. Surface normal vectors are used to indicate the outward direction for boundaries of the volume. The particle enters the volume at a and exits at b. After traveling through a different volume it again enters at c and exits at d. After traveling through a different volume it again enters at f. The next exit location should be e, but e is behind the particle's current location. The overlap between e and f is exaggerated for this example. If g is incorrectly selected as the exit intersection the particle will become lost because the surface normal is in the wrong direction. If h is incorrectly selected as the exit intersection

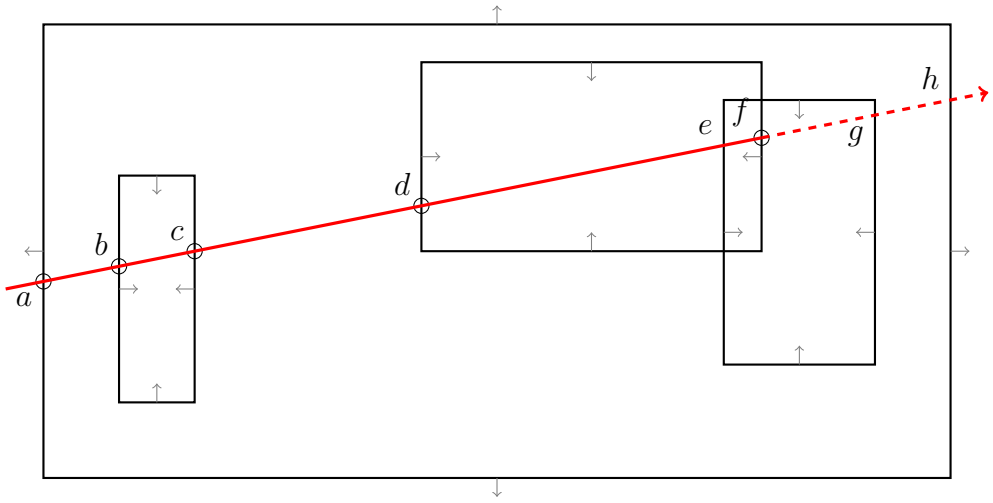


Figure 5.3: Particle track encounters an overlap.

the particle will never travel through the volume between e and g. The algorithm presented in this chapter will correctly select e as the exit intersection.

5.2.2 Tracking Through Overlaps

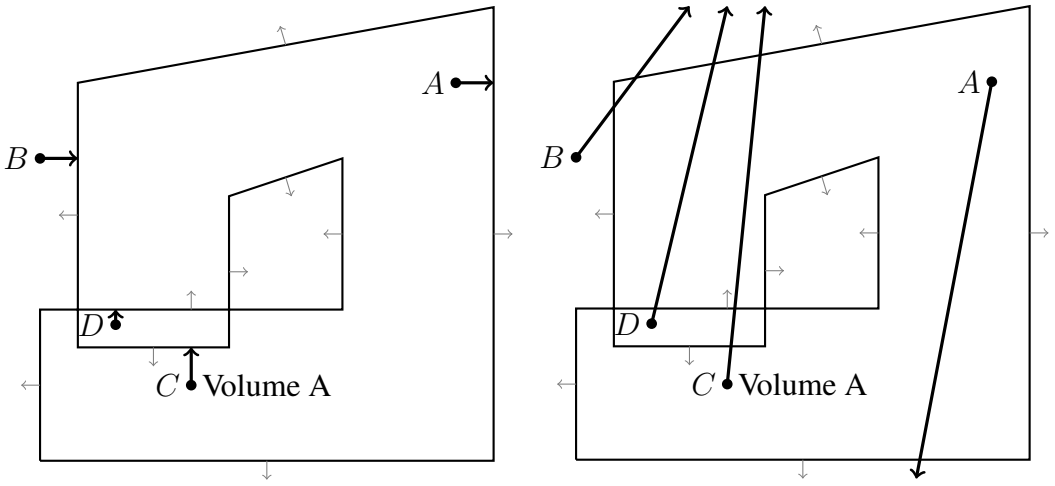
The material definition inside an overlap should be that of one of the overlapping volumes or a composition of materials among overlapping volumes. One approach to assigning materials in overlaps is to set a material priority for each volume [73]. The material with highest priority is chosen in the overlap. This approach depends on the ability to identify when a particle is traveling in an overlap.

Ray-trace on All Volumes One method of tracking through overlaps is to determine several intersections in a single ray tracing computation. In this method, all volumes of the model are searched for intersections. If a pair of exit-entrance intersection can be located within tolerance of each other a small feature has been found, which can be categorized as a gap or overlap. For discussion of this method, gaps are characterized by the exit distance being greater than the entrance distance. Conversely, overlaps are characterized by the entrance distance being greater than the exit distance. If the thickness of gaps or overlaps is less than a specified tolerance, transport

though the feature can be omitted. Avoiding insignificantly small steps along the particle trajectory is computationally efficiency. Care must be taken to conserve track length. In an implementation of a similiar approach, particles traveling through overlaps with thickness greater than a specified tolerance are reported as lost [74].

Ray-trace on the Current Volume The *ray-trace on all volumes* method differs from the tracking behavior of MCNP and DAGMC. For efficiency, MCNP and DAGMC only search for intersections against the volume which the particle is leaving. Because the particle is always leaving the current volume, every intersection must be an exit intersection. Exit intersections are characterized by the angle between the surface normal and particle trajectory being less than 180 degrees. Searching only the current volume for intersections avoids a more computationally expensive search among all volumes in the model. Upon crossing a surface and entering a new volume another ray trace, or `next_surface` call, will be performed against the new volume. Control returns to the physics code after every `next_surface` and `next_volume` computation. To cross an overlap with a single `next_surface` call, control cannot return to the physics code between overlapping volumes.

When ray-tracing on the current volume, overlaps cause lost particles because the correct exit intersection is behind the particle's numerical position. The correct exit intersection is not detected when searching for intersections in the forward direction. One method of finding the correct intersection is to search behind the particle in the reverse direction. Care must be taken to search far enough in the reverse direction to detect the correct exit intersection. Invariably, a search in the reverse direction will encounter intersections which are not valid exit intersections. A method to select the correct intersection is discussed in Section 5.3.4. If a particle is traveling nearly tangent to a surface, an overlapping surface intersection may occur at a large distance in the reverse direction. This is possible even if the overlap thickness is minimal. Overlap thickness considerations are addressed in Section 5.3.5.



(a) Enlarged orientation method.

(b) All ray intersection methods.

Figure 5.4: Point inclusion tests of a self-intersecting volume.

5.2.3 Point Inclusion Test

The original PIT used in DAGMC fails for self-intersecting volumes. A new test is needed that is tolerant of self-intersecting volumes yet efficient enough to use in production calculations. This section surveys PITs discussed in Section 3.2.3 for fitness with self-intersecting volumes.

Enlarged Orientation Method The enlarged orientation method fails for self-intersecting volumes. Figure 5.4a depicts the enlarged orientation method applied to a self-intersecting volume in two dimensions. Four test points are labeled A-D. Each point is drawn with a vector to the closest location on the boundary of the volume. Points are inside if both the vector to the closest location and the surface normal vector point in the same direction. Table 5.1 shows the result of several PITs for points A-D, with incorrect results in parentheses. The enlarged orientation method determines point C to be outside because the surface normal vector of the closest location has opposite orientation due to self intersection.

Winding Number Method In the winding number method, points within an overlap will have an additional 4π projection. The results of the winding number test on the self-intersecting volume

Table 5.1: Point inclusion test results for a self-intersecting volume.

Method	Point A	Point B	Point C	Point D
Enlarged Orientation	In	Out	(Out)	In
Winding Number	In	Out	In	In
Ray Intersection Parity	In	Out	In	(Out)
Ray Intersection Orientation *first intersection*	In	Out	(Out)	In
Ray Intersection Orientation *all intersections*	In	Out	In	In

are shown in Table 5.1. Although suitable for self intersection, the winding number test is avoided because it has a time complexity of $O(n)$ where n is the number of facets in the polyhedron.

Ray Intersection Parity Method The ray intersection parity method fails for self-intersecting volumes. Figure 5.4b illustrates ray intersections for points A-D, with results listed in Table 5.1. Extra intersections caused by overlaps reverse the even-odd state, or *parity*, of the test. An even number of intersections (2) incorrectly determines point D to be outside.

Ray Intersection Orientation Method This method, used in the robust tracking algorithm, also fails for self-intersecting volumes. Like the enlarged orientation method, the ray intersection orientation method fails because the first intersection may have opposite orientation due to self intersection. Ray intersections are depicted in Figure 5.4b. The results of this method—which uses the *first* piercing intersection—are listed in Table 5.1.

5.3 Algorithm

5.3.1 Assumptions

The collection of entities in the FBM does not form a cell complex due to overlapping volumes. In the form of a boundary representation, volumes are enclosed by one or more watertight shells. Shells are not pseudo two-manifolds because the boundary of a volume may intersect itself. Shells are resolved into surfaces composed of planar facets. Each faceted surface is non-degenerate (all

points of $d-$dimensional facets, $d > 0$, are distinct), and is oriented and non-inverted (normal of facet is consistent with that of the underlying model entity in the neighborhood of the facet). Each surface is adjacent to two volumes, including explicit and implicit representations. The overlap thickness is much less than the characteristic dimensions and tallies of the geometric model. Overlapping volumes may contain materials with vastly different nuclear properties. *It is assumed that track length through overlaps may be assigned to any of the volumes included in the overlap, affecting tallies accordingly.*

5.3.2 Overlap Degree

The *degree* of a self-intersecting region is the difference in the number of exit and entrance intersections of the volume boundary along a path from a test point to infinity.

$$degree = \#exits - \#entrances \quad [5.1]$$

$$= \#coincident\ regions \quad [5.2]$$

A region of $degree = 0$ is outside the volume. If $degree = 1$, the region is inside the volume. If $degree = 2$, the region is an overlap of the volume with itself. The volume overlaps with itself $n - 1$ times for an n-degree region. In Figure 5.5, adjacent cubes slightly overlap each other. For each set of cubes, suppose all of the cubes are regions of the same volume. The degree of the overlap is related to the dimension, as listed in Table 5.2. The significance of previous facets from Table 5.2 will be discussed in Section 5.3.5. Cube-like shapes are significant to this example because hexahedral mesh elements are used for the deformed space reactor analysis of Section 5.5.3. Several meshed reactor components could impact each other in the configurations of Figure 5.5, suggesting a limit on the overlap degree.

5.3.3 Point Inclusion Test

The original version of DAGMC used the enlarged orientation method for point inclusion testing. The enlarged orientation method fails for self-intersecting volumes because the closest surface may be inversely oriented. As proposed in Section 3.3.5, the ray intersection orientation method

(a) 1-dimensional overlap. (b) 2-dimensional overlap. (c) 3-dimensional overlap.

Figure 5.5: Multi-dimensional overlaps.

Table 5.2: For cubes, degree of overlap and number of previous facets to be stored as a function of overlap dimension.

Dimension	Degree	Previous Facets
1	2	2
2	4	6
3	8	14
d	2^d	$2(2^d - 1)$

fails for the same reason. A new point inclusion test is needed for models with self-intersecting volumes.

Ray Intersection Orientation Method The ray intersection orientation method can be altered to give correct results for self-intersecting volumes, as shown in Algorithm 5.1. Instead of considering only the first piercing intersection, all piercing intersections are used. The number of entrances and exits are summed along a ray cast from the test point to infinity. A point that is inside the volume will have at least one more exit than entrance, or *degree* greater than zero. Table 5.1 lists the results of the ray intersection orientation method—using *all* intersections along the ray—on points A-D.

A list of previously intersected facets is maintained by the tracking algorithm for each streaming event, and is an optional input to the point inclusion test. Previously intersected facets are not returned from the `ray_intersect_facets` function, thereby avoiding the on boundary result.

This method becomes less efficient for volumes with complex boundaries. As the complexity of the boundary increases, finding each intersection in the direction of the ray becomes more costly. If a direction is not specified, a random direction is used so that a ray is equally likely to cross all parts of the boundary. Test performance could suffer if particles are consistently traveling in the direction of the volume's most complex boundary. This algorithm minimizes numerical error, avoids unnecessary tolerances, and ensures consistency between the PIT and tracking algorithm.

5.3.4 Exit Intersection Considerations

As previously discussed, the exit intersection may be behind the ray origin due to overlap. The additional ray-triangle intersections (RTIs) discovered by searching behind the ray origin make selection of the correct intersection challenging. In Figure 5.6, three surfaces exist with correct orientation; the surface of the exit intersection is indicated with a dashed line. Although correct orientation is necessary, it is not sufficient to determine the exit intersection.

The correct exit intersection will be found by rejecting glancing RTIs, RTIs with incorrect orientation, and previously intersected facets. The remaining RTIs will be disambiguated by using proximity and the point inclusion test. Only the closest RTIs in both the positive and negative

```
point_inclusion( point, direction, volume, prev_facets, &result )

// If a direction does not exist, use a random direction.
if( NULL == direction ) direction = get_random_direction()

// Set distance limits of the intersection search.
nonneg_dist_limit = HUGE_VAL
neg_dist_limit    = 0

// Return all piercing intersections.
// Coplanar triangles ( 0 == alpha ) are not piercing intersections.
// Each intersection has a surface, distance, and facet.
call ray_intersect_facets( volume, point, direction, prev_facets,
                           neg_dist_limit, nonneg_dist_limit,
                           &surfs, &dists, &facets )

// Get the orientation of each intersection.
for all facets
  normal    = get_normal( volume, facet, surf )
  alphas[i] = get_dot_product( normal, direction )

// Sum the entrances/exits.
sum = 0
for all alphas
  if     ( 0 < alpha ) sum += 1
  else if( 0 > alpha ) sum -= 1

// Determine entering/leaving.
if( 0 > sum ) result = inside
else          result = outside

return SUCCESS
```

Algorithm 5.1: Overlap-tolerant point inclusion test.

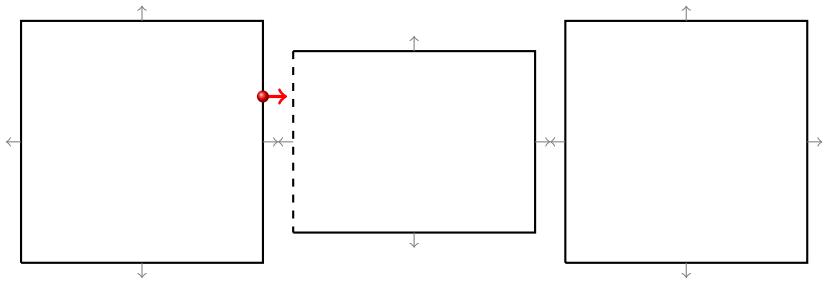
directions will be considered. Among the remaining pair of intersections, the RTI at negative distance will be rejected unless it is closer than the positive distance. Before accepting the RTI at negative distance a PIT conducted. If the particle is inside an overlap, the PIT will report the numerical location as being $inside$ the next volume. A zero-distance advance will be used to make the logical position become consistent with the numerical position, despite the existence of an overlap. The PIT is important because it guarantees that track length will be assigned to the correct volume. Track length tallies are guaranteed to be correct even if the geometric model contains overlaps.

5.3.5 Overlap-Tolerant Tracking Algorithm

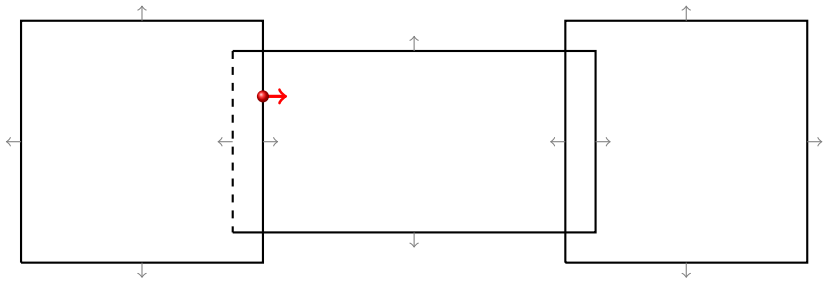
Algorithm 5.2 describes the process of finding the correct exit intersection. An exit intersection includes both numerical (location, as a distance) and logical (surface and facet) information. First it is determined if the particle is streaming. Neutral particles travel in straight lines, or *stream*, between collisions. A particle has streamed to its current location if its direction has not changed and its previous surface is known. If these two conditions are met, it is assumed that the particle is streaming. Streaming is significant because streaming particles cannot intersect the same facet twice. This property does not apply to surfaces because surfaces may not be planar.

A list of previously intersected facets is maintained for each streaming event. The algorithm prevents the use of previously intersected facets as exit intersections, to avoid infinite loops in which a particle alternates between exit intersections in close proximity. If a particle intersects a reflecting boundary, only the facet on the boundary is stored. If the particle is not streaming, the previous direction is reset to the current direction and the list of previously intersected facets is cleared.

Ray length limits are provided to the ray tracing function to increase efficiency of the intersection search. The distance to search behind the particle for overlaps is the *negative ray length*. The negative ray length is important because RTIs will be missed if is too small. If it is too large, the computation will be inefficient and extra intersections will be found. The negative ray length is set to the overlap thickness—a user-defined tolerance from the input file. The distance that the



(a) The exit intersection is in front of the particle because volumes do not overlap.



(b) As a result of overlap, the exit intersection is behind the particle.

Figure 5.6: Position of the exit intersection changes due to overlap.

ray tracing function searches for an intersection in the positive direction may be restricted. The nonnegative ray length can be limited to the distance that a particle will travel before undergoing a collision, known as the physics limit.

An oriented bounding box (OBB) tree is used to accelerate the `ray_intersect_facets` function [33], which searches the current volume's OBB tree for ray-triangle intersections (RTIs). Triangles and their bounding boxes must be closer than the negative and nonnegative distance limits. After each RTI is found, it is tested to ensure that the orientation of the triangle is correct. The intersection is screened for correct orientation, glancing vs. piercing, and previous facets. If the triangle is not oriented such that the particle is leaving the volume, the RTI is discarded. If a glancing intersection of an edge or point occurs, the RTI is discarded. The RTI is also discarded if the facet was previously intersected by the particle in the same streaming event. Distance limits decrease in magnitude as closer RTIs are accepted. The closest RTI at positive distance and, if closer, at negative distance is return to `next_surface`.

If an RTI at negative distance exists and is closer than the RTI as nonnegative distance, a point inclusion test is performed to ensure that the particle is located in the volume across the surface of the negative RTI. If the particle is not inside the volume, the RTI is discarded. By using the PIT to confirm the existence of an overlap, it is guaranteed that the numerical and logical location will be consistent; in addition the PIT will return a consistent result because it performs the same numerical ray tracing computation using the `ray_intersect_facets` function. If the particle is inside the next volume, the RTI is returned as the exit intersection. To conserve track length, the intersection is reported to occur at zero distance. The zero-distance advance is described in Section 3.3.6. Otherwise the RTI at nonnegative distance is selected exit intersection. Before exiting, the RTI's facet is added to the list of previously intersected facets.

The only tolerance used in this algorithm is the overlap thickness. From the enumeration of RTIs in Section 3.3.4, particles traveling through overlaps are of case *outside-I*. The exit intersection is guaranteed to be found if the user sets the overlap thickness to be at least as large as the maximum overlap thickness of the FBM. In addition, the PIT ensures that the particle's numerical location will be inside the next volume.

```
next_surface( prev_surf, prev_dist, ray_dir, ray_point,
              volume, physics_limit,
              &next_dist, &next_surf, &prev_facets)

// Determine if the previous surface was reflective.
if( reflecting )
  last_facet = prev_facets.back
  prev_facets.clear()
  prev_facets.push_back( last_facet )

// Determine if the particle is streaming.
else if( !streaming )
  prev_facets.clear()

// Set distance limits of the intersection search.
nonneg_dist_limit =  HUGE_VAL
neg_dist_limit    = -OVERLAP_THICKNESS
if( NULL != physics_limit )               nonneg_dist_limit =  physics_limit
if( nonneg_dist_limit < -neg_dist_limit ) nonneg_dist_limit = -neg_dist_limit

// Return the closest intersection in the positive direction and if closer, one in
// the negative direction less than neg_dist_limit. Only return piercing exit intersections.
// Each intersection has a surface, distance, and facet.
call ray_intersect_facets( volume, ray_point, ray_dir, prev_facets,
                           neg_dist_limit, nonneg_dist_limit,
                           &surfs, &dists, &facets )

// Is the RTI at negative distance inside the next volume?
if( NULL != facets[0] )
    next_vol = get_next_volume( surfs[0], volume )
    call point_inclusion( ray_point, ray_dir, next_vol, prev_facets, &result )
    if( INSIDE == result )
        next_dist = 0
        next_surf = surfs[0]
        prev_facets.push_back( facets[0] )
        return SUCCESS

// Return the RTI at positive distance if it exists.
if( NULL != facets[1] )
    next_dist = dists[1]
    next_surf = surfs[1]
    prev_facets.push_back( facets[1] )
    return SUCCESS

// If using physics_limit, assume a collision occurs before an exit intersection.
if( NULL != physics_limit )
  next_dist = physics_limit+1
  next_surf = 0
  return SUCCESS

// Otherwise the particle is lost.
next_dist = HUGE_VAL
next_surf = 0
return FAILURE
```

Algorithm 5.2: Overlap-tolerant tracking algorithm.

Revisiting the discussion from Section 5.2.1 of Figure 5.3, the particle was located at f. The `ray_intersect_facets` function will return e and h as possible intersections. Intersections a, b, c, and d are not returned because they are in the list of previously intersected facets. Intersection g is not returned because it does not have correct orientation. A point inclusion test is performed on f to make sure that it is located inside the volume adjacent to the current volume through the surface of e. This is true. e is accepted as the next intersection because the magnitude of the distance from f to e is less than f to h. To conserved track length, negative distances are reported to the physics code as zero distance.

5.4 Implementation

Instead of maintaining two separate versions of `next_surface`, the overlap-tolerant algorithm is implemented as an option inside the robust tracking algorithm. Similarly, the overlap-tolerant PIT is implemented as an option inside the PIT from Section 3.3.5. The only difference is the overlap thickness replaces numerical precision when determining the negative distance limit.

Substantial effort was invested to implement the algorithm as efficiently as possible. An early implementation temporarily move the origin of the ray backwards to avoid implementing the ray-intersection algorithms in the backward direction. This effort utilized a distance limit in only the positive direction, and return all RTIs behind the particle's numerical location. This attempt was abandoned in favor of the current approach because keeping all of the RTIs behind the particle is inefficient—only the closest RTI in the negative direction will be used. The inefficiency was most noticeable when the overlap thickness is very large.

The most inefficient component of the overlap-tolerant logic is the PIT. It requires all intersections with the boundary of the volume, instead of only the first intersection. For complex boundaries as are typical with implicit volumes, the PIT is costly. If the closest intersection is *leaving*, the point is always inside the volume. This fact does not significantly accelerate the test because if the closest intersection is *entering*, the closet intersection is not sufficient—all intersections with the boundary will be required anyhow. In addition, the closest intersection is rarely the first intersection discovered during the OBB tree traversal inside `ray_intersect_facets`.

5.5 Testing and Analysis

DAGMC's original tracking algorithm lost particles that encountered an overlap. The new overlap-tolerant algorithm performs well and has enabled new types of analyses that have previously been impossible. This section presents results for CAD models, scaling behavior, and deformed space reactor simulations.

5.5.1 ITER Benchmark Model

Overlaps due to file translation and imprecise modeling are common in CAD geometry. Overlaps prevent automated imprinting and merging of nearly-coincident surfaces. Unmerged nearly-coincident surfaces often have different faceted representations, which overlap. Using a model that does not contain overlaps, the new version of DAG-MCNP was compared with and without overlap-tolerant tracking. Overlap tolerant tracking is actuated by specifying a nonzero overlap thickness within the input file. Next, the overlap-tolerant algorithm was tested with geometry that contains overlaps due to unmerged surfaces. The ITER 40° benchmark model was selected due to its complexity and prominence in many analyses. These comparisons build upon those in Section 3.5.2.

This comparison investigates the expected best-case and worst-case performance of DAGMC. The most efficient simulation occurs with merged geometry that does not contain overlaps. The most inefficient simulation occurs with all unmerged geometry, implying all surfaces overlap. Unmerged geometry necessitates DAGMC's overlap-tolerant mode of tracking. Overlap-tolerant tracking is less efficient because intersections must be discovered in the reverse direction along the particle trajectory. In addition, the overlap-tolerant PIT requires that *all* intersections be found along the trajectory—not only the closest intersection. A PIT occurs every time the particle encounters an overlap.

The results in Table 5.3 were computed on one core of an Intel Core2 2.66 GHz CPU using 100k particle histories. No particles were lost. Cases 1-4 used materials, while cases 5-7 compared performance in voided models. Comparing cases 1 vs. 2 and 5 vs. 6 the tallies, random number

count, and collision count were unchanged by using overlap-tolerant tracking. Comparing cases 2 vs. 3 the random number and collision count did change with unmerged surfaces in the cases that used materials. This is because every surface in the merged model had an additional coincident surface in the unmerged model that was similar but not the same. The four cases without materials had the same collision and random number count. However, this agreement is not meaningful because voided models will not have any collisions, and random numbers are used only to spawn source particles. The physics limit increased tracking rate without altering results, as expected. Comparing cases 1 vs. 2 the speed penalty for using overlap-tolerant tracking on a merged model is 5%. Comparing cases 1 vs. 3 the speed penalty for using an entirely unmerged model is 53%. The tracking rate decreased because the overlap-tolerant algorithm must search behind the numerical position of the particle for intersections. The unmerged model had twice as many surfaces, demanding a corresponding increase in the number of ray tracing calculations. All of the additional ray tracing calculations were performed on the implicit complement volume, which has the most complex boundary of the entire model. Additional point inclusion tests of the overlap-tolerant algorithm also contribute to the decrease in tracking rate. For models without materials, the difference in tracking rate was more substantial. This is because overall, more execution time is spent tracking particles. These are general observations—tracking rate is highly model and tally dependent.

5.5.2 Performance of Overlapping Geometry

The comparison in Section 5.5.1 presents the best- and worst-case scenarios of entirely merged and entirely unmerged models, respectively. A more likely case is a model in which most, but not all of the surfaces are merged. This occurs when the automated *imprint and merge* step of the analysis workflow partly fails. Prior to overlap-tolerant tracking, the analyst manually repaired and merged the remaining unmerged surfaces. Overlap-tolerant tracking provides an automated alternative.

To investigate the performance of a varying fraction of unmerged surfaces, CAD models were constructed with 11 cubes positioned in a line, as shown in Figure 5.7. No materials are used. The

Table 5.3: Particle tracking of overlap-tolerant DAG-MCNP is compared using merged and unmerged versions of the 40° ITER benchmark model.

Case	Coincident Surfaces	Overlap Thickness [cm]	Physics Limit	Tracking Rate [part./min.]	Tallies	Collisions [#]	Random Numbers [#]
			with materials				
1	merged	0	no	9976	*baseline*	12701310	185958957
2	merged	10	no	9507	identical	12701310	185958957
3	unmerged	10	no	4703	identical	12701228	185958361
4	unmerged	10	yes	4851	identical	12701228	185958361
			without materials				
5	merged	0	no	81225	*baseline*	0	754270
6	merged	10	no	77527	identical	0	754270
7	unmerged	10	no	12382	identical	0	754270

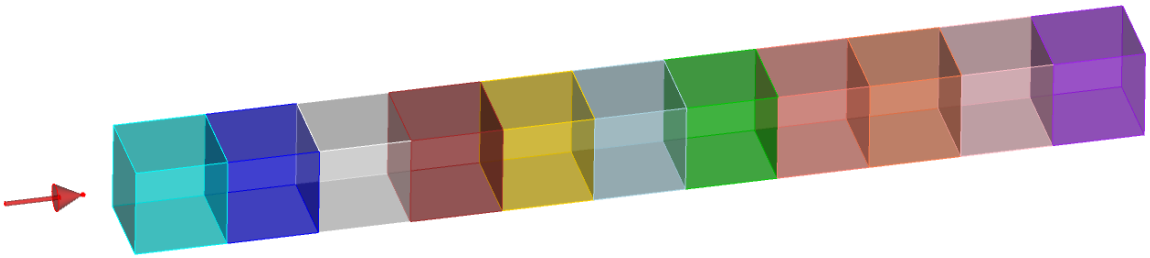


Figure 5.7: An set of models was assembled, each with 11 cubes between which 0-10 overlaps exist.

particle source is a beam positioned to travel through each cube. Six models of this pattern were created with 0-10 volumes each overlapping 0.01 cm.

The results, shown in Table 5.4 indicate the overhead for using overlap-tolerant tracking is 1%. This compares with the 5% penalty found in Table 5.3 between cases 5 vs. 6, which also did not have materials. Performance decreased, approximately linearly to case 7 with 10 pairs of unmerged surfaces. Completely unmerging the model results in a 74% penalty, compared with an 85% penalty for the similar pair of cases 5 and 7 in Table 5.3. Analysts can use this information decide the best course of action: either manually fix the CAD model or use DAGMC in the slower, overlap-tolerant mode.

5.5.3 Deformed Space Reactor

One motivation for developing an overlap-tolerant particle tracking algorithm is to analyze deformed models. In particular, this section is about modeling deformed space reactors after a launch accident as discussed in [30]. Impact causes large structural deformation of reactor components that can result in changes in the reactivity of the system. Predicting these changes is an important component of launch safety analysis. Overlap-tolerant tracking is required due to contact of adjacent bodies in the structural analysis. When calculating contact, the structural analysis permits small overlap of adjacent bodies. When initialized for DAGMC, small overlaps result in self-intersecting volumes.

Table 5.4: Tracking rate as a function of the number of overlaps.

Case	Overlaps [#]	Overlap Thickness [cm]	Tracking Speed [part./min.]	Relative Speed [%]
1	0	0.0	946490	100
2	0	0.1	940790	99
3	2	0.1	591490	62
4	4	0.1	400950	42
5	6	0.1	328960	35
6	8	0.1	232940	25
7	10	0.1	249850	26

This work represents the first known mesh-based simulations of the reactivity consequences of impact. There is a long literature studying the criticality of intact space reactors immersed in water/sand after a launch accident; however most of these studies evaluate nominal or uniformly compacted system configurations. In the past, solid models have been manually altered to approximate the shape of reactor components after impact. This method is both tedious and limited in accuracy. An alternative approach has been developed which utilizes mesh geometry exported by a structural dynamics simulation. Mesh-based radiation transport enables high-fidelity geometric description of deformed CAD models. This work resulted in a time-dependent reactivity assessment of a space reactor impacting a concrete pad after launch failure.

Structural failure presents a unique challenge when initializing deformed mesh geometry for DAGMC. During impact, mesh elements may experience structural forces that result in fracture. Mesh elements lose their stiffness to simulate fracture causing them to become geometrically invalid. Labeled as *dead*, these elements cease to participate in the remainder of the structural analysis. Adjusting the mesh topology to remove dead elements requires special treatment.

An 85-pin space reactor concept was used to demonstrate the capabilities developed, as shown in Figure 5.8 [75]. UO_2 fuel pins are clad in SS316 and cooled by NaK. Six control drums containing B_4C and Be are positioned around the perimeter of the core inside a 15.3 cm radial Be reflector. The 25 kW_e reactor is shielded by borated water. The fueled core is 38 cm high with a diameter of 22.9 cm. The structural components of the reactor are SS316. Engineering details were added by Villa for the structural analysis and described in [76, 77].

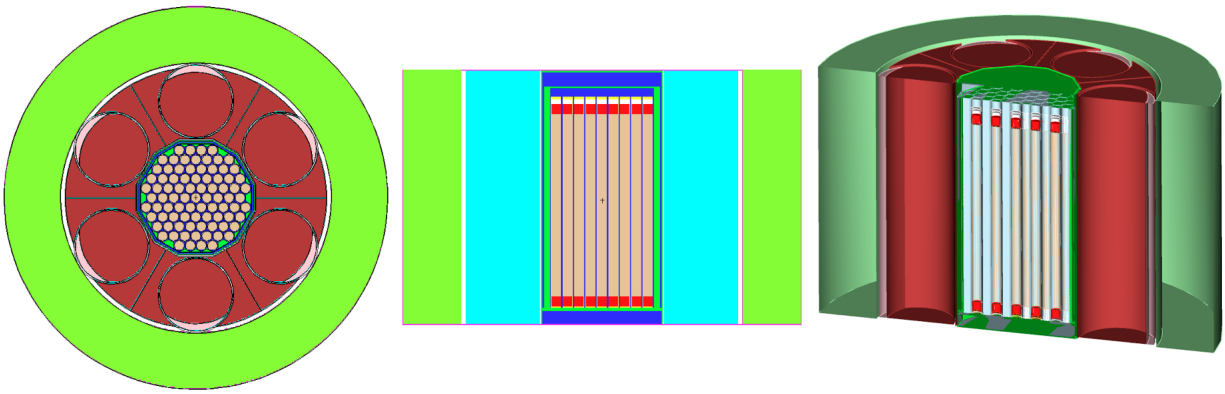


Figure 5.8: Native MCNP (left, center) and converted CAD model (right) of 85-pin space reactor.

Deformed Geometry Initialization Model creation and structural analysis were performed by Villa et al. at Sandia National Laboratories [77]. A solid model of the reactor geometry was created in CUBIT. Structural components were meshed with hexahedral elements. The mesh was exported from CUBIT using the EXODUS II file format [78]. Optionally, fluid volumes were filled with smoothed particle hydrodynamics (SPH) elements using tools developed by Villa et al. The PRESTO finite element structural dynamics code was used for structural analysis [79]. The output of the structural analysis was an EXODUS II file of the deformed mesh.

After structural analysis the model was prepared for reactivity analysis. Additional geometry was specified to define reflective and absorbing boundary conditions around the deformed model. Boundary conditions and materials were specified by assigning groups to the solid model within CUBIT. An MCNP input file was created with material definitions and other data cards, but no cell or surface cards. The MCNP input file, CUBIT geometry file, and EXODUS II deformed mesh file were needed to perform a reactivity analysis.

Analysis of deformed mesh geometry requires a unique initialization that is not required for analysis of CAD models. This is because the internal representation of mesh geometry is different than solid models. DAGMC uses solid models that specify volumes as collections of bounding surfaces, known as boundary representations. However, mesh geometry represents volumes as blocks of three-dimensional mesh elements. Dead elements and volume change are additional challenges of deformed mesh models. Geometry initialization of deformed mesh models includes the following steps:

1. reading the undeformed solid and mesh models from the CUBIT file,
2. updating the mesh point locations from the deformed mesh model in the EXODUS II file,
3. removing dead elements from the mesh,
4. converting quadrilaterals to triangles, and
5. adjusting material densities to accommodate volume changes.

A geometry initialization routine was used to convert the output of the structural analysis into a DAGMC-compatible facet-based model. First the CUBIT file was read to extract the solid and mesh models. The surfaces of the solid model were associated with the corresponding surfaces of the mesh model, using global IDs. A uniform surface sense was established for each pair of corresponding solid/mesh surfaces. This ensured that corresponding surfaces of the solid and mesh models have the same orientation.

Next the deformed model was read. Associations between points in the mesh model and deformed model were created using global IDs. Point coordinates of the mesh model were updated with positions from the deformed model.

Dead mesh elements were removed from the mesh model. Using topology, the bounding faces of the mesh model were identified. Some of the bounding faces were altered by removal of dead mesh elements. New surfaces were created for mesh faces that did not belong to original surfaces of the mesh model.

If corresponding surfaces existed, surfaces of the solid model were replaced by surfaces of the mesh model. This preserves surfaces of the solid model belonging to boundary conditions that did not exist in the mesh model, including reflecting and absorbing boundaries. Quadrilateral elements were converted to triangles by splitting along the shortest diagonal. Surfaces were removed if they no longer contained any mesh faces. Volumes were removed if all of their surfaces have been removed.

Volumes of structural components changed due to element death and material compressibility. The density of each volume was adjusted so that mass was conserved during the deformation. The undeformed volume was calculated using the solid model, faceted with a tolerance of 1 μm. The facet tolerance is the maximum distance between the faceted surface and the continuous surface of the solid model. Mass conservation occurs with a high degree of geometric accuracy because the fuel pellets are segmented into several volumes, as depicted in Figure 5.9. Geometry initialization steps for DAGMC are described with more detail in a Sandia technical report [76].

To illustrate the geometry initialization, Figure 5.10 shows a structural steel component of the 85-pin model pre/post deformation. Figure 5.10 (left) displays the CUBIT hexahedral mesh as

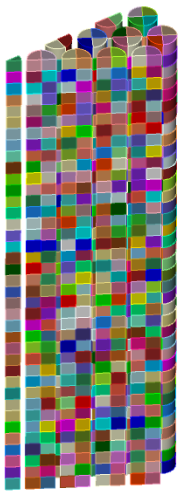


Figure 5.9: Fuel volumes of 0-degree model without SPH elements.

input to the structural analysis. Figure 5.10 (right) shows the deformed geometry as initialized for DAGMC. The point coordinates have been updated to their deformed locations. Dead elements have been removed, leaving a void. A new surface (yellow) has been created with mesh faces exposed by removing dead elements. Quadrilateral mesh faces have been converted to triangles.

Reactivity Calculations A comparison was performed between native MCNP and the overlap-tolerant version of DAG-MCNP using an undeformed solid model. The 85-pin space reactor was analyzed with control drums rotated for minimum neutron absorption. MCNP native geometry was converted to ACIS-formatted CAD geometry using the MCNP2CAD automated conversion utility. Material properties and boundary conditions were automatically mapped to the CAD file. The model has 1176 volumes and 4947 surfaces. All surfaces in the solid model were intentionally unmerged, allowing overlaps of coincident surfaces. This created overlaps to provide a rigorous test of the overlap-tolerant tracking algorithm. Surfaces of the solid model were faceted with a tolerance of 1 μm. Both cases had 10 inactive cycles, 100 active cycles, and 10,000 source neutrons per cycle. The native MCNP case had a k_{eff} of 1.01437 ($\pm$0.00075). This compares well with the DAG-MCNP k_{eff} of 1.01451 ($\pm$0.00080). No particles became lost.

Three simulations were performed of the 85-pin reactor impacting concrete at 100 m/s by Villa et al., as shown in Table 5.5. Simulations were performed at two different angles of impact, measured between the reactor's vertical axis and the surface normal vector of the concrete pad. At 0-degrees, one case included the effects of fluids in the structural analysis using SPH elements, but neither included fluids in the reactivity analysis. At 45-degrees, the fluids were excluded from both the structural and reactivity analyses. Although DAGMC cannot utilize SPH elements, adding fluids increases the accuracy of the impact simulation. For efficiency, 1/2 and 1/12 symmetry were used for the 45-degree and 0-degree models respectively. The volume, surface, and triangle count in Table 5.5 represent the last time step of the deformation, once initialized in DAGMC. The hexahedra count references the CUBIT mesh because hexahedra do not exist in the DAGMC-initialized geometry. The particle tracking rate is representative of one core of a 2.66 GHz Intel Core2 processor and has been averaged for all cases. The computational time required for structural

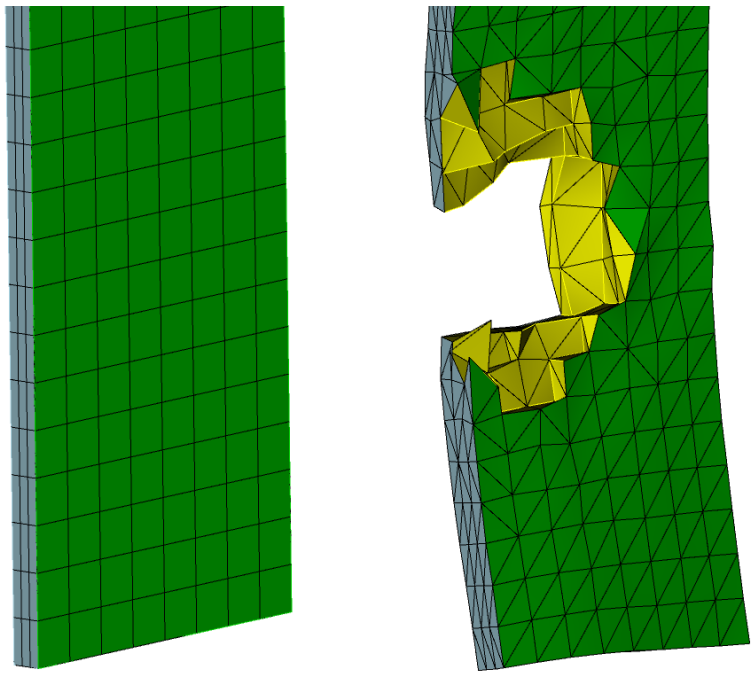


Figure 5.10: 0-degree SPH model's undeformed CUBIT mesh (left) and DAGMC-initialized geometry at 0.225 ms (right).

Table 5.5: Summary of 85-pin reactor impact models.

Case	Angle [degrees]	SPH	Symmetry	Entity Count				Tracking Rate [part./min.]
				Vols.	Surfs.	Hex.	Tris.	
1	0	no	1/12	1308	6673	962k	1.43M	3793
2	0	yes	1/12	3181	17867	966k	1.51M	3137
3	45	no	1/2	3176	11729	8.86M	11.1M	2741

analysis was decreased by doubling the thickness of the cladding to 0.1 cm. Control drums were rotated for maximum neutron absorption. Material density was adjusted to conserve mass on a per-volume basis, despite element volume change and the removal of dead elements. The time-varying point displacement and number of dead elements are shown in Figures 5.11 and 5.12.

Selected time steps for each model were analyzed as a series of DAG-MCNP cases. Each case had 10 inactive cycles, 100 active cycles, and 10,000 source neutrons per cycle. Depending on the particle tracking rate, each case took 5-7 hours on one core of a 2.66 GHz Intel Core2 processor. The neutron multiplication factor is shown for each simulation in Figure 5.13. Error bars represent one standard deviation, but are small compared to the impact-induced change in k_{eff}.

The 0-degree simulation without SPH elements had the greatest increase in k_{eff}. NaK coolant and water shielding were added to the 0-degree structural simulation through the use of SPH elements. The reactivity simulations did not include fluids. Figure 5.14 contrasts the 0-degree simulations with and without SPH elements. Green SPH elements model water shielding and orange SPH elements model NaK coolant. Water SPH elements cause the shield containment to break. The inclusion of fluids in the structural simulation restricts the increase of k_{eff} by limiting contact of adjacent fuel pins toward the bottom of the reactor. The NaK dampens impact by absorbing kinetic energy that would otherwise deform fuel and structural components. The structural analysis is explained in more detail by Villa et al. [77] and Smith et al. [76]. Despite the additional challenge of including fluids in the structural simulation, this work suggests they are crucial to producing realistic results.

Although computational expense did not permit the inclusion of SPH elements, a 45-degree collision was simulated, as shown in Figure 5.13. Although the simulation did not extend until all kinetic energy was dissipated due to time constraints, it clearly demonstrates predictive capability.

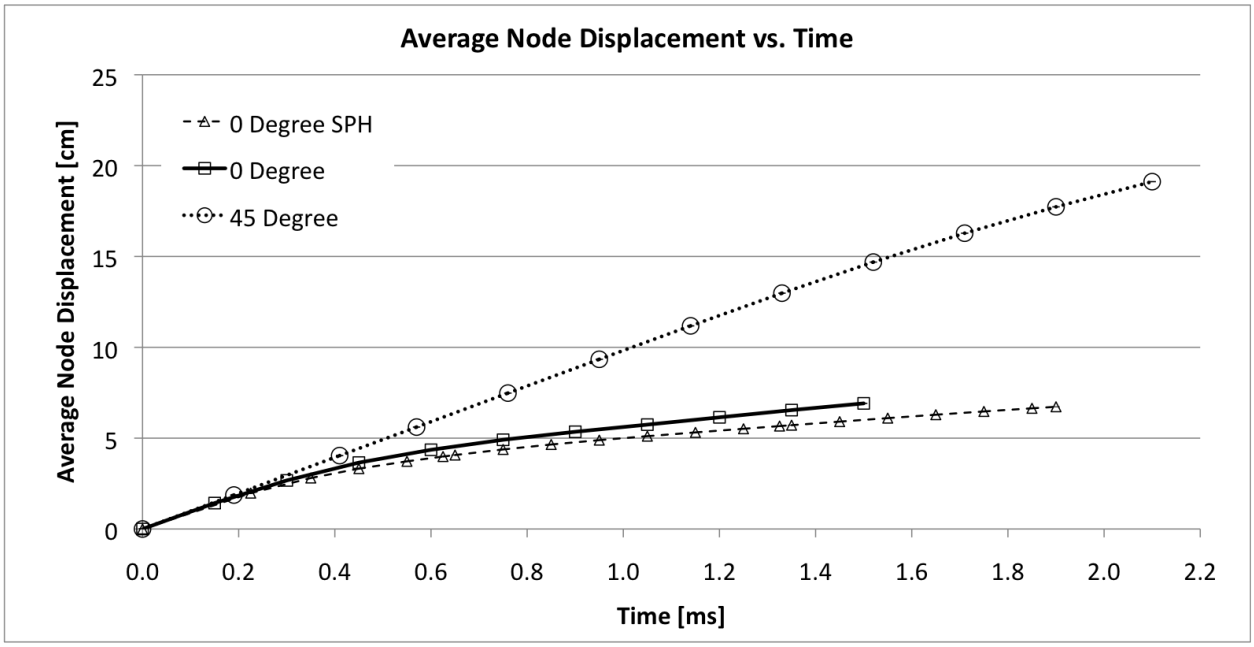


Figure 5.11: Point displacement as a function of time.

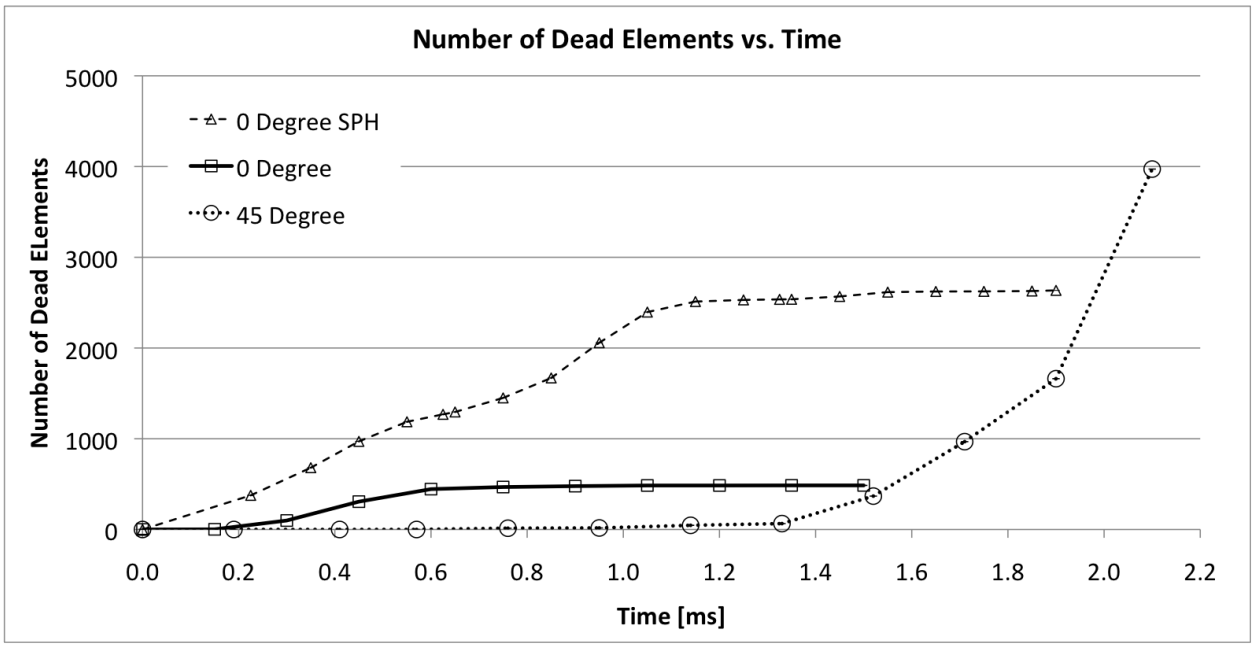


Figure 5.12: Number of dead elements as a function of time.

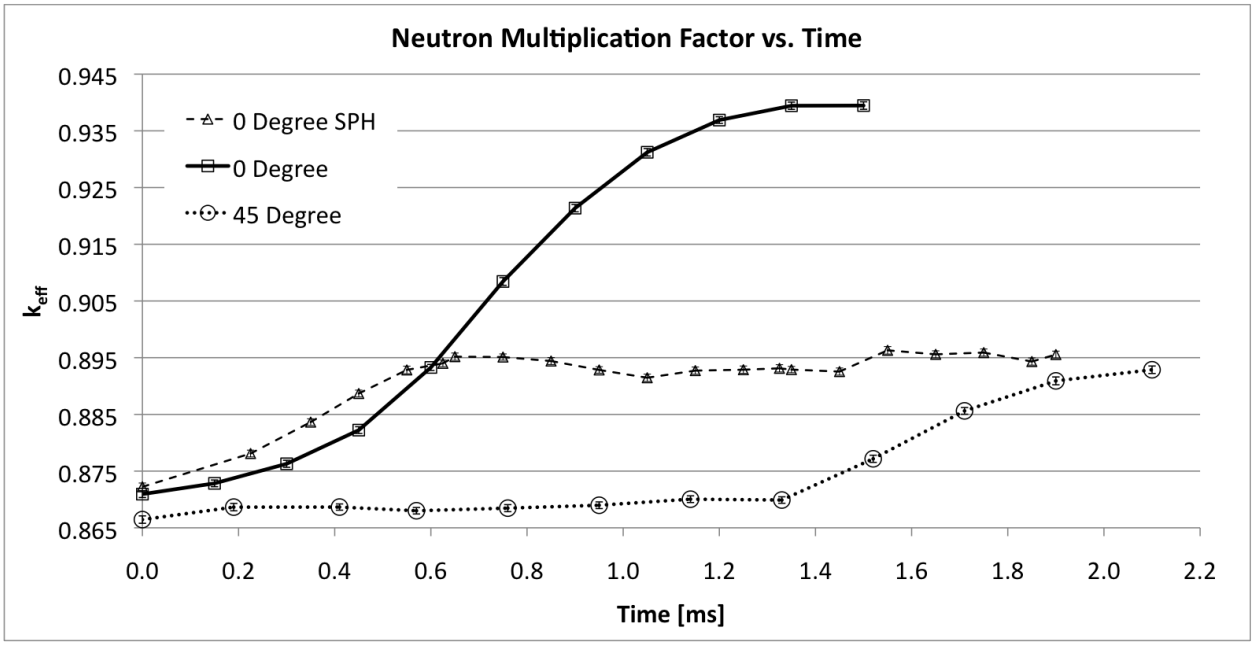


Figure 5.13: Reactivity as a function of time.

The neutron multiplication factor did not increase for the first 0.95 ms of impact because the fuel pins did not move relative to one another. Compared with the 0-degree simulation, the increase in k_{eff} occurred later in the 45-degree simulation due to the angle of impact. After 0.95 ms the distance between fuel pins began to decrease with a corresponding increase in k_{eff}, as shown in Figure 5.13.

The effect of neglecting fluids in the reactivity simulations has been estimated. The surfaces that bound fluid volumes puncture due to fracture in the structural simulation. However, fluid cannot be modeled as a continuous material without being contained in a closed volume. An alternative approach would model fluid as spherical volumes, analogous to SPH elements in the structural simulation. To determine the effect of neglecting fluids in the reactivity analysis, water shielding and NaK coolant were added to the undeformed solid model with control drums rotated for maximum neutron absorption. The neutron multiplication factor increased from 0.87112 ($\pm$0.00069) to 0.87987 ($\pm$0.00064) when fluids were included, likely due to increased reflection from the water shield. This suggests that for the 85-pin model, neglecting fluids in the reactivity analysis decreases k_{eff} by about 1%.

The average lost particle fraction was 2.4 x 10^{-6}. Initially it was thought that lost particles were caused by the creation of non-orientable surfaces when converting sets of severely deformed quadrilateral mesh faces into triangles. After further research, it appears that particles became lost after collisions occurred inside overlaps. The investigation of lost particles from deformed space reactor simulations presents an opportunity for future work.

Space Reactor Summary An initialization routine was added to DAGMC that enabled radiation transport on deformed mesh geometry. Structural simulations were performed of the reactor impacting concrete at 100 m/s with impact angles of 0 and 45 degrees. NaK coolant and borated water shielding were included in a 0-degree structural simulation to examine the effect of fluids during impact. Deformed geometry was initialized for reactivity assessment without including fluids. The neutron multiplication factor increased 2.7% and 7.7% for the 0-degree impact with and without fluids in the structural simulation, respectively. The neutron multiplication factor increased

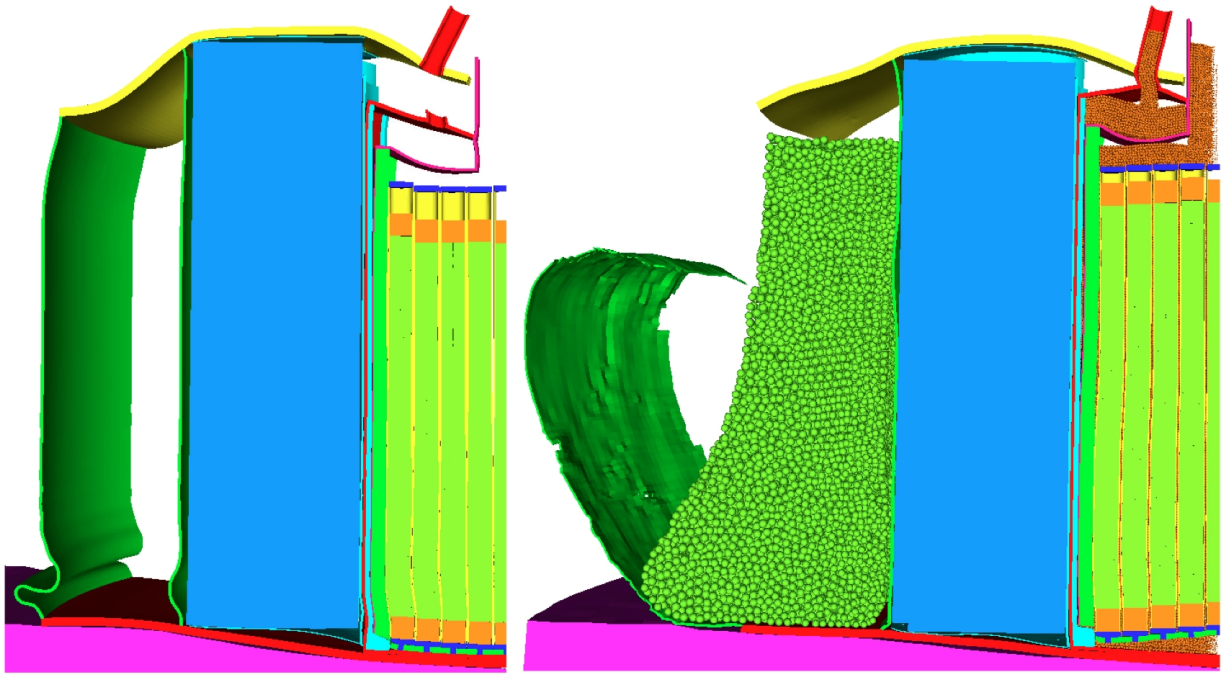


Figure 5.14: 0-degree impact at 1.5 ms without (left) and with (right) SPH elements. Image by Daniel Villa [77].

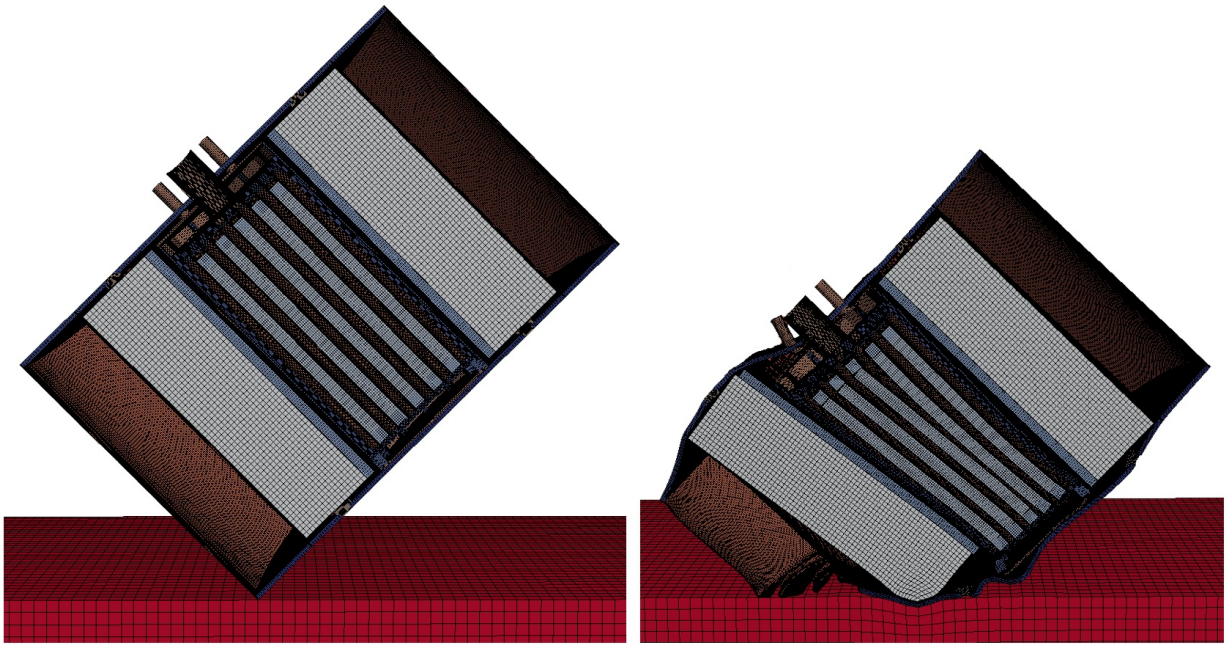


Figure 5.15: 45 degree impact at 0.0 ms (left) and 2.1 ms (right).

3.0% for the 45-degree simulation, which did not include fluids. In all cases the reactor remained subcritical due to the substantial amount of subcriticality at launch. The accuracy of these results depends on assumptions made in the structural and reactivity simulations. Material properties and fluid inclusion in the reactivity simulation are opportunities for improvement.

5.6 Conclusion

The overlap-tolerant tracking algorithm and point inclusion test has allowed models to be analyzed by DAGMC that were previously impossible. In addition to improving the workflow for CAD models, this advancement was used in the first known mesh-based reactivity analysis of deformed reactors. As discussed in Section 5.1, overlaps occur due to file format translation, imprecise draftsmanship, or deformation of mesh models. Overlap-tolerant tracking allows these models to be analyzed without investing days to imprint and merge surfaces. Merging remains an important acceleration technique, but surfaces which fail to automatically merge will be tolerated by the tracking algorithm. Avoiding the time consuming and tedious geometry repair associated with merging significantly increases the human efficiency of DAGMC analysis.

Chapter 6

Summary and Extensions

6.1 Summary

A series of improvements were added to the geometry capability of the DAGMC library. An algorithm was developed to seal together neighboring surfaces so that particles cannot leak from the discretized geometric model. The tracking algorithm was improved to be robust against numerical error and logical defects. Together, the tracking algorithm and point inclusion test are guaranteed to be consistent because they both rely on the same numerical ray tracing operation. Watertight faceting and robust tracking eliminated lost particles.

Next, improvements were made to increase the human efficiency of performing DAGMC analysis. The implicit complement allows users to avoid the tedious process of explicitly creating nonsolid space in CAD models. The boundary of an implicit volume is assembled by collecting surfaces adjacent to a single explicit volume. A tracking algorithm and point inclusion test were developed that tolerate geometric overlaps. Small overlaps occur due to file translation, imprecise design, and intentional deformation of finite element models. Manually removing overlaps in the CAD models is a tedious and time-consuming task, if not impossible. Decreasing analysis time by days, the implicit complement and overlap-tolerant tracking save human effort and enable new types of analyses.

The contribution of this work is a drastic reduction in manual CAD manipulation and a DAGMC analysis free of lost particles. Several of the improvements/logic elements used in this work are

Table 6.1: Improvements/logic elements added to DAGMC as part of this work.

Improvement	**Contribution**
Sealed Facet-Based Models	Avoid particle leakage between faceted surfaces
Plücker Ray-Triangle Test	Avoid particle leakage between facets
Edge/Point Post Processing	Avoid glancing intersections of an edge/point
Triangle Orientation	Avoid exit intersections with incorrect orientation
Negative Ray Length	Detect exit intersections behind numerical position
Previous Facets	Avoid oscillation between surface intersections
Point Inclusion Test (PIT)	Ensure consistency with `next_surface` function
Implicit Complement	Avoid explicit creation of nonsolid space in a CAD program
Overlap-Tolerance PIT	Allow analysis of CAD models with overlapping volumes

summarized in Table 6.1. The Plücker ray-triangle test, edge/point post processing, triangle orientation, negative ray length, previous facets, and point inclusion test were used in both the robust tracking algorithm and overlap-tolerant tracking algorithm.

Each improvement has been tested using CAD models from production calculations. The sealing tool was used to preprocess ten intricate CAD models. Results show that models which conform to the input assumptions of the algorithm can be sealed in seconds. DAGMC's existing test suite was used to validate and verify changes in the tracking algorithm. Testing of the tracking algorithm included over 300 computer-days searching for lost particles using the same set of 10 CAD models featured in tests of the sealing algorithm. In most cases, the implicit complement yields an identical output to that of explicitly-defined volumes. In the first known mesh-based reactivity analysis of deformed reactors, the overlap-tolerant tracking algorithm was used to predict the neutron multiplication factor of space reactors after launch failure.

6.2 Extensions

Extensions of this work would be particularly useful for additional modeling of deformed mesh geometry. These extensions include identifying disjoint regions of implicit complement, modeling fluids in deformed geometry, simplifying the mesh geometry workflow, improving quadrilateral to triangle conversion, and improving computational efficiency.

The implicit complement groups all undefined space into a single volume, perhaps containing disjoint regions. The current DAGMC methodology allows a single material to be assigned to the implicit complement, defining all nonsolid space. The implicit complement could be improved by splitting nonsolid space into disjoint volumes, if disjoint regions exist. This would permit separate material assignment for each disjoint region. In space reactor analysis, this could be used for NaK coolant, water shielding, and encompassing air, water, sand, or regolith.

Fluids are difficult to implicitly model because their enclosing structure can puncture during deformation. Punctures allow mixing between fluids, such as NaK coolant, water shielding, and outside air. Although coolant can be explicitly modeled in the structural analysis using hexahedral elements, they quickly become too deformed. Discrete spheres, or smoothed particle hydrodynamics (SPH) elements offer one approach of simulating fluids in a deformed structure. SPH elements have been used successfully to simulate fluids in the structural analysis, as demonstrated in the 0-degree collision. If the optical thickness of fluids can be approximated using spheres, SPH elements may prove acceptable for radiation transport simulations. Although adding spherical elements to MOAB is possible, new methods will need to be developed for efficient ray tracing. As currently implemented, all surfaces including spheres are faceted. This process will become inefficient if spheres are faceted into too many triangles. A better solution is likely to determine ray-sphere intersections analytically. MOAB's OBB trees can be constructed with spheres instead of triangles, leaving the current logic intact.

Faceted surfaces may cause systematic over- or underestimation of volume. For example, the volume of a faceted sphere will be less than the volume of the corresponding continuous sphere because facet points are located on the continuous, analytical surface. Therefore, triangles of the

faceted surface have a smaller radius than the corresponding continuous surface—causing an underestimation of volume. While small, this effect may be important, especially for fissionable material in a criticality calculation. Decreasing the facet tolerance causes the magnitude of the underestimation to decrease, but also decreases the efficiency of ray tracing due to additional triangles. An alternative is to improve the surface faceting algorithm of the solid modeling engine. Another approach is to use an analytical representation of curved surfaces for ray tracing.

The mesh geometry workflow assumes that models are meshed in CUBIT to ensure correspondence between solid and mesh entities. This does not represent a burden for models of medium complexity. However, for models of high complexity meshing occurs in subassemblies that are combined into a composite mesh outside of Cubit. A future DAGMC initialization routine that depends only upon the deformed mesh and boundary conditions specified inside a separate solid model would simplify the workflow.

Efficiency could be increased by selectively applying the overlap-tolerance logic only to self-intersecting volumes. Non-overlap logic could be used for all other volumes. This could be achieved automatically during preprocessing by searching each volume boundary for self-intersections. The overlap thickness could be measured and tagged on each volume in a similar process, allowing particle tracking to be optimized on a per-volume basis.

List of References

[1] E. E. Lewis and W. F. Miller, Jr., *Computational Methods of Neutron Transport*. American Nuclear Society, Inc., 1993.

[2] N. Metropolis and S. Ulam, "The Monte Carlo Method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[3] X-5 Monte Carlo Team, *MCNP—A General Monte Carlo N-Particle Transport Code, Version 5*. Los Alamos National Laboratory, revised 2/1/2008 ed., 2003. LA-UR-03-1987.

[4] I. Kawrakow, "Accurate Condensed History Monte Carlo Simulation of Electron Transport," *Medical Physics*, vol. 27, no. 3, pp. 485–498, 2000.

[5] K. V. Riper, *Moritz Geometry Tool*. White Rock Science, 2006.

[6] X. Liu, Y. Luo, and L. Tong, "Development and Application of MCNP Auto-Modeling Tool: MCAM 3.0," *Fusion Engineering and Design*, vol. 75-79, pp. 1275–1279, 2005. Proc. of the 23rd Symp. of Fusion Technology.

[7] Y. Li, L. Lu, A. Ding, H. Hu, Q. Zeng, S. Zheng, and Y. Wu, "Benchmarking of MCAM 4.0 with the ITER 3D Model," *Fusion Engineering and Design*, vol. 82, no. 15-24, pp. 2861–2866, 2007. Proc. of the 24th Symp. on Fusion Technology.

[8] Y. Wu, "CAD-Based Interface Programs for Fusion Neutron Transport Simulation," *Fusion Engineering and Design*, vol. 84, no. 7-11, pp. 1987–1992, 2009. Proc. of the 25th Sym. on Fusion Technology.

[9] FDS Team, *MCAM 4.7 Standard Edition User Manual*. Chinese Academy of Sciences, 2010.

[10] S. J. Manson, "TopAct: Automated Translation from CAD to Combinatorial Geometry," *Transactions of the American Nuclear Society*, vol. 91, no. 1, pp. 181–182, 2004.

[11] J. F. Latowski, R. P. Abbott, R. Laning, K. Morris, S. Reyes, and E. Williams, "Experience with Conversion of CAD to Monte Carlo Particle Transport Models," *Fusion Science and Technology*, vol. 52, pp. 807–811, 2007.

[12] D. E. Cullen, "TART 2002: A Coupled Neutron-Photon 3D, Combinatorial Geometry Time Dependent Monte Carlo Transport Code," tech. rep., Lawrence Livermore National Laboratory, June 2003. UCRL-ID-126455.

[13] H. Tsige-Tamirat, U. Fischer, A. Serikov, and S. Stickel, "Use of McCad for the Conversion of ITER CAD Data to MCNP Geometry," *Fusion Engineering and Design*, vol. 83, no. 10-12, pp. 1771–1773, 2008. Proc. of the 8th Int. Symp. of Fusion Nuclear Technology—ISFNT-8.

[14] A. Serikov, U. Fischer, R. Heidinger, K. Kleefeldt, L. Obholz, P. Spaeh, D. Strauss, and H. Tsige-Tamirat, "Neutronic Modeling Challenges for the ITER ECRH Launcher Shielding Design," *Nuclear Technology*, vol. 168, pp. 411–416, November 2009.

[15] U. Fischer, H. Iida, Y. Li, M. Loughlin, S. Sato, A. Serikov, H. Tsige-Tamirat, T. Tautges, P. Wilson, and Y. Wu, "Use of CAD Generated Geometry Data in Monte Carlo Transport Calculations for ITER," *Fusion Science and Technology*, vol. 56, pp. 702–709, August 2009.

[16] S. Sato, H. Iida, and T. Nishitani, "Development of CAD/MCNP Interface Program Prototype for Fusion Reactor Nuclear Analysis," *Fusion Engineering and Design*, vol. 81, pp. 2767–2772, 2006.

[17] K. B. Bekar and T. M. Evans, "MCNP-BRL: A Linkage Between MCNP and CAD Geometry," *Transactions of the American Nuclear Society*, vol. 101, pp. 623–626, 2009.

[18] L. A. Butler, E. W. Edwards, and D. L. Kregel, *BRL-CAD Tutorial Series: Volume 3—Principles of Effective Modeling*. Army Research Laboratory, 2003. ARL-SR-119.

[19] J. R. Anderson and E. W. Edwards, *BRL-CAD Tutorial Series: Volume 4—Converting Geometry Between BRL-CAD and Other Formats*. Army Research Laboratory, 2004. ARL-SR-121.

[20] P. Cowan, E. Shuttleworth, A. Bird, and A. Cooper, "The Launch of MCBEND 10," in *Proc. of the 10th Int. Conf. on Radiation Shielding and the 13th Topical Meeting on Radiation Protection and Shielding*, 2004.

[21] M. J. Armishaw and A. J. Cooper, "Current Status and Future Direction of the MONK Software Package," in *Proc. of the 8th Int. Conf. on Nuclear Criticality Safety*, vol. 2, 2007.

[22] K. Searson, F. Fleurot, and A. Cooper, "OiNC: A Comprehensive CAD Import and Tracking System for Monte Carlo Radiation Transport Calculations," *ANS Radiation Protection and Shielding Division 2010 Topical Meeting Book of Abstracts*, 2010.

[23] M. Wang, *CAD Based Monte Carlo Method: Algorithms for Geometric Evaluation in Support of Monte Carlo Radiation Transport Calculation*. Ph.D. Dissertation, University of Wisconsin-Madison, Fusion Technology Institute, August 2006. UWFDM-1353.

[24] T. J. Tautges, "The Common Geometry Module (CGM): a Generic, Extensible Geometry Interface," *Engineering with Computers*, vol. 17, no. 3, pp. 299–314, 2001.

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

[26] Lawrence Livermore National Laboratory, "VisIt Visualization Tool." `http://visit.llnl.gov`, 2010.

[27] B. M. Smith, P. P. H. Wilson, and M. E. Sawan, "Three Dimensional Neutronics Analysis of the ITER First Wall/Shield Module 13," in *Proc. of the 22nd IEEE/NPSS Symp. on Fusion Engineering—SOFE2007*, IEEE Nuclear & Plasma Sciences Society, 2007.

[28] B. M. Smith, P. P. H. Wilson, and T. D. Bohm, "Source Profile Analysis for the ITER First Wall/Shield Module 13," *Fusion Science and Technology*, vol. 56, pp. 57–62, July 2009.

[29] M. Sawan, P. Wilson, T. Tautges, L. El-Guebaly, D. Henderson, T. Bohm, E. Marriott, B. Kiedrowski, B. Smith, A. Ibrahim, and R. Slaybaugh, "Application of CAD-Neutronics Coupling to Geometrically Complex Fusion Systems," in *Proc. of 23rd Symp. on Fusion Engineering—SOFE2009*, 2009.

[30] B. M. Smith and P. P. H. Wilson, "Modeling Impact-Induced Reactivity Changes Using DAG-MCNP," in *Proc. of Nuclear and Emerging Technologies for Space—NETS2011*, American Nuclear Society, 2011.

[31] Sandia National Laboratories, "CUBIT Geometry and Mesh Generation Toolkit." `http://cubit.sandia.gov`, 2010.

[32] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: A Hierarchical Structure for Rapid Interference Detection," in *SIGGRAPH '96*, pp. 171–180, 1996.

[33] T. J. Tautges, P. P. Wilson, J. A. Kraftcheck, B. M. Smith, and D. L. Henderson, "Acceleration Techniques for Direct Use of CAD-Based Geometries in Monte Carlo Radiation Transport," in *Proc. Int. Conf. on Mathematics, Computational Methods, and Reactor Physics*, 2009.

[34] 3D Systems Inc., "Stereolithography Interface Specification," 1989.

[35] E. Béchet, J. C. Cuilliere, and F. Trochu, "Generation of a Finite Element MESH from Stereolithography (STL) Files," *Computer-Aided Design*, vol. 34, no. 1, pp. 1–17, 2002.

[36] D. Rypl and Z. Bittnar, "Generation of Computational Surface Meshes of STL Models," *Journal of Computational and Applied Mathematics*, vol. 192, no. 1, pp. 148–151, 2006. Special Issue on Computational and Mathematical Methods in Science and Engineering—CMMSE2004.

[37] Spatial Corporation, "Spatial Product Documentation." `http://doc.spatial.com/index.php/Refinements#Surface_Tolerance`, 2009.

[38] Open CASCADE, "Open CASCADE technology, 3D Modeling & Numerical Simulation." `http://www.opencascade.org`, 2010.

[39] S. Bischoff and L. Kobbelt, "Structure Preserving CAD Model Repair," *Eurographics*, vol. 24, no. 3, pp. 527–536, 2005.

[40] T. Ju, "Fixing Geometric Errors on Polygonal Models: A Survey," *Journal of Computer Science and Technology*, vol. 24, no. 1, pp. 19–29, 2009.

[41] H. Edelsbrunner and E. Mücke, "Three-Dimensional Alpha Shapes," *ACM Transactions on Graphics*, vol. 13, no. 1, pp. 43–72, 1994.

[42] N. Amenta, M. Bern, and M. Kamvysselis, "A New Voronoi-Based Surface Reconstruction Algorithm," in *Proc. of the 25th Annual Conf. on Computer Graphics and Interactive Techniques*, pp. 415–421, 1998.

[43] T. M. Murali and T. A. Funkhouser, "Consistent Solid and Boundary Representations from Arbitrary Polygonal Data," in *Proc. of the 1997 Symp. on Interactive 3D graphics—I3D1997*, pp. 155–ff., 1997.

[44] T. Ju, "Robust Repair of Polygonal Models," in *SIGGRAPH '04*, pp. 888–895, 2004.

[45] A. A. G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," *ACM Computing Surveys*, vol. 12, no. 4, pp. 437–464, 1980.

[46] J. H. Bohn and M. J. Wozny, "A Topology-Based Approach for Shell Closure," *Geometric Modeling for Product Realization*, pp. 297–319, 1993.

[47] M. Mäntylä, "Boolean Operations of 2-Manifolds through Vertex Neighborhood Classification," *ACM Transactions on Graphics*, vol. 5, no. 1, pp. 1–29, 1986.

[48] G. Barequet and M. Sharir, "Filling Gaps in the Boundary of a Polyhedron," *Computer Aided Geometric Design*, vol. 12, pp. 207–229, 1995.

[49] G. Barequet and S. Kumar, "Repairing CAD Models," in *Proc. of the 8th Conf. on Visualization '97—VIS1997*, 1997.

[50] G. T. Klincsek, "Minimal Triangulations of Polygonal Domains," *Combinatorica*, vol. 79, 1980.

[51] X. Sheng and I. Meier, "Generating Topological Structures for Surface Models," *IEEE Computer Graphics and Applications*, vol. 15, no. 6, pp. 35–41, 1995.

[52] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn, "Converting Sets of Polygons to Manifold Surfaces by Cutting and Stitching," in *Proc. of the Conf. on Visualization '98—VIS1998*, pp. 383–390, 1998.

[53] P. Borodin, M. Novotni, and R. Klein, "Progressive Gap Closing for Mesh Repairing," *Advances in Modelling, Animation and Rendering*, pp. 201–21, 2002.

[54] P. S. Patel, D. L. Marcum, and M. G. Remotigue, "Stitching and Filling: Creating Conformal Faceted Geometry," in *Proc. of the 14th Int. Meshing Roundtable*, International Meshing Roundtable, 2005.

[55] F. Kahlesz, A. Balázs, and R. Klein, "Multiresolution Rendering by Sewing Trimmed NURBS surfaces," in *Proc. of the 7th ACM Symp. on Solid Modeling and Applications—SMA2002*, pp. 281–288, 2002.

[56] O. Busaryev, T. K. Dey, and J. A. Levine, "Repairing and Meshing Imperfect Shapes with Delaunay Refinement," in *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling—SPM2009*, pp. 25–33, 2009.

[57] C. S. Chong, A. S. Kumar, and H. P. Lee, "Automatic Mesh-Healing Technique for Model Repair and Finite Element Model Generation," *Finite Elements in Analysis and Design*, vol. 43, pp. 1109–1119, 2007.

[58] J. Ruppert, "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation," *Journal of Algorithms*, vol. 18, no. 3, pp. 548–594, 1995.

[59] J. O'Rourke, *Computational Geometry in C*. Cambridge University Press, 2nd ed., 1998.

[60] B. M. Smith, T. J. Tautges, and P. P. H. Wilson, "Sealing Faceted Surfaces to Achieve Watertight CAD Models," in *Proc. of the 19th Int. Meshing Roundtable*, International Meshing Roundtable, 2010.

[61] A. F. Bielajew, "HOWFAR and HOWNEAR: Geometry Modeling for Monte Carlo Particle Transport," tech. rep., National Research Council of Canada, 1995. PIRS-0341.

[62] D. C. Williams, "GEANT4 Geometry Tracking Questions." `http://scipp.ucsc.edu/~davidw/geant4/geo.question.html`, March 2010.

[63] L. M. Popescu, "A Geometry Modeling System for Ray Tracing or Particle Transport Monte Carlo Simulation," *Computer Physics Communications*, vol. 150, no. 1, pp. 21–30, 2003.

[64] T. Möller and B. Trumbore, "Fast, Minimum Storage Ray/Triangle Intersection," in *SIGGRAPH '05*, p. 7, 2005.

[65] N. Platis and T. Theoharis, "Fast Ray-Tetrahedron Intersection using Plücker Coordinates," *Journal of Graphics Tools*, vol. 8, no. 4, pp. 37–48, 2003.

[66] R. J. Segura and F. R. Feito, "Algorithms to Test Ray-Triangle Intersection Comparative Study," in *Proc. of WSCG 2001 Conf.*, 2001.

[67] F. Duguet and G. Drettakis, "Robust Epsilon Visibility," in *Proc. of the 29th Annual Conf. on Computer Graphics and Interactive Techniques—SIGGRAPH2002*, pp. 567–575, 2002.

[68] Free Software Foundation, "GNU Multiple Precision Arithmetic Library." `http://gmplib.org`, 2010.

[69] J. Lane, B. Magedson, and M. Rarick, "An Efficient Point in Polyhedron Algorithm," *Computer Vision, Graphics, and Image Processing*, vol. 26, pp. 118–125, 1984.

[70] S. Nordbeck and B. Rydstedt, "Computer Cartography Point-In-Polygon Programs," *BIT Numerical Mathematics*, vol. 7, pp. 39–64, 1967.

[71] W. P. Horn and D. L. Taylor, "A Theorem to Determine the Spatial Containment of a Point in a Planar Polyhedron," *Computer Vision, Graphics, and Image Processing*, vol. 45, pp. 106–116, 1988.

[72] A. Khamayseh and A. Kuprat, "Deterministic Point Inclusion Methods for Computational Applications with Complex Geometry," *Computational Science and Discovery*, vol. 1, 2008.

[73] G. Yegin, "A New Approach to Geometry Modeling for Monte Carlo Particle Transport: An Application to the EGS Code System," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 211, no. 3, pp. 331–338, 2003.

[74] K. B. Bekar, J. C. Wagner, and T. M. Evans, "Testing MCNP-BRL for Nuclear Vulnerability Assessments with the M60A1 Tank," *ANS Radiation Protection and Shielding Division 2010 Topical Meeting Book of Abstracts*, 2010.

[75] T. F. Marcille, D. D. Dixon, G. A. Fischer, S. P. Doherty, D. I. Poston, and R. J. Kapernick, "Design of a Low Power, Fast-Spectrum, Liquid-Metal Cooled Surface Reactor System," in *Proc. of the Space Technology and Applications Int. Forum—STAIF2006*, pp. 319–326, 2006.

[76] J. Smith, D. Villa, B. Smith, R. Radel, T. Radel, T. Tallman, R. Lipinski, and P. Wilson, "Methods for Modeling Impact-Induced Reactivity Changes in Small Reactors," tech. rep., Sandia National Laboratories, 2010. SAND2010-6412.

[77] D. L. Villa, T. N. Tallman, and J. A. Smith, "Challenges in Structural Analysis for Deformed Nuclear Reactivity Assessments," in *Proc. of Nuclear and Emerging Technologies for Space 2011—NETS2011*, American Nuclear Society, 2011.

[78] L. A. Schoof and V. R. Yarberry, "EXODUS II: A Finite Element Data Model," tech. rep., Sandia National Laboratories Report, 1994. SAND1992-2137.

[79] SIERRA Solid Mechanics Team, *Presto 4.16 User's Guide*. Sandia National Laboratories, 2010. SAND2010-3112.