Implementation of a tetrahedral mesh phantom geometry library for EGSnrc

by

Maxwell Orok

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

Department of Mechanical Engineering
University of Ottawa

Implementation of a tetrahedral mesh phantom geometry library for EGSnrc

Maxwell Orok
Master of Applied Science

Department of Mechanical Engineering
University of Ottawa
2022

Abstract

The implementation of a general-purpose tetrahedral mesh phantom geometry library for the Monte Carlo radiation transport code EGSnrc is described. Recently, tetrahedral mesh geometries have been proposed as standard reference phantoms to advance the state of the art over rectilinear voxel phantoms. Prior to this work, EGSnrc already supported voxelized geometries, but not tetrahedral meshes. Other major radiation transport codes such as MCNP6, Geant4, and PHITS, are also capable of simulating the interaction of ionizing radiation with tetrahedral mesh phantoms. Tetrahedral mesh phantoms have a number of advantages over voxel phantoms including improved modelling fidelity and locally varying element resolution. In addition, CAD geometries can be converted into meshes, which can then be directly used in simulations. In this work, an EGSnrc tetrahedral mesh geometry library called `EGS_Mesh` is implemented. The implementation uses fast computational geometry algorithms from the literature and is accelerated using an octree spatial partitioning scheme. For a preliminary verification, results obtained using `EGS_Mesh` are compared to classical EGSnrc geometries and theoretical results (including a Fano test) and found to match within 0.1%. To demonstrate the capability of `EGS_Mesh` to simulate transport in complex mesh phantoms from the literature, results using the ICRP 145 reference human phantoms are compared to published results obtained using Geant4. The comparison has found agreement mostly within 5% of the Geant4 results, but with some differences up to 10%.

To Mikaela

Acknowledgements

This thesis is a formalization and extension of internal research conducted by Mevex Corporation from 2017 on. Mevex is a Canadian sterilization equipment manufacturer interested in simulating complex CAD geometries using EGSnrc. Dave Macrillo and Matt Ronan implemented the initial proof-of-concept EGSnrc tetrahedral mesh geometry library. The core development team at Mevex was composed of Dave Macrillo, Matt Ronan, Nigel Vezeau, Lou Thompson, and myself. In 2019, the project's intellectual property and software were donated to the National Research Council Canada to benefit the wider simulation community. I have completely rewritten the tetrahedral mesh geometry implementation presented in this thesis based on experience and lessons learned from the initial version.

Thank you to my supervisor, Professor James McDonald, for supporting me all throughout my degree. I felt free to investigate whatever interested me, but I especially appreciate James's focus on scope and iterative progress (as well as his deep knowledge of C++). I feel very lucky to have been part of his research group.

Thank you also to Dr. Frédéric Tessier and Dr. Reid Townson of the National Research Council Canada for their constant encouragement and advice as well as for their comments on the thesis manuscript. Dr. Tessier and Dr. Townson were subject to a bombardment of emails, messages and meetings regarding all things EGSnrc for many months, and I am truly thankful for their patient, invaluable counsel.

Thank you to Bangho Shin and Professor Chan Hyeong Kim of Hanyang University, for their advice and knowledge-sharing regarding the ICRP 145 mesh phantoms.

Thank you to Maria Zankl of Helmholtz Munich for her very helpful answers to my questions about the EGSnrc simulations in the ICRP 116 report.

Thank you to Professor Emily Heath and Professor Malcolm McEwen of Carleton University for inviting me to present a preliminary version of this work at the Ottawa Medical Physics Institute.

Thank you to the Compute Canada team and infrastructure, especially the supercomputer Graham. This work was partially funded by an Ontario Graduate Scholarship.

Finally, thank you to my family for always believing in me.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

Knowledge of how ionizing radiation interacts with matter is crucial for many fields. Medical physicists use ionizing radiation to treat tumours and to obtain valuable patient imaging data. The radiation processing industry uses ionizing radiation to sterilize products without even having to unpack them from their crates. In the field, dosimeters are used to obtain measurements to determine dose. But dosimeters can be awkward, expensive, unreliable, and usually, only a fraction of the area of interest can be measured. Radiation transport simulations allow academia and industry to accurately model the effects of ionizing radiation for a wide range of applications.

The gold standard for simulating radiation transport is the Monte Carlo method [1]. Monte Carlo is a statistical method that uses empirical and theoretical particle interaction probability distributions to approximate the true solution. There are many different Monte Carlo radiation transport codes, each with various features. The code used in this work is EGSnrc, which is maintained by the National Research Council Canada. EGSnrc is widely considered to be one of the most accurate and efficient codes for electron and photon transport and so is often used for medical physics applications [2, 3].

This thesis is focused on the development of a tetrahedral mesh geometry library for EGSnrc. Medical physics and related fields have historically used rectilinear voxel models for simulating the interaction of ionizing radiation with complex geometries such as human anatomy. However, there is growing interest in using tetrahedral mesh models for certain kinds of simulations (such as simulations involving CAD geometries). Tetrahedral meshes are commonly used in many engineering domains because of their ability to accurately model a wide range of geometries using a large number of individual tetrahedrons. Unlike voxel models, which usually have a single global resolution, tetrahedral meshes can have varying resolution. This means complex sub-geometries can have a higher local resolution without greatly increasing the total number of regions (and thereby the total simulation time). Some background for this thesis follows, starting with a closer look at EGSnrc.

1.1 EGSnrc and egs++

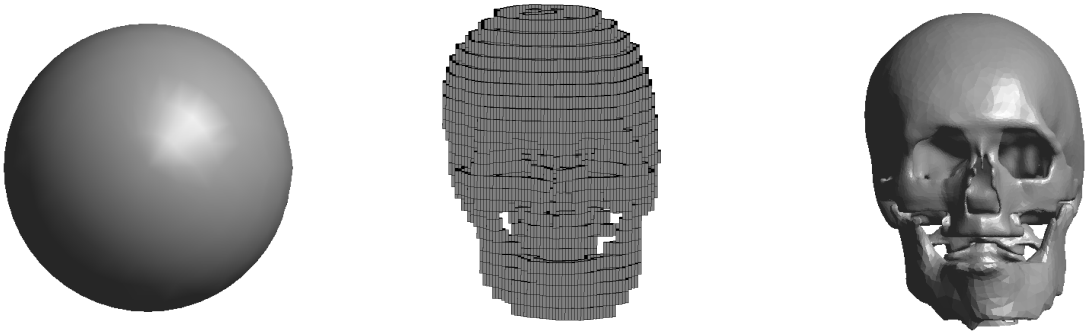
The EGSnrc radiation transport code is a general-purpose software package used to simulate the interaction of photons, electrons and positrons with matter [4]. EGSnrc is one of the major offshoots of the Electron-Gamma-Shower (EGS) code [4]. Originally conceived at the Stanford Linear Accelerator Center in the 1970s, EGS became widely used for radiation transport simulations and is considered a standard reference code [5]. More recently, the egs++ library [6] has extended the capabilities of EGSnrc by allowing users to define their own geometry classes (among other things) in C++. A number of primitive geometries are implemented directly (spheres, cubes, etc.), but more complex geometries are also supported through means of combination and transformation. The open-ended design of egs++ has facilitated the development of many useful geometry packages by the simulation community. Some of these user-defined geometries end up being incorporated into egs++ if they are deemed useful for a wider audience. This thesis is focused on the development of a tetrahedral mesh geometry library for EGSnrc called `EGS_Mesh`.

1.2 Tetrahedral mesh geometry

Because of their modelling fidelity and other useful properties, tetrahedral meshes are gaining popularity in medical physics and related fields for modelling complex geometries, including human anatomy [7]. Tetrahedral meshes can be obtained automatically from finite-element mesh generators. The import of geometries designed using computer-aided design (CAD) programs has long been a desirable feature of radiation transport codes [8] and is one of the original motivations for this work. After meshing a CAD geometry, it can then be directly used in transport simulations. However, creating meshes for complex geometries sometimes requires manual intervention and tuning. Depending on the geometry, this process can take a long time. For instance, the adult male and female models used for benchmarking by this work took years to develop [7], but they are unusually detailed and complex models intended for use as reference data. Other codes that allow for simulating radiation transport in tetrahedral mesh phantoms include MCNP6 (*Monte Carlo N-Particle*) [9], Geant4 (*GEometry ANd Tracking*) [10], and PHITS (*Particle and Heavy Ion Transport code System*) [11].

1.3 Simulation phantoms

In the context of radiation transport, the term *phantom* means a model of the simulation's subject. So, a computational human phantom refers to a model of the human body used for simulating the interaction of ionizing radiation with human anatomy. The development of computational human phantoms for radiation transport has been an active area of research



(a) Stylized skull (b) Voxelized skull [13] (c) Tetrahedralized skull [7]

Figure 1.1: Examples of stylized, voxelized and tetrahedralized phantoms.

for more than 50 years [12]. As safety regulations become more stringent and computing power grows, computational phantoms become more sophisticated. Skull phantoms of increasing complexity are shown in Figure 1.1. The first phantom, Figure 1.1a, is a sphere. Early phantoms like this were composed of basic shapes. As computing power and memory were limited, phantoms that could be described using simple equations were often the only choice [12]. The second phantom, Figure 1.1b, uses rectilinear voxel elements. Voxel phantoms emerged in the 1980s and are still a standard format used today [14]. They offer better modelling capability than stylized phantoms in most cases but requiring a uniform voxel resolution means that thin geometries can have defects (e.g. the skin of one standard voxel phantom [13] has holes due to resolution limitations) [14]. The last phantom, Figure 1.1c, represents the current state of the art: a tetrahedral mesh phantom [7]. The tetrahedral mesh geometry implemented in this work is able to model complex modern tetrahedral mesh phantoms and has recovered results comparable to those obtained using other major transport codes.

1.4 Thesis overview

The main contributions of this thesis are:

- the development and implementation of a general-purpose tetrahedral mesh geometry in EGSnrc,
- a preliminary verification of the implementation, and
- a comparison with benchmark results obtained using Geant4.

An overview of radiation transport using EGSnrc is given in Chapter 2. A review of existing tetrahedral mesh geometry libraries implemented in other Monte Carlo transport

codes is conducted in Chapter 3. Chapter 3 also surveys the history of computational human phantoms. Chapter 4 introduces the tetrahedral mesh geometry, `EGS_Mesh`, including all of the necessary geometry algorithms required for its implementation. First, a naive version is constructed, and then the final octree-accelerated version is presented with performance results. Results of the preliminary verification work, including comparisons to existing egs++ geometries, a Fano test, and results obtained for the ICRP 145 adult reference mesh phantoms are detailed in Chapter 5. Future work and conclusions are given in Chapter 6.

Chapter 2

Radiation transport using EGSnrc

EGSnrc, maintained by the National Research Council Canada (NRC), is a complete overhaul of the original Electron-Gamma-Shower (EGS) program. The EGS family of codes are used to simulate the interaction of ionizing photons, electrons and positrons with matter [4]. Parts of EGS are older than C, let alone C++ [15]. EGS has been widely used by the radiation transport community ever since its initial release.

The first version of EGS was developed at the Stanford Linear Accelerator Center (SLAC) in the 1970s [15]. It started as a revision to a program known as SHOWER, written by the visiting German scholar H. H. Nagel [15, 16]. As fate would have it, Nagel never met the next generation of SHOWER developers [15]. His program was apparently "recovered from a trash receptacle" by Walter Nelson [17] and then "quickly accepted and put to immediate use" at SLAC [15]. But SHOWER was not powerful enough for the SLAC team, and Nelson, his collaborator Richard Ford, and others began implementing extensive changes [15]. Their goal was "to develop the Nagel code into a general, multi-media, modularized, and versatile form" [15]. And so, SHOWER became Electron-Gamma-Shower.

The development of EGS at SLAC continued throughout the seventies culminating in EGS3 [15]. Next, the combined efforts of SLAC, the NRC, and the National Laboratory for High Energy Physics (KEK) in Japan led to the next major release, EGS4 [18]. But it was pioneering work conducted at the NRC by Kawrakow, Bielajew, Rogers and others that resulted in the version still used today, EGSnrc [4].

EGSnrc improved upon its predecessors through the use of a new electron multiple-scattering algorithm [19] among other advances [20, 21]. Taken together, these features give EGSnrc its nominal 0.1% accuracy (with respect to its own cross-sections), a massive improvement over the prior state of the art [4]. Most of the low-level EGSnrc physics routines are written in MORTRAN3, a macro language for FORTRAN. Over the years, EGSnrc has also introduced the use of C++ for new features. The parts of EGSnrc written in C++ are collectively known as egs++ [6]. Nowadays, EGSnrc development efforts usually build on top of the hardened MORTRAN and egs++ core. New developments include geometry libraries

(e.g. `EGS_Lattice` [22] for repeated cell geometries) and specialized applications such as `egs_brachy` [23] for brachytherapy dose calculations.

2.1 Physics overview

EGSnrc simulates the interaction of photons, electrons and positrons with matter [4]. In EGSnrc, photons interact with matter through pair production, Compton scattering, photoelectric absorption, and Rayleigh scattering, while electrons interact through bremsstrahlung, positron annihilation, elastic scattering and inelastic scattering [4]. EGSnrc and other Monte Carlo codes aim to solve the Boltzmann transport equation, which governs particle fluence. Particle fluence, ϕ, is defined by the number of particles, $\mathrm{d}n$, flowing through an infinitesimal sphere with cross section $\mathrm{d}A$ [24]

$$\phi = \frac{\mathrm{d}n}{\mathrm{d}A}. \tag{2.1}$$

Informally, the transport equation for the time total derivative of ϕ, can be in written in terms of a source term, S, particle velocity magnitude, v, and a collision term, $I[\phi]$, as [4]

$$\frac{\mathrm{d}\phi}{\mathrm{d}t} = S + vI[\phi]. \tag{2.2}$$

In other words, the change in particle fluence over time depends on particles generated by sources and particles generated or destroyed as part of the collision term. The transport equation for a particle in terms of position, $\vec{x}$, angular direction, $\vec{\Omega}$, energy, E, and time, t, is

$$\frac{\mathrm{d}\phi(\vec{x},\vec{\Omega},E,t)}{\mathrm{d}t} = S(\vec{x},\vec{\Omega},E,t) + v(E)\cdot I[\phi(\vec{x},\vec{\Omega},E,t)]. \tag{2.3}$$

The collision term contains most of the equation's complexity, since it represents all of the potential atomic interactions. Because of the collision term, this is an integro-differential equation which is intractable analytically. It is also very computationally expensive to solve using deterministic numerical methods without approximations because of the many degrees of freedom.

Monte Carlo offers an alternative approach, by solving the equation stochastically. Using knowledge of particle interaction probabilities in different materials (i.e. cross-section data), randomly generated particles are simulated one at a time to build up a statistical average of the solution [4]. Particles take discrete steps between interactions in the simulation geometry. Each particle is simulated until it either leaves the geometry or all of its energy has been deposited. Secondary (tertiary, etc.) particles that are produced by the initial particle's interactions are themselves tracked in the same way. Particle histories are assumed to be independent, meaning that separate histories do not influence each other [4]. Photon transport is fairly straightforward to implement efficiently, as photons tend to have a relatively large

mean free path between interactions [25]. Unfortunately, charged particle transport is not as inherently efficient, as the mean free path tends to be much smaller because charged particles continually interact with the media's charged atomic components. As electrons (and other charged particles) move through matter, there are usually a large number of collisions that have a small effect on the particle and a small number of collisions that have a large effect on the particle [25]. Being able to "condense" the less important collisions into fewer explicitly simulated particle steps allows for massive improvements in simulation performance [25]. This is the idea behind Berger's foundational condensed history technique [25], which is used by EGSnrc to simulate charged particles [4]. Berger divides condensed history schemes into two classes. Class I schemes group all kinds of collisions together, relying on precalculated path length data [25]. Though simple to implement, this can be a severe approximation of reality. In contrast, Class II schemes only group less important collisions together while simulating "catastrophic" particle collisions separately (i.e. when the particle's energy loss from a collision breaches some threshold value) [25]. This is the technique used by EGSnrc [19]. As Berger notes, Class II condensed history schemes tend to be more accurate, but unfortunately "computations become more laborious" [25]. Although the condensed history technique was originally proposed without a strong theoretical foundation, it was widely adopted by various Monte Carlo codes because of its demonstrated agreement with experimental data [26]. Later, it was proven that the condensed history technique does indeed solve the Boltzmann equation as the condensed history step size approaches zero [26]. Approximation errors from improper condensed history step sizes are known as step-size artefacts [27]. The EGSnrc electron-step algorithm, PRESTA-II [19], is notable for its relative lack of step-size artefacts compared to its predecessors [4].

2.2 The Monte Carlo method

Monte Carlo methods are an extremely general class of numerical integration methods that work by randomly sampling points in the integration domain. Formalized by the Los Alamos brain trust in the 1940s [28] to aid the development of atomic weapons [29], Monte Carlo is the premier choice for accuracy and reliability in radiation transport. The world's major general-purpose radiation transport codes (EGSnrc, Geant, MCNP, etc.) all employ Monte Carlo to perform their calculations.

Monte Carlo integration of a continuous function, f, works by estimating the mean function value, $\bar{f}$. Given the mean value, the integral of f over the domain $[a, b]$ can be then found using the average value theorem,

$$\bar{f} = \frac{1}{b-a} \int_a^b f(x) \mathrm{d}x. \tag{2.4}$$

The Monte Carlo estimate of the mean function value, $\bar{X}$, is found by taking the average of N samples,

$$\bar{X} = \frac{1}{N} \sum_{n=1}^{N} f(x_n) \approx \bar{f}. \tag{2.5}$$

As the number of samples (for radiation transport: particle histories) increases, results become more accurate. In the limit of $N \to \infty$, Monte Carlo exactly solves the integral. The statistical error of Monte Carlo decreases at a rate of $\frac{1}{\sqrt{N}}$, regardless of the number of dimensions [3]. This is in contrast to other methods such as numerical quadrature, where the convergence rate is a function of the number of dimensions [3]. As the number of dimensions grows, Monte Carlo becomes the most efficient numerical integration method (formally, for integrals of more than four dimensions [3]). Since electron-photon-positron transport at a single moment in time has 18 degrees of freedom, Monte Carlo is the most appropriate numerical integration method.

For a given Monte Carlo calculation, the simulation efficiency, ε, is defined by

$$\varepsilon = \frac{1}{T\sigma^2}, \tag{2.6}$$

with calculation time, T, and estimated variance, σ^2, with units of s^{-1} [30]. So, efficiency improvements can either come from reducing the required calculation time or by reducing the variance of the results. There exist a number of specialized techniques to increase Monte Carlo simulation efficiency, while retaining high-quality results, which are known as variance reduction techniques [31]. Some techniques make assumptions about the problem at hand, meaning they are only valid for a subset of possible simulation inputs [32] (e.g. only being applicable for either photons or electrons). One example of a variance reduction technique implemented in EGSnrc is electron range rejection [4]. Given an electron's energy, EGSnrc can determine the maximum distance the electron can travel in a medium before falling below the energy cutoff. If the distance to a geometry boundary is larger than the maximum electron range, the electron's energy is deposited in the current region and transport is terminated without having to explicitly simulate the electron's final steps [4]. However, this can lead to inaccuracies in certain cases, as the electron, before being cut off, could have emitted a bremsstrahlung photon capable of exiting the current region [4]. Whether this is an acceptable approximation depends on the simulation, meaning some care is required to apply range rejection without affecting results (as is usually the case for most variance reduction techniques).

To perform a Monte Carlo calculation, a source of random numbers is required. For practical purposes, pseudorandom number generators, implemented in software (as opposed to truly random numbers), are employed to obtain reproducible results [33]. Pseudorandom number generators for Monte Carlo are typically held to stringent standards. History shows

the risks of placing too much faith in widely-used generators. In the past, "high-quality" random number generators have been found to cause "subtle, but dramatic, systematic errors for some algorithms" [34]. In a recent review, James and Moneta list only the generators RANLUX [35, 36], RANLUX++ [37] and MIXMAX [38] as "suitable for use in the most demanding Monte Carlo simulations" [33]. Of the three, RANLUX++ was considered "hard to beat" [33] with the following caveat: since the generator was implemented directly in assembly, portability may be an issue. This restriction has recently been lifted by Hahnfeld and Moneta's pure C++ implementation [39], which nearly matches the speed of Sibidanov's original assembly, thanks to careful optimization efforts. Both RANLUX and a simpler generator called RANMAR are available for EGSnrc applications written in MORTRAN, but only the RANMAR generator is currently available for egs++ applications, which could in theory give lower-quality results than the higher-quality generators endorsed by [33].

The trust placed in the Monte Carlo method is readily justified by the method's power and accuracy especially compared to results obtained by alternative methods. Nevertheless, Rogers and Bielajew warn that "Too often, systematic errors are not considered and one tends to lend too much credence to Monte Carlo results" [40]. They note that Monte Carlo codes are not free of programming errors (found in virtually all software), modelling inaccuracies (arising from the limitations of the chosen physics model) and roundoff or truncation errors inherent to floating-point math routines [40]. Since proofs of program correctness are not realistic, statistically equivalent results obtained using multiple Monte Carlo codes are often considered the highest standard for simulation results (assuming a lack of experimental data).

2.2.1 Uncertainty estimates

As with any Monte Carlo method, quantities such as region energy deposition calculated by EGSnrc have an associated uncertainty value. EGSnrc uses the history-by-history technique [41] for estimating uncertainties. Generally speaking, a history is defined as a statistically independent event [41]. For EGSnrc, a history is "all particle tracks associated with one initial particle" [41]. So, if a photon, emitted from a particle source, generates an electron through photoelectric absorption, that electron belongs to the same history as the initial photon. For a per-history calculated quantity, X_i, the uncertainty value of the mean, $\sigma_{\bar{X}}$, given N total histories is [41]

$$\sigma_{\bar{X}} = \sqrt{\frac{1}{N-1}\left(\frac{\sum_{i=1}^{N} X_i^2}{N} - \left(\frac{\sum_{i=1}^{N} X_i}{N}\right)^2\right)} \quad . \tag{2.7}$$

For example, the quantity of interest, X, can be energy deposition in a certain medium. Then, the reported value will be the sum of all deposited energies, divided by particle fluence to obtain a mean value (energy deposition per particle fluence). For each quantity, EGSnrc

Table 2.1: Classifying relative uncertainties. Adapted from Table 7 of [42].

Relative uncertainty	Meaning
$> 50\%$	Meaningless
20% to 50%	Factor of a few
$< 10\%$	Generally reliable
$< 5\%$	Reliable

keeps a running total of both X and X^2 for use in uncertainty calculations [41].

Time and resources permitting, it is desirable to obtain as low uncertainty values as possible, although different applications have different uncertainty requirements. Roughly speaking, EGSnrc is accurate up to 0.1% with respect to its own cross-section data for multiple-scattering calculations [4]. So running simulations past 0.1% uncertainty is usually a waste of time for the practitioner. If a quantity's uncertainty estimate is too large, the value is not reliable. It is important to note that the uncertainty estimate is also a statistical value with an associated uncertainty [42]. This means that quantities with large relative uncertainties can exhibit much greater errors than implied by just the relative uncertainty alone [42]. Table 2.1, adapted from the MCNP documentation [42], can be used as a rough guide. With this brief introduction to EGSnrc and the Monte Carlo method complete, the relevant literature for the development of a tetrahedral mesh geometry library for EGSnrc is reviewed in Chapter 3.

Chapter 3

Literature review

The `EGS_Mesh` implementation mainly relies on experience from other radiation transport codes that support tetrahedral mesh phantoms. First, EGSnrc and the wider ecosystem of Monte Carlo transport codes, along with some alternative radiation transport solution methods, are considered in Section 3.1. Then, the existing tetrahedral mesh geometry libraries of MCNP6, Geant4, and PHITS are reviewed in Section 3.2. And finally, a review of computational human phantoms (including the ICRP 145 tetrahedral mesh human phantoms [7] used to benchmark `EGS_Mesh`) is conducted in Section 3.3. As is shown, modern computational human phantoms can be extremely complex, making them a good stress test for transport codes.

3.1 Monte Carlo radiation transport codes

Along with EGSnrc, other Monte Carlo radiation transport codes include MCNP [9], Geant4 [10], PHITS [11], FLUKA [43], and PENELOPE [44]. Of these, the most widely used are probably MCNP, Geant4 and EGSnrc. In his 2013 history of Monte Carlo radiation transport, Bielajew notes that EGS and MCNP have been "historically dominant" while also drawing attention to the somewhat more recent rise of Geant4 [3]. He also concludes EGS and MCNP are "very different in nature" in that their users tend to come from different fields, since EGS is mainly used in medical physics while MCNP also has a huge nuclear engineering user base, thanks to its ability to simulate neutrons as well [3]. The same holds for the other codes, in that each usually has a dedicated group of users clustered within a certain field. In general, having many Monte Carlo codes aids verification and validation efforts as each code can help confirm the others' results.

EGSnrc is somewhat unusual among other modern transport codes in that it can only simulate photons, electrons, and positrons [4]. Both MCNP and Geant4 (among others) can simulate a wider range of particles including neutrons and protons [9, 10]. But on the other hand, for photon and electron transport, EGSnrc is usually considered to be one of the most

accurate and efficient codes, and often serves as a reference implementation [2, 45, 46].

The Monte Carlo method for radiation transport is used by a wide range of applications and has over fifty years of research history [47]. Rogers credits "massive increases in computing power per unit cost" for the method's rise to prominence [47], since Monte Carlo is computationally expensive. Over time, the growth in computing power has enabled the use of Monte Carlo for an increasing number of applications. A somewhat recent example is the development of Monte Carlo-based commercial radiotherapy treatment planning systems. Historically, the high cost of Monte Carlo meant less accurate deterministic methods were often used by treatment planning systems in clinical contexts [48]. But increases in simulation performance and superior accuracy have led to Monte Carlo being widely accepted for radiotherapy calculations [1]. Much progress in this area is owed to the OMEGA (Ottawa-Madison Electron Gamma Algorithm) project of the nineties [49], which resulted in the EGS-based radiation therapy beam simulation code BEAM [50], and successor BEAMnrc [51].

As far as computing hardware goes, most major Monte Carlo codes are CPU-based. But the rise of graphics processing units (GPUs) for scientific computing has also resulted in fast GPU-based Monte Carlo codes. In 2011, Lippuner and Elbakri reimplemented the photon transport routines of EGSnrc using the CUDA GPU programming environment and reported up to 40 times faster performance for equivalent results [52]. More recently, the GPU-based code ARCHER has demonstrated near real-time performance for treatment planning calculations, while obtaining comparable results to EGSnrc [53].

The EGSnrc manual holds that Monte Carlo "is the only known solution method that can be applied for any energy range of interest" [4]. However, the high computational cost of Monte Carlo means alternative methods can be an attractive choice for practitioners if simulation time and computing power are limited. Deterministic methods are the main alternative to Monte Carlo. Under certain conditions, deterministic methods can obtain accurate results (usually compared to Monte Carlo as a benchmark). For example, the point kernel method can calculate the dose from a point source along a line (neglecting scattering from other areas) using a single equation [54]. This can be a good approximation for low-density materials with low scattering, but is less accurate for high-density materials [54]. Another deterministic method of recent interest are linear Boltzmann transport equation (LBTE) solvers [55]. Instead of a stochastic solution to the Boltzmann equation, LBTE methods numerically solve the Boltzmann equation (with some approximations) [55]. For some cases, LBTE methods have obtained very similar results to Monte Carlo for radiotherapy calculations (within 2%) [56] and a recent review article concluded that LBTE solvers are "very promising", though less mature than Monte Carlo [55]. But in general, Monte Carlo is still considered the most accurate and realistic method available today and as computing power grows, the case for using less-accurate deterministic transport algorithms becomes weaker

and weaker [57].

3.2 Tetrahedral meshes in radiation transport codes

As tetrahedral meshes become supported by more transport codes, a body of work has emerged regarding their implementation and benchmarking. Of first importance are the existing tetrahedral mesh geometries of Geant4 [10], MCNP6 [9], and PHITS [11]. Geant4 was the first to implement this feature, using a large number of individual tetrahedron objects (`G4Tet`s) to make up a mesh. Looking at the source code history, `G4Tet` was created in 2004, so tetrahedral meshes have been technically realizable in Geant4 ever since [58]. Next came the initial MCNP6 implementation in 2012 [59], which was followed by PHITS in 2015 [60]. But much of the contemporary tetrahedral mesh geometry landscape was foreseen by a triangle mesh geometry published in 2009 for the transport code PENELOPE. That code, `penMesh`, improved on prior work from 2004 on triangle mesh transport [61] by making great efficiency and usability improvements [62]. The `penMesh` geometry implementation has a very similar design to the tetrahedral mesh geometry described in this thesis [62]. Transport acceleration is done using an octree data structure [62], and the implementation makes use of fast ray-triangle [63], ray-box and triangle-box overlap [64] algorithms from the computer graphics literature [62].

One of the original motivations for `EGS_Mesh` was to simulate the interaction of ionizing radiation with CAD geometries. Direct use of CAD geometries in transport simulations has long been a desirable feature of radiation transport codes. Because CAD tools are so often used to design objects used in simulations (e.g. particle accelerators), the ability to directly simulate transport in CAD designs avoids the ostensibly redundant step of redefining the geometry for the transport code. CAD geometry import has long been a strength of Geant4. For example, Geant4 has had a STEP file interface for over 20 years [65], though reportedly with limitations for complex geometries and sometimes requiring the use of commercial software [66]. MCNP also has a similar STEP file import capability with the use of external tools [67]. In 2012, Poole *et al.* implemented an alternative method for simulating radiation transport in CAD geometries for Geant4 [66]. Instead of translating each geometry directly into Geant4's geometry input language, triangular surface meshes are used to represent each volume. Such surface meshes are provided by common geometry file formats such as PLY and STL [66]. Though elegantly achieving the goal of CAD import, transport in surface meshes (termed "tessellated solids" by Geant4) was found to be a "performance bottleneck", especially for surface meshes made of many facets [68] (later on, the "DagSolid" library was developed to accelerate surface mesh transport in Geant4 [69]). To address this issue, Poole *et al.* went on to implement a tetrahedral mesh-based CAD interface [68]. They used TetGen, a freely available mesh generator [70] to convert STL files to a tetrahedral mesh format,

which after being read in by Geant4, was converted to a large number of `G4Tet` objects. The tetrahedral mesh simulations were found to be roughly two orders of magnitude faster than naive surface mesh simulations [68] (their implementation relied on Geant4's "smart-voxel" capability, a kd-tree-like partitioning method used to discretize arbitrary Geant4 geometries and accelerate particle intersection calculations [71]). Poole *et al.*'s conclusions on the usefulness of tetrahedral meshes for modelling CAD geometries in Geant4 confirmed the findings of earlier tetrahedral mesh transport research (e.g. [72, 73, 74]). So over time, tetrahedral meshes have proven to be an efficient and accurate way to simulate radiation transport in CAD geometries, though requiring an additional mesh generation step.

Around the same time as the Geant4 developments, work was ongoing on the MCNP mesh implementation [75]. Again, CAD import was the goal, and again, an unstructured mesh approach was selected out of the alternatives [75]. In addition to supporting first-order tetrahedrons, the MCNP unstructured mesh library is also capable of modelling second-order tetrahedrons as well as first- and second-order pentahedrons and hexahedrons [59]. The implementation uses an skd-tree to accelerate particle transport and also makes use of mesh element face neighbour information [59]. The commercial software Abaqus CAE is used to prepare input mesh files [59]. Unlike other codes, such as `EGS_Mesh`, which require a conformal mesh of first-order tetrahedrons, the MCNP implementation tries to be robust even when there are gaps or overlap between mesh elements [76].

Finally, the PHITS tetrahedral mesh geometry implementation is considered. The PHITS implementation uses two octrees to accelerate transport [77]. First, a volume octree containing every mesh element is built, to accelerate nearest-element point searches [77]. A surface element octree is also constructed to improve the performance of external particles intersecting with the mesh surface [77]. This idea is also used by `EGS_Mesh`, following the initial PHITS implementation. However, `EGS_Mesh` uses triangle-box overlap testing (Section 4.4.4) to check if a tetrahedron is inside a bounding box, while the PHITS implementation tests if the element centroid or any nodes are inside the box [77]. This approach can lead to false negatives, since tetrahedrons can still intersect a box even if none of the element nodes or centroid are inside it. The PHITS implementation handles this possibility by continuing to search other surrounding elements [77], though it is possible this reduces simulation performance. Inside mesh elements, PHITS uses ray-plane intersections to transport particles and also makes use of element face neighbours [77]. Of the other tetrahedral mesh geometry implementations considered here, `EGS_Mesh` is most similar to the PHITS implementation, as they both use two octrees to accelerate particle transport.

A recent study comparing the performance of the ICRP 145 tetrahedral mesh phantoms versus similar voxel geometries of varying resolution using Geant4, MCNP6, and PHITS was conducted by Yeom *et al.* [78]. Relative to the voxel phantom performance, the PHITS mesh simulations were generally faster than the voxel simulations, the Geant4 mesh simula-

tions had mixed results, and the MCNP6 mesh simulations were generally much slower than the voxel simulations. The authors attributed MCNP6's relatively slow performance to the "overly sophisticated" unstructured mesh implementation, in contrast with the specialized tetrahedral mesh implementations of Geant4 and PHITS [78]. A later study on the performance benefits of multithreading for simulating tetrahedral meshes concluded that the Geant4 implementation had the best scaling [79]. Meanwhile, the ICRP 145 report concluded that Geant4 was the fastest code for simulating photons and electrons. Relative to Geant4, the report found that PHITS is "three to 20 times" slower while MCNP6 is "six to 30 times" slower [7]. Performance differences can be attributed to the different transport algorithms used in each code as well as mesh-specific implementation details.

Before moving on, it should be noted that generating a high-quality tetrahedral mesh from surface meshes or CAD files can be a difficult process. Researchers plainly state that "Tetrahedral meshing is a hard problem" [80]. Meshing errors are commonplace, often requiring manual intervention and a skilled operator. Geometry simplification is sometimes necessary if the mesh generator fails. Another issue is correctly associating material information with mesh elements. But if a suitable mesh can be obtained, the resulting simulation will probably be relatively efficient.

3.3 Computational human phantoms

Computational human phantoms are crucial for studying the interaction of ionizing radiation with human anatomy. Many experiments are only possible by way of simulation, for obvious reasons. As radiation simulations have become more sophisticated, so have computational human phantoms. Calculations using the first "Reference Man" in the 1970s were thought to be accurate to one significant figure using the methods available at the time [81]. Today, modern phantoms offer far better predictive power. An important entity in this field is the International Commission on Radiological Protection (ICRP), which publishes reference material and makes recommendations to regulatory bodies [7].

In his history of the field, Xu outlines three major generations of computational human phantoms: stylized phantoms, voxel phantoms, and most recently boundary representation or BREP phantoms [12]. Examples of each generation are shown in Figures 1.1a, 1.1b, and 1.1c. Another review of computational human phantoms is given by Kainz *et al.* [82].

In the beginning (circa 1960) [12] there were the "stylized" phantoms: a set of "mathematical expressions describing idealised body organs" [14]. These stylized phantoms had "obvious shortcomings" [12] (e.g. some stylized phantoms assumed a uniform density throughout the phantom [83]) and their inherent defects "sometimes led to inaccurate results" [12]. Variations on standard reference data were considered out of scope at the time, with researchers concluding that since numerical results "are often only specified to one significant figure, it

is clear that minor changes in the definition to bring it into precise agreement with national or regional averages are not warranted" [81]. Though stylized phantoms were widely used at the time (and still used today in certain cases [84]), their limitations were the impetus for a new cohort of researchers to improve the state of the art [12].

Voxel-based phantoms emerged in the 1980s [12], bringing "an excitement to the research community because of their anatomical realism" [85] and have since become a standard reference format (e.g. the "Adult Reference Computational Phantoms" of ICRP 110 [13] and the "Pediatric Computational Reference Phantoms" of ICRP 143 [86]). Voxel phantoms are the standard format used today and are universally considered to be a massive improvement over stylized phantoms [12], but still have "limitations in representing organs and tissues that are complex or very thin" [87]. These limitations mean that stylized subregions are still required in order to perform accurate calculations for very thin organs and tissues [88]. For example, the eye lens is at risk of cataracts from ionizing radiation [89]. But the structure of the eye is such that the required voxelization for an accurate phantom would make simulation prohibitively expensive [13], so stylized phantoms are recommended for this case. For example, the widely-used ICRP 110 voxel models [13] are often enhanced using the stylized eye model of Behrens *et al.* [84]. However, requiring two types of phantoms in a single simulation complicates the setup process. Voxel phantoms also have limited deformability, meaning that transforming a reference phantom to another posture (e.g. from standing to sitting) is not feasible, even though it would be valuable for researchers [87]. Voxel phantom simulation is implemented in EGSnrc using the `EGS_XYZGeometry` library [6].

The third-generation boundary-representation (BREP) phantoms, including tetrahedral mesh phantoms, seek to overcome the modelling fidelity and deformability limitations of voxel phantoms [7]. Other kinds of BREP phantoms include polygonal surface mesh phantoms [90] and phantoms composed of non-uniform rational basis splines (NURBS) [91], but these are considered more complicated to implement in radiation transport codes (i.e. NURBS phantoms require voxelization in order to be used by most transport codes) [92]. As a recent example, in 2020 the ICRP released two adult tetrahedral mesh phantoms [7] as an alternative to their voxel reference meshes [13]. The ICRP considers tetrahedral mesh phantoms to be "the most advanced type of computational phantoms" because of their superior modelling power [7]. The phantoms were created by converting the previous generation of voxel phantoms to a surface mesh format, which was then meshed [7]. One of the motivations for the development of mesh phantoms was an investigation of dose to thin organs (which are difficult to represent using uniform voxelization). A preliminary study comparing the ICRP 110 male voxel phantom to an analogous polygonal surface phantom concluded that for penetrating radiation like photons and neutrons, organ doses were fairly similar [93]. But for weakly penetrating radiation like electrons, large dose differences were found in thin organs.

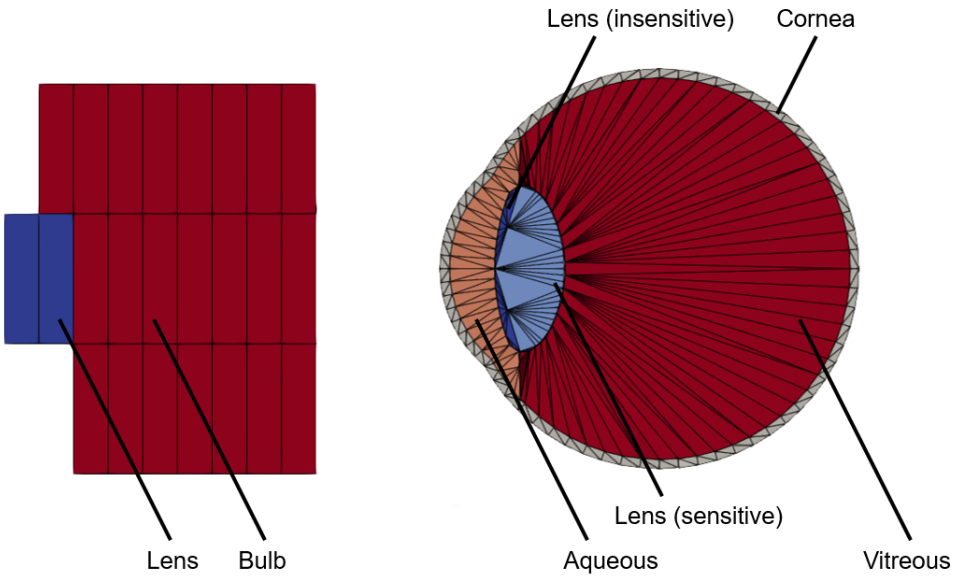


Figure 3.1: Voxelized phantom eye [13] and mesh phantom eye [7] cross-sections.

The researchers found the polygonal surface phantom had up to five times higher dose to skin and other thin organs than predicted using the voxel phantom [93]. Development and dissemination work is ongoing, e.g., the ICRP has begun work on pediatric mesh-type reference phantoms to complement the adult phantoms [94]. Researchers have demonstrated the improved deformability of the phantoms over their voxel counterparts by quickly developing a number of derived phantoms (e.g. representing the 10^{th} and 90^{th} percentile population height and weight) [95]. The deformability of BREP phantoms means they are well suited to 4D transport simulations, "in which the organs and tissue move and deform" [92]. A classic use case for 4D simulation is to account for a patient's lung movement, since this complication can cause "deviations between the planned and delivered doses" [96, 97]. The increased resolution of the mesh phantoms meant the use of supplementary stylized phantoms for certain organs is no longer required for most cases [7]. As an example of the difference in modelling ability, Figure 3.1 shows the voxelized eye of ICRP 110 compared to the tetrahedralized eye of ICRP 145. The tetrahedral mesh is much better able to represent the delicate eye tissues and better approximates the curved eye geometry. And with the improved resolution comes improved accuracy. The mesh eye distinguishes between the aqueous and vitreous humours, cornea, and sensitive/insensitive lens tissues, each with its own material composition data [7]. Meanwhile, the voxel eye only distinguishes between lens and bulb (an amalgamation of the remaining eye tissues) [13].

The development of the phantoms was a multi-year effort. A number of papers were written for the design of phantom organs and tissues, including:

- the alimentary and respiratory systems [98],

- the eyes [89],
- the skin [99],
- the skeleton [100], and
- the small intestine [101].

But the work was worth it and the adult mesh phantoms have been readily accepted and put to use by the simulation community. Researchers have already used the phantoms for

- determination of organ dose coefficients for external exposures for photons and electrons [88], and for neutrons, protons and helium ions [102],
- dosimetry of the radiotracer ^{18}F-FDG accounting for body size variations [103],
- proof-of-concept design of a digital tomosynthesis (low-dose x-ray imaging) device [104],
- development of over 200 body-size dependent derived phantoms [105],
- investigation of the influence of different postures (sitting, squatting, kneeling, etc.) on dose from an external photon source [106],
- investigation of organ dose from radioactive iodine depending on thyroid location based on the ICRP 145 mesh phantoms [107], and another study adjusted to Korean national average reference data [108], and
- estimation of dose received by a patient by an overexposure event during a medical procedure [109].

Even after the phantoms were initially released, development and refinement work continues. For instance, the ICRP 145 phantom teeth are modelled as a single tissue [7]. However, some researchers are interested in separating the teeth into constituent tissues [110]. For instance, tooth enamel is useful for assessing absorbed dose after exposure to radiation [111]. In order to calculate enamel doses, Shin *et al.* created a model of the teeth separated by tissue, which allowed them to more accurately calculate enamel dose after embedding the model in the reference mesh [110]. And in another recent study, researchers interested in obtaining a more accurate understanding of liver dose developed a refined liver mesh model including blood vessels [112] to differentiate dose to functional tissue and dose to organ blood. They found up to a 14% difference in absorbed dose with the homogeneous ICRP 145 liver [112], showing the importance of ever more detailed models. These examples demonstrate the extensibility of the reference phantoms, wherein the base phantoms can be augmented with more refined sub-models as required.

At the time of writing, mesh-type paediatric reference phantoms have been created and will soon be released as part of an ICRP reference document [94]. These new mesh phantoms will complement the existing voxel paediatric phantoms of ICRP 143 [86]. In some cases the pediatric phantoms have more complex organs than the corresponding adult phantoms. For instance, the paediatric phantoms include a set of deciduous teeth along with their permanent teeth (i.e. incorporating the detailed teeth model of Shin *et al.* [110]). Like the adult phantoms, the development of certain paediatric phantom organs and tissues have also merited publications [113, 114]. Hybrid phantoms made from combining the mesh phantoms with patient CT scans have also been created [115].

However, this new generation of phantoms is not yet the final word in modelling accuracy. In their review paper on Monte Carlo-based internal dosimetry, Fum *et al.* caution that organ dose "depends more on the location and shape of the organ rather than on the organ volume" [116], so just scaling the phantoms by volume may introduce significant errors for specific individuals. They see "internal organ displacement or reshaping" as the next step in the development of patient-specific phantom design [116]. This comes with a new set of challenges, as non-trivial phantom modifications are currently very labour-intensive.

Chapter 4

Tetrahedral mesh geometry: EGS_Mesh

EGS_Mesh is a tetrahedral mesh class defined in C++. The implementation includes basic tetrahedral mesh geometry routines (such as finding the closest point on a plane), and the necessary EGS_BaseGeometry interface methods. The EGS_BaseGeometry interface is accelerated using an octree.

Simple code is preferred, even at the cost of some performance. Keeping the code base as small as possible eases the future maintenance burden and simplifies debugging. But since EGS_Mesh simulations can run for days on end, performance is also an important feature. Because EGS_Mesh is a general-purpose library, only optimizations that apply to most meshes are considered. Advanced users can modify the library to achieve higher performance than the reference version in certain cases. As a concrete example, the egs++ method isWhere, which returns the bounding region for a particle, is accelerated using an element octree. This improves performance for virtually all meshes. But, if the particle source location is known beforehand to be inside a certain element, it could be faster to maintain a cache of nearby elements to check before conducting a full octree search.

First, the egs++ geometry interface is introduced in Section 4.1. Basic tetrahedral mesh geometry algorithms are then defined in Section 4.2. A naive geometry implementation which satisfies the egs++ interface is given in Section 4.3 and then the final accelerated version is given in Section 4.4. EGS_Mesh performance tests are conducted in Sections 4.5, 4.6, and 4.7. EGS_Mesh memory requirements are discussed in Section 4.8.

4.1 The egs++ geometry interface

The design of the egs++ geometry library allows for user-defined geometry packages [6]. All egs++ geometries must inherit from the abstract base class EGS_BaseGeometry to properly interface with the EGSnrc physics engine [117]. The egs++ documentation is a valuable resource for would-be geometry implementers and contains many useful notes and examples [6]. Over the years, EGSnrc maintainers and users have developed special-purpose geometries

that are best suited to their requirements. A recent example is EGS_Lattice, a library for simulating a large number of repeated cell geometries [22].

EGSnrc geometries are divided into regions, each of which may be composed of a different medium (e.g. water, air, etc.). One widely used geometry with multiple regions is EGS_XYZGeometry. EGS_XYZGeometry divides the domain into hexahedral elements (voxels), just as EGS_Mesh divides the domain into tetrahedral elements. The most important required methods of the EGS_BaseGeometry interface are:

- isWhere: given a point, return the enclosing geometry region, if any,
- hownear: determine the minimum distance from a point to a geometry region boundary along any direction (used both internally and externally),
- howfar: determine the minimum distance from a point to a geometry region boundary along a given direction (used both internally and externally), and
- medium: given a geometry region index, return the medium of that region.

There also are some other, technically necessary, methods that are trivial to implement:

- inside: an obsolete method equivalent to isWhere,
- isInside: determines whether a given position is inside the geometry (simplified case of isWhere), and
- getType: returns the type of this geometry as a string.

And finally, all derived geometries have to set EGS_BaseGeometry::nreg to the number of regions in the derived geometry so that the regions method returns the correct number [117]. At the time of writing, EGS_Mesh stores the following data explicitly:

- node coordinate vectors,
- element node offsets,
- element tags,
- element neighbour offsets,
- element face normal vectors,
- element boundary flags, and
- element medium offsets.

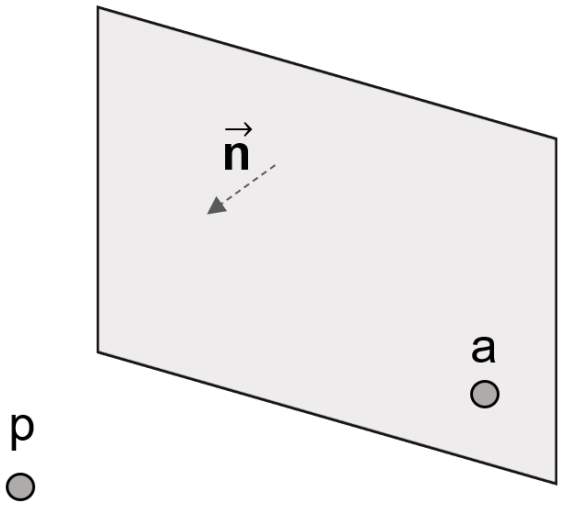


Figure 4.1: Detecting if a point is in front of a plane.

4.2 Tetrahedral mesh geometry basics

Before implementing the `EGS_BaseGeometry` interface, there are a number of basic geometry algorithms that are required. Then, the actual algorithms are implemented in terms of these geometry primitives. For example, the `hownear` method returns the absolute distance to a geometry boundary [4], which means the implementation will require a number of closest-point geometry routines. As another example, the `howfar` method returns the distance to the geometry given a particle position and direction vector [4], so ray-geometry intersection algorithms are also required. This section relies heavily on Ericson's textbook [118]. The primitive geometry methods considered here are:

- detecting if a point is in front of a plane (used by `isWhere`),
- distance from point to plane (used by `hownear`),
- closest point on triangle to point (used by `hownear`),
- ray-plane intersection (used by `howfar`),
- ray-triangle intersection (used by `howfar`), and
- finding neighbouring elements (used to reduce the element search space).

4.2.1 Detecting if a point is in front of a plane

Given a plane defined by a normal, $\vec{n}$, and a point, $\vec{a}$, the goal is to find whether the query point, $\vec{p}$, is in front of the plane (i.e. whether the normal points toward $\vec{p}$). The geometry is shown in Figure 4.1. For all points not on the plane, the plane divides space into two

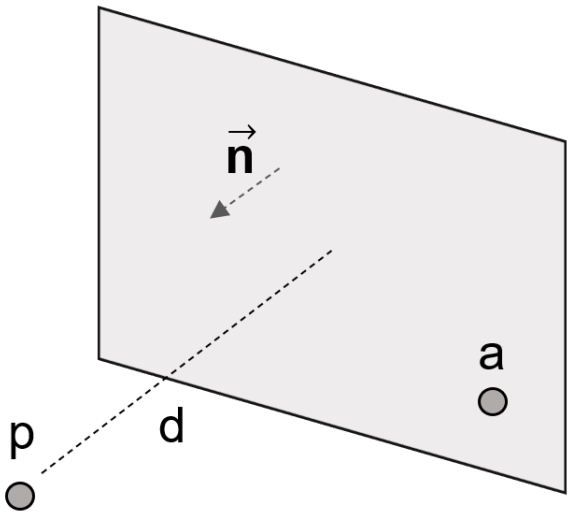


Figure 4.2: Distance from a point to a plane.

halfspaces. An arbitrary point, $\vec{p}$, is in the plane's positive halfspace if the normal points towards it and in the negative halfspace if the normal points away

$$\vec{n} \cdot (\vec{p} - \vec{a}) \begin{cases} < 0 & \text{if } \vec{p} \text{ is in the negative halfspace} \\ = 0 & \text{if } \vec{p} \text{ is on the plane} \\ > 0 & \text{if } \vec{p} \text{ is in the positive halfspace} \end{cases} \tag{4.1}$$

For the four triangular faces of a tetrahedron with vertices $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$, Section 5.1.6 of [119] shows how a precalculated face normal is not required to determine whether a given point is outside a triangle's face plane. For a triangle face with vertices, $\vec{a}$, $\vec{b}$, $\vec{c}$, the fourth vertex, $\vec{d}$, can used to orient the query. If the dot products $(\vec{d} - \vec{a}) \cdot ((\vec{b} - \vec{a}) \times (\vec{c} - \vec{a}))$ and $(\vec{p} - \vec{a}) \cdot ((\vec{b} - \vec{a}) \times (\vec{c} - \vec{a}))$ have the same sign, $\vec{p}$ and $\vec{d}$ both lie on the same side of the plane. Otherwise, they are on opposite sides of the plane.

4.2.2 Distance from point to plane

The objective is to find the shortest distance, d, from a query point, $\vec{p}$, to the plane. The plane is defined by a normal, $\vec{n}$, and plane point, $\vec{a}$. The geometry is shown in Figure 4.2. Intuitively, the shortest distance to the plane will always be along the plane's normal vector. The distance in terms of the normal's magnitude, $\|n\|$, can be found by projecting the vector $\vec{pa}$ onto the negative normal vector $-\vec{n}$ (towards the plane). This projection is the dot product between the two vectors [118],

$$d = -\frac{\vec{n} \cdot \vec{pa}}{\|\vec{n}\|}. \tag{4.2}$$

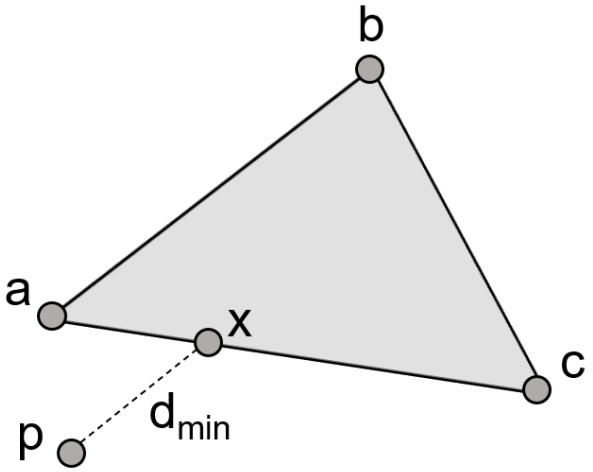


Figure 4.3: Closest point on a triangle for a given point.

After assuming a unit normal, (i.e. $\|\vec{n}\| = 1$) and taking the absolute value of the distance, $|d|$ (as required by the `hownear` specification [4]), the equation becomes

$$|d| = |\vec{n} \cdot \vec{pa}|. \tag{4.3}$$

Inside a tetrahedron, this routine can be used to find the minimum distance to the tetrahedron surface by calculating the distance to each face plane and taking the minimum value.

4.2.3 Closest point on triangle to point

The objective is to find the closest point, $\vec{x}$, on the triangle with vertices $\vec{a}$, $\vec{b}$, $\vec{c}$, for a query point, $\vec{p}$. The geometry is shown in Figure 4.3. The implementation from Ericson [119] uses the concept of Voronoi regions. Essentially, the Voronoi region of a triangle feature is the space bounding all points which are closest to that feature [119]. The closest point is then found by projecting onto the corresponding triangle feature. So, the Voronoi region of vertex $\vec{a}$ is the space containing all points that are closer to $\vec{a}$ than to any other triangle feature. There are seven Voronoi regions for a triangle: three vertex regions, ($\vec{a}$, $\vec{b}$, $\vec{c}$), three edge regions, ($\vec{ab}$, $\vec{bc}$, $\vec{ca}$), and the region inside the triangle itself. The triangle Voronoi regions are defined in terms of negative halfspaces (4.1). A point, $\vec{p}$, is in the Voronoi region of an arbitrary triangle vertex, $\vec{v_0}$, if the following conditions are true given the other triangle vertices, $\vec{v_1}$ and $\vec{v_2}$,

$$(\vec{p} - \vec{v_0}) \cdot (\vec{v_1} - \vec{v_0}) < 0,$$
$$(\vec{p} - \vec{v_0}) \cdot (\vec{v_2} - \vec{v_0}) < 0.$$

For example, the test for containment in the Voronoi region of vertex b is

$$(\vec{p}-\vec{b})\cdot(\vec{a}-\vec{b})<0,$$
$$(\vec{p}-\vec{b})\cdot(\vec{c}-\vec{b})<0.$$

Now for the edge Voronoi regions. For a triangle edge, $\vec{e}=\vec{v_1}-\vec{v_0}$, it is necessary that $\vec{p}$ is in the positive halfspaces of the two planes defined using the edge vertices [119]

$$(\vec{p}-\vec{v_0})\cdot(\vec{v_1}-\vec{v_0})>0,$$
$$(\vec{p}-\vec{v_1})\cdot(\vec{v_0}-\vec{v_1})>0.$$

In addition, the point must be outside or on the edge (otherwise it is within the triangle). This can be checked using the following "triple product" equation [119] making use of the triangle normal, $\vec{n}$,

$$\vec{n}\cdot((\vec{v_0}-\vec{p})\times(\vec{v_1}-\vec{p}))\leq 0. \tag{4.4}$$

For example, $\vec{p}$ is in the Voronoi region of edge $\vec{ab}$ if the following holds

$$(\vec{p}-\vec{a})\cdot(\vec{b}-\vec{a})>0,$$
$$(\vec{p}-\vec{b})\cdot(\vec{a}-\vec{b})>0,$$
$$\vec{n}\cdot((\vec{a}-\vec{p})\times(\vec{b}-\vec{p}))\leq 0.$$

Then, if a point is in an edge region, the closest point, $\vec{x}$, is found by projecting onto the edge. The parametric equation of an edge from $\vec{a}$ to $\vec{b}$ is [119]

$$P(t)=\vec{a}+t(\vec{b}-\vec{a}), \tag{4.5}$$

where t scales the direction vector, $\vec{ab}$. The value of t in terms of real distance (not just in terms of $\|\vec{ab}\|$) corresponding to the projection of $\vec{p}$ onto the edge is [119]

$$t=\frac{(\vec{p}-\vec{a})\cdot(\vec{b}-\vec{a})}{\|\vec{b}-\vec{a}\|^2}, \tag{4.6}$$

which can then be substituted back into the parametric edge equation to find the closest point on the edge.

Finally, if the point does not belong to any vertex or edge Voronoi regions, it must lie inside the triangle and the closest point is found by projecting onto the triangle. Barycentric coordinates are used to determine this closest point. In the barycentric formulation, an arbitrary triangle point, $\vec{q}$, can be expressed as a weighted sum of the three triangle vertices,

$\vec{a}$, $\vec{b}$, and $\vec{c}$, with respective weights, w, u and v

$$\vec{q}(w,u,v) = w\vec{a} + u\vec{b} + v\vec{c}. \tag{4.7}$$

Each weight can be found using the triple products, T_i, already calculated for each edge (4.4)

$$\begin{aligned} w &= \frac{T_a}{T_a + T_b + T_c}, \\ u &= \frac{T_b}{T_a + T_b + T_c}, \\ v &= \frac{T_c}{T_a + T_b + T_c}. \end{aligned}$$

with T_a, T_b, and T_c given by

$$T_a = \vec{n} \cdot ((\vec{b} - \vec{p}) \times (\vec{c} - \vec{p})), \tag{4.8}$$

$$T_b = \vec{n} \cdot ((\vec{c} - \vec{p}) \times (\vec{a} - \vec{p})), \tag{4.9}$$

$$T_c = \vec{n} \cdot ((\vec{a} - \vec{p}) \times (\vec{b} - \vec{p})). \tag{4.10}$$

The actual implementation makes use of Ericson's efficient reformulation of these equations to minimize the number of floating point operations (see Section 5.1.5 of [119]).

4.2.4 Ray-plane intersection

Given a plane defined by a unit normal vector, $\vec{n}$, and plane point, $\vec{a}$, the goal is to find the distance, d, to the intersection point, $\vec{x}$, if any, with the ray defined by the particle position, $\vec{p}$, and direction vector, $\vec{u}$. In practice, the particle direction vector is normalized by the EGSnrc implementation. The geometry is shown in Figure 4.4. The intersection point, $\vec{x}$, with the ray can be described using the parametric ray equation [118],

$$\vec{x} = \vec{p} + t\vec{u}. \tag{4.11}$$

A positive t value means the ray is pointing towards the plane, and a negative t value means the ray is pointing away from the plane. So, only positive t values are considered valid intersections. Since $\vec{x}$ is on the plane, the vector from another plane point, $\vec{a}$, to $\vec{x}$ is perpendicular to the plane normal, so the dot product between the two vectors is zero

$$\vec{n} \cdot (\vec{x} - \vec{a}) = 0. \tag{4.12}$$

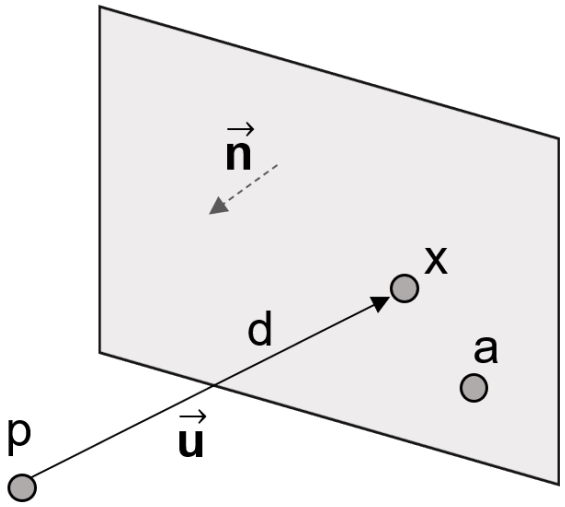


Figure 4.4: Ray-plane intersection.

Substituting (4.11) for $\vec{x}$ and isolating for t yields

$$\begin{aligned}
\vec{n} \cdot (\vec{p} + t\vec{u} - \vec{a}) &= 0 \\
\vec{n} \cdot ((\vec{p} - \vec{a}) + t\vec{u}) &= 0 \\
\vec{n} \cdot \vec{ap} + t(\vec{n} \cdot \vec{u}) &= 0 \quad \text{(dot product is distributive)} \\
t &= -\frac{\vec{n} \cdot \vec{ap}}{\vec{n} \cdot \vec{u}}. \qquad (4.13)
\end{aligned}$$

If $\vec{u}$ is normalized, $t = d$ since t is the intersection distance in terms of $\|u\|$. So assuming a unit $\vec{u}$, the distance, d, is obtained

$$d = -\frac{\vec{n} \cdot \vec{ap}}{\vec{n} \cdot \vec{u}}. \qquad (4.14)$$

This final equation includes a negative sign, which implies a signed distance value. One can confirm that the equation returns a positive distance under regular conditions. If the point, $\vec{p}$, is in front of the plane, $\vec{n} \cdot \vec{ap}$ is positive. If $\vec{u}$ is pointed towards the plane, $\vec{n} \cdot \vec{u}$ is negative. The two negative signs cancel out, returning a positive distance

$$\begin{aligned}
d &= -\frac{\vec{n} \cdot \vec{ap}}{\vec{n} \cdot \vec{u}}, \\
d &\approx (-)\frac{(+)}{(-)}, \\
d &\approx (+).
\end{aligned}$$

Since there is a division by $\vec{n} \cdot \vec{u}$, it initially appears that care must be taken to avoid cases where the particle velocity is perpendicular to the normal and the dot product is zero (i.e.

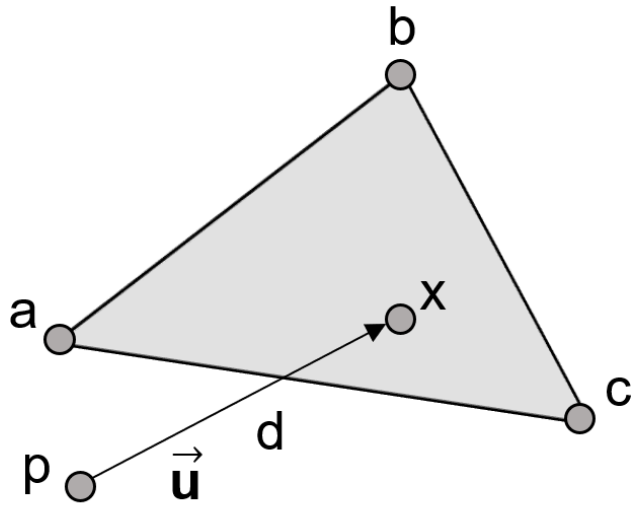


Figure 4.5: Ray-triangle intersection.

the particle is travelling parallel to the plane). In the EGS_Mesh implementation, a non-zero check is performed before carrying out the division. If the two vectors are perpendicular, there is considered to be no intersection (even if the particle lies on the plane and is travelling perpendicular to it, which would imply infinite intersection points). Other implementations choose to rely on IEEE 754 infinity arithmetic [120], which returns the conventional answer of no intersection in this case, to avoid a branch (see Section 5.3.1 of [119]).

4.2.5 Ray-triangle intersection

The objective is to find the distance, d, to the intersection point, $\vec{x}$, on the triangle with vertices $\vec{a}$, $\vec{b}$, $\vec{c}$, for the particle with position, $\vec{p}$, and direction, $\vec{u}$. The geometry is shown in Figure 4.5. EGS_Mesh uses the standard Möller-Trumbore intersection algorithm [63]. Möller-Trumbore again uses barycentric coordinates (introduced in Section 4.2.3) for an efficient implementation [63]. For all points inside the triangle, the barycentric weights are normalized, so $w + u + v = 1$. Rewriting w in terms of u and v yields the conventional barycentric coordinate equation for a triangle

$$\vec{q}(u, v) = (1 - u - v)\vec{a} + u\vec{b} + v\vec{c}. \tag{4.15}$$

Substituting the parametric ray equation (4.11) for the intersection point $\vec{q}$ on the triangle from (4.15) yields the following to be solved (one should note the difference between u, the weight, and $\vec{u}$, the direction vector)

$$\vec{p} + t\vec{u} = (1 - u - v)\vec{a} + u\vec{b} + v\vec{c}. \tag{4.16}$$

Collecting like terms yields

$$\begin{aligned} p + t\vec{u} &= (1 - u - v)\vec{a} + u\vec{b} + v\vec{c}, \\ p + t\vec{u} &= \vec{a} - u\vec{a} - v\vec{a} + u\vec{b} + v\vec{c}, \\ \vec{p} - \vec{a} &= -t\vec{u} - u(\vec{b} - \vec{a}) + v(\vec{c} - \vec{a}). \end{aligned}$$

Isolating for the three unknowns t, u, and v leads to this matrix equation

$$\vec{p} - \vec{a} = \begin{bmatrix} & & \\ -\vec{u}, & \vec{b} - \vec{a}, & \vec{c} - \vec{a} \\ & & \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix}. \tag{4.17}$$

There are three linear equations and three unknowns, so Cramer's rule [63] can be applied to directly solve the system in terms of matrix determinants where $\vec{b} - \vec{a} = \vec{E_1}$, $\vec{c} - \vec{a} = \vec{E_2}$ and $\vec{p} - \vec{a} = \vec{T}$

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{\det\begin{bmatrix} -\vec{u}, & \vec{E_1}, & \vec{E_2} \end{bmatrix}} \begin{bmatrix} \det\begin{bmatrix} \vec{T}, & \vec{E_1}, & \vec{E_2} \end{bmatrix} \\ \det\begin{bmatrix} -\vec{u}, & \vec{T}, & \vec{E_2} \end{bmatrix} \\ \det\begin{bmatrix} -\vec{u}, & \vec{E_1}, & \vec{T} \end{bmatrix} \end{bmatrix}. \tag{4.18}$$

There are four 3 by 3 matrix determinants to calculate. The determinant of a 3 by 3 matrix $[\vec{A}, \vec{B}, \vec{C}]$ can be found using one cross product and one dot product [118]

$$\det\begin{bmatrix} \vec{A}, & \vec{B}, & \vec{C} \end{bmatrix} = \vec{A} \cdot (\vec{B} \times \vec{C}). \tag{4.19}$$

Rewriting (4.18) with this formula yields

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{-\vec{u} \cdot (\vec{E_1} \times \vec{E_2})} \begin{bmatrix} \vec{T} \cdot (\vec{E_1} \times \vec{E_2}) \\ -\vec{u} \cdot (\vec{T} \times \vec{E_2}) \\ -\vec{u} \cdot (\vec{E_1} \times \vec{T}) \end{bmatrix}. \tag{4.20}$$

Finally, these two cross-product identities [118]

$$\vec{A} \cdot (\vec{B} \times \vec{C}) = (\vec{A} \times \vec{B}) \cdot \vec{C}, \tag{4.21}$$

$$\vec{B} \times \vec{C} = -(\vec{C} \times \vec{B}), \tag{4.22}$$

are used to rearrange (4.20) until there are only two cross products ($\vec{P} = \vec{u} \times \vec{E}_2$ and $\vec{Q} = \vec{T} \times \vec{E}_1$) required to calculate all four determinants,

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{-\vec{u} \cdot (\vec{E}_1 \times \vec{E}_2)} \begin{bmatrix} \vec{T} \cdot (\vec{E}_1 \times \vec{E}_2) \\ -\vec{u} \cdot (\vec{T} \times \vec{E}_2) \\ -\vec{u} \cdot (\vec{E}_1 \times \vec{T}) \end{bmatrix} = \frac{1}{\vec{P} \cdot \vec{E}_1} \begin{bmatrix} \vec{Q} \cdot \vec{E}_2 \\ \vec{P} \cdot \vec{T} \\ \vec{u} \cdot \vec{Q} \end{bmatrix}. \tag{4.23}$$

If $u > 0$, $v > 0$ and $u + v \leq 1$, there is an intersection and t is the signed intersection distance. Only positive values of t (i.e. along the direction vector $\vec{u}$) are considered valid intersections. Before calculating any of the unknowns, the divisor, $\vec{P} \cdot \vec{E}_1$, is compared to a small epsilon range around zero. If it is too close to zero, there is no intersection and the routine exits early [63]. Another version of these equations using the triangle normal is described by Ericson (Section 5.3.6 [119]), which could improve performance over the current version, since `EGS_Mesh` already precalculates triangle normals.

Though efficient, this algorithm is not watertight at the edges, meaning valid intersections can be missed due to floating point roundoff error [121]. Woop *et al.* have designed a watertight ray-triangle intersection algorithm for single precision floats [121] using the guarantees of the floating point math standard IEEE-754 [120]. This watertight algorithm has not yet been implemented for `EGS_Mesh`, so its relative performance for this use case is unknown, though, by the authors' measurement, it is "at most 10% slower than Möller-Trumbore" for rendering scenes with tens of millions of triangles [121]. Their 32-bit implementation relies on a double-precision fallback case [121], which might necessitate the use of 128-bit floats for a fully conservative double-precision `EGS_Mesh` implementation. For the initial implementation of this thesis, the simpler Möller-Trumbore algorithm was selected, but the watertight algorithm exists and may prove a superior choice (see also Section 4.3.3 and Section 6.6).

4.2.6 Finding neighbouring elements

To increase simulation efficiency, element face connectivity (i.e. element neighbours) is determined and stored. When a particle is in a tetrahedron, the four element faces need to be checked for intersection with the particle path. When the particle crosses a face, the new element can be immediately determined thanks to the stored connectivity information and passed back as the new region number to the EGSnrc physics engine (there are some exceptions due to floating point errors, see Section 4.3.4). In the case of a boundary element, the particle may also leave the mesh if exiting through a boundary face. The internal case of `howfar` benefits the most from this optimization (Section 4.3.4).

Determining tetrahedron face connectivity is done once at the start of the simulation, so performance is less critical than for core simulation routines such as `howfar`. There exists a straightforward quadratic algorithm to determine tetrahedron (and more generally,

finite-element) face connectivity. This quadratic algorithm is shown in Algorithm 4.1.

Algorithm 4.1 Quadratic finite-element neighbour-finding algorithm

```
for element i of elements do
    for face f_i of i do
        for element j of elements, i ≠ j do
            for face f_j of j do
                if f_i = f_j then
                    neighbours[i][f_i] ← j
                end if
            end for
        end for
    end for
end for
```

Nevertheless, time-to-simulation is an important metric for users, as certain errors may only be detected after simulation start-up has completed. The increasing cost of a quadratic algorithm is especially bad for larger meshes of interest, such as the eight-million element ICRP 145 meshes [7]. For this reason, an efficient algorithm was selected over the naive quadratic one. This algorithm is described in Löhner's *Applied CFD Techniques* [122] and shown in Algorithm 4.2. This algorithm shrinks the search space by only considering other elements as neighbour candidates if they share a face point.

Algorithm 4.2 Linear finite-element neighbour-finding algorithm

```
for element i of elements do
    for node n of i do
        nodes2elts[n].push(i)
    end for
end for
for element i of elements do
    for face f_i of i do
        n ← f_i.nodes[0]                          ▷ Pick an arbitrary face node
        for element j of nodes2elts[n], i ≠ j do
            for face f_j of j do
                if f_i = f_j then
                    neighbours[i][f_i] ← j
                end if
            end for
        end for
    end for
end for
```

The actual implementation has some further optimizations compared to the pseudocode of Algorithm 4.2: if j is found to be a neighbour of i, then i is also a neighbour of j, and one can skip elements that have already had their neighbours determined in this way.

4.3 Naive egs++ mesh geometry implementation

With these basic geometry routines in hand, a naive implementation of the egs++ interface for the mesh geometry is realizable. Since each tetrahedron has four triangular faces, tetrahedral geometry routines can be built using the triangle routines as primitives. However, care is required to avoid robustness issues, especially for the `howfar` implementation.

4.3.1 `isWhere`

The `isWhere` routine returns the bounding geometry region index of a given point or -1 if the point is outside the geometry [4, 117]. As this is a true or false answer for each element, exact distances need not be calculated. Given these parameters, the algorithm from Section 4.2.1 suffices. In the `EGS_Mesh` implementation, triangle face normals point inwards (their direction is swapped if called by the EGSnrc visualization tool `egs_view`). If all four tetrahedron face normals are directed towards the point, the point is inside the tetrahedron. For the naive implementation, brute-force search over all elements is used. Though easy to implement, this method has terrible performance: linear in the number of mesh elements, $O(n)$. The worst case occurs when the query point is outside the mesh. Every mesh element must be checked before it is concluded the point is not inside the mesh, which is an issue for simulations with external particle sources. This implementation is improved upon in Section 4.4.5.

4.3.2 `hownear`

The `hownear` method is used to find the minimum distance to a geometry boundary in any direction for a given point, $\vec{x}$. A geometry region number, i, is also provided. If $i = -1$, `hownear` must find the minimum distance to a geometry surface from outside, which can be expensive for an arbitrary mesh. Roughly speaking, both EGSnrc electron step algorithms, PRESTA-I [123] and PRESTA-II [21], rely on `hownear` to decide whether to use multiple- or single-scattering [4]. If possible, multiple-scattering is preferred because of its better performance (usually an order of magnitude faster than single-scattering). If the particle is close to a boundary and `hownear` returns a small distance, the algorithm switches to single-scattering mode if the exact boundary crossing algorithm is used (see Section 3.6 of [4]). This will not negatively impact simulation accuracy, because single-scattering is more accurate than multiple-scattering. Therefore, even a trivial `hownear` implementation suffices to perform simulations. If implementing `hownear` is too difficult or if the resulting performance is too slow, geometry implementers can return 0.0 for all cases, forcing the electron transport algorithm to assume it is always near a boundary [4]. However, to use multiple-scattering, a fast `hownear` implementation is required. A less drastic implementation choice than returning 0.0 in all cases is to instead return a lower

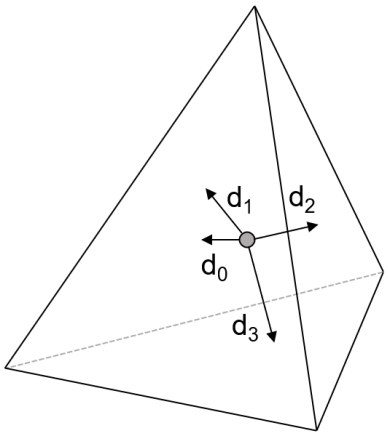


Figure 4.6: Tetrahedral mesh internal `hownear` geometry.

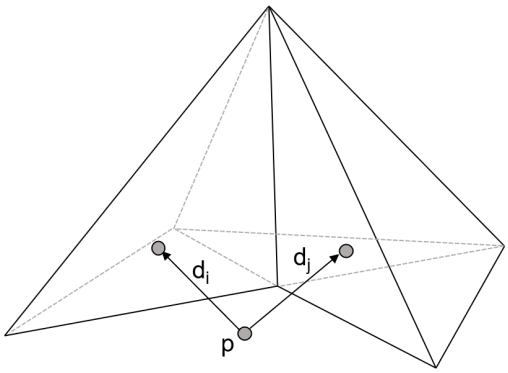


Figure 4.7: Tetrahedral mesh external `hownear` geometry.

bound on the distance to a boundary in any direction (see Section 3.6 of [4]). This is what is done by the accelerated `EGS_Mesh hownear` implementation below in Section 4.4.6. But first, a naive implementation of `hownear` for an arbitrary tetrahedral mesh is considered.

If the particle is inside the mesh (i.e. $i \neq -1$), then the particle is assumed to be inside region i that was passed into `hownear` (there are rare cases where the exact position of the particle and region disagree because of floating point issues, but this is neglected). Then, the minimum distance to a boundary is the minimum distance to the four tetrahedral faces. The implementation calculates the distance to each triangle's plane using (4.3) from Section 4.2.2 and returns the minimum distance. The four required plane distances, d_0, d_1, d_2 and d_3 are shown in Figure 4.6. If the particle is outside the mesh ($i = -1$), then a search over all mesh surface faces is required. The minimum distance to a surface face is stored and updated as the search progresses. Unlike for the internal case, the plane distance equation is not applicable here since the closest point on the plane could be outside the triangle bounds. Instead, the slower triangle-point minimum distance algorithm introduced in Section 4.2.3 is used. The external `hownear` geometry is shown in Figure 4.7. The number of surface elements is typically much less than the total number of elements, so at least a full mesh

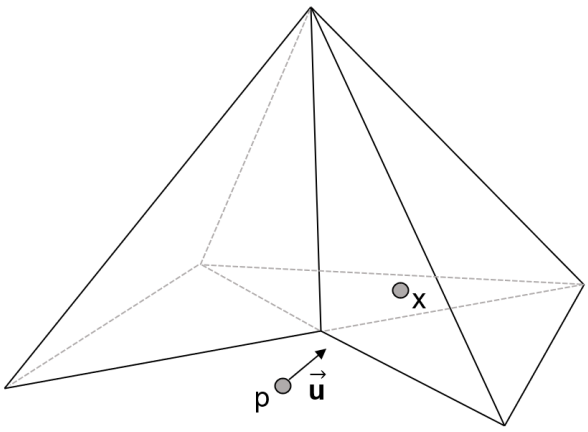


Figure 4.8: Tetrahedral mesh external `howfar` geometry.

search is not necessary. For example, the ICRP 145 adult male phantom [7] has over 8×10^6 elements, but only around 9×10^4 surface elements. However, this naive implementation is still extremely slow since `hownear` is called many times during electron transport. To summarize, the naive implementation has good, constant-time, performance for the internal case and slow (linear in the number of surface elements) performance for the external case. External `hownear` performance is improved upon in Section 4.4.6.

4.3.3 `howfar` (external)

The `howfar` method is generally the most complex routine to implement for a new EGSnrc geometry [4]. Given a particle position, $\vec{p}$, and normalized direction, $\vec{u}$, `howfar` must find the intersection distance, t_i, with the next geometry boundary. The intended step size, t, is passed into `howfar` and if there is an intersection with distance $t_i < t$, t is lowered to t_i. If the particle will enter a new geometry region after the step, the new region number should also be returned. Like for `hownear`, there are two cases: external and internal. The external `howfar` geometry with particle position, $\vec{p}$, direction, $\vec{u}$, and intersection point, $\vec{x}$, is shown in Figure 4.8. The naive implementation requires a loop over all surface elements, like the naive external `hownear` algorithm. Every surface face of each tetrahedron is checked for an intersection. Ending the search when the first intersection is found is not possible, since there could exist a closer intersection given a concave mesh. Like the external `hownear` case, the performance of the naive external `howfar` mesh routine is linear in the number of mesh surface elements. However, it turns out that naive `howfar` is simpler and potentially faster than naive `hownear`, since many surface elements can be quickly determined to not intersect the particle during the `howfar` calculation. Meanwhile, `hownear` has to find the distance to every surface element, even in cases where the particle is moving away from the element in question.

For the external case, the Möller-Trumbore ray-triangle intersection algorithm [63] from

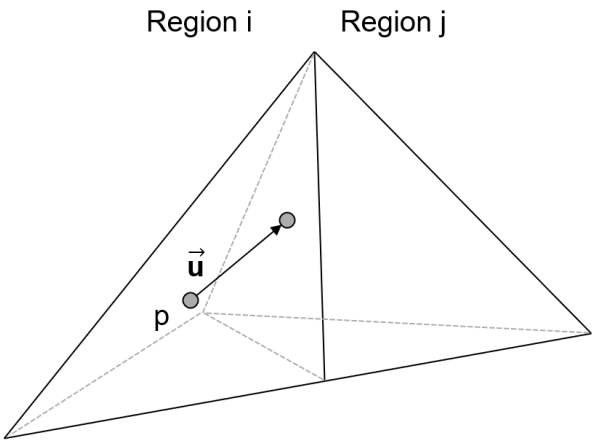


Figure 4.9: Tetrahedral mesh internal `howfar` geometry.

Section 4.2.5 is used. The performance and simplicity of this algorithm are its strengths. However, Möller-Trumbore is not watertight at the edges and vertices, meaning valid intersections can be missed due to floating point issues, resulting in false negatives [121]. The amount of false negatives will depend on the mesh and source position. As an estimate of the false negative rate, Woop *et al.* designed a test case with a ray source shooting 10^8 rays inside a fully enclosing sphere mesh [121]. For a mesh of 4×10^4 triangles, there were around 40 missed intersections, and for a mesh of 4×10^6 triangles, there were around 340 missed intersections [121], meaning the respective false negative rates were 0.00004% and 0.00034%. Based on these numbers and given EGSnrc's nominal accuracy of 0.1% [4], the presence of false negatives should not have a noticeable effect on the end result of typical `EGS_Mesh` simulations, but it is possible that certain meshes have higher false negative rates. As an aside, a useful extension to this work would be the implementation of a watertight triangle-ray intersection algorithm from the literature [121] and using it to measure the false negative rate of Möller-Trumbore for a complex mesh such as the ICRP 145 phantoms [7] (see Section 6.6). The performance of this naive `howfar` implementation is improved upon in Section 4.4.7.

4.3.4 `howfar` (internal)

The internal `howfar` case is similar to the external case, except the current bounding mesh region of the particle is also known. The geometry with particle position, $\vec{p}$, direction, $\vec{u}$, and two tetrahedral geometry regions, i and j, is shown in Figure 4.9. The particle is in region i, and will be transported along $\vec{u}$ until it intersects the triangular face between i and j. After the step ends, the particle is now considered to be in j and transport continues. Since `EGS_Mesh` precalculates tetrahedron neighbours, it might be concluded that given a face intersection the next geometry region is known immediately (Section 4.2.6). Although

usually true, this is not always the case, since the particle might exit the current tetrahedron exactly through a vertex or edge into a tetrahedron other than the immediate face neighbours. Another possibility is that calculating the distance to a boundary results in numerical undershoot, where the particle is considered to have exited through a face after a `howfar` step, but it actually remains geometrically inside the previous region [124]. Overshoot is also possible, given very thin neighbouring tetrahedrons or intersections near a vertex or edge.

Bielajew has considered both undershoot and overshoot for numerous quadric geometries in the context of EGS [124] and gives implementation and error-recovery advice for common shapes such as spheres, planes, cones, etc. The exact recommendations are not immediately applicable to a tetrahedral mesh geometry, but they have been very helpful in developing the actual implementation. In particular, the recommended error-recovery method for quadrics is to return 0.0 as the step distance if a particle is not inside the expected region. For a tetrahedral mesh, this has been observed to lead to an infinite loop if a particle is exactly coincident with a mesh vertex or edge. Thanks to numerical issues, particles exactly on the vertex or edge can be considered part of multiple tetrahedrons at once, causing the proposed recovery process to loop forever if care is not taken.

To avoid truncation errors at geometry boundaries, a lower bound on the minimum value of `howfar`, t_{min}, is introduced as well. This is slightly different than an enforced minimum step, as if the EGSnrc physics engine decides to transport a particle by a step smaller than t_{min}, it will not be modified by the mesh `howfar` implementation. Instead, if the mesh `howfar` calculation returns a distance result, t, less than t_{min} (which can occur for a degenerate calculation), t is raised to t_{min} and transport continues. In the current implementation, $t_{min} = 10^{-10}$ cm, or one picometre. This value was chosen since tens of picometres is roughly on the order of an atomic radius measured in angstroms (1 pm = 0.01 Å) [125], yet 10^{-10} is large enough to avoid floating point truncation errors of the same magnitude when performing additions or subtractions with floating point numbers with magnitude similar to those used by typical simulations. The current value of t_{min} may not be optimal and can be refined later on with justification.

The nature of floating point means that a small number added to a large number may be swallowed up by rounding error [126]. This means choosing a t_{min} value that is too small will fail to allow the particle to move far enough so regular transport can resume. Using the guarantees of IEEE 754 [120], the neighbourhood where t_{min} becomes too small can be found. The standard model of floating point arithmetic is [126]

$$fl(x \operatorname{op} y) = (x \operatorname{op} y)(1 + \delta), \quad |\delta| \leq u, \quad op = +, -, *, /, \tag{4.24}$$

where $fl(a)$ is the floating point representation of a real number, a, and u is the unit roundoff

specific to the floating point number system with base, β, and precision, t,

$$u = \frac{1}{2}\beta^{1-t}. \tag{4.25}$$

For double precision floats with base $\beta = 2$ (binary) and precision $t = 53$, u is

$$u = \frac{1}{2}2^{1-53} = 2^{-53} = 1.110\,223\,025\,\ldots \times 10^{-16}. \tag{4.26}$$

Specific values of u are also called the machine epsilon, ϵ_m [127]. In other words, the standard model says the rounding error on the floating point result of $x\,op\,y$ in double precision is bounded by $|\epsilon_m \cdot (x\,op\,y)|$

$$\begin{aligned} fl(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta), \quad |\delta| \leq \epsilon_m, \quad op = +, -, *, /. \\ \Rightarrow fl(x \operatorname{op} y) \in [(x \operatorname{op} y)(1-\epsilon_m), (x \operatorname{op} y)(1+\epsilon_m)] \end{aligned} \tag{4.27}$$

As a concrete example, double precision addition with an arbitrary number, x, and minimum step value, t_{min}, is considered. The error, δ, on the result is bounded by (4.27)

$$\begin{aligned} \delta &\leq |x+y| \cdot \epsilon_m \\ \delta &= |x+y| \cdot \epsilon_m \quad \text{(worst case)} \\ \delta &= |x+t_{min}| \cdot \epsilon_m. \end{aligned} \tag{4.28}$$

Then, one can find the neighbourhood of x where the error can equal t_{min},

$$\begin{aligned} \delta = t_{min} &= |x+t_{min}| \cdot \epsilon_m \\ \Rightarrow |x| &= \frac{t_{min}}{\epsilon_m} - t_{min} \quad (t_{min} > 0). \end{aligned} \tag{4.29}$$

For illustration, one can pick a much too small value of t_{min} for most purposes (e.g. $t_{min} = \epsilon_m$),

$$\begin{aligned} |x| &= \frac{\epsilon_m}{\epsilon_m} - \epsilon_m \\ |x| &= 1 - \epsilon_m \\ |x| &\approx 1. \end{aligned} \tag{4.30}$$

Therefore, when $|x|$ is near 1, rounding error could completely cancel out the floating point addition with t_{min}, i.e. $x + t_{min} = x$. One can then find the approximate magnitude of x

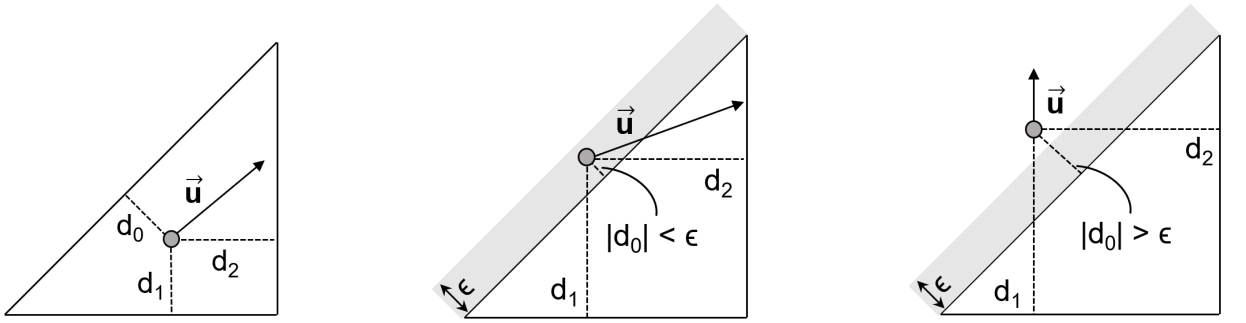
where floating point addition rounding errors approach $t_{min} = 10^{-10}$, as used by EGS_Mesh

$$\begin{aligned} |x| &= \frac{t_{min}}{\epsilon_m} - t_{min} \\ |x| &\approx \frac{t_{min}}{\epsilon_m} \\ |x| &\approx 900720. \end{aligned} \tag{4.31}$$

Since EGSnrc uses centimetres internally, this corresponds to roughly 9 km, which is much larger than the typical mesh simulated by EGS_Mesh (e.g. the ICRP 145 phantoms are 200 cm tall). As long as meshes are on the order of metres or smaller and near the origin (i.e. not 9 km away), this t_{min} value should be sufficient to avoid an infinite loop due to rounding error.

For robustness, Möller-Trumbore is not used for the internal case, as edge and vertex intersections can be missed. As far as the EGSnrc geometry interface is concerned, this should be impossible if the particle is inside a tetrahedron. Though the same issue is present for the external case, a false negative for an external intersection calculation means the next particle is simulated and the simulation continues. An early version of EGS_Mesh using Möller-Trumbore for the internal howfar implementation was prone to infinite loops if an intersection could not be found, requiring a fallback case using ray-plane intersections (Section 4.2.4). Ray-plane intersections are watertight, guaranteeing a intersection if the particle is inside the four face planes. But since ray-plane intersections are required with or without Möller-Trumbore for a conservative implementation, the routine was redesigned to use ray-plane intersections from the start. Quantities obtained from the ray-plane intersections are also useful for recovering from region-position mismatch errors, as is shown later on. Using ray-plane intersections to calculate polygon face intersections is a well-known and straightforward method (see [128]).

The final implementation uses thick planes (or fuzzy planes) for each tetrahedron face. A particle can be considered part of an element if it is close enough to the element (i.e. $d < \epsilon$ for some small ϵ value). For consistency, $\epsilon = t_{min}$, (the minimum howfar bound previously discussed). Four ray-plane intersections are tested using (4.14). There are three potential cases depending on the results of the intersection tests. Figure 4.10 shows each case in 2D for clarity. Case 1, shown in Figure 4.10a, is the "normal" case, where the signed distance from (4.2) to each tetrahedron face plane is positive. Since each distance is positive, the particle is strictly inside the element and can be transported using the minimum distance of the ray-plane intersection equation (4.14) for each side. This is the fast path. Case 2 (Figure 4.10b) uses thick planes to try and transport the particle as if it was inside the element. Since EGSnrc provides the element number the particle is supposed to be inside, boundary crossing calculations are assumed to have occurred already. For example, the



(a) Inside the element (b) Within a thick plane (c) Outside the thick plane

Figure 4.10: Three cases of an interior `howfar` method using thick planes.

previous `howfar` step may have resulted in the particle being slightly outside the destination element due to numerical undershoot [124]. If the particle is close enough to the element (i.e. the magnitude of the minimum negative signed distance to a face plane is smaller than ϵ) and the particle is not travelling away from the thick plane face, transport continues as in Case 1 using the ray-plane intersection algorithm. Finally, Case 3 is an error-recovery case (i.e. the slow path). If the particle is outside the assumed element's thick planes or the particle is travelling away from the element entirely, the particle can no longer be considered inside the assumed element. To recover, the particle is shifted by t_{min} and a search is conducted to find the actual element the particle is inside. First, the immediate element neighbours are checked, and then a general search using `isWhere` is conducted if the particle is not in a neighbouring element. Since this occurs fairly rarely and the final `isWhere` implementation is accelerated using an octree, this case has been observed to have negligible impact on overall performance. An alternative approach is to ignore floating-point issues, as most of the particles will not fall into the last two cases. However, this more conservative algorithm was chosen for the first implementation in an effort to be as accurate as possible.

4.3.5 `medium`

In the mesh input file, each tetrahedron has an assigned medium. During `EGS_Mesh` setup, the `EGS_BaseGeometry::addMedium` method [117] is used to add each unique medium to the list of known media. The corresponding medium offset index is stored for each element and is used to implement the `medium` interface.

4.4 Accelerated egs++ mesh geometry implementation

With the naive implementation complete, the final accelerated version of `EGS_Mesh` is now considered. One should note however, that the naive version, though slow, fully implements

the required EGS_BaseGeometry interface. Turning off the acceleration features and comparing to the naive brute-force implementation can aid debugging and development. However, there are huge performance gains from acceleration that are necessary for EGS_Mesh to have competitive performance with the other transport code mesh implementations.

4.4.1 Octree partitioning

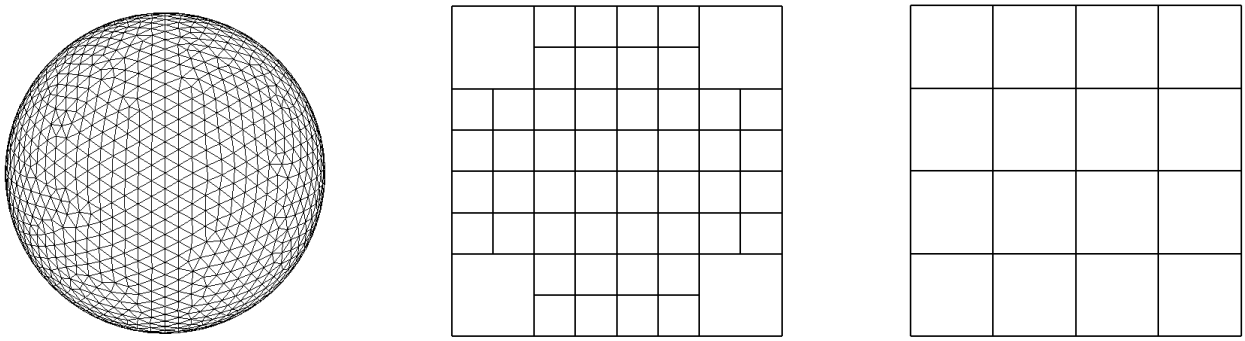
EGS_Mesh uses octrees to accelerate its geometry routines. Given a global axis-aligned bounding box, octrees recursively divide the box into smaller boxes [129]. Octrees are a relatively simple example of what the ray-tracing community calls a spatial partitioning technique [130]. Other spatial partitioning techniques include kd-trees and BSP trees (both considered generalizations of the plain octree). It is possible that using another spatial partitioning technique could increase the performance of EGS_Mesh, but the octree's straightforward implementation makes it a good initial choice of acceleration structure. For future reference, an efficient kd-tree implementation with extensive commentary can be found in [127].

The goal of spatial partitioning is to exclude as many elements as possible from the intersection search. For example, if a particle intersects a single octant, only elements in that octant need to be tested for intersection. Compared to brute-force search, this can yield multiple order of magnitude improvements in simulation efficiency [130].

EGS_Mesh uses two element octrees: one octree containing every element, and another just for surface elements. This setup is also used by the PHITS tetrahedral mesh implementation [77]. Having a second octree just for surface elements does slightly increase the simulation's memory use, but the number of surface elements is usually much less than the total number of elements. This means the surface octree can have much larger octant bounding boxes and therefore exhibit superior exterior ray-mesh intersection performance. The surface octree will tend to improve performance more and more as the total number of elements increases and the volume octree grows deeper at a faster rate than the surface octree. Examples of volume and surface octrees for a sample mesh are shown in Figure 4.11.

The EGS_Mesh octrees use the simplest subdivision strategy of halving the box on each axis to obtain eight octants of equal volume. This process recurses in each octant until a minimum number of intersected elements is obtained. So child octants may have different depths if one octant intersects more elements than another. Other subdivision strategies exist for creating more balanced trees [127]. For example, a tree could search for a division where a roughly equal number of elements are on either side. For the initial implementation, fixed leaf octant element thresholds were set, following the default values of the PHITS implementation (100 for the surface octree and 200 for the volume octree) [77].

Spatial partitioning is not the only way to accelerate ray-tracing geometry routines. The major alternative is to use a bounding volume hierarchy [127, 131]. Both approaches aim



(a) Input mesh (b) Volume octree (c) Surface octree

Figure 4.11: Volume and surface octrees for an example mesh.

to reduce the number of elements checked for intersection, but bounding volume hierarchies can overlap in space and elements usually only belong to a single bounding volume [131]. In contrast, spatial partitioning schemes usually have elements that overlap the partition and so elements can belong to more than one subdivision [131]. This is certainly the case for the `EGS_Mesh` octrees, as elements cut by an octree subdivision plane are assigned to both octants. Usually, input meshes are very dense and most elements are connected to other elements. This dense, connected mesh means that strict partitioning to construct a bounding volume hierarchy would become more expensive, compared to say, a ray-tracing scene with objects scattered around the domain.

As for which acceleration technique is best suited for `EGS_Mesh`, spatial partitioning appears to be the best choice for dense tetrahedral meshes. In their rendering textbook, Pharr, Jakob and Humphreys compare bounding volume hierarchies to spatial partitioning using kd-trees as an example [127], writing

> BVHs are more efficient to build than kd-trees, which generally deliver slightly faster ray intersection tests than BVHs but take substantially longer to build. On the other hand, BVHs are generally more numerically robust and less prone to missed intersections due to round-off errors than kd-trees are.

Taking octrees as a special case of kd-tree, it is a good tradeoff to have increased construction time versus runtime performance, because the `EGS_Mesh` octrees are initialized when the simulation begins and then used as-is for the entire simulation.

To properly implement particle transport using an octree, a few further geometry routines are required. First, an algorithm to find the closest point on an axis-aligned bounding box, required for `hownear`, is considered in Section 4.4.2. Next, a ray-box intersection test used to accelerate `howfar` is shown in Section 4.4.3. Section 4.4.4 implements a test for whether a given tetrahedron lies inside an axis-aligned bounding box. Finally, accelerated `isWhere`, `hownear`, and `howfar` routines are implemented using these basic algorithms (Sections 4.4.5, 4.4.6, and 4.4.7 respectively).

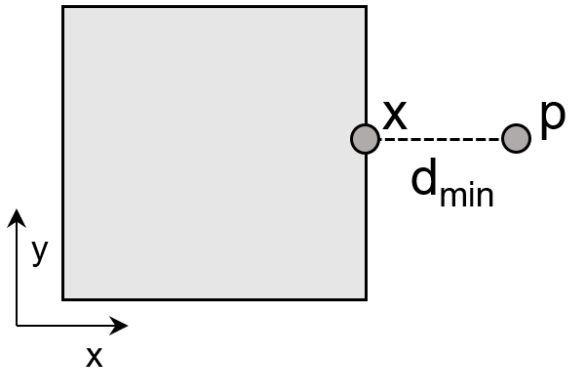


Figure 4.12: Closest point on an axis-aligned bounding box for a given point.

4.4.2 Closest point on axis-aligned bounding box to point

Given a point, $\vec{p}$, and an axis-aligned bounding box, the goal is to find the closest point, $\vec{x}$, on the box to $\vec{p}$. Since the box is axis-aligned (the box extents are parallel to the x, y and z axes), the closest point can be found by clamping each point coordinate, p_i, to the box's boundaries, (i_{min}, i_{max}), for each axis [119]

$$x_i = \text{clamp}(p_i, i_{min}, i_{max}) = \begin{cases} i_{min} & \iff p_i < i_{min} \\ p_i & \iff i_{min} < p_i < i_{max} \\ i_{max} & \iff p_i > i_{max} \end{cases} \quad i \in \{x, y, z\}. \tag{4.32}$$

A 2D version of this geometry is shown in Figure 4.12. If the query point, $\vec{p}$, is inside the box, this algorithm considers $\vec{p}$ itself to be the closest point. As implemented in `EGS_Mesh`, this algorithm is only used if the point is already known to be outside the box, avoiding this potential ambiguity.

4.4.3 Ray-axis-aligned bounding box intersection

To implement `howfar` with an octree made of axis-aligned bounding boxes, a ray-axis-aligned box intersection algorithm is required. Given a point, $\vec{p}$, with direction, $\vec{u}$, the goal is to find the intersection with an axis-aligned box. The 2D analogue of this geometry is shown in Figure 4.13. A common way to implement this test uses three ray-slab intersections [119] [127]. Each slab (made of two parallel planes) is checked for intersection. An intersection test for the y-slab is shown in Figure 4.14. The ray does not intersect the y-slab's near plane, but it does intersect the far plane. Slab intersection t-values can be found using the ray-plane intersection equation (4.14) (rearranged to avoid the minus sign), with slab plane

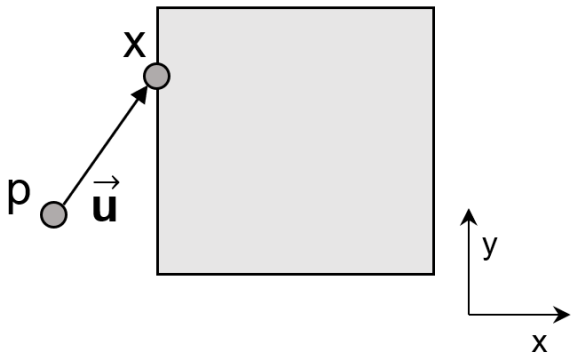


Figure 4.13: Ray-axis-aligned bounding box intersection.

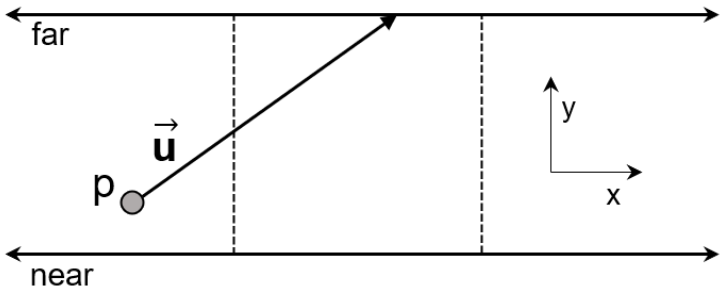


Figure 4.14: Single ray-slab intersection.

normal, $\vec{n}$, plane point, $\vec{a}$, ray position, $\vec{p}$, and ray velocity, $\vec{u}$,

$$t = \frac{\vec{n} \cdot (\vec{a} - \vec{p})}{\vec{n} \cdot \vec{u}}.$$

The general equation can be simplified for each component since each plane is axis-aligned. For example, the normal vector of the x-slab is $(1, 0, 0)$, so the y and z components of the dot product will be zero. So the intersection distance, t_i, for each component, i, is

$$t_i = \frac{a_i - p_i}{u_i}, \quad i \in \{x, y, z\}. \tag{4.33}$$

If the ray is parallel to the slab, and not inside the near and far planes, there is no intersection with the box (this check is done before the division by u_i, since $u_i = 0$ if the ray is parallel to the slab). If the first slab test concludes there might be an intersection with the box, the next slab is tested, and so on. For a test using multiple slabs, an intersection is found if the intersection distance intervals, $[t_{near}, t_{far}]$, for each test overlap, as shown in Figure 4.15. If an intersection is found, the distance to the box is the smallest intersection distance, t_{min}, and the closest point, $\vec{x}$, can be found using $\vec{x} = \vec{p} + t_{min}\vec{u}$. If any slab test distance intervals are instead found to be disjoint, the ray does not intersect the box. An improved ray-box intersection test accounting for some robustness issues in this basic test is considered

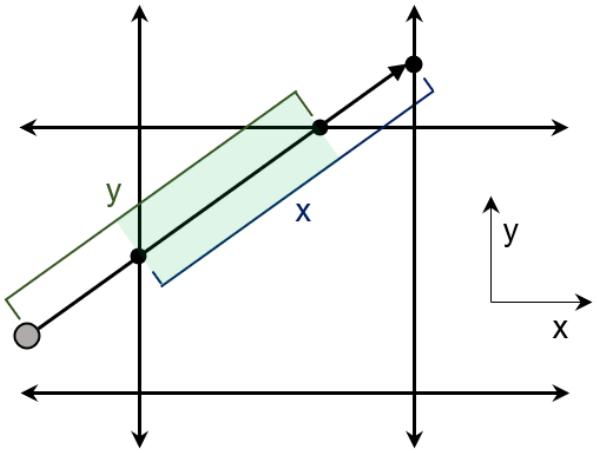


Figure 4.15: Overlap of two ray-slab intersections (adapted from Figure 3.2 of [127]).

in Section 3.9.2 of [127], but this has not been implemented in `EGS_Mesh` yet.

4.4.4 Triangle-axis-aligned bounding box overlap

During octree construction, each bounding box needs to determine which tetrahedrons are intersected by the box. Even if a tetrahedron is not contained fully within an octant, it should be included if there is any overlap at all to properly implement `isWhere` and the other geometry methods. Since each tetrahedron has four triangular faces, this can be done using four triangle-box overlap tests per tetrahedron. Triangle axis-aligned bounding box overlap testing is conventionally implemented using multiple separating axis tests [119]. Separating axis tests are a general way to test whether two convex objects overlap (i.e. the approach can be used for any convex polyhedra) [119]. Both objects are projected onto a potential separating axis, and if the projection intervals overlap, the axis is not a separating axis and the next potential separating axis is tested, if any. If a separating axis is found (as shown in Figure 4.16), it is concluded the objects do not overlap. But if no separating axis can be found, the test instead concludes the two objects do overlap. To construct a separating axis test for two convex shapes, A and B, it is known the minimum number of axes required to check are [119]:

- Axes parallel to the face normals of A,
- Axes parallel to the face normals of B, and
- Axes parallel to the cross products of the edge vectors of A with the edge vectors of B.

So for an axis-aligned box, A, and triangle, B, there are 13 total axes to check:

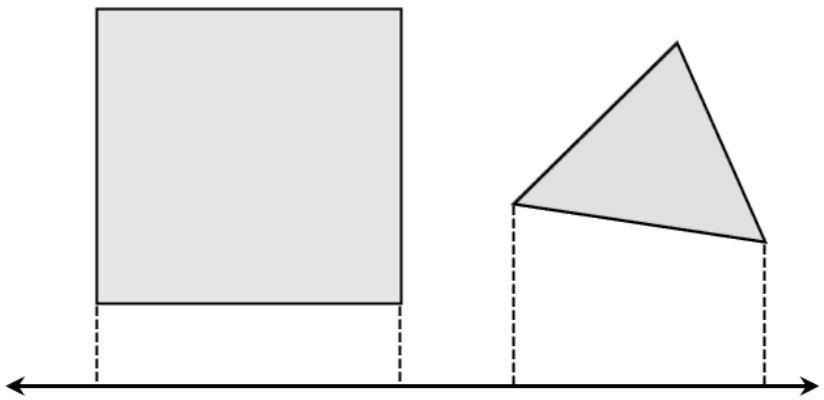


Figure 4.16: A separating axis for a box and triangle.

- Three axes parallel to the face normals of the box (i.e. in the x, y, and z directions),
- One axis parallel to the triangle face normal, and
- Nine axes resulting from the cross products of the three box edge vectors (the three unique x,y,z box edges are used four times to make the box) with the three triangle edge vectors.

Conventional wisdom holds the edge-edge tests are performed first, then the box face normal tests, and finally the triangle face normal test [119].

The three box face normals (aligned with the Cartesian axes) can be tested by transforming the triangle coordinates relative to the centre of the box. Then, for each component, if all vertices are outside the box bounds, a separating axis has been found and there is no overlap. Given a box centre point, $\vec{o}$, and triangle with vertices, $\vec{a}$, $\vec{b}$, $\vec{c}$, the transformed triangle vertices, $\vec{v_0}$, $\vec{v_1}$, $\vec{v_2}$, are $(\vec{a}-\vec{o})$, $(\vec{b}-\vec{o})$, $(\vec{c}-\vec{o})$, respectively. Then, if either of the following tests are true for a box with extents (half-widths), e_i, for $i \in \{x, y, z\}$, a separating axis i has been found

$$\begin{aligned} v_{ni} &< -e_i \qquad \forall n \in \{0, 1, 2\}, \\ v_{ni} &> e_i \qquad \forall n \in \{0, 1, 2\}. \end{aligned} \tag{4.34}$$

Next, the triangle face normal can be tested using a box-plane overlap test. If the box does not overlap the triangle plane, the box cannot overlap the triangle. The test geometry for a triangle face plane, P, is shown in Figure 4.17. First, the distance $|s|$ in terms of $\vec{n}$ from the box centre point to the plane is found using the point-plane distance equation (4.3). Next, the distance in terms of $\vec{n}$ from the box centre point to the furthest box vertex, $|r|$, is found. If $|s| < |r|$, the box intersects the plane (i.e. the distance from the centre to the plane, $|s|$, is less than the distance to the further vertex, $|r|$). The distance $|r|$ from the box

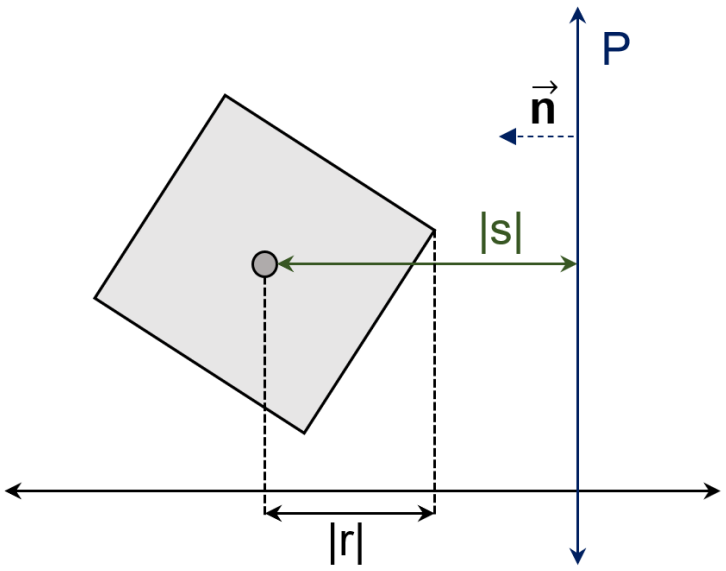


Figure 4.17: Box-plane separating axis test.

centre, $\vec{o}$, to a vertex, $\vec{v}$, in terms of $\vec{n}$ is the projection (dot product) of $\vec{o}-\vec{v}$ onto $\vec{n}$

$$|r| = (\vec{o} - \vec{v}) \cdot \vec{n}. \tag{4.35}$$

Now, each of the eight box vertices, $\vec{v_n}$, can be expressed in terms of the box extents, e_x, e_y, e_z, box face normals, $\vec{f_x}$, $\vec{f_y}$, $\vec{f_z}$ and box centre, $\vec{o}$

$$\vec{v_n} = \vec{o} \pm e_x\vec{f_x} \pm e_y\vec{f_y} \pm e_z\vec{f_z}, \quad 0 \leq n \leq 7. \tag{4.36}$$

Substituting back into (4.35), the following is obtained

$$\begin{aligned} |r| &= (\vec{o} - (\vec{o} \pm e_x\vec{f_x} \pm e_y\vec{f_y} \pm e_z\vec{f_z})) \cdot \vec{n} \\ |r| &= (\pm e_x\vec{f_x} \pm e_y\vec{f_y} \pm e_z\vec{f_z}) \cdot \vec{n} \\ |r| &= \pm(e_x\vec{f_x} \cdot \vec{n}) \pm (e_y\vec{f_y} \cdot \vec{n}) \pm (e_z\vec{f_z} \cdot \vec{n}). \end{aligned} \tag{4.37}$$

To maximize r, the absolute value of each component term can be taken

$$|r| \leq |e_x\vec{f_x} \cdot \vec{n}| + |e_y\vec{f_y} \cdot \vec{n}| + |e_z\vec{f_z} \cdot \vec{n}|. \tag{4.38}$$

And since the box is axis-aligned, the face vectors are the Cartesian unit vectors (e.g. $\vec{f_x} \cdot \vec{n} = n_x$). Assuming positive box extents, the final equation for $|r|$ is

$$|r| = e_x|n_x| + e_y|n_y| + e_z|n_z|. \tag{4.39}$$

Again, after finding $|s|$ and $|r|$, the box overlaps the plane if $|s| < |r|$.

The final category of axes to test are the nine edge-edge cross products. Each cross product results in a potential separating axis. Both the box and triangle are projected onto the axis and checked for overlap. The box's projection radius, $|r|$, on an axis, $\vec{a}$, is given by (4.38), substituting $\vec{a}$ for $\vec{n}$

$$|r| = |e_x \vec{f_x} \cdot \vec{a}| + |e_y \vec{f_y} \cdot \vec{a}| + |e_z \vec{f_z} \cdot \vec{a}|. \tag{4.40}$$

The vectors from the box origin to the triangle vertices are $\vec{v_0}$, $\vec{v_1}$, and $\vec{v_2}$ and the projection of each relative vertex vector onto the axis is $\vec{v} \cdot \vec{a}$. If the projection intervals of the box and triangle onto $\vec{a}$ are disjoint, $\vec{a}$ is a separating axis. This is the case if either of the following tests are true

$$\begin{aligned} \vec{v_n} \cdot \vec{a} < -r \quad & \forall n \in \{0, 1, 2\}, \\ \vec{v_n} \cdot \vec{a} > r \quad & \forall n \in \{0, 1, 2\}. \end{aligned} \tag{4.41}$$

The triangle-box intersection test is now complete. If all 13 possible separating axes are tested without finding a true separating axis, the box and triangle overlap.

A potential issue with the separating axis test can occur for the separating axes created by an edge-edge cross product. If the two edges are parallel, the resulting vector is the zero vector and so could mistakenly be counted as a separating axis [119]. There are a few ways to try and recover from this issue, such as constructing a new potential separating axis which is perpendicular to the two parallel edge vectors [119]. The `EGS_Mesh` implementation instead skips the axis, and continues with testing the remaining axes. This choice has the advantage of simplicity, but skipping an axis does risk potential false positive intersections (i.e. an octant may include elements which do not overlap it). However, any false positives should only lead to a theoretical slight performance loss from having to check elements which do not intersect the octant. A far worse issue would be false negatives — tetrahedrons actually intersected by the box but not detected as intersecting, which could affect the final simulation result. Now, with these additional geometry routines in hand, the final accelerated `EGS_Mesh` implementation is considered.

4.4.5 Octree-accelerated `isWhere`

The mesh `isWhere` implementation is accelerated using the element volume octree from Section 4.4. Thanks to the triangle-box overlap algorithm from Section 4.4.4, each octant can find the list of tetrahedrons that overlap it. The naive implementation from Section 4.3.1 used a brute-force search over all elements to find the bounding element (if any) for a query point. With the octree, random position search, which was $O(n)$ for brute-force, becomes a $O(\log_8 n)$ algorithm in theory as the vast majority of elements are completely excluded from the search.

First, the query point is tested against the volume octree's root octant (i.e. the mesh's

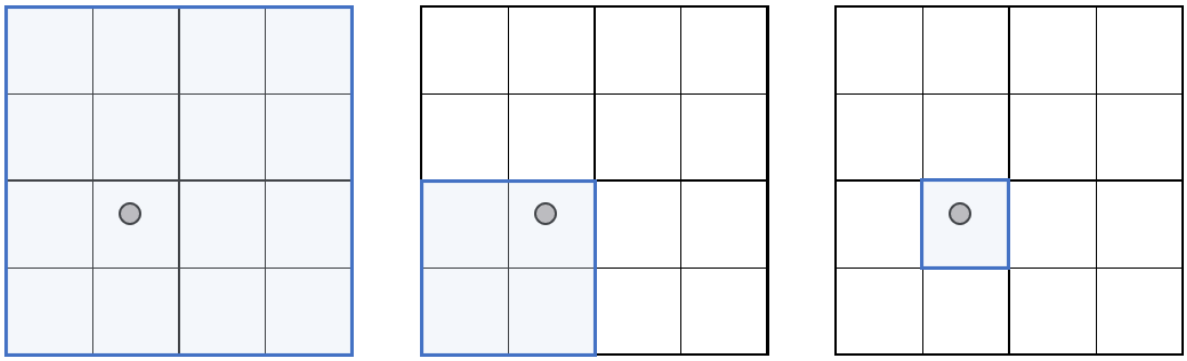


Figure 4.18: The `isWhere` octree search process.

global bounding box). If the point is outside the mesh's global bounding box, it cannot be inside any mesh elements and the search ends. If the point is inside the root octant, the appropriate child octant (based on the point position) is then tested. The search descends the tree until a leaf node is reached. Then, each element bounded by the leaf node is checked using the query point. The maximum number of elements to check is determined by the octree's maximum number of bounded elements. This process is shown in Figure 4.18. Since each octant has a list of overlapping tetrahedrons, if the leaf node search fails to find an element that contains the query point, the search can end once a leaf node is processed (i.e. the search does not have to consider neighbouring octants). Section 4.5 shows the performance improvement obtained using this accelerated version of `isWhere`.

4.4.6 Octree-accelerated `hownear`

There are two `hownear` cases: internal and external. The internal implementation from Section 4.3.2 cannot be accelerated using an octree, since the bounding element is known and all that is required are four point-plane distances. However, the external case requires further optimization. The naive external `hownear` implementation calculates the minimum distance to all surface elements. This can be improved thanks to the octree of surface elements and an approximation allowed by the `hownear` specification from the EGSnrc manual [4]. In Section 3.6, the manual notes that for "complex geometries, the mathematics of `HOWNEAR` can become difficult and sometimes almost impossible! If it is easier for the user to compute some lower bound to the nearest distance, this could be used in `HOWNEAR`" [4]. This simplification is crucial for quickly determining `hownear` for an arbitrary mesh. Without allowing for a lower bound, the `hownear` search becomes more complicated, since the bounding box containing the point is not guaranteed to have the closest element, as shown in Figure 4.19. But allowing for a lower bound means only one leaf octant needs to be visited, instead of continuing to search the neighbouring octants. The `hownear` lower bound in an leaf octant is found by first determining the minimum distance to all octant elements. Then, the minimum distance,

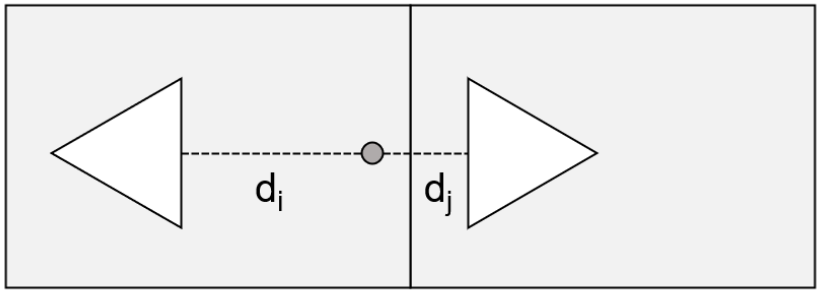


Figure 4.19: The minimum distance to a tetrahedron in the query point's octree bounding box is not necessarily the global minimum distance to an element for `hownear`.

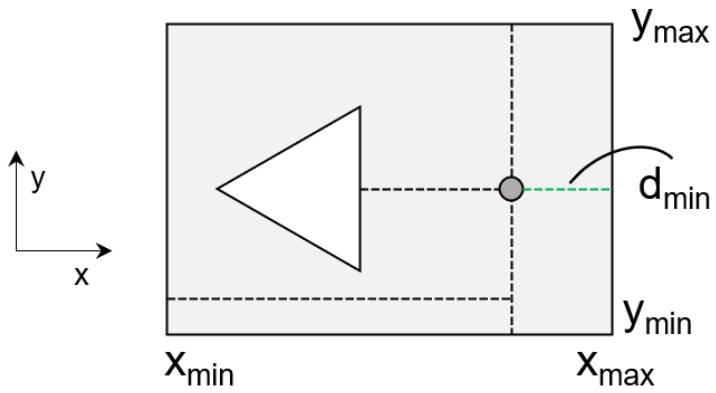


Figure 4.20: Finding the lower bound for a point in an octree bounding box (2D view).

d_{min}, to the octant boundary is found using (4.42), given bounding box extents, i_{min}, i_{max}, and particle position vector, $\vec{p}$,

$$\begin{aligned} d_i &= \min(p_i - i_{min}, i_{max} - p_i), \quad i \in \{x, y, z\} \\ d_{min} &= \min(d_x, d_y, d_z). \end{aligned} \tag{4.42}$$

After both potential minimums (to an element, and to the octant boundary) are found, the smaller of the two values is the final value of `hownear`, as shown in Figure 4.20. The naive linear search has become a search over the maximum number of elements in an octree leaf node. Finally, since the surface element octree only partitions the mesh's bounding box, particles can exist outside the octree. In that case, the minimum distance to the mesh's global bounding box is found using the algorithm from Section 4.4.2. The accelerated version of `hownear` is now complete. A performance comparison of the naive version of `hownear` and the accelerated version can be found in Section 4.6.

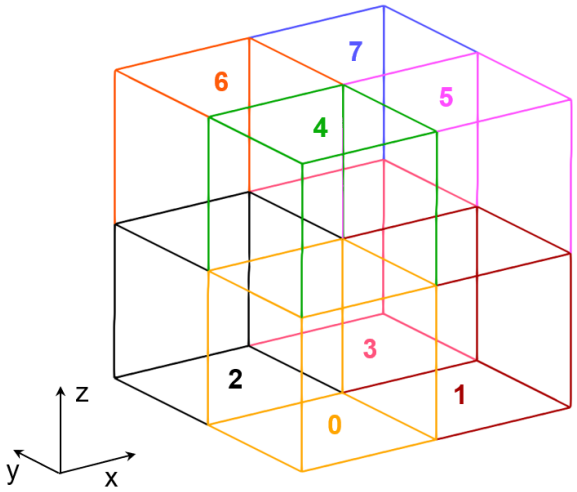


Figure 4.21: Octree child numbering.

4.4.7 Octree-accelerated `howfar`

After implementing improved versions of `isWhere` and `hownear`, it is time to implement a better `howfar`. The naive external case of `howfar` from Section 4.3.3 requires a loop over all surface elements and so is the most important case to optimize. The internal case of `howfar` from Section 4.3.4 is already constant-time for the fast path, and so is not optimized further. A performance test for this version of `howfar` compared to the naive version is given in Section 4.7.

Ray-tracing an octree is a common operation in computer graphics. There exist many different ray-octree traversal schemes, e.g. [132, 133, 134]. This work uses a simple top-down traversal method, which is probably less efficient than the wealth of more complex schemes. In other words, further optimizations are possible at a later date. The `EGS_Mesh` surface octree, used to implement the external case of `howfar`, can have varying suboctant depth. Octree nodes stop subdividing when a set number (100) of elements are bounded by the node, so two neighbouring octants may have different volumes.

The ray-octree intersection routine starts with the root octree node. If the ray does not intersect the root node's bounding box (Section 4.4.3), it cannot intersect any child nodes and `howfar` can exit with no intersection. However, if the ray does intersect the root node, the search process begins. Each octant is either a parent or leaf, and the search progresses according to the type of octant.

Given a parent octant, the first intersected child is determined using the ray intersection point. The eight octree children are indexed by their relative positions as in Figure 4.21. For example, the child octant at index 0 bounds the region of the parent octant with centre point, c, where $d \leq c_d$, $\forall d \in \{x, y, z\}$. So the first octant to search can be determined using three comparisons against c_x, c_y and c_z given the intersection point. If there are

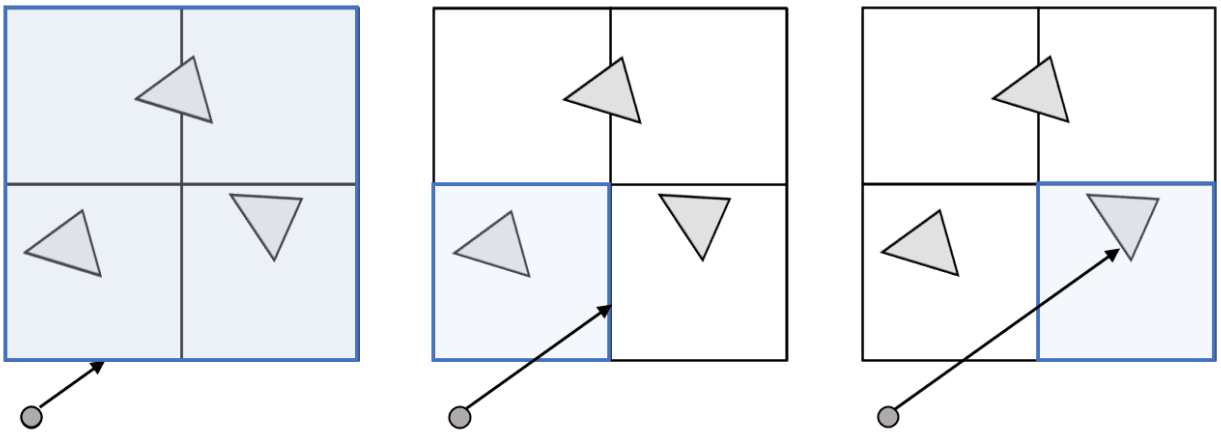


Figure 4.22: The octree search process used by `howfar`.

no element intersections found in the first child octant, intersection tests are performed using the remaining child octants. If any other child octants are intersected by the ray, they are searched in order of minimum intersection distance. An example octree search as implemented in `howfar` is shown in Figure 4.22. If the octant is a leaf octant, all bounded surface tetrahedron faces are checked for intersection (4.5). If an intersection is found, the search still has to check all bounded tetrahedrons in case there is a closer intersection. Once an intersection is found in a leaf node, `howfar` is complete, since leaf octants are searched in order of minimum distance to the ray. Given the surface octree decomposition, `howfar`'s complexity is $O(\log_8 n)$. But unlike `hownear`, which can terminate as soon as a single leaf node has been completely searched, `howfar` has to continue searching until either an intersection is found or the ray exits the octree. The worst case for `howfar` is having to traverse many octants without finding any intersections (e.g. the ray is travelling parallel to many surface tetrahedrons but not far enough away to be excluded from the search).

4.5 Accelerated `isWhere` performance testing

The performance of the naive brute-force `isWhere` from Section 4.3.1 was compared against the accelerated version of Section 4.4.5 that uses an octree. Transport simulations using sphere meshes of increasing resolution (shown in Figure 4.23) were conducted using both implementations to check the performance of the naive version of `isWhere` against the accelerated version. The test geometry is a mesh sphere of 1 cm radius made of water (`H2O521ICRU` from the EGSnrc media file `521icru.pegs4dat`) in a 2 cm vacuum cube. An isotropic 2 cm cube photon source was used to generate particles both inside and outside the mesh sphere. The total number of elements and the number of surface elements for each mesh is given in Table 4.1. The simulation geometry is shown in Figure 4.24. Figure 4.24a shows the coarsest sphere mesh (roughly 700 tetrahedrons) in the vacuum cube while Figure 4.24b shows the

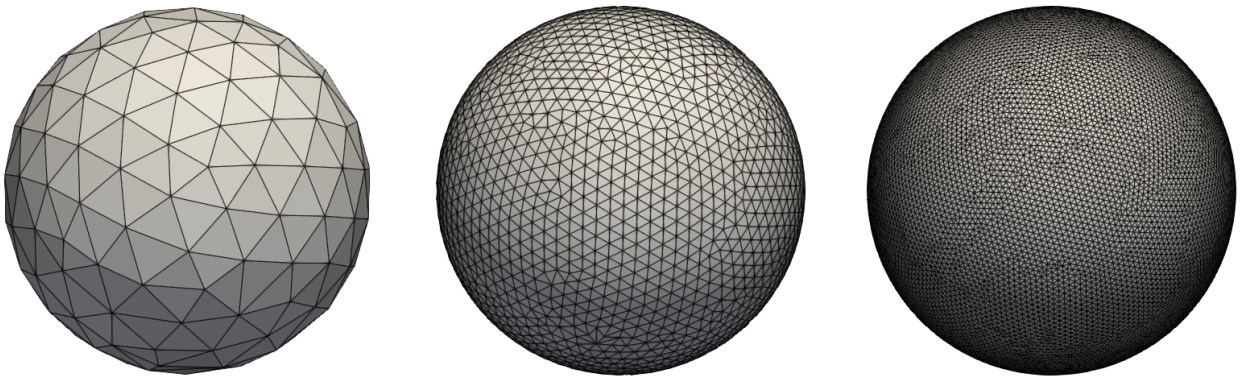


Figure 4.23: Three sphere meshes of increasing resolution

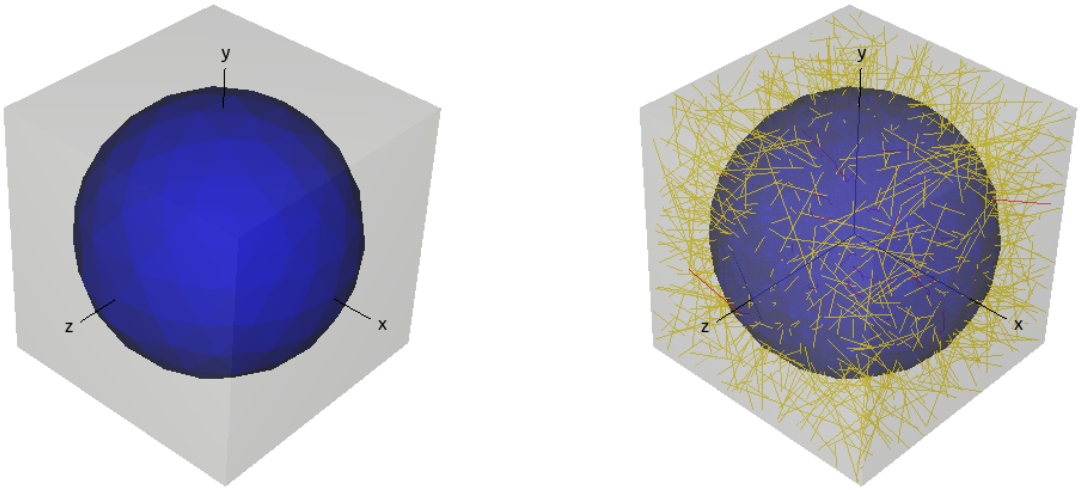
Table 4.1: Element information for performance test meshes

N elements	N surface elements
690	318
11,070	2,112
31,115	4,242
109,495	9,960
442,046	25,198
1,033,252	44,448
3,585,980	100,088
10,118,604	203,794

particle tracks from 1000 primary photons generated throughout the geometry.

To emphasize the performance effect of `isWhere`, the final accelerated implementation of `EGS_Mesh` was used (i.e. `hownear` and `howfar` both use their final accelerated implementations), but the brute-force `isWhere` was swapped in for the naive timings. A single core of an Intel Core i5-4690 @ 3.50GHz computer with 8 GB of RAM was used for the simulations. A total of 10^5 1 MeV photon histories were simulated. Simulation setup and finalization time was neglected, so only the time taken to actually simulate the particles was counted (i.e. the time to simulate a single batch of 10^5 particles). The default EGSnrc transport parameters were used, with energy cutoffs set to 10 keV kinetic.

The performance results are shown in Figure 4.25. The average of five measurements was taken and the standard deviation for each measurement is less than 5%. As Figure 4.25 shows, simulation performance using naive `isWhere` diminishes greatly as the number of mesh elements grows. Meanwhile, performance of the accelerated version of `isWhere` also diminishes as the number of mesh elements grows, but to a much smaller degree. For all measured cases, the accelerated implementation shows better performance. This will likely always be the case unless the mesh has a very small number of elements, where the two algorithms should have roughly equal performance. Comparing the runtimes of the smallest



(a) Sphere mesh in vacuum box (b) Sphere mesh with particle tracks

Figure 4.24: Performance test geometry for `isWhere`.

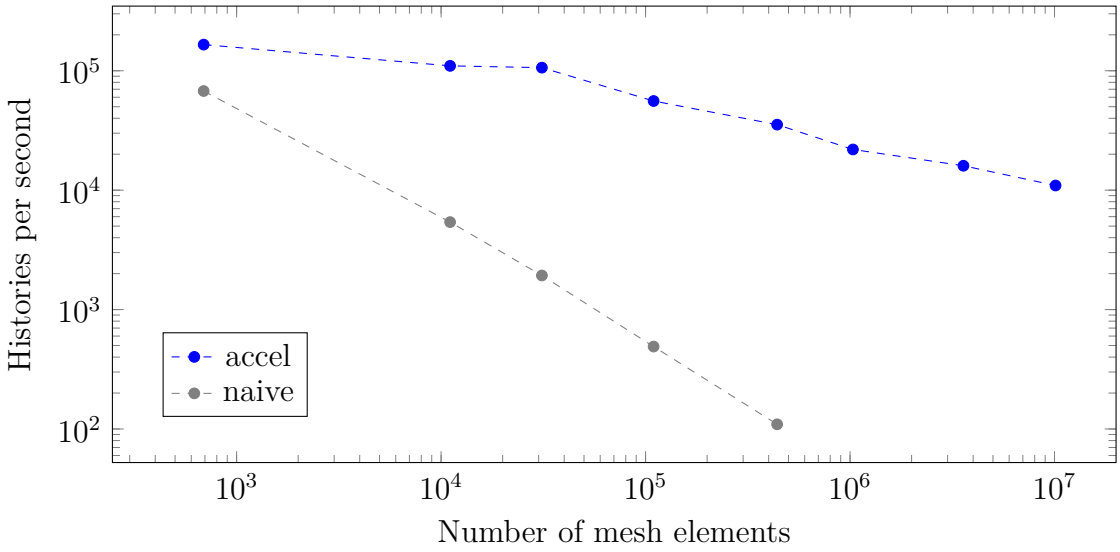


Figure 4.25: Performance of the naive and accelerated `isWhere` implementations.

and largest meshes simulated by both implementations, the runtime of the smaller mesh for the naive implementation is roughly 600 times faster than the runtime for the largest mesh (the larger mesh has around 640 times more elements than the smallest mesh). This is in line with the expected result, since naive `isWhere` has linear complexity for the total number of mesh elements. For the accelerated implementation, the runtime of the smallest mesh is only around five times faster than for the largest mesh. This is a larger difference than predicted by pure algorithm complexity, since given the $\log_8 n$ octree lookup algorithm, the predicted difference is 1.99 times. However, the measured value is the same order of magnitude as the predicted difference. The additional performance hit for larger meshes is possibly caused by increased cache misses, because as the number of mesh elements gets larger, the likelihood a given element is in cache decreases. Cache behaviour is known to have a strong effect on CPU performance [135]. The accelerated `isWhere` version is also shown to scale to at least 10^7 elements, allowing users to simulate much larger meshes than the naive version.

To ensure the accelerated implementation does not affect the end result, the results for the sphere mesh with 4.4×10^5 elements using both implementations were compared. That mesh was used because it is the largest mesh simulated by both implementations. The simulation output files were bitwise identical (i.e. element doses and uncertainties were exactly the same). This may not be the case for longer simulations, since only 10^5 particles were simulated, but this serves as a check the accelerated version is working as expected for this case. A potential difference between the two versions could arise for a query point lying on the intersection of an element boundary and an octree boundary. Due to floating-point issues, it is possible the point could be considered part of two or more elements. In the current implementation, the brute-force search will always return the lowest numbered element, while the octree search could potentially return a higher numbered element, depending on which leaf octant is searched. This could cause a very small difference in dose for the elements in question for the two implementations, but overall any differences from this rare possibility should be negligible. Next, the performance of the accelerated `hownear` implementation is tested.

4.6 Accelerated `hownear` performance testing

The performance of the naive implementation of `hownear` was compared against the octree-accelerated version. The goal is to measure `hownear` performance for the external case, since the internal case of `hownear` is the same for both implementations. The same hardware and sphere meshes from Section 4.5 were used. But the simulation setup was changed from the `isWhere` configuration, since `hownear` is more specialized than `isWhere`. First, `hownear` is only called for electrons (since it is used as input to the EGSnrc electron step algorithm [4]), so a benchmark using a photon source would erroneously conclude that the two algorithms

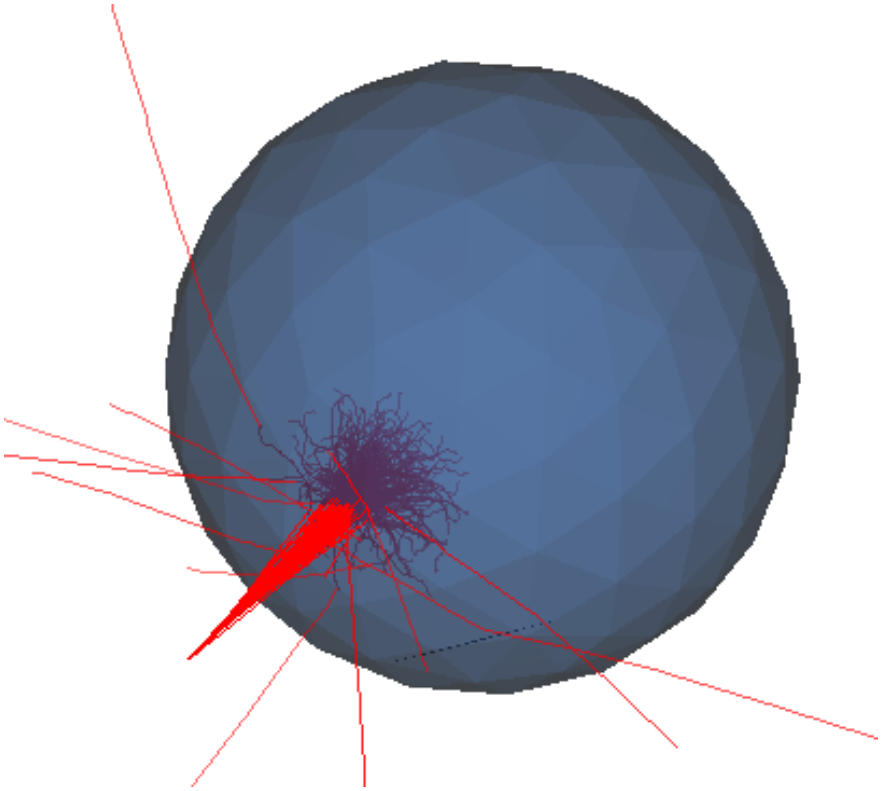


Figure 4.26: Performance test geometry for `hownear`.

have the same performance. And since `hownear` is only called in non-vacuum media, the mesh envelope medium must be changed as well. This benchmark uses another common medium for geometry envelopes: air (e.g. `AIR521ICRU` from `521icru.pegs4dat`). To summarize, for this test case, the external mesh `hownear` routine is responsible for finding a lower bound on the minimum distance to the mesh in any direction once the electrons enter the air box. The naive algorithm searches all surface elements to find the minimum distance, while the accelerated version uses an octree of the surface elements to greatly reduce the number of elements to search. Once electrons enter the mesh, both implementations use the fast internal case of `hownear`. The `hownear` benchmark geometry is shown in Figure 4.26. A pencil beam of 1 MeV electrons from the left hand side of the figure irradiates the mesh sphere of water in a 4 cm cube of air. The electrons only reach a relatively small depth into the sphere. The same transport parameters as for the `isWhere` test were used.

Since the naive algorithm has linear complexity for the number of surface elements, it is expected that simulation performance is reduced by the ratio of mesh surface elements (from Table 4.1). The performance results are shown in Figure 4.27. Like the `isWhere` benchmark, an average of five measurements was taken. The standard deviation for each measurement is less than 2%. As expected, the accelerated version of `hownear` has superior performance for each of the meshes considered and performance decreases at a slower rate than the naive version as the number of mesh elements grows. The runtime ratio of the smallest mesh versus

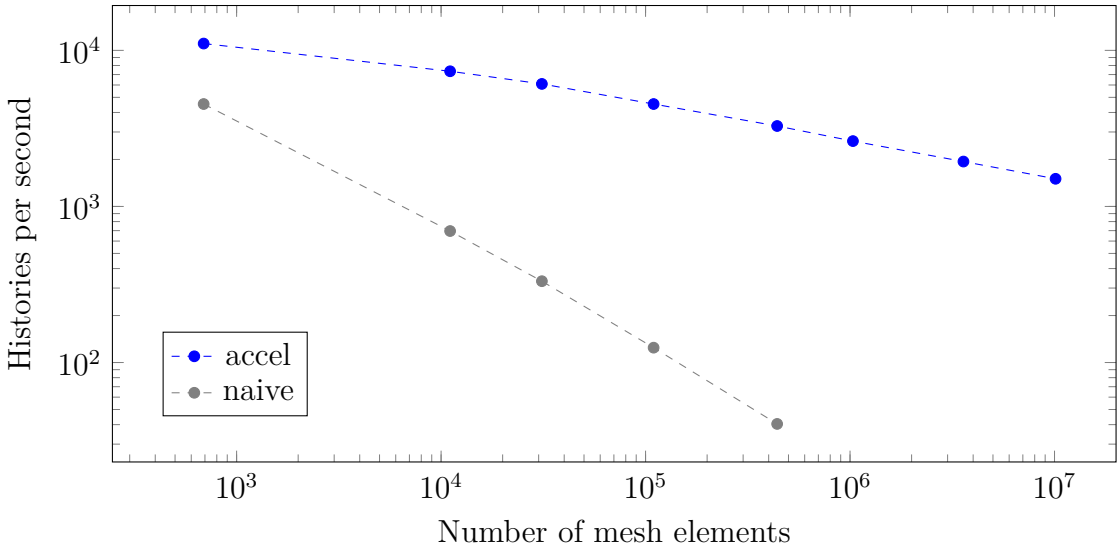


Figure 4.27: Performance of the naive and accelerated `hownear` implementations.

the largest mesh simulated is 3.4 for the accelerated version and 70 for the naive version. These observed results are similar to the theoretical performance differences of the $O(\log_8 n)$ and $O(n)$ algorithms (2.3 and 79, respectively).

A comparison between the results of the two implementations for the same simulation conditions was conducted to ensure that the accelerated version of `hownear` does not noticeably affect the end result. Unlike `isWhere` and `howfar`, the accelerated version of `hownear` will often return a different value than the naive version (i.e. returning a lower bound instead of the exact distance in some cases). This means the random number generator states are almost certain to diverge, meaning a completely bitwise identical result as obtained for the other methods is very unlikely. Instead, the comparison was designed to account for this uncertainty. Simulations using the same mesh for 10^7 histories were conducted with both implementations. Dose ratios for each region were calculated, and to avoid uncertainty-related discrepancies, only ratios with combined uncertainties under 1% were examined. Out of the initial 4.4×10^5 elements, this meant only around 5.5×10^3 elements were retained. Dose ratios for these elements are shown in Figure 4.28. Most of the dose ratios are within a single standard deviation of the expected value, though some are within two or three standard deviations. So, for this simulation, the difference between the two implementations can be explained by statistical uncertainty alone. However, this does not prove that the accelerated version of `hownear` yields statistically equivalent results to the naive version in all cases. Nevertheless, further evidence can be drawn from Chapter 5, since every `EGS_Mesh` simulation there uses the accelerated version of `hownear`.

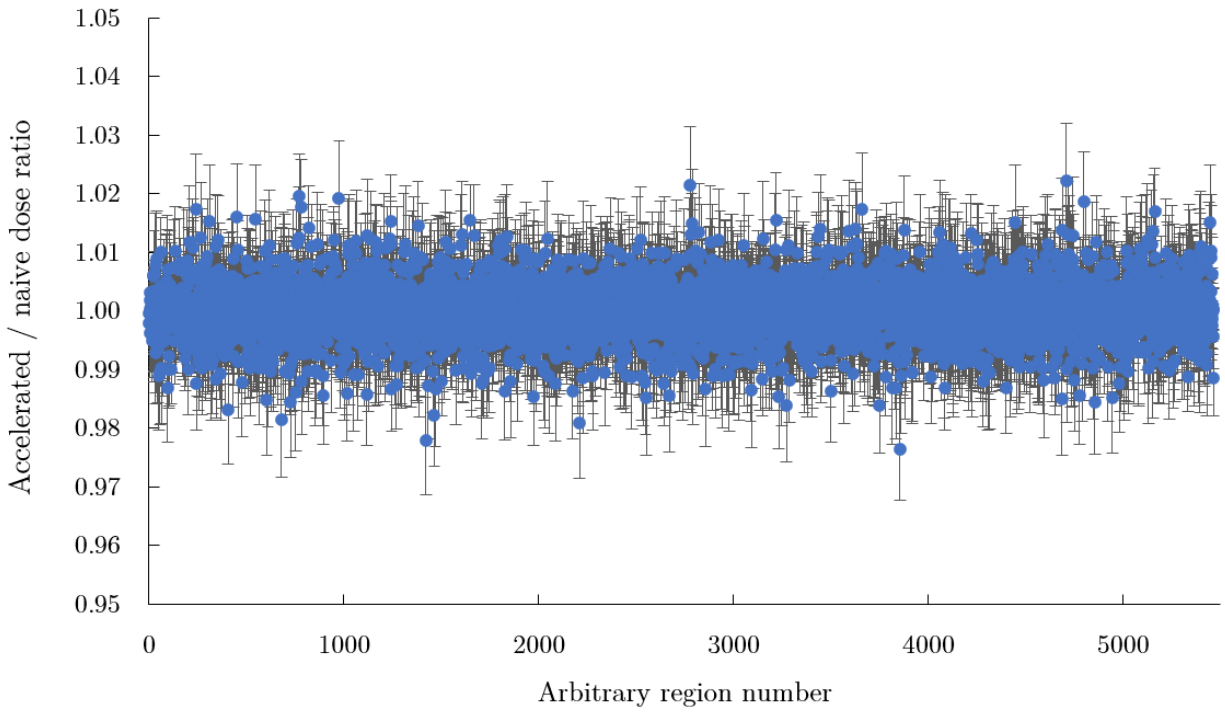


Figure 4.28: Dose ratios between the naive and accelerated versions of `hownear` for geometry regions with less than 1% uncertainty.

4.7 Accelerated `howfar` performance testing

For testing `howfar`, the same simulation setup as the `hownear` test was used. The performance results for an average of five runs are shown in Figure 4.29. The standard deviation for each measurement is less than 1%. Like the `hownear` comparison, the naive implementation has linear complexity for the number of surface elements and the accelerated implementation has logarithmic complexity for the number of surface elements. However, the naive version of `howfar` is faster than naive `hownear`, because `hownear` is forced to check all surface elements to find the minimum distance, while `howfar` can skip any elements facing away from the particles velocity vector since there will not be an intersection. The observed runtime ratio between the largest and smallest mesh is 3.4 for the accelerated version and 70 for the naive version. Like for the `hownear` test, these are similar to the theoretical performance results given each algorithm's complexity.

As done for the other accelerated implementations, a comparison between the results of the naive and accelerated versions of external `howfar` was conducted to ensure the accelerated version obtains comparable results to the naive version. As was the case for `isWhere` (but not `hownear`), the two result files were bitwise identical, meaning the exact same `howfar` results for 10^5 particles were obtained by both implementations for the given mesh. This is not guaranteed for all simulations, whether from rounding error in determining ray-box intersections or intersecting an element edge exactly on an octant boundary, but again serves

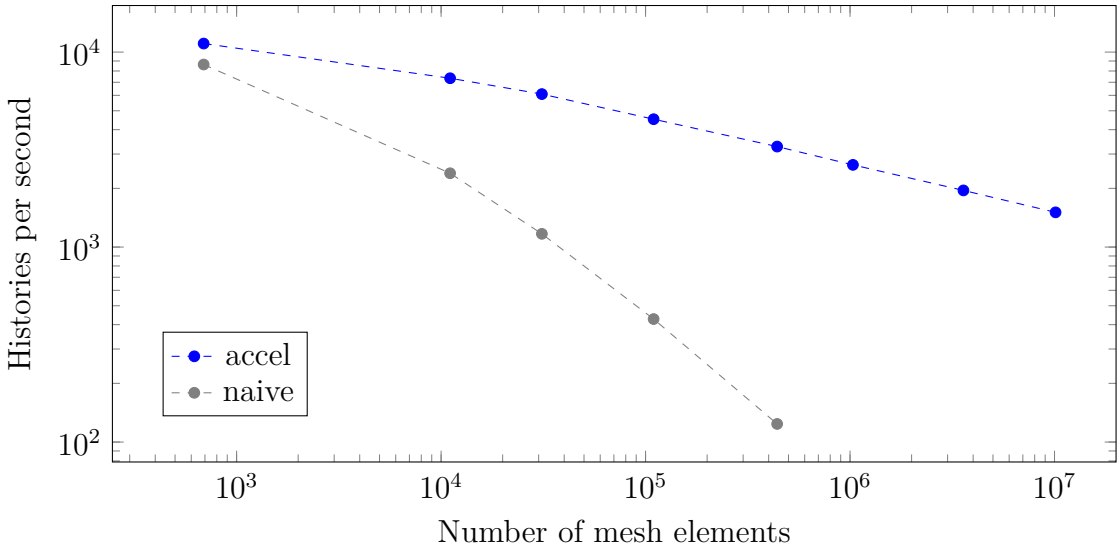


Figure 4.29: Performance of the naive and accelerated `howfar` implementations.

as some verification that the accelerated version is performing as expected.

4.8 `EGS_Mesh` memory use

The `EGS_Mesh` implementation aims to have predictable and stable memory use for a given mesh. There is minimal use of dynamic memory after the `EGS_Mesh` setup steps are finished. A memory profile recorded by the `massif` heap profiler for a typical `EGS_Mesh` simulation using the ICRP 145 adult male phantom is shown in Figure 4.30. Memory usage rises during `EGS_Mesh` setup until a steady state is reached when the actual simulation begins (around 70% on the plot). For this sample, only a small number of particles were simulated to highlight the `EGS_Mesh` setup steps. The profile can be divided into five regions:

1. Mesh data parsing (0-9%),
2. Element neighbour-finding (9-20%),
3. Octree building (20-56%),
4. Face normal generation (56-70%), and
5. Simulation steady state (70-100%).

Memory use starts to rise when the mesh file is parsed into node and element data. Next, there is a local maximum due to the fast neighbour-finding algorithm, which uses a large amount of temporary memory. After the neighbour-finding temporary memory is deallocated, memory use rises again while building the volume and surface octrees. After the octrees are complete, there is another large jump due to explicitly storing mesh face normal

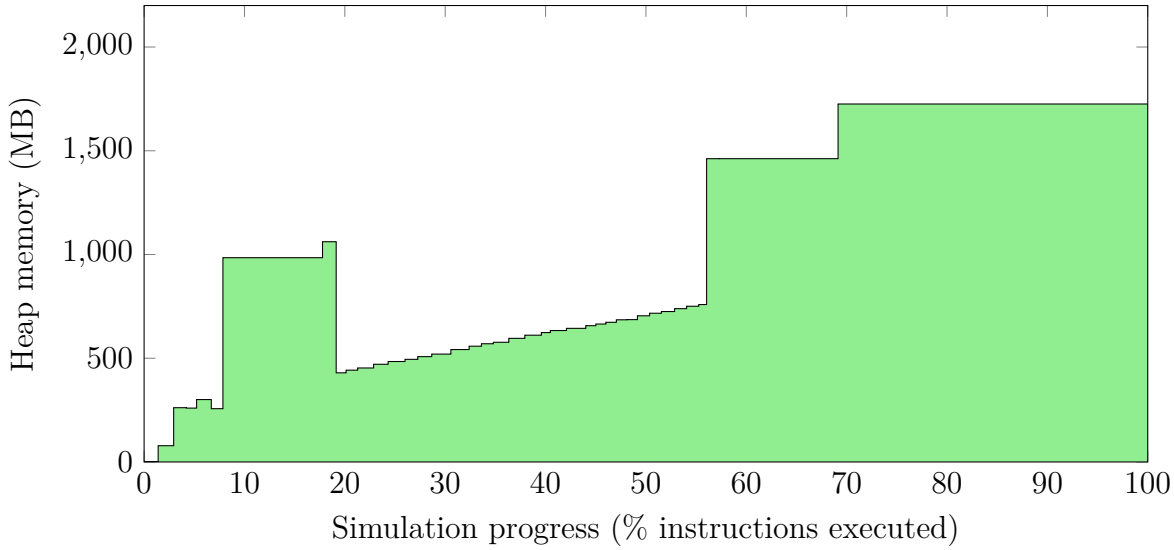


Figure 4.30: EGS_Mesh memory profile for a simulation using the ICRP 145 adult male mesh phantom. The actual simulation begins when the memory profile reaches steady state (near 70%). Only a small number of particles were simulated to emphasize the EGS_Mesh setup steps.

vectors. Finally, after allocating memory for per-region dose scoring, steady state is reached and the simulation begins. Relative memory requirements of different EGS_Mesh simulation components for simulating the ICRP 145 male phantom is shown in Figure 4.31. For this mesh, explicitly storing face normals (which are used to speed up EGS_Mesh geometry routines) is responsible for almost 50% of the total memory. If EGS_Mesh memory usage becomes an issue, changes to how EGS_Mesh stores face normals will most likely have the largest impact. Memory requirements in other transport codes for simulating the ICRP 145 mesh phantoms have been studied by Yeom *et al.* [78]. Table 4.2 gives reported memory requirements for Geant4, MCNP6, and PHITS, as well as for EGS_Mesh. PHITS uses the least amount of memory, while EGS_Mesh uses around 0.5 GB more than PHITS. Geant4 and MCNP6 use roughly an order of magnitude more memory then PHITS and EGS_Mesh. Along with memory, simulation setup time for various codes was also studied. Results for the ICRP 145 male phantoms are given in Table 4.3. For EGS_Mesh, the peak setup time for five runs was taken. Setup time also includes other EGSnrc setup steps such as media initialization (EGS_Mesh-specific setup took 1.7 minutes). PHITS has the fastest setup time at 0.2 minutes, while the other codes take 2-3 minutes to begin the simulation. This difference likely comes from the PHITS implementation's ability to save a given tetrahedral mesh's data as a binary file to reduce the setup time of subsequent simulations [77].

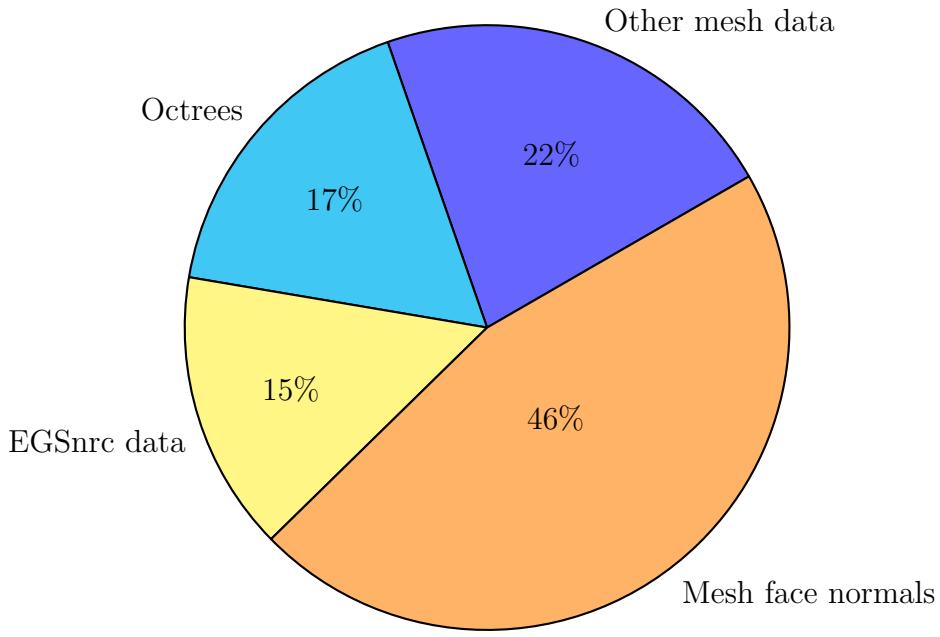


Figure 4.31: Percentage memory breakdown from an `EGS_Mesh` simulation of the ICRP 145 adult male phantom. "Other mesh data" combines node coordinates, element node offsets, element neighbours, and media offsets. The "EGSnrc data" portion is mostly due to the large per-region energy deposition scoring array, which is an optional simulation component.

Table 4.2: Memory required to simulate the ICRP 145 adult male phantom using various transport codes. `EGS_Mesh` memory usage was measured using `massif`. Geant4, MCNP6, and PHITS results are from Yeom *et al.* [78].

Software	Memory (GB)
Geant4	10.6
MCNP6	13.7
PHITS	1.2
`EGS_Mesh`	1.7

Table 4.3: Setup time required to simulate the ICRP 145 adult male phantom using various transport codes. Geant4, MCNP6, and PHITS results are from Yeom *et al.* [78].

Software	Setup time (minutes)
Geant4	3.3
MCNP6	2.3
PHITS	0.2
`EGS_Mesh`	3.3

Chapter 5

Verification results

This chapter describes the methodology and results used to verify the `EGS_Mesh` implementation. Tactics from both theory and practice are employed. First, simulations using `EGS_Mesh` are compared to simulations using other egs++ geometries, given that simulations of two phantoms representing the same geometry should obtain statistically equivalent results. For example, the dose to a sphere of water should be independent of the underlying geometry library used to carry out the simulation (i.e. whether the sphere is made of a tetrahedral mesh or a sphere geometry should not matter, assuming the mesh accurately models the sphere's curvature). Following this idea, a comparison of an `EGS_Mesh` water-box phantom against an equivalent hexahedral voxel phantom is conducted in Section 5.1. Next, the stringent Fano test, which serves as a consistency check of radiation transport codes, is applied to a sample `EGS_Mesh` phantom in Section 5.2. An ion chamber simulation using `EGS_Mesh` is conducted in Section 5.3 and results are compared to a classical egs++ geometry simulation. Finally, Section 5.4 goes through a real-world test case using the ICRP 145 adult reference mesh phantoms including a comparison to results obtained using Geant4 [78]. All simulations were conducted with EGSnrc 2021 (using `EGS_Mesh` as a custom geometry library). Unless otherwise specified, the default EGSnrc transport parameters were used for the simulations.

Most simulation results in this chapter are reported in terms of absorbed dose. Absorbed dose measures the energy, E, deposited in a mass, m, by ionizing radiation

$$D = \frac{E}{m}. \tag{5.1}$$

The SI unit of absorbed dose is joules per kilogram, also known as Gray (Gy) [136]. For the human phantoms used in this work, overall organ dose is usually calculated by dividing the total energy deposited in that organ by the organ mass (there are some exceptions). It should be mentioned that for most radiological protection applications, absorbed dose is not precise enough. For instance, if the same dose is given to the brain and the left foot, should they both count as equally risky? And what about various kinds of radiation? Is there a

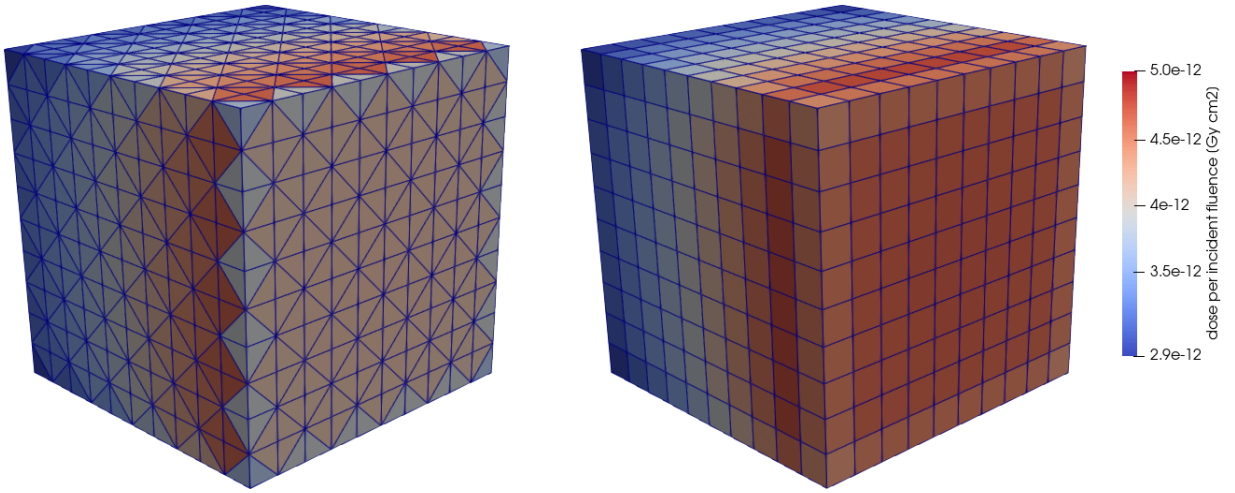


Figure 5.1: `EGS_Mesh` results (left) and `EGS_XYZGeometry` (right) dose per incident fluence results for a broad parallel beam of 1 MeV photons irradiating a 10 cm cube water phantom from the lower right.

difference for dose from photons and dose from electrons, neutrons, alpha particles and so on? The answer is to use equivalent dose and effective dose, derived quantities that account for these factors, instead of plain absorbed dose. The unit of equivalent dose is joules per kilogram multiplied by a unitless weight and so is technically equivalent to absorbed dose's Gray. But equivalent dose uses another unit, Sievert (Sv), to differentiate reported values from absorbed dose. For radiological protection, having doses calculated in Sieverts has more meaning than plain absorbed dose. That said, this chapter mainly uses absorbed dose for simplicity.

5.1 Comparison vs. structured voxel mesh

One straightforward verification method is to compare `EGS_Mesh` results with those obtained using the existing EGSnrc structured hexahedral voxel geometry `EGS_XYZGeometry` [6]. It should be noted that dose results for coarse unstructured meshes can look quite different than structured meshes with roughly the same number of elements (see Figure 5.1). Usually, all structured mesh elements have the same volume, so for example, a parallel beam source irradiating all regions would cause two elements beside each other to have very similar dose values. But an unstructured mesh can have elements that are much larger than others. Element size and position can have a large impact on the final absorbed dose value. This is especially noticeable for coarse unstructured meshes, and as the mesh is refined, this phenomenon diminishes. But how to detect potential implementation errors for coarse meshes? One method is to divide hexahedral elements into tetrahedrons, run the simulation, and check that the sum of the split element energy depositions is statistically equivalent to the

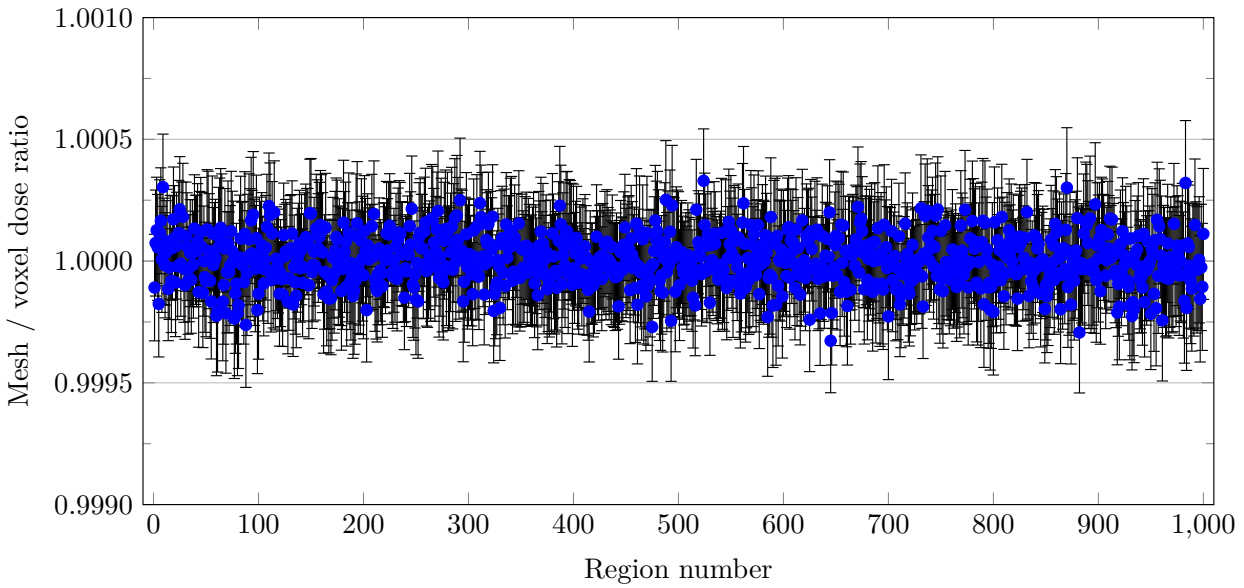


Figure 5.2: Dose ratios for a 10 cm `EGS_Mesh` cube of water compared to an equivalent `EGS_XYZGeometry` phantom calculated using Equation (5.2)

original hexahedral voxel energy deposition. This method was also used to verify the PHITS tetrahedral mesh implementation [77]. Each voxel, V, is divided into five tetrahedrons, T_i, $i \in \{1, 2, 3, 4, 5\}$, so volume-averaged dose ratios can be calculated by [77]

$$\frac{D_T}{D_V} = \frac{\sum_{i=1}^{5} E_{T_i}}{\sum_{i=1}^{5} m_{T_i}} \frac{m_V}{E_V} = \frac{\sum_{i=1}^{5} E_{T_i}}{E_V}. \tag{5.2}$$

An ideal dose ratio is 1, but given EGSnrc's nominal accuracy of 0.1%, ratios within 0.1% of 1 are considered to show excellent agreement. To compare the two geometries, a 20 cm diameter circular 1 MeV photon parallel beam was used to irradiate a voxel geometry of 1000 elements (and a corresponding tetrahedral mesh of 5000 elements) in vacuum. Results from simulating 10^{12} histories are shown in Figure 5.2. All values are within the 0.1% band, meaning for this case, the difference between the two geometries is less than that of EGSnrc's nominal precision and can be considered negligible. So, for this simulation, the `EGS_Mesh` geometry results are shown to be statistically indistinguishable from the voxel geometry, serving as evidence that the `EGS_Mesh` library has been implemented correctly.

5.2 Fano test

One of the most powerful verification tools for radiation transport simulations is the Fano test. Though passing the Fano test does not guarantee a simulation is accurate, it is an important consistency check for Monte Carlo transport codes (i.e., a known analytical solution). The test relies on the Fano theorem, which gives a theoretical reference dose value for

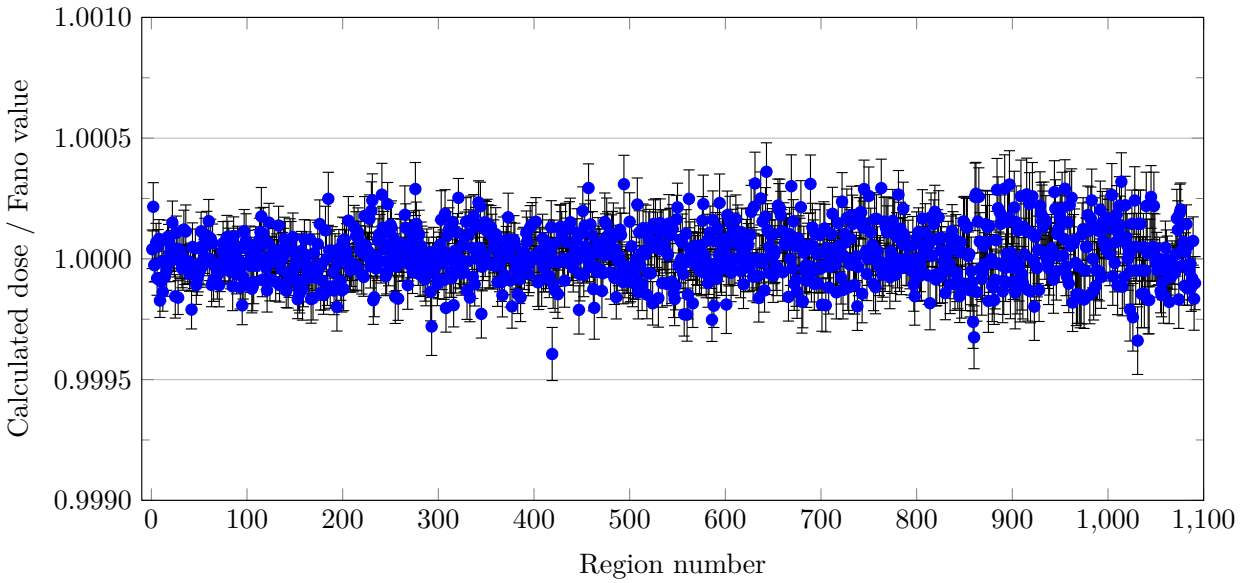


Figure 5.3: Fano test results for a sample mesh of 1.1×10^3 tetrahedrons.

certain types of radiation transport problems [137]. Briefly stated, given a uniform, isotropic, density-scaled source of particles in an infinite medium, the secondary particle flux in the medium is independent of the density including any density variations throughout the geometry [137]. In practical terms, this means the dose in every geometry region should equal the theoretical Fano dose value. This theoretical Fano dose, F, can be calculated given the primary source energy, E_S, and the total geometry mass, m_T,

$$F = \frac{E_S}{m_T}. \tag{5.3}$$

A Fano test for a sample `EGS_Mesh` geometry was conducted using the `egs_fano_source` particle source library [138] and an egs++ user code called `egs_fano` that implements periodic boundary conditions to model an infinite medium (provided by Dr. Frédéric Tessier of the NRC). A 2 MeV Fano electron source was used to irradiate a 10 cm meshed cube of water in a larger 20 cm cube of water. For the test, 500×10^9 histories were simulated. The photon cutoff energy was 10 keV and the electron cutoff energy was also 10 keV (kinetic). Figure 5.3 shows the Fano test results for this simulation. All mesh regions are within 0.1% of the theoretical value for three-sigma uncertainty bounds, meaning this specific `EGS_Mesh` simulation passes the Fano test at the 0.1% level with a 99.7% confidence interval. However, this does not imply that all `EGS_Mesh` simulations (or even other simulations using this mesh) pass the Fano test, because each unique simulation can be used to construct a separate test. Resources permitting, the Fano test will be a crucial tool for practitioners working to verify specific `EGS_Mesh` simulations.

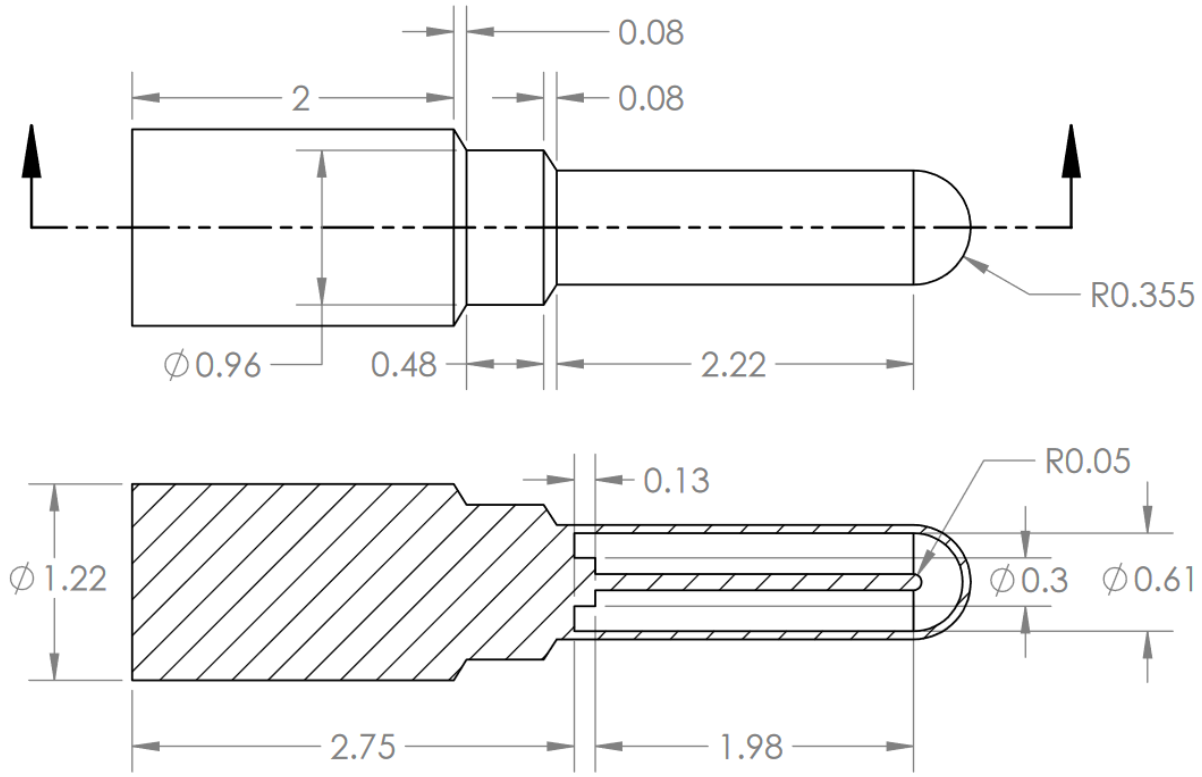


Figure 5.4: Ion chamber drawing, adapted from [139]. Units are in centimetres.

5.3 Ion chamber simulation

Part of the motivation for `EGS_Mesh` is to model complex geometries which are tedious and difficult to model using existing egs++ geometries. One example of such a geometry is an ion chamber. Ion chambers are relatively complex geometries often used in EGSnrc simulations. To demonstrate the ability of `EGS_Mesh` to model complex geometries, an ion chamber was drawn in CAD and then meshed to perform an `EGS_Mesh` simulation. Results are compared to an equivalent "classic" egs++ chamber model. For ease of comparison, the ion chamber from Chapter 6 of the EGSnrc *Getting Started* manual [139] is used. Figure 5.4 shows the dimensions of the ion chamber. The chamber was drawn in SolidWorks, exported into the STEP CAD format [140], and then meshed using the mesh generator Gmsh [141]. Geometry media (C552 plastic, and air) were also assigned using Gmsh. A few meshes of varying resolution were generated to check the effect of mesh resolution on simulation results. One of the meshes in shown in Figure 5.5. The mesh ion chamber was visually inspected to make sure it matched the classic egs++ model. Figure 5.6 shows a render of both chambers using the EGSnrc visualization tool `egs_view`.

For the simulation, the ion chamber was embedded in a 10 cm cube of water centred at the origin. A 10 MeV collimated photon point source was used, as in Chapter 6 of [139]. Both `ECUT` and `PCUT` were set to 10 keV kinetic and all variance reduction settings were turned off.

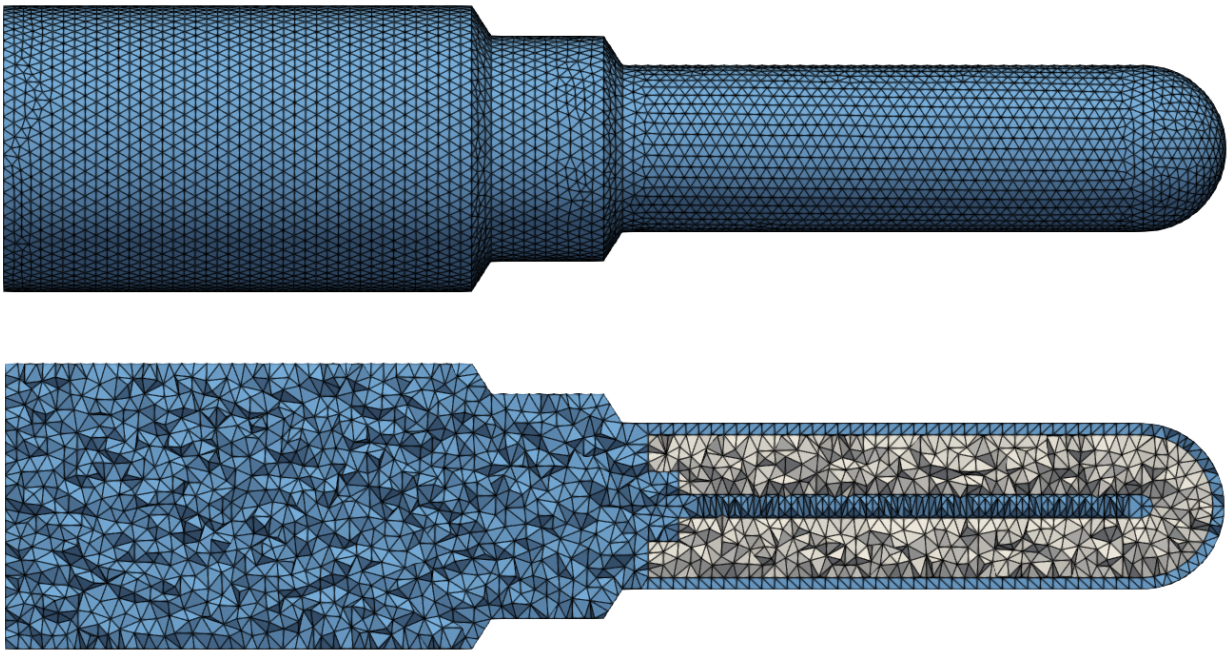


Figure 5.5: Ion chamber mesh with 1.1×10^5 elements.

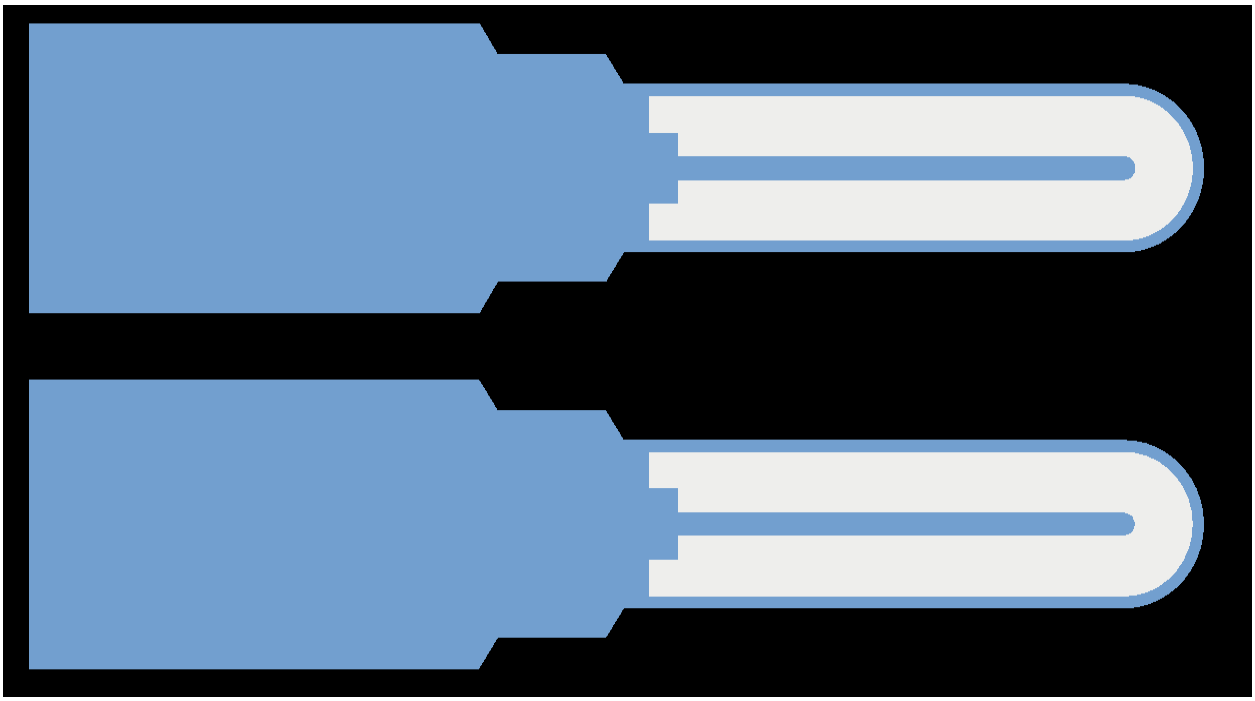


Figure 5.6: An `egs_view` rendering of the two ion chamber models. The classic egs++ model is above and the mesh model is below.

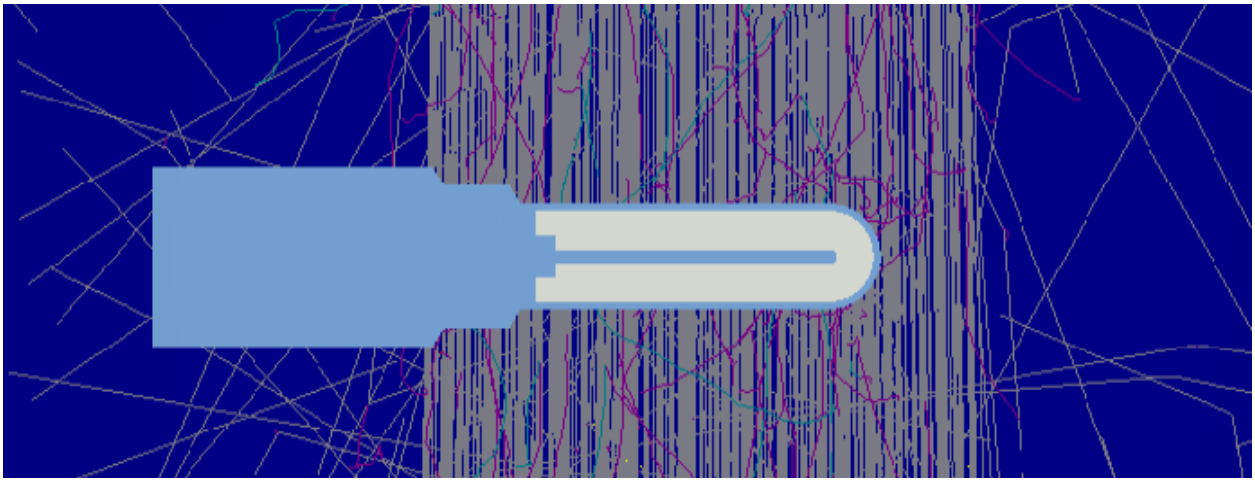


Figure 5.7: Particle tracks for the ion chamber simulation.

Sample particle tracks for the mesh chamber simulation are shown in Figure 5.7. Figure 5.8 shows air dose results for a number of meshes of increasing resolution. It appears the two coarsest meshes slightly over-predict dose to air compared to the classic model, though they are still within 0.2% of the expected value. The more refined meshes are all within 0.1% of the expected value, though there seems to be a slight downwards trend in dose ratios as the mesh becomes extremely refined. This does not necessarily imply a problem with the mesh implementation however, as the more refined meshes could passively increase the overall simulation accuracy by reducing the average particle step size (in theory, as the particle step size approaches zero, the condensed history approach exactly solves the transport equation [26]). Further simulations using only single-scattering instead of multiple-scattering (more accurate but much slower) and the lowest possible EGSnrc energy cutoffs (1 keV kinetic) would be useful to determine whether the refined ion chamber mesh simulations are indeed more accurate than the classic model.

Simulation performance measured in histories per second is shown in Figure 5.9. For this benchmark, 10^6 histories were simulated and the average of five measurements was taken. The standard deviation for each point is less than 1%. As expected, the general trend is that higher mesh resolution leads to longer simulation times. Overall, there is not an enormous runtime penalty from using `EGS_Mesh` instead of a classical egs++ geometry for this simulation, even for very refined meshes. The largest mesh studied, with 1.3×10^7 elements, was roughly 50% slower than the classic geometry. A few of the coarsest meshes are actually slightly faster than the classic geometry, which may be due to the `EGS_Mesh` surface octree being able to rapidly determine whether particles will intersect the geometry at all. It is possible that the classic model, being made of a few disparate geometries, may have to do more work in this case (but this is conjecture and further study would be beneficial). However, coarse meshes can have relatively large discretization errors compared to the classic model—for example, the volume of air in the coarsest ion chamber mesh is 11% smaller than

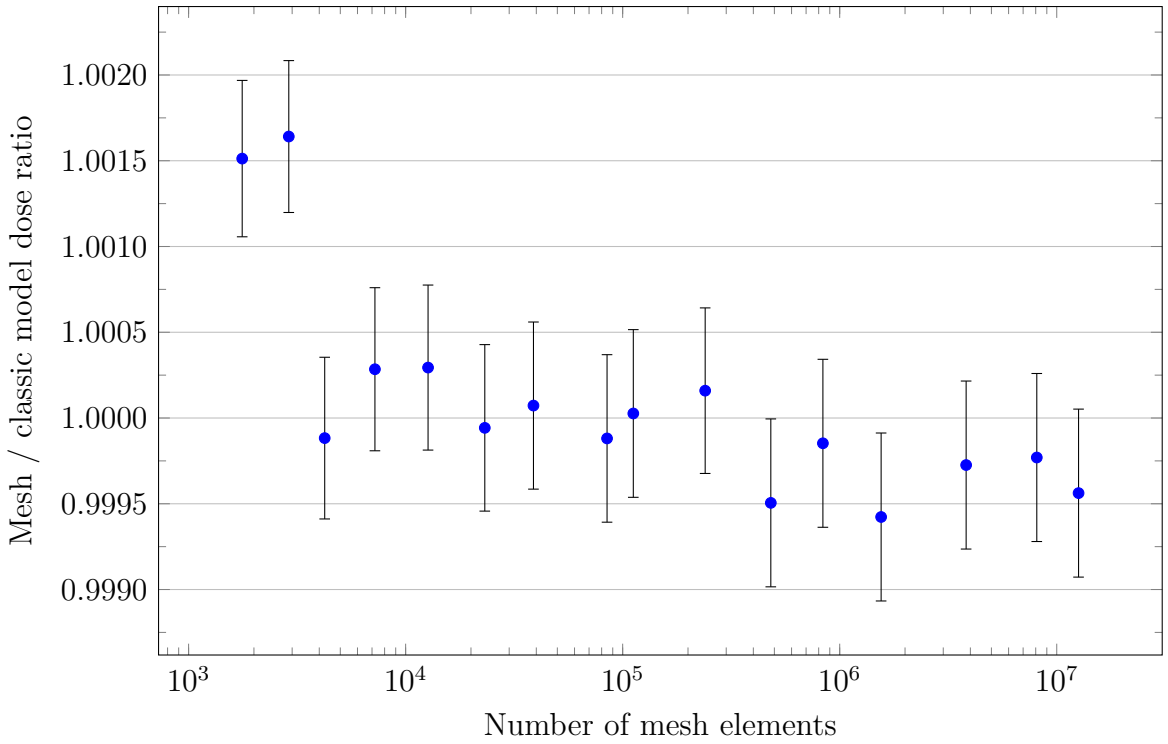


Figure 5.8: Air dose ratios for ion chamber meshes of increasing resolution.

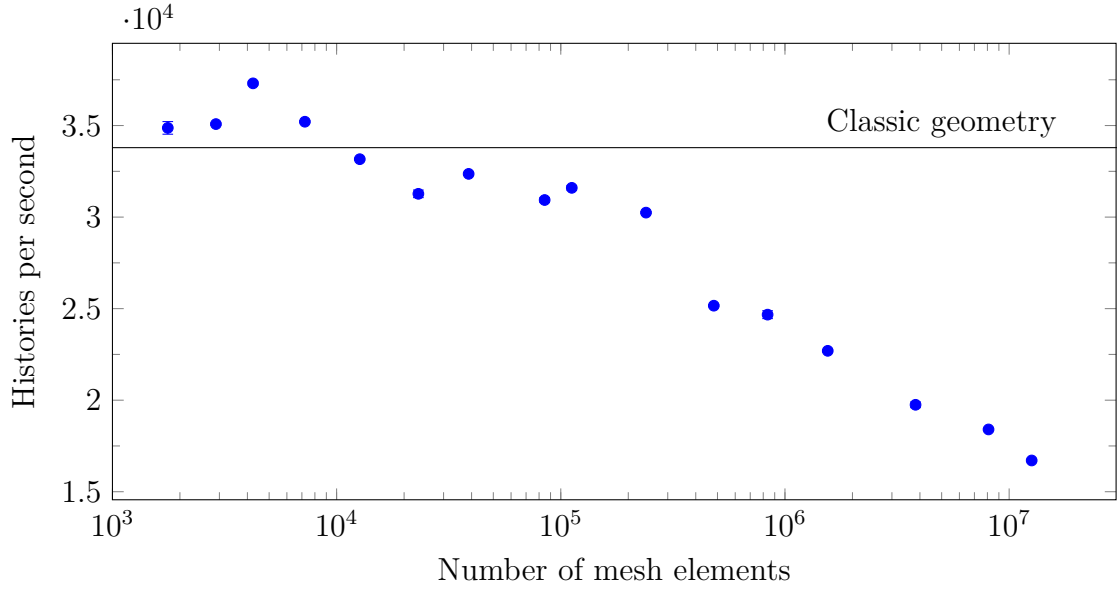


Figure 5.9: Simulation performance for ion chamber meshes of increasing resolution.

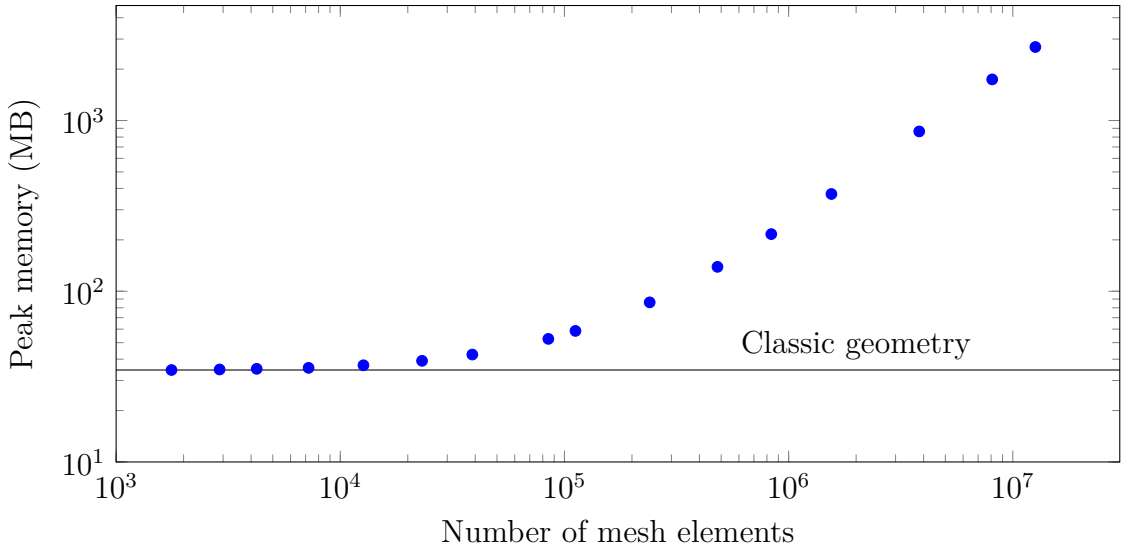


Figure 5.10: Peak memory for ion chamber meshes of increasing resolution.

the ideal value. Though as Figure 5.8 shows, in this case, the discrepancy does not lead to a noticeable difference in overall air dose.

There are some unexpected local performance maxima. The measurement uncertainties are not large enough to explain this phenomenon. One possible explanation comes from the octree partitioning scheme used by `EGS_Mesh`. Since each leaf octant has a maximum number of bounded elements, increasing the total number of elements could force the tree to subdivide again, leading to smaller leaf octants with fewer elements. These new leaf octants, having fewer elements, are then faster to do intersection tests with (assuming the cost of the additional bounding-box intersection test is less important than the speed up from being able to check fewer tetrahedrons). As plotted, this effect is more prominent when the maximum number of bounded elements is a larger percentage of the total number of elements (i.e. when the mesh is coarser).

Peak memory usage for each simulation is shown in Figure 5.10. The average of five measurements (using `/usr/bin/time -v`) was found. The standard deviation for each measurement is less than 1%. The baseline value from the classic geometry simulation is shown using a solid black line. As expected, memory use increases monotonically with the number of mesh elements. For the coarsest meshes, the required amount of memory is roughly equal to the amount used by the classic model (around 35 MB). But as the mesh becomes more refined, memory usage grows rapidly. The largest ion chamber mesh of 1.3×10^7 elements requires 2.7 GB of memory. However, this is less than the typical amount of memory available even on typical laptop computers (8 GB), so memory usage for single-core calculations should not be an issue in practice except for extremely large meshes. However, running separate simulations on multiple cores will require a proportionally larger amount of memory, which might necessitate a larger memory resource.

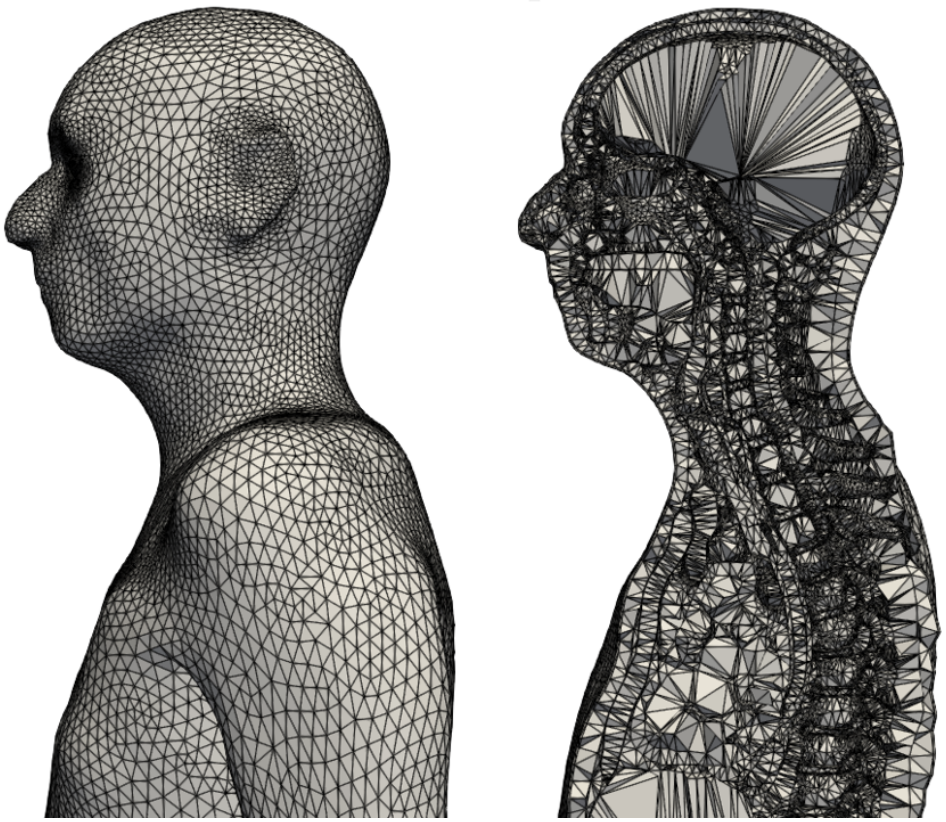


Figure 5.11: Adult male ICRP 145 reference phantom with internal view.

5.4 ICRP 145 reference phantoms

The ICRP 145 adult mesh reference phantoms and accompanying report were released in 2020 [7]. The report authors consider these new mesh phantoms as the "most advanced" kind of computational human phantoms because they can be directly used in simulations without requiring further voxelization [7] (e.g. NURBS phantoms require voxelization to be used in most transport codes [92]). The male phantom has roughly 8.2 million elements and 1.2 million nodes, while the female phantom has roughly 8.6 million elements and 1.3 million nodes. Each phantom has 52 specific media definitions and nearly 200 separate tissues and organs. The upper half of the male phantom along with an internal view is shown in Figure 5.11. As an example of the mesh phantom's improved modelling fidelity, the gallbladders of both the ICRP 110 voxel male phantom and ICRP 145 tetrahedral mesh male phantom are shown in Figure 5.12. Resolution limitations for the voxel phantom mean the voxelized organ has holes where the voxels cannot accurately represent the thin wall. Meanwhile the mesh organ has no holes and can better model the thin, curved geometry. The ICRP 145 phantom data files are freely available as supplemental material to the report [142]. But report access is required to correctly associate media with organ identification numbers and generate media data files. To use these meshes for EGSnrc simulations, node coordinates, element node numbers, and element media are required. The polygonal mesh phantoms are

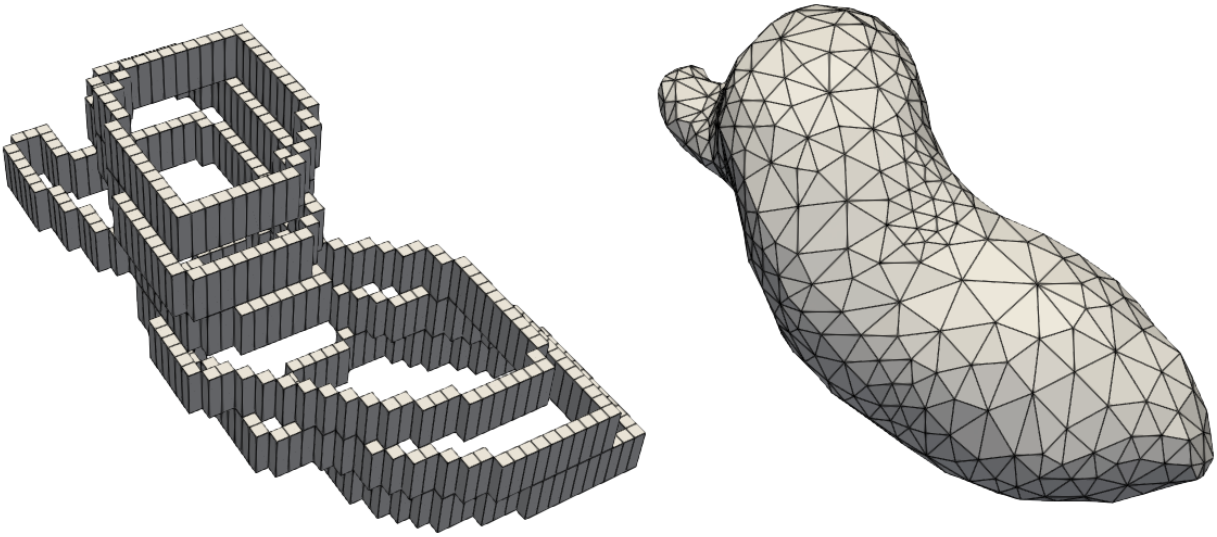


Figure 5.12: Voxelized gallbladder of ICRP 110 vs meshed gallbladder of ICRP 145.

also specified in the supplemental material, but only the tetrahedral mesh phantoms are considered here.

5.4.1 Conversion to `EGS_Mesh` input format

A dedicated computer program was written to convert the provided TetGen input files to the Gmsh `msh4.1` format. Both phantoms have a node coordinate file and element file (e.g., `MRCP_AM.node` and `MRCP_AM.ele` for the adult male phantom). The node file has ID and coordinate data for all the unique mesh nodes and the element file has the four node ids and tissue numbers of each element. The conversion process was fairly straightforward. Each tissue was assigned an entity-physical group pair and then all node and element data were converted to the Gmsh format. To test the converter, the generated Gmsh files were then converted back into the TetGen format and compared against the original inputs. No differences were found. Next, EGSnrc organ media were created following the report specifications in Annex B [7].

5.4.2 Organ media

Phantom organ media were specified using EGSnrc density correction files [4]. Each of the density correction files for both the male and female phantoms was generated using the National Institute of Standards and Technology (NIST) ESTAR database [143] and the media elemental composition tables B.1 and B.2 from ICRP 145, Annex B [7]. Media composition files for each phantom are part of the report's supplementary material (e.g. `MRCP_AM_media.dat`). Using the site manually to generate data for over 100 media ran the

Table 5.1: `egs_dose_scoring` media energy deposition

medium	Edep [MeV · cm^2]	D [Gy · cm^2]
100	$1.1036e^{-6} \pm 0.569\%$	$6.3698e^{-20} \pm 0.569\%$
200	$9.2266e^{-7} \pm 0.621\%$	$5.3577e^{-20} \pm 0.621\%$
300	$6.7504e^{-8} \pm 1.819\%$	$2.2422e^{-21} \pm 1.819\%$
⋮	⋮	⋮

risk of input errors and tedium (in case the media ever have to be regenerated or new phantoms are introduced, such as the forthcoming paediatric phantoms [94]), so a short program was written to automate the media generation process. Spot checks were conducted to ensure the automatically generated media matched with the manually created files.

Both phantoms each have 52 specific media definitions. The media names are the same, except that medium 43 for the male phantom is the prostate, while for the female phantom it is the uterus [7]. There are small elemental composition differences between the male and female phantom media. For example, the density of the upper humerus spongiosa of the male phantom is 1.233 g/cm^3, while for the female phantom the density is 1.185 g/cm^3 [7]. Each phantom has 187 numbered tissues as specified in ICRP 145 Table A.1 [7]. Most tissue numbers (with varying media compositions) are shared by both phantoms except for the testes and prostate of the male phantom and the ovaries and uterus of the female phantom.

To ease post-processing, an EGSnrc medium was defined in the input file for each organ and tissue using the identification number from Table A.1 of ICRP 145 [7]. In the input file, medium energy deposition values and uncertainties were scored using the `egs_dose_scoring ausgab` object [6]. It should be noted that turning region dose `off` in the `egs_dose_scoring` object is not strictly required, but having it `on` means every region dose value (over 8 million total) will be written to `stdout` or a log file, which can quickly take up a large amount of disk space for batch jobs. With the dose scoring object, a summary table (e.g. Table 5.1) is printed at the end of the simulation, which can be used for further post-processing to calculate organ absorbed doses and other quantities.

One should note that the energy deposition values reported by `egs_dose_scoring` shown in Table 5.1 are normalized by source fluence and that the dose values are not valid, since it is assumed that the media volume is 1. Media doses must be calculated as a separate step. The values in Table 5.1 can be used for an example calculation. For the organ with ID 100, one knows from ICRP 145 Table A.1 that it is the left adrenal gland. Assuming these results are for the male phantom, the left adrenal has a mass of 8.683 g (Table A.1 again). So, the absorbed dose per fluence, D, calculated using Equation 5.1, organ mean energy deposition,

E and the mass, m, (using the joules per MeV conversion factor [144]) is

$$D = \frac{E}{m} = \frac{1.1036 \times 10^{-6}\,\text{MeV} \cdot \text{cm}^2 \cdot 1.602 \times 10^{-13}\,\frac{\text{J}}{\text{MeV}}}{8.683 \times 10^{-3}\,\text{kg}} = 2.036 \times 10^{-17}\,\text{Gy} \cdot \text{cm}^2.$$

5.4.3 Anterior-posterior external beam organ doses

To verify the ICRP 145 implementation in EGSnrc using `EGS_Mesh`, a comparison to results from the literature was conducted. Researchers have previously calculated ICRP 145 mesh phantom organ doses for various external exposure cases using Geant4 [88]. The Geant4 study considered six external irradiation geometries [88]. Of these, only the anterior-posterior (AP) configuration is studied here. For the AP case, the phantom is irradiated by a broad parallel particle beam incident on the front of the phantom in vacuum. This case was simulated in EGSnrc using a 2 m diameter circular source transformed to face the phantom and centered at the origin. The phantom was enclosed in a vacuum box centred at the origin with side lengths of 60 cm in the x-direction, 40 cm in the y-direction and 180 cm in the z-direction. For each calculation, 10^9 histories were simulated without the use of any variance reduction techniques. Following the EGSnrc simulation settings used by the ICRP 116 report, the electron cutoff energy was 0.531 MeV (i.e. 20 keV kinetic), the photon cutoff energy was 2 keV, and the `NIST` brems cross-section data base was used [145]. Figure 5.13 shows absorbed dose results for the adult male phantom liver for 1 to 10 MeV photons in the anterior-posterior external irradiation geometry. Also plotted are results obtained using Geant4 [88], and results from ICRP 116 using the previous generation of voxel phantoms [145]. It should be noted that the ICRP 116 data are a combination of results from various transport codes (including EGSnrc) [145]. The photon results show agreement between both the voxel and mesh phantoms, as well as agreement between the two mesh phantom implementations. This confirms the ICRP 145 finding that "the MRCPs [mesh phantoms] provide effective dose DCs very similar to those of the voxel-type ICRP reference phantoms for penetrating radiations" [7] such as photons.

Figure 5.14 shows absorbed dose results for the adult male gallbladder. These results also show that the photon organ dose is very close for the voxel and mesh phantoms. However the gallbladder photon results show more disagreement than the liver results, possibly owing to the difference in phantom geometries. The voxel gallbladder has holes due to voxel resolution limitations, while the mesh gallbladder does not have holes as shown in Figure 5.12. But for electron exposures, there are larger discrepancies between the voxel and mesh results. Figure 5.15 shows electron results for the liver and Figure 5.16 shows electron results for the gallbladder. The electron doses are plotted using a log scale, as the electron organ doses increase exponentially for this energy range considered, while the photon doses increase linearly. Unlike for photons, the electron beam liver doses show worse agreement than the

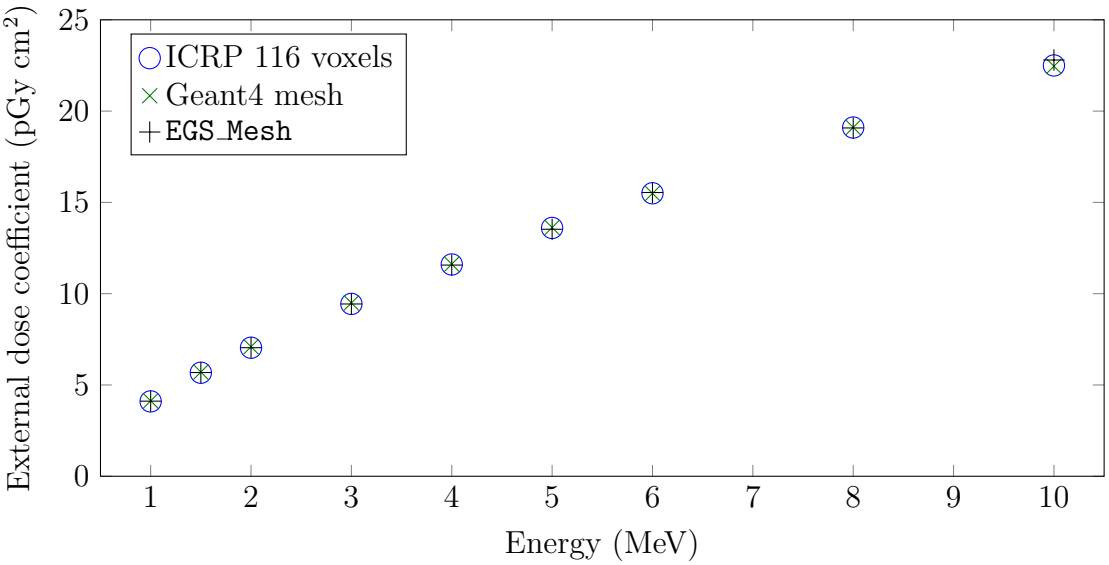


Figure 5.13: Photon absorbed dose per incident fluence results for the adult male liver, AP external irradiation geometry. All uncertainties are less than 1%.

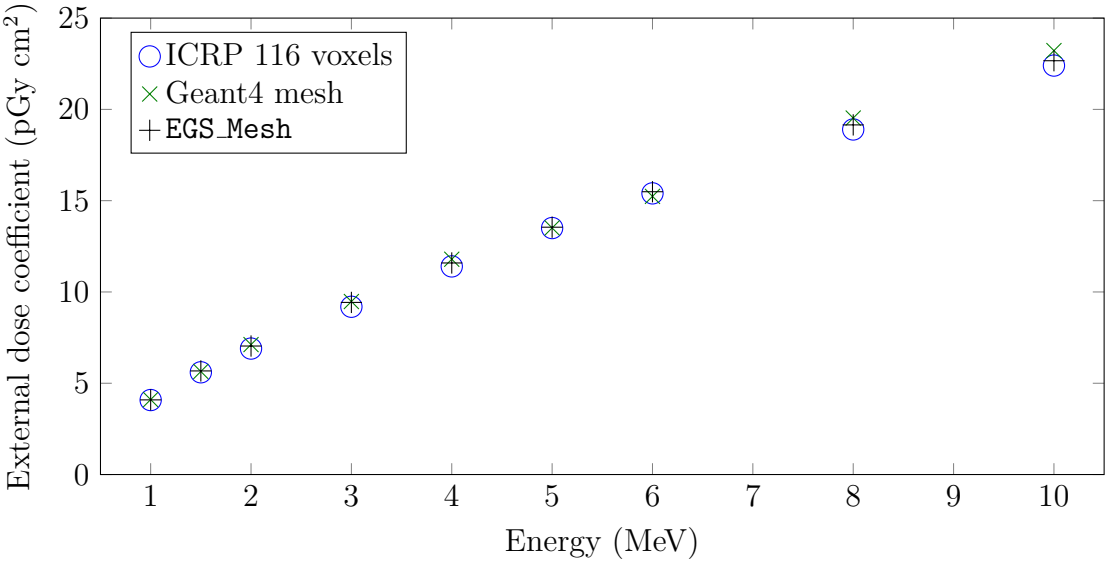


Figure 5.14: Photon absorbed dose per incident fluence results for the adult male gallbladder, AP external irradiation geometry. All uncertainties are less than 1%.

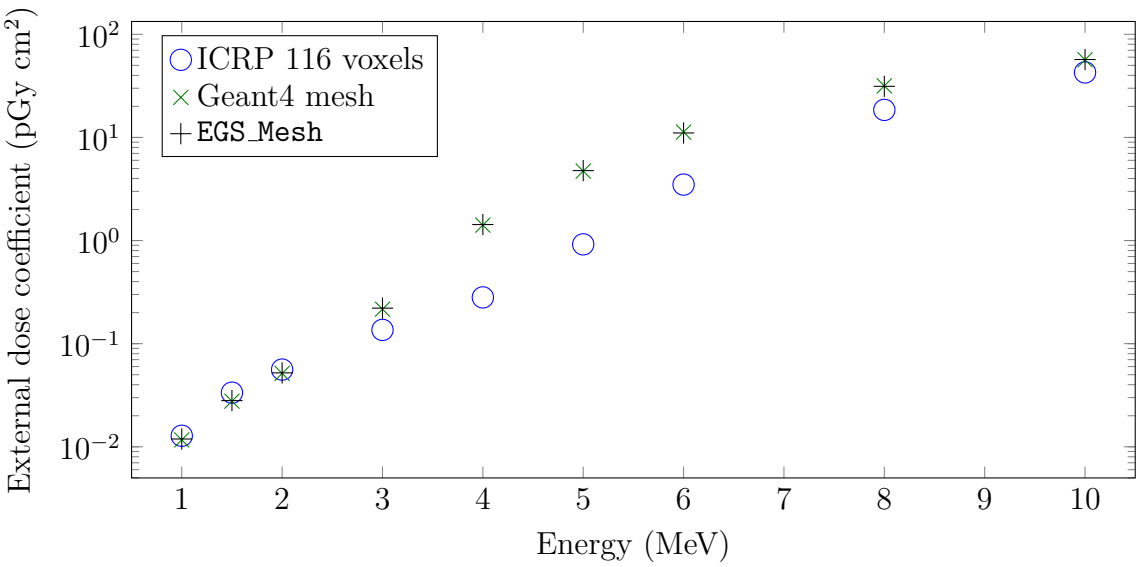


Figure 5.15: Electron absorbed dose per incident fluence results for the adult male liver, AP external irradiation geometry. All uncertainties are less than 1%.

gallbladder doses, especially for the 4, 5 and 6 MeV sources. This may be because the liver is closer to the phantom surface than the gallbladder. Since electrons are weakly-penetrating, most of their energy is deposited near the phantom's surface, before reaching the gallbladder. This means that discrepancies between the voxel and mesh gallbladder doses for electrons will likely be smaller than the discrepancies seen for the liver.

Yeom *et al.* concluded that differences between the voxel and mesh phantoms are "mainly owing to the improved representation of the organs" in the mesh phantoms (i.e. differences in geometry) [88]. Generally, mesh phantom organ positions and sizes are very similar to the corresponding voxel organs, but there are differences due to the different element shapes (i.e. curvature, resolution, etc.). This explains why the electron results show less agreement than photon results, since electrons, being weakly-penetrating, will often be more sensitive to subtle geometry differences than photons, which tend to spread out through the phantom.

Both the 5 MeV photon and electron results for the adult male liver in the anterior-posterior external irradiation geometry are shown in Figure 5.17. The photon liver dose is much more uniform than the electron dose. The electron dose has two hot spots where the liver is closest to the phantom's surface. Because electrons cannot penetrate as far as photons, the dose falls off quickly afterwards. Dose is shown using a log scale, so, on a linear scale, the hot spots would be even more pronounced compared to the other areas of the liver. In the high-dose regions, the 5 MeV electron dose is higher than for the photon case, but taking the liver as a whole, the total dose is around 2.9 times higher for photons than for electrons.

Figures 5.18, 5.19, 5.20 and 5.21 plot organ absorbed doses for the ICRP 145 adult female and male phantoms using EGSnrc and Geant4 (excluding the lungs, red bone marrow

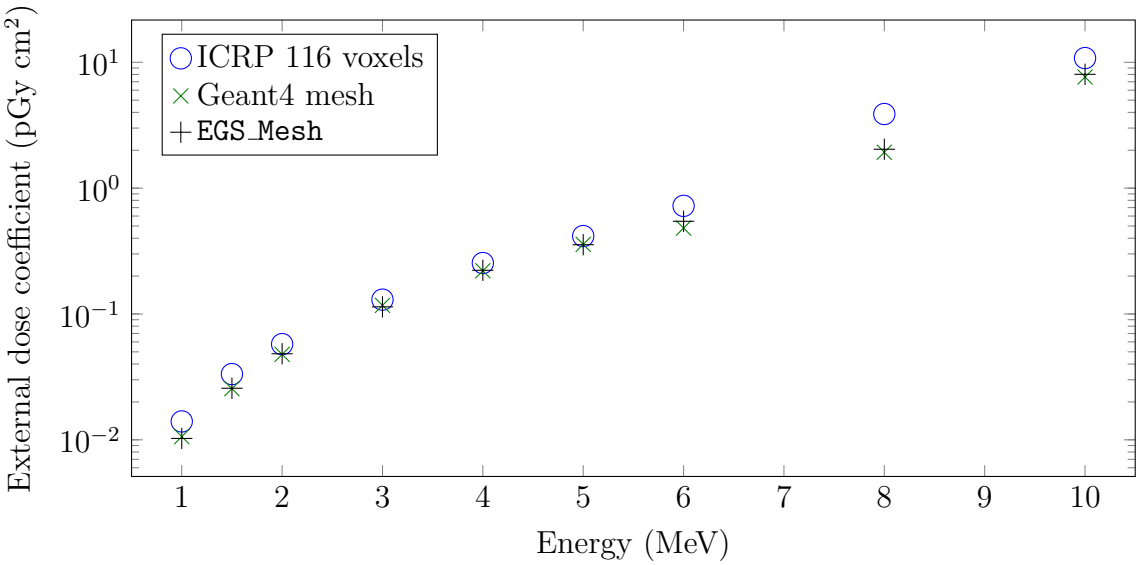


Figure 5.16: Electron absorbed dose per incident fluence results for the adult male gallbladder, AP external irradiation geometry. Uncertainties for the ICRP 116 and Geant4 data are less than 1%, while the `EGS_Mesh` uncertainties are less than 5%.

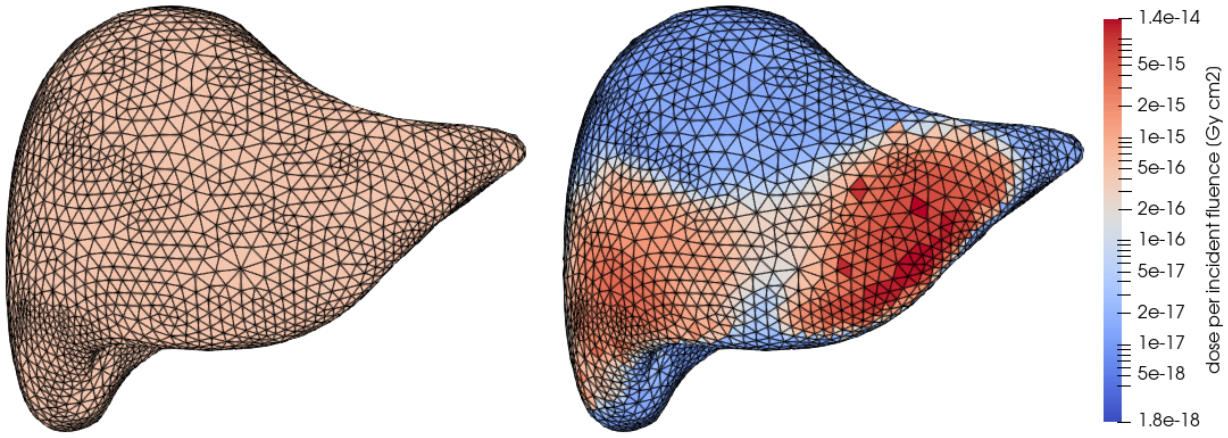


Figure 5.17: 5 MeV photon and electron adult male liver dose per incident fluence, anterior-posterior external irradiation geometry. The beam is directed into the page.

Table 5.2: ICRP 145 tissue ids for calculating absorbed dose to radiosensitive organs.

Organ	ICRP 145 tissue ids
Extrathoracic region	302, 402 (see Equation (5.4))
Eye lens	6600, 6800
Stomach	7201
Small intestine	7401
Colon	7601, 7801, 8001, 8201, 8401
Oesophagus	11001
Skin	12201, 12301, 12401, 12501
Urinary bladder	13701

and endosteum, as these require special calculation methods). For the EGSnrc results, 10^9 histories were simulated using the same transport parameters as before. There are many different types of organs represented, including thin-walled organs such as the stomach and urinary bladder, superficial organs such as the skin, and simple organs like the kidneys. Most of the results obtained using `EGS_Mesh` are within 5% of the Geant4 value. Generally, the photon organ doses show less variance than the electron organ doses. Following [78], certain organ absorbed doses are calculated by averaging only over radiosensitive organ regions. These organs are given in Table 5.2. In addition, the extrathoracic region absorbed dose was calculated using the weighting scheme from Table 2.3 of ICRP 133 [146], where the final extrathoracic absorbed dose, D_{ET}, is calculated using the following weighted sum,

$$D_{ET} = 0.001 \cdot D_{302} + 0.999 \cdot D_{402}, \tag{5.4}$$

where D_{302} is the absorbed dose of region 302 and D_{402} is the absorbed dose of region 402.

In general, it can be difficult to determine the root cause of differences between Monte Carlo transport codes. Differences can arise from various parts of each code, including differences between transport algorithms, cross-section data and geometry implementations. It is also possible that one or both implementations might also have errors, further confounding explanation. At the time of writing, the majority of dose ratios studied so far have been within 5% of the Geant4 value, and all are within 10%, demonstrating that `EGS_Mesh` is able to obtain similar results to Geant4, the code primarily used by the initial ICRP 145 report [7]. Further work is required to completely verify the use of the ICRP 145 phantoms with `EGS_Mesh`.

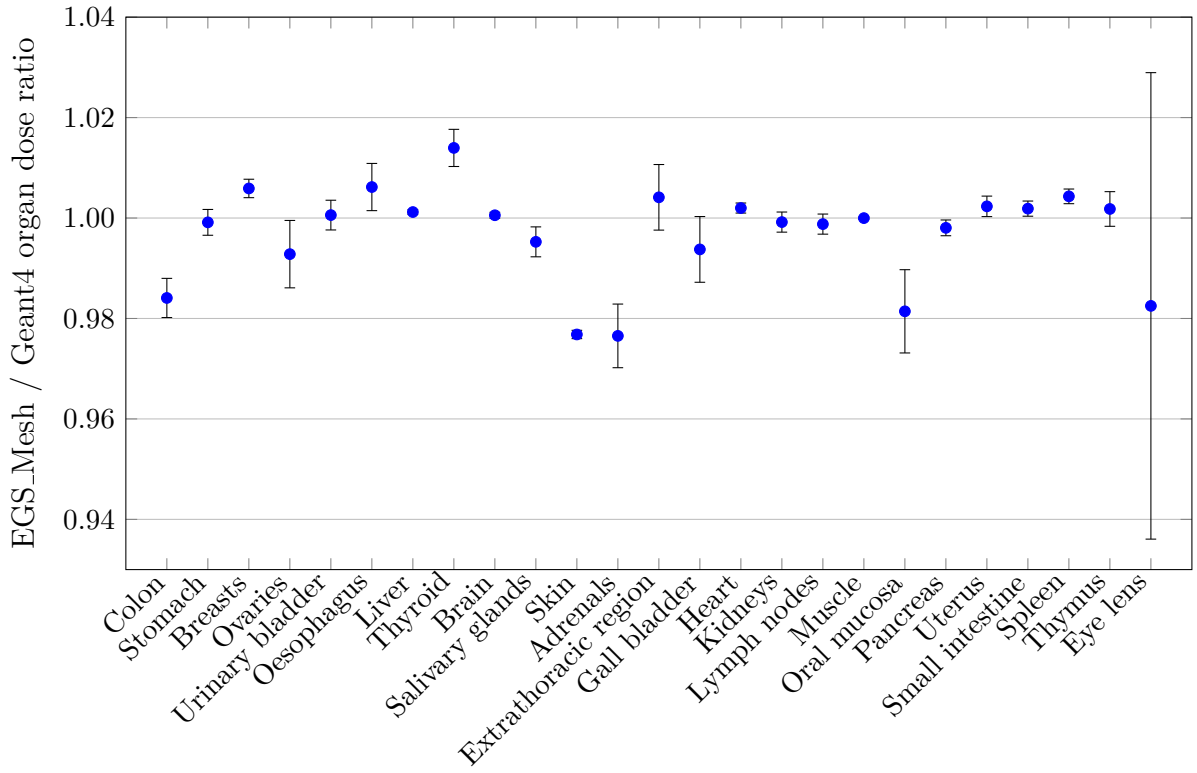


Figure 5.18: ICRP 145 adult female phantom organ dose ratios between EGSnrc and Geant4 for a 5 MeV photon anterior-posterior beam.

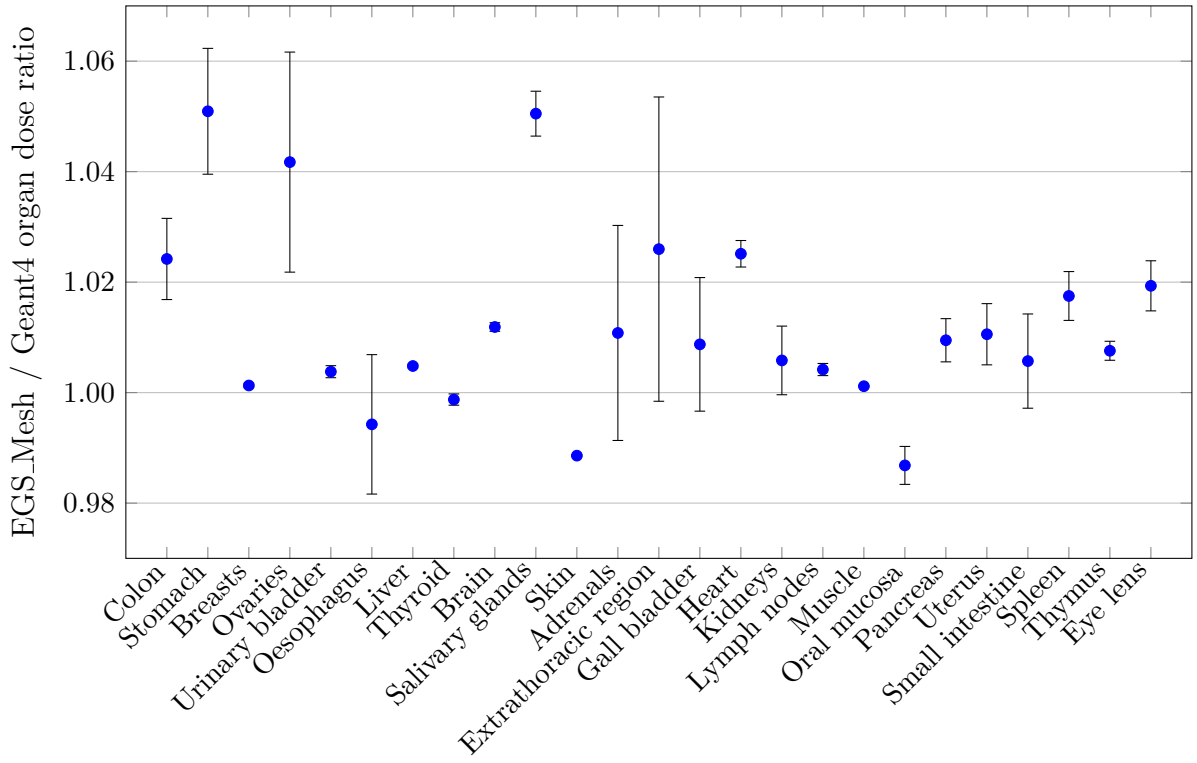


Figure 5.19: ICRP 145 adult female phantom organ dose ratios between EGSnrc and Geant4 for a 5 MeV electron anterior-posterior beam.

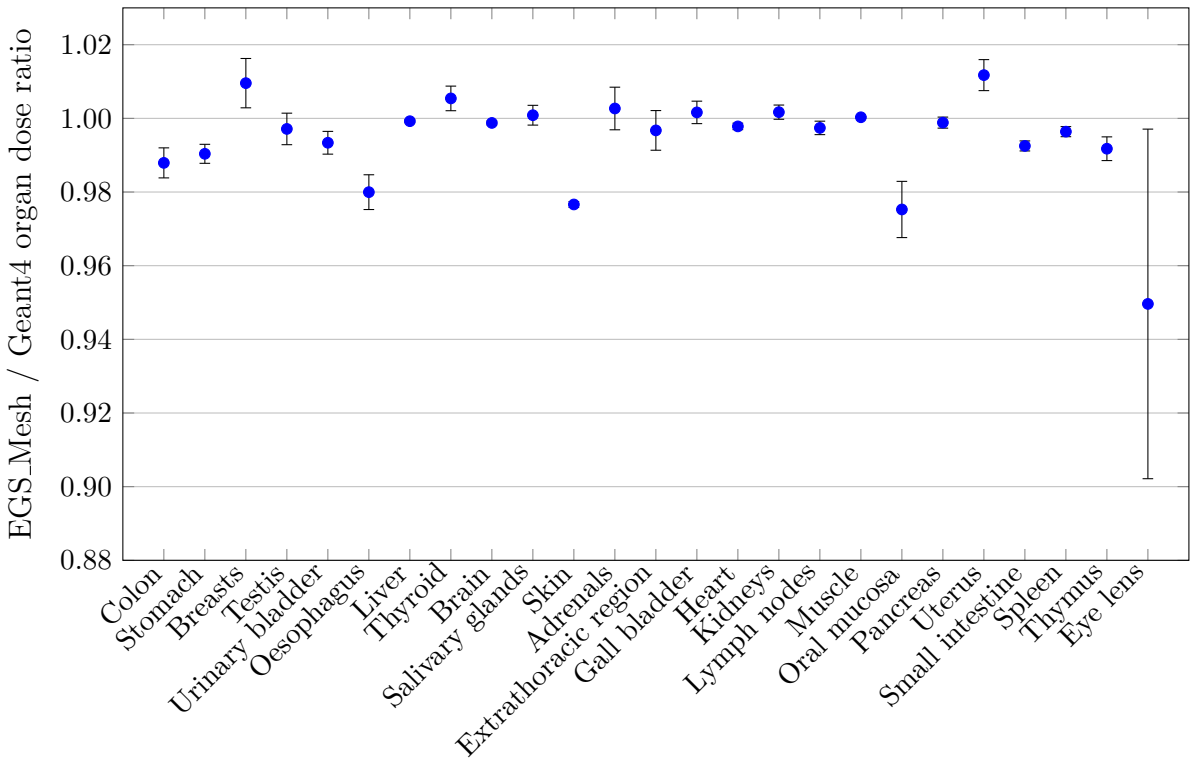


Figure 5.20: ICRP 145 adult male phantom organ dose ratios between EGSnrc and Geant4 for a 5 MeV photon anterior-posterior beam.

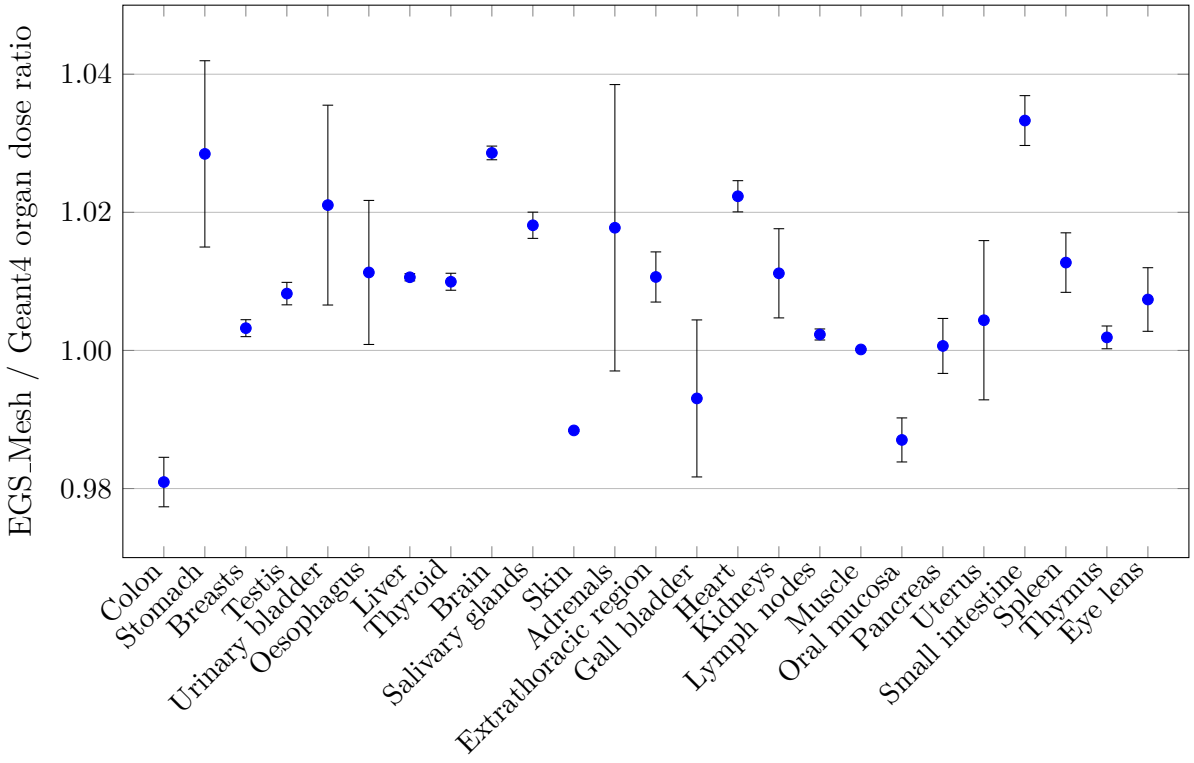


Figure 5.21: ICRP 145 adult male phantom organ dose ratios between EGSnrc and Geant4 for a 5 MeV electron anterior-posterior beam.

Chapter 6

Future work and conclusions

The tetrahedral mesh geometry implementation described in this thesis can be used as a basis for further research. Research directions include new features, such as tetrahedral mesh sources (Section 6.1), embedded ICRP 145 lung airway modelling (Section 6.2) and an improved transport algorithm for homogeneous mesh media (Section 6.3). The actual mesh implementation is fairly naive, meaning further performance optimizations are possible as discussed in Section 6.4. The ongoing verification work must continue and further comparisons to the literature and other real-world calculations will be invaluable as support for the implementation's validity as detailed in Section 6.5. Section 6.6 highlights some `EGS_Mesh` numerical routines with inherent floating point robustness issues, that could be addressed with algorithmic improvements or alternative implementations. The thesis is concluded in Section 6.7.

6.1 Internal mesh sources

The ICRP 145 mesh phantoms were designed with a number of "source regions", organs and tissues that themselves emit radiation after, for example, ingesting a radioactive substance for a medical procedure [7]. Some source organs of the ICRP 145 adult male phantom are shown in Figure 6.1. For example, thyroid cancer is treated with radioactive iodine, which is concentrated in the thyroid, but other organs receive dose as a consequence of treatment [107]. As a complement to the external source results considered in Chapter 5, the development of a tetrahedral mesh particle source for comparison against the ICRP 145 data [7] and other studies (e.g. [107]) would be a straightforward extension to this work. Annex I of ICRP 145 has simulation results for different internal organ sources including the thyroid, liver, and cortical bone [7]. A comparison against values obtained using the voxel reference phantoms in ICRP 133 [146] would be immediately realizable.

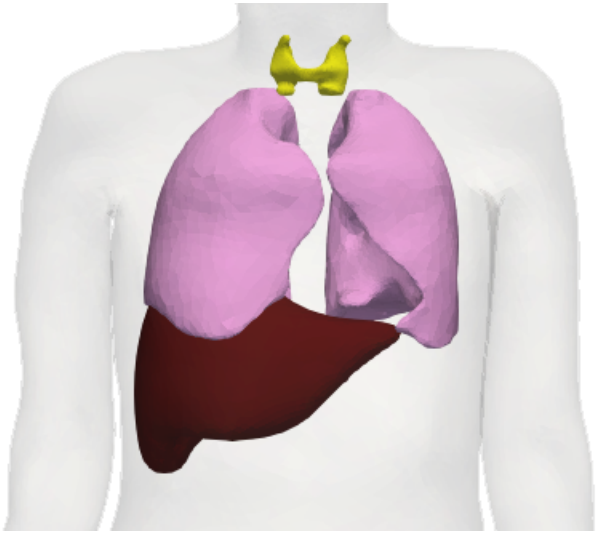


Figure 6.1: Three source organs of the ICRP 145 adult male phantom (thyroid, lungs and liver).

6.2 ICRP 145 lung airway modelling

Though not strictly relating to the tetrahedral mesh implementation, implementing a combined model of the ICRP 145 mesh and the detailed lung model developed by Kim *et al.* [98] would be especially valuable for researchers interested in the lungs and more accurate results for dose to different lung tissues. Implementing the lung model using a tetrahedral mesh is considered too expensive because the large number of tetrahedrons required for modelling it would use around 50GB of memory [98]. Instead, a constructive solid geometry approach is used to model the fine structure of the lung airways [98]. This detailed model is then overlaid with the mesh geometry for a combined simulation [98]. So far, there is only a Geant4 implementation of this enhanced model [7, 98], meaning another implementation would be useful for verification. More generally, implementation of refined mesh organ models such as the teeth model of Shin *et al.* [110] would further improve the calculation power of the ICRP 145 mesh phantoms considered here.

6.3 `howfarless`: efficient transport in homogeneous tetrahedral meshes

For many applications, a complete set of region doses like those output by `EGS_Mesh` simulations is unnecessary. For example, calculation of organ dose coefficients using the ICRP 145 meshes [7] only requires total organ average dose. In that case, only considering the distance to a medium boundary and transporting the particle to the next medium boundary without having to explicitly simulate transport in every medium voxel could increase

simulation efficiency without noticeably affecting result accuracy. Walters and Kawrakow implemented such a feature in the rectilinear voxel code `DOSXYZnrc` [147]. They found up to a 5 times increase in efficiency for certain simulation configurations [147]. They observed that efficiency increased as the source energy increased [147], which makes intuitive sense as particles with more energy tend to travel farther. Other researchers have extended their implementation for further efficiency improvements in certain slab geometries [148]. However, it may be the case that a similar implementation for a tetrahedral mesh would result in lower efficiency improvements because of the increased difficulty (compared to a voxel geometry) of determining the distance to an arbitrary mesh boundary.

6.4 Other performance improvements

The computer graphics literature has a wealth of performance-oriented research. It is virtually certain that the implementation presented in this thesis can be improved with regards to performance. The performance-oriented parts of the current implementation are:

- storing tetrahedron neighbours,
- volume and surface octree spatial partitioning, and
- precalculating triangle face normals used heavily in the geometry routines.

Further performance improvements could be realized from either improving on the current implementation or adding to the list of optimizations. One major optimization challenge is that unstructured meshes are notoriously cache-unfriendly. On a case by case basis, reordering the input mesh file so elements closest to the source come first in memory would likely improve performance somewhat. More sophisticated mesh reordering algorithms also exist in the literature [149]. It is possible another spatial partitioning scheme like a kd-tree [127] would improve performance over the octree used in the initial implementation. The use of an industrial-grade ray-tracing library such as `embree` [150] would also likely improve performance, though it would introduce a heavy external dependency. In computer graphics, geometry routines are typically accelerated using single instruction multiple data (SIMD) intrinsics that allow for multiple intersection tests to be performed at once. The use of double precision values in `EGS_Mesh` instead of the single precision floats often used in computer graphics [127] means more recent instruction sets like Intel's Advanced Vector Extensions (AVX) are required to process four doubles at once [151]. A SIMD implementation of the mesh geometry routines could improve performance, though it may be that the cost of cache misses would drown out any improvements obtained through vectorization.

6.5 Continued verification and benchmarking

The comparison work begun in this thesis regarding results obtained using the ICRP 145 phantoms compared to the ICRP 116 data [145] and those results obtained using Geant4 [88] is not yet complete. Only the anterior-posterior geometry was studied in depth so far, (the other five irradiation geometries being posterior-anterior, left lateral, right lateral, rotational and isotropic). Implementation of the rotational source might be accomplished using the `EGS_DynamicSource` class [6]. At the moment, there is no direct analogue to the inwardly directed spherical isomorphic source in the egs++ class library, so would-be implementers may have to design their own special-purpose source. At the time of writing, an ICRP report on mesh paediatric phantoms is being finalized [94]. Once the report is released, implementation of the paediatric phantoms following this work would be valuable both for `EGS_Mesh` verification and for further study of the phantoms themselves using EGSnrc. More performance testing of the system is also crucial. A thorough examination of `EGS_Mesh` performance for different cases would be a useful resource and would help guide the efficiency expectations of practitioners setting up a new simulation.

6.6 Robustness improvements

Geometry routines making use of floating point numbers like C++'s `float` and `double` can have some inherent robustness issues. Not all real numbers can be described using floating point, meaning truncation is inevitable in the general case. However, for a performance sensitive project like EGSnrc, floating point alternatives like arbitrary precision arithmetic are probably too expensive. The frequency of robustness issues is estimated to be below the sensitivity threshold of EGSnrc ($< 0.1\%$), but nevertheless a better understanding of the various issues and measurement of their true frequency would be very beneficial for the implementation. Section 3.9 of [127] has a thorough introduction to these concepts and some immediately applicable algorithms, such as a more robust version of the ray-box intersection algorithm considered in Section 4.4.3. Along with an improved ray-box test, another interesting result would be to compare the existing Möller-Trumbore ray-triangle intersection algorithm [63] with the watertight intersection algorithm of Woop *et al.* [121] for various meshes.

6.7 Conclusions

A tetrahedral mesh geometry library, `EGS_Mesh`, for the radiation transport code EGSnrc, was implemented. `EGS_Mesh` can be used in combination with other egs++ libraries, allowing users to design ever-more powerful and accurate simulations. `EGS_Mesh` also allows users

to simulate radiation transport in CAD geometries after a meshing step. The `EGS_Mesh` implementation uses known algorithms from the computational geometry literature and is accelerated using octree spatial partitioning. `EGS_Mesh` has been verified by comparison to theoretical results (including a Fano test and a comparison against an equivalent voxelized geometry) and by comparison to results obtained using Geant4, another transport code capable of simulating radiation transport in tetrahedral meshes. `EGS_Mesh` has been shown to be capable of simulating radiation transport in large and complex tetrahedral mesh phantoms from the literature, such as the ICRP 145 adult reference male and female phantoms [7]. ICRP 145 organ dose comparisons to Geant4 for the energies considered here found agreement mostly within 5%, but some organs had differences of up to 10%, requiring further study to be completely assured of the use of the ICRP 145 phantoms with EGSnrc. Future research directions include the design of an accelerated transport algorithm for use in homogeneous tetrahedral meshes as well as continued performance and benchmarking studies (including the use of tetrahedral mesh phantoms from the wider simulation community). With this work, an initial implementation of tetrahedral mesh transport in EGSnrc has been realized.

Bibliography

[1] J. Cygler and G. Ding. "Electrons: Clinical Considerations and Applications". In: *Monte Carlo Techniques in Radiation Therapy*. CRC Press, 2013. Chap. 11, pp. 155–164.

[2] J. Lee et al. "Fano cavity test for electron Monte Carlo transport algorithms in magnetic fields: comparison between EGSnrc, PENELOPE, MCNP6 and Geant4". In: *Physics in Medicine & Biology* 63.19 (Oct. 2018), p. 195013. DOI: 10.1088/1361-6560/aadf29. URL: https://doi.org/10.1088/1361-6560/aadf29.

[3] A. F. Bielajew. "History of Monte Carlo". In: *Monte Carlo Techniques in Radiation Therapy*. CRC Press, 2013. Chap. 1, pp. 3–12.

[4] I. Kawrakow et al. *The EGSnrc code system*. Tech. rep. PIRS-701. National Research Council of Canada, 2021.

[5] *EGS4 code system*. Tech. rep. Stanford Linear Accelerator Center, Menlo Park, CA (USA), Dec. 1985. URL: https://www.osti.gov/biblio/6137659.

[6] I. Kawrakow et al. *The EGSnrc C++ class library*. Tech. rep. PIRS-898. National Research Council of Canada, 2019.

[7] ICRP. "Adult Mesh-type Reference Computational Phantoms". In: *Annals of the ICRP* 49 (2020). ICRP Publication 145.

[8] M. Constantin et al. "Linking computer-aided design (CAD) to Geant4-based Monte Carlo simulations for precise implementation of complex treatment head geometries". In: *Physics in Medicine & Biology* 55.8 (Mar. 2010), N211–N220. DOI: 10.1088/0031-9155/55/8/n03. URL: https://doi.org/10.1088/0031-9155/55/8/n03.

[9] *MCNP Users Manual - Code Version 6.2*. Tech. rep. Los Alamos National Laboratory, 2018.

[10] S. Agostinelli et al. "Geant4—a simulation toolkit". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303. ISSN: 0168-9002. DOI: https://doi.org/10.1016/S0168-9002(03)01368-8. URL: https://www.sciencedirect.com/science/article/pii/S0168900203013688.

[11] T. Sato et al. “Particle and Heavy Ion Transport code System, PHITS, version 2.52”. In: *Journal of Nuclear Science and Technology* 50.9 (2013), pp. 913–923. DOI: 10.1080/00223131.2013.814553. eprint: https://doi.org/10.1080/00223131.2013.814553. URL: https://doi.org/10.1080/00223131.2013.814553.

[12] X. G. Xu. “An exponential growth of computational phantom research in radiation protection, imaging, and radiotherapy: a review of the fifty-year history”. In: *Physics in Medicine & Biology* 59.18 (Aug. 2014), R233–R302. DOI: 10.1088/0031-9155/59/18/r233. URL: https://doi.org/10.1088/0031-9155/59/18/r233.

[13] ICRP. “Adult Reference Computational Phantoms”. In: *Annals of the ICRP* 39 (2009). ICRP Publication 110.

[14] M. Zankl et al. “Computational phantoms, ICRP/ICRU, and further developments”. In: *Annals of the ICRP* 47.3-4 (2018). PMID: 29652167, pp. 35–44. DOI: 10.1177/0146645318756229. eprint: https://doi.org/10.1177/0146645318756229. URL: https://doi.org/10.1177/0146645318756229.

[15] R. L. Ford, W. R. Nelson, et al. *The EGS code system: Computer programs for the Monte Carlo simulation of electromagnetic cascade showers (version 3)*. Tech. rep. SLAC National Accelerator Lab., Menlo Park, CA (United States), 1978.

[16] H. H. Nagel and C. Schlier. “Berechnung von Elektron-Photon-Kaskaden in Blei für eine Primärenergie von 200 MeV”. In: *Zeitschrift fur Physik* 174.4 (Dec. 1963), pp. 464–471. DOI: 10.1007/BF01380630.

[17] A. F. Bielajew et al. “History, overview and recent improvements of EGS4”. In: *Conference on Radiation Transport Calculations Using the EGS4*. June 1994.

[18] W. R. Nelson, H. Hirayama, and D. W. O. Rogers. *The EGS4 code system*. Tech. rep. Stanford Linear Accelerator Center, Menlo Park, CA (USA), 1985.

[19] I. Kawrakow. “Accurate condensed history Monte Carlo simulation of electron transport. I. EGSnrc, the new EGS4 version”. In: *Medical Physics* 27.3 (2000), pp. 485–498.

[20] I. Kawrakow. “Accurate condensed history Monte Carlo simulation of electron transport. II. Application to ion chamber response simulations”. In: *Medical Physics* 27.3 (2000), pp. 499–513.

[21] I. Kawrakow and A. F. Bielajew. “On the condensed history technique for electron transport”. In: *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms* 142.3 (1998), pp. 253–280.

[22] M. P. Martinov and R. M. Thomson. “Taking EGSnrc to new lows: Development of egs++ lattice geometry and testing with microscopic geometries”. In: *Medical Physics* 47.7 (2020), pp. 3225–3232.

[23] M. J. P. Chamberland et al. “egs_brachy: a versatile and fast Monte Carlo code for brachytherapy”. In: *Physics in Medicine & Biology* 61.23 (2016), p. 8214.

[24] J. Seuntjens, W. Strydom, and K. Shortt. “Dosimetric principles, quantities and units”. In: *Radiation oncology physics: A handbook for teachers and students*. 2005, pp. 45–70.

[25] M. J. Berger. “Monte Carlo calculation of the penetration and diffusion of fast charged particles.” In: *Methods in Computational Physics*. 135 (1963).

[26] E. W. Larsen. “A theoretical derivation of the condensed history algorithm”. In: *Annals of Nuclear Energy* 19.10-12 (1992), pp. 701–714.

[27] A. F. Bielajew and D. W. O. Rogers. “Electron step-size artefacts and PRESTA”. In: *Monte Carlo transport of electrons and photons*. Springer, 1988, pp. 115–137.

[28] N. Metropolis and S. Ulam. “The Monte Carlo method”. In: *Journal of the American statistical association* 44.247 (1949), pp. 335–341.

[29] N. Metropolis et al. “The beginning of the Monte Carlo method”. In: *Los Alamos Science* 15.584 (1987), pp. 125–130.

[30] J. Wulff, K. Zink, and I. Kawrakow. “Efficiency improvements for ion chamber calculations in high energy photon beams: Efficiency improvements for ion chamber calculations”. In: *Medical Physics* 35.4 (2008), pp. 1328–1336. ISSN: 0094-2405.

[31] A. F. Bielajew and D. W. O. Rogers. “Variance-Reduction Techniques”. In: *Monte Carlo Transport of Electrons and Photons*. Springer, 1988, pp. 407–419.

[32] I. Kawrakow and M. Fippel. “Investigation of variance reduction techniques for Monte Carlo photon dose calculation using XVMC”. In: *Physics in Medicine & Biology* 45.8 (2000), pp. 2163–2183. ISSN: 0031-9155.

[33] F. James and L. Moneta. “Review of high-quality random number generators”. In: *Computing and Software for Big Science* 4.1 (2020), pp. 1–12.

[34] A. M. Ferrenberg, D. P. Landau, and Y. J. Wong. “Monte Carlo simulations: Hidden errors from “good” random number generators”. In: *Phys. Rev. Lett.* 69 (23 Dec. 1992), pp. 3382–3384. DOI: 10.1103/PhysRevLett.69.3382. URL: https://link.aps.org/doi/10.1103/PhysRevLett.69.3382.

[35] F. James. “RANLUX: A Fortran implementation of the high-quality pseudorandom number generator of Lüscher”. In: *Computer Physics Communications* 79.1 (1994), pp. 111–114.

[36] M. Lüscher. “A portable high-quality random number generator for lattice field theory simulations”. In: *Computer physics communications* 79.1 (1994), pp. 100–110.

[37] A. Sibidanov. “A revision of the subtract-with-borrow random number generators”. In: *Computer Physics Communications* 221 (2017), pp. 299–303.

[38] K. G. Savvidy. “The MIXMAX random number generator”. In: *Computer Physics Communications* 196 (2015), pp. 161–165.

[39] J. Hahnfeld and L. Moneta. “A Portable Implementation of RANLUX++”. In: *EPJ Web Conf.* 251 (2021), p. 03008. DOI: 10.1051/epjconf/202125103008. URL: https://doi.org/10.1051/epjconf/202125103008.

[40] D. W. O. Rogers and A. F. Bielajew. “Monte Carlo techniques of electron and photon transport for radiation dosimetry”. In: *The dosimetry of ionizing radiation* 3 (1990), pp. 427–539.

[41] B. R. B. Walters, I. Kawrakow, and D. W. O. Rogers. “History by history statistical estimators in the BEAM code system”. In: *Medical Physics* 29.12 (2002), pp. 2745–2752.

[42] J. K. Shultis and R. E. Faw. *An MCNP primer*. Tech. rep. 2011.

[43] G. Battistoni et al. “The FLUKA code: Description and benchmarking”. In: *AIP Conference proceedings*. Vol. 896. 1. American Institute of Physics. 2007, pp. 31–49.

[44] F. Salvat, J. M. Fernández-Varea, and J. Sempau. “PENELOPE-2006: A code system for Monte Carlo simulation of electron and photon transport”. In: *Workshop proceedings*. Vol. 4. 6222. 2006, p. 7.

[45] H. Koivunoro et al. “Accuracy of the electron transport in MCNP5 and its suitability for ionization chamber response simulations: A comparison with the EGSnrc and PENELOPE codes”. In: *Medical Physics* 39.3 (2012), pp. 1335–1344.

[46] J. P. Archambault and E. Mainegra-Hing. “Comparison between EGSnrc, Geant4, MCNP5 and Penelope for mono-energetic electron beams”. In: *Physics in Medicine & Biology* 60.13 (June 2015), pp. 4951–4962. DOI: 10.1088/0031-9155/60/13/4951. URL: https://doi.org/10.1088/0031-9155/60/13/4951.

[47] D. W. O. Rogers. “Fifty years of Monte Carlo simulations for medical physics”. In: *Physics in Medicine & Biology* 51.13 (2006), R287–R301. ISSN: 0031-9155.

[48] J. Cygler et al. “Evaluation of the first commercial Monte Carlo dose calculation engine for electron beam treatment planning”. In: *Medical Physics* 31.1 (2004), pp. 142–153.

[49] T. Mackie et al. “The Ottawa-Madison electron gamma algorithm (OMEGA) project: feasibility of two Monte Carlo techniques”. In: *The Use of Computers in Radiation Therapy: Proceedings of the X ICCR. Lucknow (India): ICCR* (1990), pp. 250–2.

[50] D. W. O. Rogers et al. “BEAM: A Monte Carlo code to simulate radiotherapy treatment units”. In: *Medical Physics* 22.5 (1995), pp. 503–524. ISSN: 0094-2405.

[51] D. W. O. Rogers, B. R. B. Walters, I. Kawrakow, et al. *BEAMnrc users manual*. Tech. rep. PIRS-509a. National Research Council of Canada, 2009.

[52] J. Lippuner and I. A. Elbakri. “A GPU implementation of EGSnrc’s Monte Carlo photon transport for imaging applications”. In: *Physics in Medicine & Biology* 56.22 (2011), pp. 7145–7162. ISSN: 0031-9155.

[53] L. Su et al. “ARCHER$_{RT}$ – A GPU-based and photon-electron coupled Monte Carlo dose computing engine for radiation therapy: Software development and application to helical tomotherapy”. In: *Medical Physics* 41.7 (2014), p. 071709.

[54] E. Craven, J. Mittendorfer, and C. Howard. “Software for Food Irradiation Simulation and Equipment Validation”. In: *Food irradiation technologies: Concepts, applications and outcomes*. Royal Society of Chemistry, 2018. Chap. 7, pp. 105–122.

[55] J. L. Bedford. “Calculation of absorbed dose in radiotherapy by solution of the linear Boltzmann transport equations”. In: *Physics in Medicine & Biology* 64.2 (Jan. 2019), 02TR01. DOI: 10.1088/1361-6560/aaf0e2. URL: https://doi.org/10.1088/1361-6560/aaf0e2.

[56] L. Hoffmann et al. “Validation of the Acuros XB dose calculation algorithm versus Monte Carlo for clinical treatment plans”. In: *Medical Physics* 45.8 (2018), pp. 3909–3915.

[57] F. De Martino et al. “Dose Calculation Algorithms for External Radiation Therapy: An Overview for Practitioners”. In: *Applied Sciences* 11.15 (2021). ISSN: 2076-3417. DOI: 10.3390/app11156806. URL: https://www.mdpi.com/2076-3417/11/15/6806.

[58] *G4Tet.hh*. https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/geometry/solids/specific/include/G4Tet.hh. Accessed: 2022-04-27.

[59] R. L. Martz. “MCNP6 Unstructured Mesh Initial Validation and Performance Results”. In: *Nuclear Technology* 180.3 (2012), pp. 316–335. DOI: 10.13182/NT12-A15347. eprint: https://doi.org/10.13182/NT12-A15347. URL: https://doi.org/10.13182/NT12-A15347.

[60] PHITS Development team. *Features of PHITS 2.82*. 2015. URL: https://phits.jaea.go.jp/image/phits282-upgrade-en.pdf.

[61] N. Borglund et al. “Geometry package for Monte Carlo simulations on CAD files”. In: *Nuclear Instruments and Methods in Physics Research* 525 (1-2 2004), pp. 417–420.

[62] A. Badal et al. “penMesh — Monte Carlo Radiation Transport Simulation in a Triangle Mesh Geometry”. In: *IEEE transactions on medical imaging* 28.12 (2009), pp. 1894–1901. ISSN: 0278-0062.

[63] T. Möller and B. Trumbore. “Fast, minimum storage ray-triangle intersection”. In: *Journal of graphics tools* 2.1 (1997), pp. 21–28.

[64] T. Akenine-Möller. “Fast 3D Triangle-Box Overlap Testing”. In: *ACM SIGGRAPH 2005 Courses*. SIGGRAPH ’05. Los Angeles, California: Association for Computing Machinery, 2005, 8–es. ISBN: 9781450378338. DOI: 10.1145/1198555.1198747. URL: https://doi.org/10.1145/1198555.1198747.

[65] A. Boehnlein et al. *GEANT 4: an Object-Oriented toolkit for simulation in HEP*. Tech. rep. CERN, 1994.

[66] C. M. Poole et al. “A CAD interface for GEANT4”. In: *Australasian physical & engineering sciences in medicine* 35.3 (2012), pp. 329–334. ISSN: 0158-9938.

[67] J. Yang et al. “Research on converting CAD model to MCNP model based on STEP file”. In: *2013 International Joint Conference on Awareness Science and Technology Ubi-Media Computing (iCAST 2013 UMEDIA 2013)*. 2013, pp. 541–547. DOI: 10.1109/ICAwST.2013.6765499.

[68] C. M. Poole et al. “Fast Tessellated Solid Navigation in GEANT4”. In: *IEEE Transactions on Nuclear Science* 59.4 (2012), pp. 1695–1701. DOI: 10.1109/TNS.2012.2197415.

[69] M. C. Han et al. “DagSolid: a new Geant4 solid class for fast simulation in polygon-mesh geometry”. In: *Physics in Medicine & Biology* 58.13 (June 2013), pp. 4595–4609. DOI: 10.1088/0031-9155/58/13/4595. URL: https://doi.org/10.1088/0031-9155/58/13/4595.

[70] H. Si. “TetGen, a Delaunay-based quality tetrahedral mesh generator”. In: *ACM Transactions on Mathematical Software (TOMS)* 41.2 (2015), pp. 1–36.

[71] G. Cosmo. “The Geant4 geometry modeler”. In: *IEEE Symposium Conference Record Nuclear Science 2004*. Vol. 4. IEEE, 2004, 2196–2198 Vol. 4. ISBN: 9780780387003.

[72] T. Barker et al. “Use of tetrahedral mesh geometry to import a converted CAD file for shielding and criticality calculations with MONK and MCBEND”. In: *Proc. 11th International Conference on Radiation Shielding (ICRS-11) and 14th Topical Meeting on Radiation Protection and Shielding (RPS-2008), Georgia, USA (April-2008)*. 2008.

[73] Q. Fang. “Mesh-based Monte Carlo method using fast ray-tracing in Plücker coordinates”. In: *Biomedical optics express* 1.1 (2010), pp. 165–175.

[74] H. Shen and G. Wang. "A tetrahedron-based inhomogeneous Monte Carlo optical simulator". In: *Physics in Medicine & Biology* 55.4 (Jan. 2010), pp. 947–962. DOI: 10.1088/0031-9155/55/4/003. URL: https://doi.org/10.1088/0031-9155/55/4/003.

[75] K. C. Kelley, R. L. Martz, and D. L. Crane. "Riding Bare-Back On Unstructured Meshes For 21st Century Criticality Calculations". In: *Proc. PHYSOR 2010* (2010), pp. 9–14.

[76] R. L. Martz and D. L. Crane. *The MCNP6 Book on Unstructured Mesh Geometry: Foundations*. Tech. rep. LA-UR-12-25478 Rev. 1, MCNP6 code release to RSICC, Oak Ridge, TN, General . . ., 2014.

[77] T. Furuta et al. "Implementation of tetrahedral-mesh geometry in Monte Carlo radiation transport code PHITS". In: *Physics in Medicine & Biology* 62.12 (2017), pp. 4798–4810.

[78] Y. S. Yeom et al. "Computation Speeds and Memory Requirements of Mesh-Type ICRP Reference Computational Phantoms in Geant4, MCNP6, and PHITS". English. In: *Health Physics* 116.5 (May 2019), pp. 664–676. ISSN: 0017-9078. DOI: 10.1097/HP.0000000000000999.

[79] M. C. Han et al. "Multi-threading performance of Geant4, MCNP6, and PHITS Monte Carlo codes for tetrahedral-mesh geometry". In: *Physics in Medicine & Biology* 63.9 (May 2018), 09NT02. DOI: 10.1088/1361-6560/aabd20. URL: https://doi.org/10.1088/1361-6560/aabd20.

[80] Y. Hu et al. "Tetrahedral Meshing in the Wild". In: *ACM Trans. Graph.* 37.4 (July 2018), p. 3. ISSN: 0730-0301. DOI: 10.1145/3197517.3201353. URL: https://doi.org/10.1145/3197517.3201353.

[81] ICRP. *Report of the Task Group on Reference Man*. ICRP Publication 23. Pergamon Press, 1975, p. 4.

[82] W. Kainz et al. "Advances in Computational Human Phantoms and Their Applications in Biomedical Engineering-A Topical Review". In: *IEEE transactions on radiation and plasma medical sciences* 3.1 (2019), pp. 1–23. ISSN: 2469-7311.

[83] K. F. Eckerman et al. "Stylized Computational Phantoms Developed at ORNL and Elsewhere". In: *Handbook of Anatomical Models for Radiation Dosimetry*. Ed. by X. G. Xu and K. F. Eckerman. San Francisco: CRC Press, 2009, pp. 43–61.

[84] R. Behrens, G. Dietze, and M. Zankl. "Dose conversion coefficients for electron exposure of the human eye lens". In: *Physics in Medicine & Biology* 54.13 (June 2009), pp. 4069–4087. DOI: 10.1088/0031-9155/54/13/008. URL: https://doi.org/10.1088/0031-9155/54/13/008.

[85] X. G. Xu and K. F. Eckerman. "Preface". In: *Handbook of Anatomical Models for Radiation Dosimetry*. Ed. by X. G. Xu and K. F. Eckerman. San Francisco: CRC Press, 2009, pp. xi–xiii.

[86] ICRP. "Paediatric Computational Reference Phantoms". In: *Annals of the ICRP* 49 (2020). ICRP Publication 143.

[87] C. H. Kim et al. "The reference phantoms: voxel vs polygon". In: *Annals of the ICRP* 45.1_suppl (2016). PMID: 26969297, pp. 188–201. DOI: 10.1177/0146645315626036. eprint: https://doi.org/10.1177/0146645315626036. URL: https://doi.org/10.1177/0146645315626036.

[88] Y. S. Yeom et al. "Dose coefficients of mesh-type ICRP reference computational phantoms for idealized external exposures of photons and electrons". In: *Nuclear Engineering and Technology* 51.3 (2019), pp. 843–852. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2018.12.006. URL: https://www.sciencedirect.com/science/article/pii/S1738573318306910.

[89] T. T. Nguyen et al. "Incorporation of detailed eye model into polygon-mesh versions of ICRP-110 reference phantoms". In: *Physics in Medicine & Biology* 60.22 (2015), pp. 8695–8707. ISSN: 0031-9155.

[90] C. H. Kim et al. "A polygon-surface reference Korean male phantom (PSRK-Man) and its direct implementation in Geant4 Monte Carlo simulation". In: *Physics in Medicine & Biology* 56.10 (Apr. 2011), pp. 3137–3161. DOI: 10.1088/0031-9155/56/10/016. URL: https://doi.org/10.1088/0031-9155/56/10/016.

[91] C. Lee et al. "Hybrid computational phantoms of the male and female newborn patient: NURBS-based whole-body models". In: *Physics in Medicine & Biology* 52.12 (May 2007), pp. 3309–3333. DOI: 10.1088/0031-9155/52/12/001. URL: https://doi.org/10.1088/0031-9155/52/12/001.

[92] Y. S. Yeom et al. "Tetrahedral-mesh-based computational human phantom for fast Monte Carlo dose calculations". In: *Physics in Medicine & Biology* 59.12 (May 2014), pp. 3173–3185. DOI: 10.1088/0031-9155/59/12/3173. URL: https://doi.org/10.1088/0031-9155/59/12/3173.

[93] Y. S. Yeom et al. "Conversion of ICRP male reference phantom to polygon-surface phantom". In: *Physics in Medicine & Biology* 58.19 (2013), p. 6985.

[94] C. Choi et al. "Development of paediatric mesh-type reference computational phantom series of International Commission on Radiological Protection". In: *Journal of Radiological Protection* 41.3 (Aug. 2021), S160–S170. DOI: 10.1088/1361-6498/ac0801. URL: https://doi.org/10.1088/1361-6498/ac0801.

[95] H. Lee et al. “Percentile-specific computational phantoms constructed from ICRP mesh-type reference computational phantoms (MRCPs)”. In: *Physics in Medicine & Biology* 64.4 (Feb. 2019), p. 045005. DOI: 10.1088/1361-6560/aafcdb. URL: https://doi.org/10.1088/1361-6560/aafcdb.

[96] S. Gholampourkashi et al. “Development of a deformable phantom for experimental verification of 4D Monte Carlo simulations in a deforming anatomy”. In: *Physica Medica* 51 (2018), pp. 81–90. ISSN: 1120-1797. DOI: https://doi.org/10.1016/j.ejmp.2018.05.012. URL: https://www.sciencedirect.com/science/article/pii/S1120179718304691.

[97] T. Bortfeld, S. B. Jiang, and E. Rietzel. “Effects of motion on the total dose distribution”. In: *Seminars in Radiation Oncology* 14.1 (2004). High-Precision Radiation Therapy of Moving Targets, pp. 41–51. ISSN: 1053-4296. DOI: https://doi.org/10.1053/j.semradonc.2003.10.011. URL: https://www.sciencedirect.com/science/article/pii/S1053429603000924.

[98] H. S. Kim et al. “Inclusion of thin target and source regions in alimentary and respiratory tract systems of mesh-type ICRP adult reference phantoms”. In: *Physics in Medicine & Biology* 62.6 (Feb. 2017), pp. 2132–2152. DOI: 10.1088/1361-6560/aa5b72. URL: https://doi.org/10.1088/1361-6560/aa5b72.

[99] Y. S. Yeom et al. “Construction of new skin models and calculation of skin dose coefficients for electron exposures”. In: *Journal of the Korean Physical Society* 69 (2016), pp. 512–517.

[100] Y. S. Yeom et al. “Development of skeletal system for mesh-type ICRP reference adult phantoms”. In: *Physics in Medicine & Biology* 61.19 (Sept. 2016), pp. 7054–7073. DOI: 10.1088/0031-9155/61/19/7054. URL: https://doi.org/10.1088/0031-9155/61/19/7054.

[101] Y. S. Yeom et al. “New small-intestine modeling method for surface-based computational human phantoms”. In: *Journal of Radiological Protection* 36.2 (Mar. 2016), pp. 230–245. DOI: 10.1088/0952-4746/36/2/230. URL: https://doi.org/10.1088/0952-4746/36/2/230.

[102] Y. S. Yeom et al. “Dose coefficients of mesh-type ICRP reference computational phantoms for external exposures of neutrons, protons, and helium ions”. In: *Nuclear Engineering and Technology* 52.7 (2020), pp. 1545–1556. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2019.12.020. URL: https://www.sciencedirect.com/science/article/pii/S1738573319309726.

[103] L. M. Carter et al. “Patient Size-Dependent Dosimetry Methodology Applied to 18F-FDG Using New ICRP Mesh Phantoms”. In: *Journal of Nuclear Medicine* 62.12 (2021), pp. 1805–1814. ISSN: 0161-5505. DOI: 10.2967/jnumed.120.256719. eprint: https://jnm.snmjournals.org/content/62/12/1805.full.pdf. URL: https://jnm.snmjournals.org/content/62/12/1805.

[104] T. G. Primidis et al. “3D chest tomosynthesis using a stationary flat panel source array and a stationary detector: a Monte Carlo proof of concept”. In: *Biomedical Physics & Engineering Express* 8.1 (Nov. 2021), p. 015006. DOI: 10.1088/2057-1976/ac3880. URL: https://doi.org/10.1088/2057-1976/ac3880.

[105] C. Choi et al. “Body-size-dependent phantom library constructed from ICRP mesh-type reference computational phantoms”. In: *Physics in Medicine & Biology* 65.12 (June 2020), p. 125014. DOI: 10.1088/1361-6560/ab8ddc. URL: https://doi.org/10.1088/1361-6560/ab8ddc.

[106] Y. S. Yeom et al. “Posture-dependent dose coefficients of mesh-type ICRP reference computational phantoms for photon external exposures”. In: *Physics in Medicine & Biology* 64.7 (Apr. 2019), p. 075018. DOI: 10.1088/1361-6560/ab0917. URL: https://doi.org/10.1088/1361-6560/ab0917.

[107] Y. S. Yeom et al. “INVESTIGATION OF THE INFLUENCE OF THYROID LOCATION ON IODINE-131 S VALUES”. In: *Radiation Protection Dosimetry* 189.2 (Apr. 2020), pp. 163–171.

[108] Y. S. Yeom et al. “Iodine-131 S values for use in organ dose estimation of Korean patients in radioiodine therapy”. In: *Nuclear Engineering and Technology* 54.2 (2022), pp. 689–700. ISSN: 1738-5733. DOI: https://doi.org/10.1016/j.net.2021.08.027. URL: https://www.sciencedirect.com/science/article/pii/S1738573321005155.

[109] Y. Yin et al. “Physical dosimetric reconstruction of a case of large area back skin injury due to overexposure in an interventional procedure”. In: *Radiation Medicine and Protection* (2022). ISSN: 2666-5557. DOI: https://doi.org/10.1016/j.radmp.2022.02.001. URL: https://www.sciencedirect.com/science/article/pii/S2666555722000077.

[110] B. Shin et al. “Detailed tooth models for ICRP mesh-type reference computational phantoms”. In: *Journal of Radiological Protection* 41.4 (Sept. 2021), pp. 669–688. DOI: 10.1088/1361-6498/abeaf9. URL: https://doi.org/10.1088/1361-6498/abeaf9.

[111] P. Fattibene and F. Callens. “EPR dosimetry with tooth enamel: A review”. In: *Applied Radiation and Isotopes* 68.11 (2010), pp. 2033–2116.

[112] C. Correa-Alfonso et al. “A mesh-based model of liver vasculature: implications for improved radiation dosimetry to liver parenchyma for radiopharmaceuticals”. In: *EJNMMI Physics* 9.28 (2022). URL: https://doi.org/10.1186/s40658-022-00456-0.

[113] C. Choi et al. “Development of skeletal systems for ICRP pediatric mesh-type reference computational phantoms”. In: *Journal of Radiological Protection* 41.2 (June 2021), pp. 139–161.

[114] H. Han, C. Choi, B. Shin, et al. “Pediatric lens dose coefficients for photons and electrons: dosimetric impact of detailed eye models for ICRP paediatric mesh-type reference computational phantoms”. In: *Trans. Korean. Nucl. Soc., Virtual Autumn Meeting*. 2020.

[115] E. Kollitz et al. “A patient-specific hybrid phantom for calculating radiation dose and equivalent dose to the whole body”. In: *Physics in Medicine & Biology* 67.3 (Feb. 2022), p. 035005.

[116] W. K. Fum, J. H. D. Wong, and L. K. Tan. “Monte Carlo-based patient internal dosimetry in fluoroscopy-guided interventional procedures: A review”. In: *Physica Medica* 84 (2021), pp. 228–240.

[117] I. Kawrakow et al. *EGS_BaseGeometry*. Accessed: 2022-03-23. 2019. URL: https://nrc-cnrc.github.io/EGSnrc/doc/pirs898/classEGS__BaseGeometry.html.

[118] C. Ericson. “Chapter 3 - A Math and Geometry Primer”. In: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 23–73.

[119] C. Ericson. “Chapter 5 - Basic Primitive Tests”. In: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 125–233.

[120] *IEEE Std 754-2019 (Revision of IEEE 754-2008): IEEE Standard for Floating-Point Arithmetic*. IEEE, 2019. ISBN: 1-5044-5924-5.

[121] S. Woop, C. Benthin, and I. Wald. “Watertight ray/triangle intersection”. In: *Journal of Computer Graphics Techniques (JCGT)* 2.1 (2013), pp. 65–82.

[122] R. Löhner. “Data Structures and Algorithms”. In: *Applied Computational Fluid Dynamics Techniques*. John Wiley & Sons, Ltd, 2008. Chap. 2, pp. 7–33.

[123] A. F. Bielajew and D. W. O. Rogers. “PRESTA: The Parameter Reduced Electron-Step Transport Algorithm for Electron Monte Carlo Transport”. In: *Nucl. Instrum. Meth. B* 18 (1986), p. 165. DOI: 10.1016/S0168-583X(86)80027-1.

[124] A. F. Bielajew. *HOWFAR and HOWNEAR: Geometry modeling for Monte Carlo particle transport*. Tech. rep. PIRS-0341. National Research Council of Canada, 1995.

[125] S. Hardinger. *Illustrated Glossary of Organic Chemistry: Atomic Radius*. http://www.chem.ucla.edu/~harding/IGOC/A/atomic_radius.html. Accessed: 2022-03-22.

[126] N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM, 2002.

[127] M. Pharr, W. Jakob, and G. Humphreys. *Physically based rendering: From theory to implementation*. 3rd ed. Morgan Kaufmann, 2016.

[128] M. Garrity. "Raytracing irregular volume data". In: *Proceedings of the 1990 workshop on volume visualization*. VVS '90. ACM, 1990, pp. 35–40. ISBN: 9780897914178.

[129] A. S. Glassner. "Space subdivision for fast ray tracing". In: *IEEE Computer Graphics and Applications* 4.10 (1984), pp. 15–24. DOI: 10.1109/MCG.1984.6429331.

[130] C. Ericson. "Chapter 7 - Spatial Partitioning". In: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 285–347.

[131] C. Ericson. "Chapter 6 - Bounding Volume Hierarchies". In: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 235–284.

[132] A. S. Glassner. "Space subdivision for fast ray tracing". In: *IEEE computer graphics and applications* 4.10 (1984), pp. 15–24. ISSN: 0272-1716.

[133] I. Gargantini and H. H. Atkinson. "Ray Tracing an Octree: Numerical Evaluation of the First Intersection". In: *Computer graphics forum* 12.4 (1993), pp. 199–210. ISSN: 0167-7055.

[134] S. Laine and T. Karras. "Efficient Sparse Voxel Octrees". In: *IEEE Transactions on Visualization and Computer Graphics* 17.8 (2011), pp. 1048–1059. DOI: 10.1109/TVCG.2010.240.

[135] C. Ericson. "Chapter 13 - Optimization". In: *Real-Time Collision Detection*. The Morgan Kaufmann Series in Interactive 3D Technology. San Francisco: Morgan Kaufmann, 2005, pp. 511–551.

[136] ICRP. "The 2007 Recommendations of the International Commission on Radiological Protection". In: *Annals of the ICRP* 37 (2007). ICRP Publication 103.

[137] U. Fano. "Note on the Bragg-Gray Cavity Principle for Measuring Energy Dissipation". In: *Radiation research* 1.3 (1954), pp. 237–240. ISSN: 0033-7587.

[138] I. Kawrakow et al. *EGS_FanoSource Class Reference*. Accessed: 2022-05-02. 2019. URL: https://nrc-cnrc.github.io/EGSnrc/doc/pirs898/classEGS__FanoSource.html.

[139] R. Townson et al. *Getting Started with EGSnrc*. Tech. rep. National Research Council of Canada, 2021.

[140] M. J. Pratt. "Introduction to ISO 10303—the STEP Standard for Product Data Exchange". In: *Journal of Computing and Information Science in Engineering* 1.1 (Jan. 2001), pp. 102–103.

[141] C. Geuzaine and J.-F. Remacle. "Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities". In: *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331.

[142] ICRP. *ICRP 145: Adult Mesh-type Reference Computational Phantoms*. `https://www.icrp.org/publication.asp?id=ICRP%20Publication%20145`. Accessed: 2022-03-07.

[143] M. Berger, J. Coursey, and M. Zucker. *ESTAR, PSTAR, and ASTAR: Computer Programs for Calculating Stopping-Power and Range Tables for Electrons, Protons, and Helium Ions (version 1.21)*. Jan. 1999. URL: `http://physics.nist.gov/Star`.

[144] E. Tiesinga et al. *NIST Values of Fundamental Physical Constants*. 2018. URL: `https://physics.nist.gov/cuu/Constants/index.html`.

[145] ICRP. "Conversion Coefficients for Radiological Protection Quantities for External Radiation Exposures". In: *Annals of the ICRP* 40 (2010). ICRP Publication 116.

[146] ICRP. "The ICRP Computational Framework for Internal Dose Assessment for Reference Adults: Specific Absorbed Fractions". In: *Annals of the ICRP* 45 (2010). ICRP Publication 133, pp. 1–74.

[147] B. R. B. Walters and I. Kawrakow. "A "HOWFARLESS" option to increase efficiency of homogeneous phantom calculations with DOSXYZnrc". In: *Medical Physics* 34.10 (2007), pp. 3794–3807.

[148] K. Babcock, G. Cranmer-Sargison, and N. Sidhu. "An enhanced HOWFARLESS option for DOSXYZnrc simulations of slab geometries". In: *Medical Physics* 35.9 (2008), pp. 4106–4111.

[149] S.-E. Yoon et al. "Cache-Oblivious Mesh Layouts". In: *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. Los Angeles, California: Association for Computing Machinery, 2005, pp. 886–893. ISBN: 9781450378253.

[150] I. Wald et al. "Embree: A Kernel Framework for Efficient CPU Ray Tracing". In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: `10.1145/2601097.2601199`. URL: `https://doi.org/10.1145/2601097.2601199`.

[151] Intel. *Intel Intrinsics Guide*. `https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html`. Accessed: 2022-03-07.