Politecnico di Torino

Corso di Laura Magistrale in
Ingegneria Energetica e Nucleare

Sustainable Nuclear Energy

Academic Year 2024/2025


# Methods for verification and preliminary validation test of the geometry navigation engine in the Monte Carlo transport code TRIPOLI-5®.

Academic Tutor:
Prof. Sandra Dulla

Company Tutors:
Coline Larmier
Davide Mancusi

Candidate:
Vincenzo Di Blasi
s289048

17 March 2025

# Abstract

Monte Carlo simulation is a widely used methodology in particle physics, nuclear medicine, reactor engineering, and many other scientific and engineering fields. One of the crucial aspects of these simulations is the transport of particles through complex geometries, which requires a highly accurate and efficient geometry navigation engine. The geometry navigation engine is responsible for determining the path of particles through materials and structures with complex geometries. This process includes managing particle interactions with various materials, determining collision points, and updating particle trajectories based on their interactions. The accuracy of this component is essential to ensure that the simulation results are reliable.
TRIPOLI-5® represents a cutting-edge advancement in the realm of massively parallel and natively HPC-oriented stochastic particle transport codes, a collaborative development effort between CEA and IRSN initiated in 2022. Central to the efficacy of a Monte Carlo transport code is the accurate representation of the system geometry under investigation. This involves decomposing the domain into volumes, each characterized by a specific material composition. The geometry navigation engine plays a crucial role, providing critical information such as the material composition in which a particle resides, the distance to the nearest geometrical boundary, and the subsequent volume the particle will traverse. Initially, TRIPOLI-5® relied exclusively on a geometry navigator utilizing the ROOT package, a robust tool developed by CERN. However, recognizing the need for enhanced performance and flexibility, a novel native geometry engine, AGORA, has been integrated into TRIPOLI-5®.
AGORA adeptly handles nested geometries through the use of "universes", with each universe comprising elementary volumes defined using Constructive Solid Geometry (CSG). A CSG volume is depicted using a combination of half-spaces defined by elementary surfaces (such as planes, spheres, cylinders, cones, or quadrics) and Boolean operators (intersection, union, or complement). The overarching geometry adopts a tree-like structure, where each volume may contain a material (acting as a leaf of the tree), a universe, or a lattice, which itself contains a list of universes.
The objective of this thesis is to develop methods to rigorously verify the reliability of the AGORA native geometry engine and to conduct preliminary validation test to ensure its integration with the TRIPOLI-5® Monte Carlo transport code. In this work we conducted analyses comprising two primary approaches: internal comparisons within TRIPOLI-5® between AGORA and the ROOT-based navigator, and external comparisons between TRIPOLI-5® (utilizing AGORA) and other Monte Carlo transport codes such as OpenMC or TRIPOLI-4®.
We developed specific datasets and scripts for each code to describe a variety of

geometrical configurations, ranging from elementary shapes like nested spheres or cylinders, to complex geometries inspired by international benchmarks, such as the Hoogenboom-Martin benchmark, which features multiple lattice levels.
Our methodology involved sampling a substantial number of particles from given positions and comparing the materials identified by the different geometry engines. Furthermore, we verified particle tracking by comparing the chord lengths traversed in each material by the simulated particles from a given point in a specified direction.
A preliminary verification test study of AGORA was conducted in tandem with the TRIPOLI-5® Monte Carlo code, with a focus on the SPERT-III experimental benchmark. This study aimed to assess the accuracy and reliability of the AGORA module in a valuable benchmark case in reactor physics calculations.
In addition, we have employed AGORA to verify Cauchy's formula in simple but not trivial configuration, examining the average chord length for isotropic and uniform surface irradiation in a given body, which is articulated as a function of the body volume and its bounding surface.

# Acknowledgements

I am deeply grateful to Professor Sandra Dulla for sparking my interest in this field through her clear explanations and contagious passion during my course of study. Her unwavering dedication to fostering the growth of her students and their integration into professional environments deserves my deepest thanks.

I would like to express my sincere gratitude to my tutors, Coline Larmier and Davide Mancusi. Throughout my internship, their guidance, insightful discussions and willingness to challenge my perspective, have been invaluable in raising a space for learning and personal growth. Their highly collaborative approach to work will remain an exemplary model for me, one that I will always strive to follow.

I extend my sincere appreciation to the Head of LTSD, Andrea Zoia, for the always enlightening discussions, the incisive advices, and his exceptional availability. His genuine interest in the work of all his collaborators and his willingness to share his expertise has been a source of inspiration and admiration.

My final thanks go to the Intern Antoine Dervaux, to his Tutor Matthieu Lemaire as well as to Shai Kinast (Nuclear Research Center Negev, Israel) for generously sharing their work with me. Their collaboration has been truly important, and I sincerely hope I have been as much of help to their work as they have been to mine. This experience has significantly contributed to my personal growth, strengthening my teamwork skills and teaching me the immense value of cooperation.

# Contents

# List of Figures

# List of Tables

# Introduction

The continuous advancement in high performance and parallel computing has driven the ongoing development of Monte Carlo codes, which today serve as a scientific gold standard for simulating complex physical phenomena. Thanks to their ability to model stochastic processes with a high degree of accuracy, Monte Carlo codes are employed not only for the direct resolution of engineering and physics problems but also for the validation of deterministic codes.
The evolution of computing architectures has led to a significant increase in computational performance, allowing simulations to be conducted on an increasingly larger scale and with an unprecedented level of detail. The reduction in simulation times and the enhancement of computational efficiency represent a true breakthrough, with important implications in both academic and industrial fields. These improvements not only enable the exploration of new physical and engineering scenarios with greater precision but also overlay the way for an increasing integration between Monte Carlo methods and deterministic approaches, combining the strengths of both to optimize the modeling and analysis of complex systems. Alongside advancements in numerical computation, there has been a significant evolution in the geometric representation within simulations. The geometric models used in modern transport codes are becoming increasingly detailed and more faithful to the physical reality of experiments and structures under analysis. The ability to accurately represent intricate and complex geometries is a crucial step in ensuring reliable simulations, reducing uncertainties in results, and improving the predictive capability of the models.
Accurate geometric definition is not only a technical requirement but also a strategic element for the reliability of simulations: a geometrically realistic model allows for results that are closer to experimental conditions, improving quality of results and reducing the need for simplifying approximations that could compromise the accuracy of calculations. In this context, the development of accurate Geometry Navigation Engines and the associated tools for their verification are an important research area within Monte Carlo codes.
Recent advancements in this field have focused on enhancing the efficiency and robustness of geometry navigation algorithms, enabling faster and more precise calculations within increasingly complex models. These improvements not only refine the accuracy of simulations but also contribute to greater computational efficiency, ensuring that Monte Carlo methods remain a gold standard across scientific and engineering applications.
During my internship at the French *Commissariat à l'Énergie Atomique et aux*

*Énergies Alternatives* (CEA) at Paris-Saclay, I had the opportunity to deepen my understanding of Monte Carlo codes for neutron transport within the context of nuclear research. The CEA is a major player in high-level research, providing concrete solutions to a wide range of needs in four main areas: low-carbon energies (nuclear and renewable), digital technologies, technologies for the medicine of the future, defense and security.
Within the CEA, the *Service d'Études des Réacteurs et de Mathématiques Appliquées* (SERMA) is a specialized unit dedicated to the development of advanced numerical simulation software for both deterministic and stochastic particle transport. This thesis is based on the work conducted during a six-month internship at the *Laboratoire du Transport Stochastique et des Données* (LTSD), a laboratory within SERMA focused on the development and optimization of stochastic transport models for particles, under the supervision of Drs. Coline Larmier and Davide Mancusi.
The LTSD is actively involved in the development of Monte Carlo transport codes, ensuring their efficiency and reliability for simulations that are crucial to reactor design and safety assessments. The methodologies and computational tools developed within this laboratory play a fundamental role in advancing neutron transport simulations, contributing to key areas such as reactor physics, radiation shielding, and safety evaluations. Research in this field continues to evolve, with the goal of enhancing computational performance and improving simulation accuracy, thereby supporting technological advancements in nuclear engineering.
Some of the findings related to the methodology developed for the verification of the AGORA geometry navigation engine, as well as the results obtained, will be featured in the paper *"C. Larmier, D. Mancusi, V. Di Blasi, A. Zoia, Verification Methods for the AGORA Geometry Navigation Engine of the TRIPOLI-5® Monte Carlo Code"*, which will appear in the proceedings of the M&C2025 conference.

# Chapter 1

# Theoretical Background

## 1.1 Neutron Transport Theory

Neutron transport theory plays a fundamental role in nuclear science and engineering, as it provides a framework for studying the behavior and propagation of neutrons within various media, along with the associated phenomena such as transmutation, activation, and radiation damage. Accurate modeling of neutron transport is essential for the design, optimization, and safety analysis of any nuclear system.
The neutron transport equation describes the time-dependent evolution of the neutron flux $\phi(\mathbf{r}, E, \mathbf{\Omega}, t)$ in space ($\mathbf{r}$), energy ($E$), and direction ($\mathbf{\Omega}$):

$$\frac{1}{v}\frac{\partial\phi(\mathbf{r}, E, \mathbf{\Omega}, t)}{\partial t} + \mathbf{\Omega}\cdot\nabla\phi(\mathbf{r}, E, \mathbf{\Omega}, t) + \Sigma(\mathbf{r}, E)\phi(\mathbf{r}, E, \mathbf{\Omega}, t) =$$

$$\oint_{4\pi}\int_0^\infty \Sigma_s(\mathbf{r}, E')f_s(\mathbf{r}, \mathbf{\Omega}', \mathbf{\Omega}, E' \to E)\phi(\mathbf{r}, E', \mathbf{\Omega}', t)\, d\mathbf{\Omega}'\, dE' +$$

$$\frac{\chi(\mathbf{r}, E)}{4\pi}\int_0^\infty \nu(\mathbf{r}, E')\Sigma_f(\mathbf{r}, E')\phi(\mathbf{r}, E', \mathbf{\Omega}', t)\, dE' + S(\mathbf{r}, E, \mathbf{\Omega}, t)$$

The first term on the left-hand side describes the time evolution of the neutron population, representing how the neutron flux changes over time. This change is governed by the neutron velocity $v$, where the rate of variation in the flux is influenced by both the motion of neutrons through the medium and their interactions.
The second term on the left-hand side, known as the streaming term, accounts for the movement of neutrons through the medium, describing how neutrons propagate in different directions. The third term represents the collision term, capturing the loss of neutrons due to interactions with the material. This term incorporates the total reaction rate, represented by $\Sigma\phi$, where $\Sigma$ is the total macroscopic cross section, quantifying the probability per unit path length that a neutron will interact with the material, either through absorption, scattering, or other nuclear processes.
The right-hand side of the neutron transport equation contains the source terms, which represent the various ways neutrons are introduced into the system.
The scattering source term involves the scattering macroscopic cross section $\Sigma_s$, which quantifies the probability of a neutron undergoing scattering per unit path

length. This term also accounts for the type of scattering (elastic or inelastic) and incorporates the scattering probability density function $f_s$. This density function depends on the position, energy, and direction of the neutron both before and after the collision. When integrated over both angle and energy, this function gives the probability that a neutron will scatter into a specific energy and direction range.
The fission source term represents neutrons that are emitted due to fission events within the system. The term $\chi(\mathbf{r}, E)$, known as the fission spectrum, describes the probability of neutron emission at a given energy $E$. Fission events are often modeled as isotropic, which introduces a factor of $1/4\pi$ to normalize the emission direction. Additionally, $\nu$ represents the number of neutrons emitted per fission event, and $\Sigma_f$ is the fission macroscopic cross section, quantifying the probability of a neutron inducing a fission event per unit path length.
Finally, the external source term $S$ accounts for any external sources of neutrons acting on the system, such as neutron generators or external radiation sources, which contribute neutrons into the system from outside the medium being modeled.

## 1.2 Deterministic and Stochastic Methods

The integro-differential equation which govern neutron transport can be solved using both deterministic and stochastic numerical approaches.
The deterministic approach involves solving a system of partial differential equations that describe the behavior of neutrons in the medium of interest. These equations typically cannot be solved analytically due to their complexity, requiring discretization in space, energy, angular direction, and time. Over the past five decades, various advanced numerical methods have been developed to solve the transport equation with high accuracy. Notable examples include the method of characteristics, the spherical harmonics ($P_N$) method, and the discrete ordinates ($S_N$) method. Despite their sophistication, deterministic methods yield approximate solutions due to the inherent limitations introduced by discretization and truncation errors.
One significant advantage of deterministic methods is that, once the transport equation is solved, the results provide a reliable first approximation of the behavior of a nuclear device. These methods are particularly useful in systems where a balance between computational efficiency and accuracy is required.
The stochastic numerical approach relies on probability and statistical theories, commonly referred to as the Monte Carlo method. This method provides an accurate representation of the propagation of randomly moving particles, with no inherent approximations in principle. It is suited for handling complex geometries and generally delivers results that surpass the accuracy of deterministic methods. However, a notable limitation of the Monte Carlo method is its inability to provide a rapid, approximate solution to the transport equation. This arises from its probabilistic nature, which requires the convergence of stochastic results to a mean value. Achieving results with low statistical error necessitates many simulations, which in turn demands substantial computational resources.
However, the Monte Carlo method has become the golden standard for neutron

transport calculations due to its high accuracy and flexibility. Its adoption is gaining even more momentum thanks to the rapid development of High Performance Computing (HPC), which significantly reduces the computational time required to perform a sufficient number of simulations.

## 1.3 Mathematical Foundation of the Monte Carlo Method

The primary principle behind Monte Carlo simulations is that the macroscopic behavior of a large ensemble of neutrons can be predicted by applying statistical laws governing random processes. A useful analogy is the roll of a die: while the result of any individual roll is uncertain, the average result of many rolls will converge toward the expected value, with approximately equal occurrences of each possible outcome. Similarly, Monte Carlo simulations track the histories of a large number of neutrons to predict the average behavior of the entire population.
Monte Carlo simulations rely on the axiomatic framework of probability theory, which provides a rigorous foundation for modeling random phenomena. The following key properties of probability are fundamental to the formulation of stochastic models:

- **Non-negativity**: The probability of any event $E$ is non-negative, i.e., for any event $E \subset \Omega$, the probability $P(E) \geq 0$, where $\Omega$ is the sample space.

- **Normalization**: The probability of the entire sample space is unity, i.e., $P(\Omega) = 1$, ensuring that one of the possible outcomes must occur.

- **Additivity**: For any mutually exclusive events $E_i$ and $E_j$ (i.e., $E_i \cap E_j = \emptyset$), the probability of their union is the sum of their individual probabilities, i.e., $P(E_i \cup E_j) = P(E_i) + P(E_j)$.

## 1.4 Random Variables and Distributions

A random variable $\xi$ is a measurable function that maps outcomes from the sample space $\Omega$ to real numbers in $\mathbb{R}$. It serves as a mathematical model for quantifying random phenomena. For example, consider the toss of a fair coin. The sample space $\Omega$ consists of two possible outcomes: $\{\text{Head}, \text{Tail}\}$, which we can label as $\{1, 0\}$, where 1 represents head and 0 represents tail. The random variable $\xi$ maps these outcomes to real numbers: $\xi = 1$ for heads and $\xi = 0$ for tails. The probability of each outcome is $\frac{1}{2}$, so we have $P(\xi = 1) = \frac{1}{2}$ and $P(\xi = 0) = \frac{1}{2}$.
In this case, the random variable $\xi$ models the result of a coin toss, and its distribution is defined by the probabilities of getting heads or tails.

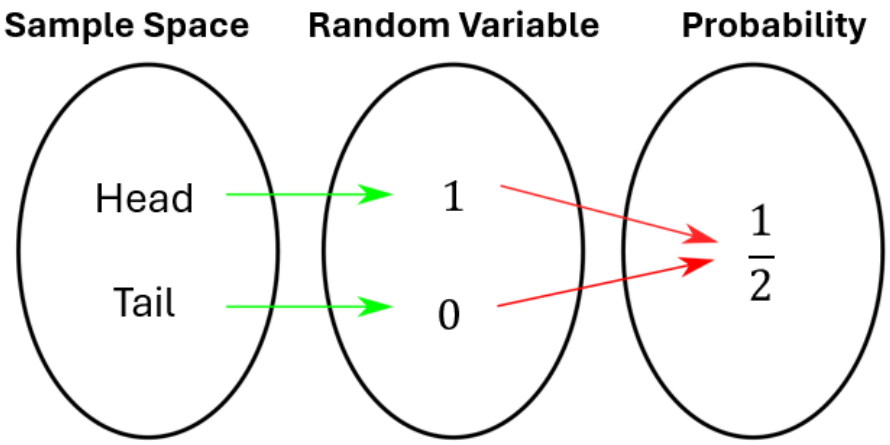


Figure 1.1: Example of random variable definition.

In Monte Carlo simulations for neutron transport, random variables are used to model key quantities such as particle displacement, which represents the neutron's position as it moves through a medium, and interaction types, determining whether a neutron undergoes a scattering event, is absorbed, or undergoes a fission event. Additionally, scattering angles are modeled by random variables to simulate the direction change of neutrons during collisions. These random variables enable the probabilistic simulation of neutron behavior, allowing predictions of macroscopic properties like flux, reaction rates, and energy deposition.
The *Cumulative Distribution Function* (CDF) $F_\xi(x)$ of a random variable $\xi$ is defined as:

$$F_\xi(x) = P[\xi \leq x],$$

where $P[\xi \leq x]$ represents the probability that $\xi$ takes a value less than or equal to $x$.
The CDF possesses the following fundamental properties:

$$\begin{cases} \lim_{x\to-\infty} F_\xi(x) = 0, \\ \lim_{x\to+\infty} F_\xi(x) = 1, \\ \lim_{x\to x_0^+} F_\xi(x) = F_\xi(x_0), \\ F_\xi(x) \text{ is monotonically non-decreasing.} \end{cases}$$

These properties ensures that the CDF approaches 0 as $x$ approaches negative infinity, and approaches 1 as $x$ increases to plus infinity. The monotonicity of $F_\xi(x)$ ensures that the probability of $\xi$ being less than or equal to any value $x$ never decreases as $x$ increases.

The *Probability Density Function* (PDF), denoted by $f_\xi(x)$, is the derivative of the CDF with respect to $x$:

$$f_\xi(x) = \frac{dF_\xi(x)}{dx}.$$

The PDF provides a continuous representation of the probability distribution of the random variable, and the integral of the PDF over any interval $[a, b]$ gives the probability that the random variable $\xi$ falls within that interval:

$$P(a \leq \xi \leq b) = \int_a^b f_\xi(x)\, dx.$$

## 1.5 Moments and Statistical Quantities

In order to obtain average information about a statistical phenomenon, it is necessary to define the concept of the *moments* of a probability distribution. The $n$-th order moment is expressed as:

$$E[x^n] = \int_{-\infty}^{+\infty} x^n f_\xi(x)\, dx,$$

where $f_\xi(x)$ is the PDF associated with the random variable $\xi$.
The PDF $f_\xi(x)$ gives the likelihood that $\xi$ will take a value in an infinitesimally small range around $x$.
For $n = 1$, the result of the integral is the mean value:

$$E[x] = \int_{-\infty}^{+\infty} x f_\xi(x)\, dx$$

Here, $E[x]$ represents the expected value or mean of the random variable $\xi$. The integral computes the weighted average of all possible values of $\xi$, where each value is weighted by the probability density function $f_\xi(x)$. This formula gives us the central tendency or average value of the distribution of $\xi$.
Another fundamental quantity is the variance (the mean squared error):

$$\sigma^2[x] = \int_{-\infty}^{+\infty} (x - E[x])^2 f_\xi(x)\, dx = E[(x - E[x])^2]$$

The variance $\sigma^2[x]$ quantifies the spread or dispersion of the random variable $\xi$ around its mean $E[x]$. The variance provides insight into the variability of the phenomenon under study. A higher variance indicates that $\xi$ can take values that are far from the mean, while a lower variance suggests that $\xi$ is tightly clustered around its mean.
The square root of the variance is called the standard deviation:

$$\sigma[x] = \sqrt{\sigma^2} = \sqrt{E[(x - E[x])^2]}$$

The standard deviation $\sigma[x]$ serves as a measure of the average deviation of the values of the random variable $\xi$ from the mean, and, unlike the variance, it has the same units as the original variable, making it more interpretable and providing a more intuitive understanding of the spread of the values. It is commonly used to express the precision of a statistical phenomenon, as it reflects the typical magnitude

of deviation of individual values from the mean. Finally, the relative standard deviation is defined as:

$$RSD = \frac{\sigma}{E[x]}$$

The RSD is the ratio of the standard deviation to the mean. It is often used to express the dispersion relative to the size of the mean, providing a dimensionless quantity that is particularly useful when comparing the variability of different data sets or variables with different units. The RSD gives a sense of how large the spread is compared to the central value of the distribution.

## 1.6 Sample Average and its Statistical Properties

Consider performing $N$ experiments, each associated with a value of the random variable $\xi_i$. The values $\xi_i$ obtained in the different experiments are random numbers, identically distributed according to the same probability density function (PDF), and statistically independent.
In this context, a new random variable, known as the sample average, can be defined as:

$$\bar{\xi}_N = \frac{1}{N}\sum_{i=1}^{N}\xi_i,$$

where the sample average $\bar{\xi}_N$ represents the arithmetic mean of the observed values over the $N$ experiments. The following statistical properties of the sample average can be derived, namely the expected value and variance:

$$E[\bar{\xi}_N] = E[x] = \mu,$$

$$\sigma^2[\bar{\xi}_N] = \frac{\sigma^2[x]}{N},$$

These equations are fundamental as they indicate that the random variable $\bar{\xi}_N$ serves as an estimator of the expected value $\mu$, and its dispersion with respect to the mean value is characterized by the variance $\sigma^2[\bar{\xi}_N]$.
Importantly, as the number of experiments $N$ increases, the dispersion of the sample average decreases, resulting in a more precise estimate of the expected value.

## 1.7 Fundamental Theorems and Statistical Errors

The mathematical foundation of the Monte Carlo method is rooted in two key theorems. The first of these is the Tchebycheff Inequality applied to the sample average:

$$P[|\bar{\xi}_N - \mu| > k] \leq \frac{\sigma^2}{Nk^2},$$

which states that as the number of experiments $N$ increases, the sample average converges in probability to the true mean value $\mu$. This result provides a bound on

the probability that the deviation of the sample average from the true mean exceeds a given threshold $k$.
While the Tchebycheff Inequality provides valuable information about the behavior of the sample average, it does not specify the difference between the sample average and the true mean for a finite number of experiments. This type of information is provided by the Central Limit Theorem (CLT), which asserts that, as the number of experiments $N$ tends to infinity, the distribution of the sample average $\bar{\xi}_N$ approaches a normal distribution.
Applying this theorem to the relations derived from the Tchebycheff Inequality it can be demonstrated that:

$$P\left(\left|\frac{\xi^{(N)}-\mu}{\sqrt{\frac{\sigma^2}{N}}}\right| \leq k\right) \approx \begin{cases} 0.68 & \text{if } k=1 \\ 0.95 & \text{if } k=2 \\ 0.99 & \text{if } k=3 \end{cases}$$

This relation indicates that the probability that the relative error of the sample average is within $k$ times the relative standard deviation of the sample average takes specific values.
By using the CLT, it becomes possible to quantify the statistical error associated with the results of Monte Carlo simulations. For example, if $k = 1$, this implies that there is approximately a 68% probability that the expected value lies within the error bar. Another important result from the CLT is that the convergence rate of the sample average to the true mean follows a rate of $1/\sqrt{N}$, which is critical for estimating the precision of Monte Carlo simulations.

## 1.8 Random Walks

The Monte Carlo method, fundamental in the study of neutron behavior, relies on probabilistic modeling to simulate the random walks of neutrons as they interact with their surrounding medium. By tracking the trajectory of each neutron from its birth, likely initiated by a fission event, to its death through absorption or leakage, this technique offers a detailed representation of the neutron's life cycle. The trajectory is represented as a series of events, or a "history," which depends on the position, energy, and direction of the particle, the distance between two consecutive events, and the type of interaction.
The simulation process of a neutron's random walk in a homogeneous medium is based on a series of successive steps that model its free motion and the interactions it may undergo, such as scattering or absorption. The initial position of the neutron, represented by $\vec{r}$, and its direction of motion $\vec{\Omega}$, can originate from a fission source or from a neutron that has previously undergone an interaction. The neutron's movement is described through a probability distribution, which reflects the distance traveled before the neutron undergoes an interaction.
The probability that a neutron moves from a point $P = (\vec{r}, \vec{\Omega}, E)$ to another point $P' = (\vec{r'}, \vec{\Omega}, E)$ without interacting depends on the distance between the two points

and the energy of the neutron, and it is expressed by the free flight PDF $f_l$. This function, which takes into account the macroscopic total cross section $\Sigma_t(E)$, is given by:

$$f_l(|\vec{r} - \vec{r'}|, E) = \Sigma_t(E) e^{-[|\vec{r}-\vec{r'}|\Sigma_t(E)]}$$

In this context, the macroscopic cross section $\Sigma_t(E)$ represents the probability of interaction per unit distance traveled by the neutron. The cumulative distribution function that describes the probability that a neutron will reach the point $P'$ is:

$$F_l(|\vec{r} - \vec{r'}|, E) = 1 - e^{-[|\vec{r}-\vec{r'}|\Sigma_t(E)]}$$

To generate random free flight trajectories following this distribution, the inverse transform method is used. In this approach, a random number $\rho \in [0, 1]$ is generated, and the equation $F_l(s, E) = \rho$ is solved, where $s$ is the distance traveled by the neutron. Solving for $s$, we obtain:

$$s = -\frac{1}{\Sigma_t(E)} \log(1 - \rho)$$

This step allows the determination of the distance traveled by the neutron before any potential interaction.
Once the distance traveled is calculated, the neutron's position is updated. It is then determined whether the neutron remains within the domain of interest. If the neutron exits from the domain of interest, the random walk ends due to leakage outside the geometry. Otherwise, the next step is to determine the type of interaction the neutron may undergo. Two possible events must be considered: collision or crossing a geometry boundary. If the neutron crosses a boundary within the geometry but remains inside the overall domain, the random walk does not end. Instead, the neutron is repositioned at the boundary; the current volume and corresponding material properties are updated and a new distance to the next collision is sampled based on the properties of the new material.
In a collision event, the neutron may undergo one of three possible interactions: scattering, capture, or fission, with probabilities proportional to the respective macroscopic cross sections, $\Sigma_s(E)$, $\Sigma_c(E)$, and $\Sigma_f(E)$. If capture occurs, the neutron is removed from the system, and the random walk ends. In the case of fission, the neutron is absorbed, but the interaction produces secondary particles, including new neutrons, which contribute to the overall neutron population. If the neutron undergoes scattering, it changes direction and resumes its free flight process. The random walk continues as the neutron travels through the medium, undergoing successive interactions until either absorption occurs or the neutron leaks from the domain.
The random walk model simulates the behavior of a neutron moving through a medium, where the distance traveled between interactions is determined by the macroscopic cross section. This approach provides a detailed and probabilistic simulation of neutron transport, continuously updating the neutron's position and determining its interactions based on the probabilities associated with each physical

event. To ensure accurate results with relatively low uncertainty, a large number of neutron histories must be simulated, which, while computationally demanding, guarantees high precision.

# Chapter 2

# TRIPOLI® and AGORA

## 2.1 The Evolution of TRIPOLI® Codes

The TRIPOLI® family of codes represents a sophisticated suite of tools designed for radiation transport simulations based on the Monte Carlo method. Developed to address diverse applications such as reactor physics, radiation protection, and nuclear instrumentation, these codes have been continuously refined over decades. Their development began in the 1960s under the CEA, initially at the Fontenay-aux-Roses laboratories before moving to Saclay [1]. Over time, TRIPOLI® has evolved into a cornerstone of nuclear analysis and simulation, supported by an ecosystem of complementary tools. This ecosystem includes deterministic codes for lattice and core simulations, software for modeling fuel depletion, photon transport tools, and advanced systems for evaluating and processing nuclear data.

At the heart of this family lies TRIPOLI-4® [2], the fourth generation of the series, which serves as a key resource for CEA's simulation activities.

TRIPOLI-4® has established itself as the reference code for not only CEA laboratories but also for major external organizations. For example, *Électricité de France* (EDF), responsible for operating a fleet of nuclear reactors in France, relies on this tool for its simulations and operational needs.

This Monte Carlo code excels in its ability to simulate neutral particles such as neutrons, photons, and even charged particles like electrons and positrons. Its energy ranges are broad, covering neutrons from 20 MeV to extremely low energies and photons from 20 MeV to 1 keV. For electrons and positrons, simulations extend down to 1 keV, enabling precise modeling in areas like radiation detection and nuclear instrumentation.

The particle transport capabilities rely on continuous-energy nuclear data, which encompass detailed characteristics such as scattering behaviors, secondary particle yields, and fission spectra. Compatibility with a wide range of internationally recognized nuclear data libraries ensures flexibility and precision in its applications.

In addition to its advanced computational features, TRIPOLI-4® includes a robust geometry package. This native package supports both surface-based and combinatorial representations, offering versatility in constructing geometries and the ability to combine these approaches when needed.

A particularly valuable feature of TRIPOLI-4® is its compatibility with geometries developed in the ROOT [3] software format. ROOT is an open-source data analysis framework developed at Conseil Européen pour la Recherche Nucléaire (CERN), designed for building, browsing, and visualizing complex geometries.
This compatibility enhances TRIPOLI-4®'s flexibility by allowing users to seamlessly integrate both geometries created with its native package and those from the ROOT geometry package. This ensures broad applicability and interoperability across diverse simulation scenarios.

## 2.2 TRIPOLI-5®

In 2022, a collaboration between CEA, *Institut de Radioprotection et de Sûreté Nucléaire* (IRSN), and EDF initiated the development of TRIPOLI-5®, a next-generation Monte Carlo particle transport code designed to perform highly parallel simulations on advanced computing architectures [4].
This new code leverages the expertise gained from earlier research into the portability of particle transport algorithms in high-performance computing environments, particularly those that integrate hybrid systems combining traditional processors and graphical processing units.
TRIPOLI-5® focuses primarily on reactor physics, with capabilities that include modeling both stationary and dynamic reactor configurations while incorporating multi-physics feedback. These features allow for simulations that account for interdependent phenomena such as temperature, density, and nuclide concentration variations, which are critical in complex reactor systems.
In the short term, the primary goal of TRIPOLI-5® is to address challenges in areas such as reactor kinetics and fuel depletion, all while optimizing performance on modern parallel computing platforms, ranging from large scale systems to standard workstations. Looking to the future, TRIPOLI-5® is intended to evolve into a versatile tool capable of addressing a broader range of applications. These include radiation shielding, advanced nuclear instrumentation, and the implementation of variance reduction techniques to enhance simulation efficiency. This new code represents a significant step forward in leveraging advanced computing technologies to meet the increasing complexity and precision demands of modern nuclear science and engineering.
During the first phase of its development, TRIPOLI-5® relied on a geometry engine based on the ROOT geometry package. This choice was driven by the fact that TRIPOLI-4® also relies on the ROOT geometry engine, facilitating comparisons between the two codes. However, to enhance the flexibility and performance of geometry modeling, the integration of a native geometry engine package was recently introduced. AGORA, the new geometry navigation engine implemented within TRIPOLI-5® and developed at CEA, is specifically designed to enhance the geometry modeling process through its multi-nested structure, based on the Constructive Solid Geometry (CSG) paradigm. This architecture facilitates efficient representation and navigation of complex geometries while improving computational performance. To complement these advancements, TRIPOLI-5® provides advanced

visualization tools to facilitate the inspection of geometries modeled with AGORA. A dedicated tool recently implemented in TRIPOLI-5® allows for the visualization of AGORA geometries through two-dimensional sections. The tool supports various modes, including point and track representations, with or without wireframes. Sections at different heights can be plotted to analyze the spatial distribution of volumes and materials, providing valuable insights into the geometry's structure and composition. Additionally, the T4G visualization tool [5], originally developed for TRIPOLI-4® and ROOT geometries, has been extended to handle AGORA geometries. T4G offers comprehensive 3D navigation, including zooming, rotating and slicing, as well as features for visualizing volume decomposition and material distribution.
These visualization capabilities play a crucial role in assessing the spatial arrangement of the modeled geometry, improving the accuracy and confidence in the representation of the system under scrutiny.
To verify the accuracy and robustness of AGORA, we developed a series of verification methods and conducted preliminary verification test as part of my internship. However, before presenting these methods, along with the adopted procedures and the obtained results, it is essential to first provide an overview of AGORA and its role as the new geometry navigation engine of TRIPOLI-5®, to establish the necessary context for understanding the following sections.

## 2.3 AGORA and Constructive Solid Geometry

In AGORA, the construction of geometries is grounded in the CSG paradigm, which allows for the creation of complex, nested structures by combining simple geometric primitives. This approach facilitates the modeling of intricate configurations, essential for accurately representing nuclear systems.
To fully appreciate the advantages of the CSG paradigm, it is essential to first understand the key features of AGORA in geometry construction.

### 2.3.1 Surfaces Definition

The first concept to introduce is that of surfaces. Mathematically, a surface is defined as the set of points that satisfy a given equation in Cartesian coordinates $f(x, y, z) = 0$. Common examples of surfaces include:

- A plane perpendicular to the x axis: $x - x_0 = 0$
- A cylinder parallel to the z axis: $(x - x_0)^2 + (y - y_0)^2 - R^2 = 0$
- A sphere: $(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0$

A half-space of a surface refers to the region of space whose points satisfy either a positive or negative inequality of the surface equation.
To elucidate, consider a sphere centered at the coordinates $(x_0, y_0, z_0)$ with a radius $R$. The canonical equation of the sphere is:

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$$

By reformatting this equation through the subtraction of the right-hand term from both sides, we derive the surface equation for the sphere as:

$$f(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - R^2 = 0$$

From this formulation, one can infer that any point internal to the sphere satisfies $f(x, y, z) < 0$, while any point external to the sphere satisfies $f(x, y, z) > 0$.
Hence, the coordinates for which $f(x, y, z) < 0$ define the negative half-space, whereas the coordinates for which $f(x, y, z) > 0$ define the positive half-space.

Each surface in AGORA is defined by several parameters specific to each geometric type. For instance, consider two cylindrical surfaces used to model a fuel rod:

```
s1 = agora.ZCylinder(0.0, 0.0, inner_radius)
s2 = agora.ZCylinder(0.0, 0.0, outer_radius)
```

Here thee first cylinder, `s1`, defines the inner surface of the fuel rod, while the second cylinder, `s2`, defines the outer surface. The parameters in the brackets of `ZCylinder` refers to the x-coordinate of the cylinder's center, to the y-coordinate of the cylinder's center and to the radius of the cylinder respectively.
The first cylinder has a radius of dimensions `inner_radius`, while the second cylinder has a radius of dimensions `outer_radius`. Both the cylinders `s1` and `s2` are defined along the z-axis meaning their length extends infinitely in the z-direction and both are centered in $x = 0.0$ and $y = 0.0$.

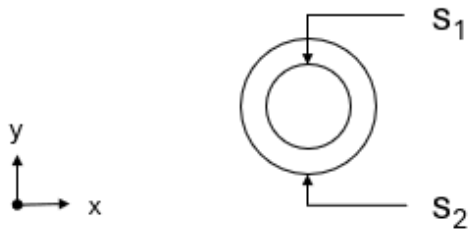


Figure 2.1: Graphic representation of surfaces.

### 2.3.2 Shapes Definition

Once the surfaces have been defined, the next step involves specifying the spatial regions by applying Boolean operators to the corresponding surface half-spaces. In AGORA, the Boolean operators included are:

- & (intersection): Represents a logical AND operation.
- | (union): Represents a logical OR operation.
- - (complement): Represents a logical NOT operation.

By applying these operators to the surfaces, we can create shapes that represent specific regions within the geometry model in AGORA.
Continuing the previous example, we can define the following shapes:

```
   r1 = agora.Shape( -s1 )
r2 = agora.Shape( +s1 & -s2 )
   r3 = agora.Shape( +s2 )
```

Here `r1` defines the region inside the first cylinder `s1`. The unary minus (-) operator indicates that we are taking the negative half-space of the surface `s1`, which could represent the fuel region.
The Shape `r2` defines the region between `s1` and `s2`. The plus (+) operator denotes the positive half-space of `s1`, while the minus (-) operator applied to `s2` refers to its negative half-space. The intersection operator (&) between `+s1` and `-s2` then defines the space that lies within the outer cylinder and the inner cylinder, which could represent the cladding region.
Finally `r3` defines a semi-infinite region outside the outer cylinder `s2`. The unary plus (+) operator indicates that we are taking the positive half-space of the surface `s2`, which could represent the space surrounding the fuel rod such as the coolant region.

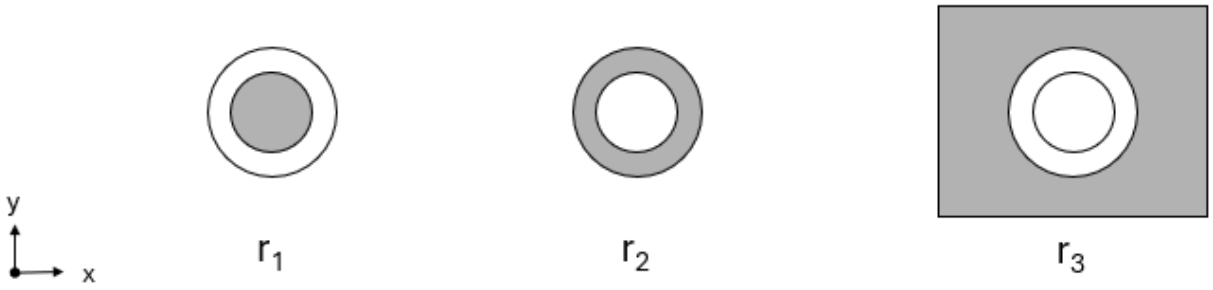


Figure 2.2: Graphic representation of shapes.

### 2.3.3 Volumes Definition

In AGORA, the definition of volumes begins by assigning material properties to geometric shapes, which then form the basis of spatial volumes.
These volumes represent the distinct regions within the geometry, each associated with specific material composition that define their physical properties.
By assigning materials to shapes, AGORA generates volumes that model the corresponding regions of space. This approach is crucial for accurate simulation and analysis, as each material influences the interaction of particles within its assigned volume.
Continuing from the previous example, the definition of three volumes `c1`, `c2` and `c3` is built upon the previously introduced shapes and surfaces. These surfaces

serve as the boundaries of the shapes, which are subsequently linked to material compositions to create the corresponding volumes:

```
v1 = agora.Volume(r1, "FUEL")
v2 = agora.Volume(r2, "CLADDDING")
v3 = agora.Volume(r3, "COLD_WATER")
```

The volume `v1` associates the shape `r1` with the material `"FUEL"`, representing the region inside the inner cylinder. The volume `v2` associates the shape `r2` with the material `"CLADDING"`, corresponding to the region between the inner and outer cylinders. The volume `v3` associates the shape `r3` with the material `"COLD_WATER"`, represented by the positive half space of the outer cylinder defined by `r3`.
These resulting volumes represent the fuel, the cladding, and the surrounding coolant regions of a fuel rod.

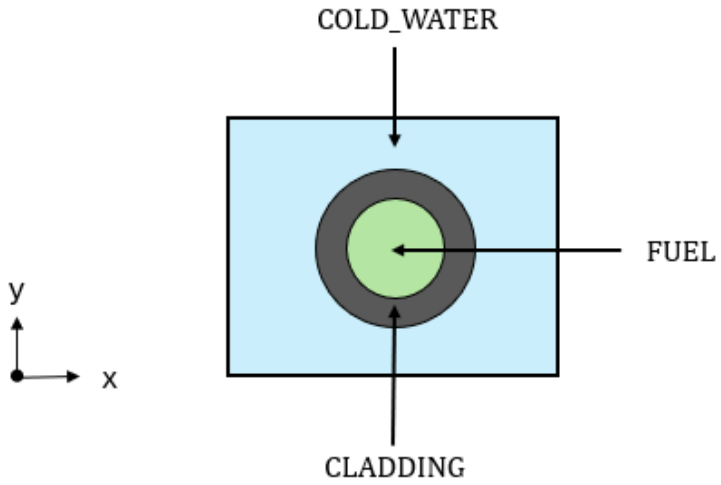


Figure 2.3: Graphic representation of volumes.

### 2.3.4 Universes Definition

AGORA's universe-based approach simplifies the modeling of repeated structures by allowing users to define a single universe and instantiate it multiple times throughout the geometry. This is particularly useful for modeling reactor components, where individual fuel rods are repeated within assemblies, and assemblies are repeated within the reactor core.
For instance, a pincell of a fuel rod can be represented as a universe containing the previously defined volumes:

```
fuel_cold = agora.Universe([v1, v2, v3])
```

In this definition `fuel_cold` is the name assigned to the universe, representing a complete fuel rod. The attribute `[v1, v2, v3]` refers to the list of volumes, previously defined, that comprise the universe.
By grouping these volumes into a universe, the fuel rod's geometry and material

configuration are encapsulated in a reusable structure. This approach is highly beneficial when variations of the fuel rod need to be modeled.
For instance, replacing the coolant volume, containing `COLD_WATER` and represented by `v3`, with a hot coolant allows for modeling the thermal behavior of a fuel rod in hotter regions of the reactor core.
A universe representing a hot fuel rod can be defined as follows:

```
r4 = agora.Shape(+s2)
v4 = agora.Volume(r4, "HOT_WATER")
fuel_hot = agora.Universe([v1, v2, v4])
```

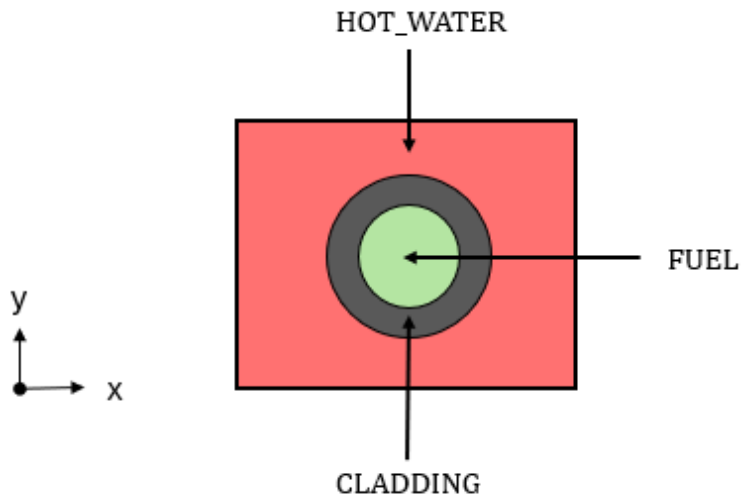


Figure 2.4: Graphic representation of the fuel_hot universe.

This new universe, created by replacing the cold coolant volume `v3` with the hot coolant volume `v4`, can be used to represents a hot fuel rod configuration. These universes can be further combined into larger assemblies or used in lattices, as shown in the next section.

### 2.3.5 Lattices Definition

Frequently, repeated structures in a geometry occur in a regular pattern, such as a rectangular or hexagonal lattice. In such cases, it would be cumbersome for the user to define the boundaries of each region to be filled with a universe.
In AGORA lattices are used to efficiently handle these repeated structures. A lattice represents a regular, repeating grid of universes, where each cell in the lattice can contain an individual universe that models a component or a group of components. This approach drastically reduces the complexity and size of the geometry description by allowing the user to define a single instance of a structure and replicate it multiple times within the lattice without the need to define each instance separately.
For example, consider the following rectangular lattice in AGORA:

```
lattice = agora.RectLattice
      (Point(x, y, z),
     Vector(dx, dy, dz),
nx, ny, nz, replicated_universe)
```

Here `Point(x, y, z)` specifies the starting coordinate of the lattice. The parameters $x$, $y$, and $z$ define the position of the lattice origin in 3D space.
`Vector(dx, dy, dz)` defines the spacing between adjacent cells in the lattice. The parameters `dx`, `dy`, and `dz` specify the cell pitch in the $x$, $y$, and $z$ directions.
The parameters `nx`, `ny`, and `nz` represent the number of cells in the $x$, $y$, and $z$ directions respectively, while `replicated_universe` represents the object (universe) to be repeated in each cell of the lattice. This can be a fuel rod, moderator, control rod, or any other defined universe in the geometry.

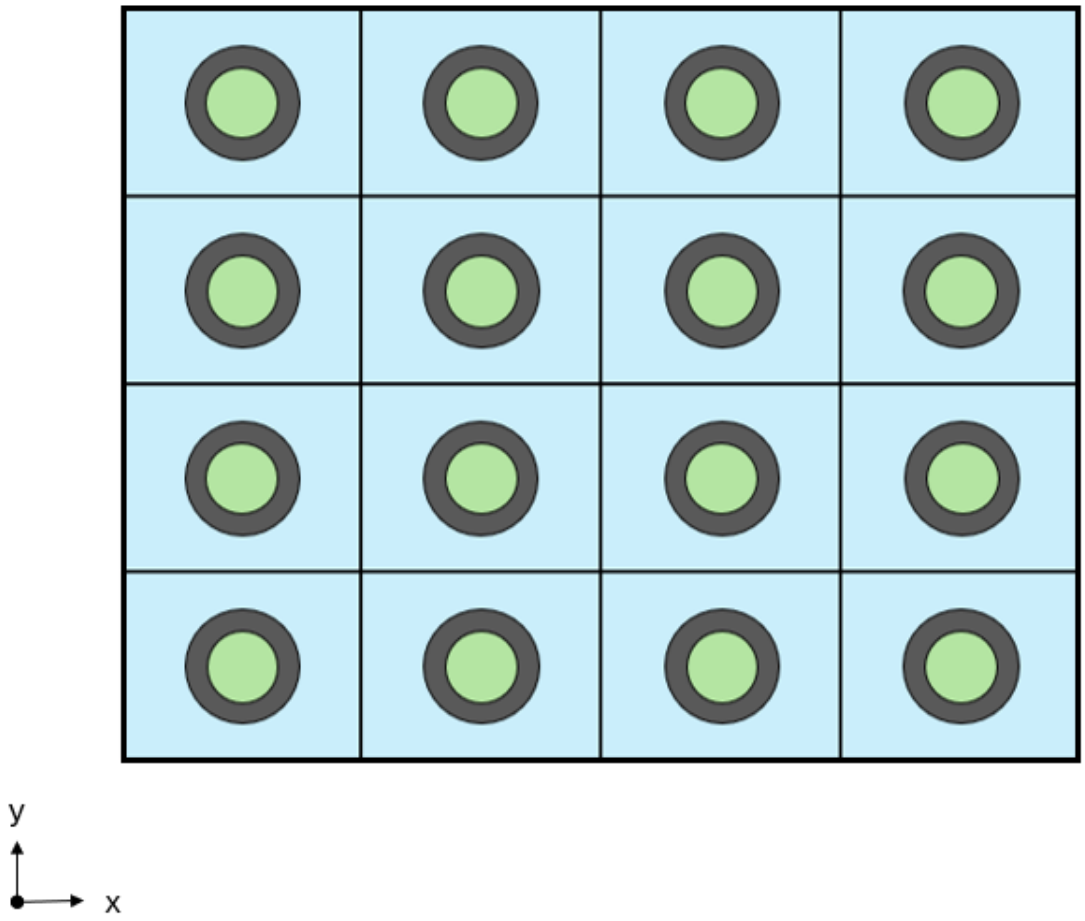


Figure 2.5: Graphical representation of a 4 x 4 lattice filled with the fuel_cold universe previously defined.

### 2.3.6 Composite Volumes Definition

In AGORA, a Composite Volume is used to combine a geometric shape with more complex structures such as lattices or universes. This approach allows for the representation of spatially restricted regions within larger ones, effectively focusing on localized configurations while managing more complex structures. Composite Volumes are particularly valuable in reactor simulations, where repeated structures need to be confined to specific finite regions of space. The shape acts as a boundary, and only the portion of the lattice or universe that falls within the boundary of the shape is considered in the Composite Volume.
Continuing from the previous example lattice definition, we introduce a Composite Volume to confine the lattice to a finite cylindrical region.
This is a typical scenario for modeling a localized section of the reactor core:

```
core_surface = agora.ZCylinder(0.0, 0.0, core_radius)
core_bottom = agora.ZPlane(bottom_z)
core_top = agora.ZPlane(top_z)
core_shape = agora.Shape([-core_surface & +core_bottom & -core_top])
composite_volume = agora.CompositeVolume(core_shape, lattice)
```

The `core_shape` defines a cylindrical region with radius r = core_radius, bounded vertically by planes at z = bottom_z and z = top_z. The `composite_volume` restricts the visibility of the previously defined lattice to the region enclosed by this finite shape.

Composite Volumes are particularly useful when dividing a geometric model into different sections. For example, they can be used to represent distinct regions of a reactor core, such as the hot and cold zones, a common approach for simulating thermal behavior within a reactor. By using different sets of planes and filling the lattice with different universes, it is possible to create distinct regions that reflect the varying temperature profiles within the core.

Below is an example demonstrating how this approach can be implemented:

```
core_shape_hot = agora.Shape([-core_surface & +core_bottom_hot & -core_top_hot])
hot_core = agora.CompositeVolume(core_shape_hot, lattice_hot)
```

In this case, the `core_bottom_hot` and `core_top_hot` are positioned at different heights compared to the previously defined planes, thus distinguishing a different zone within the reactor core. The `hot_core` Composite Volume confines the hot section of the core, where the `lattice_hot` has been filled with the `hot_fuel` universe.

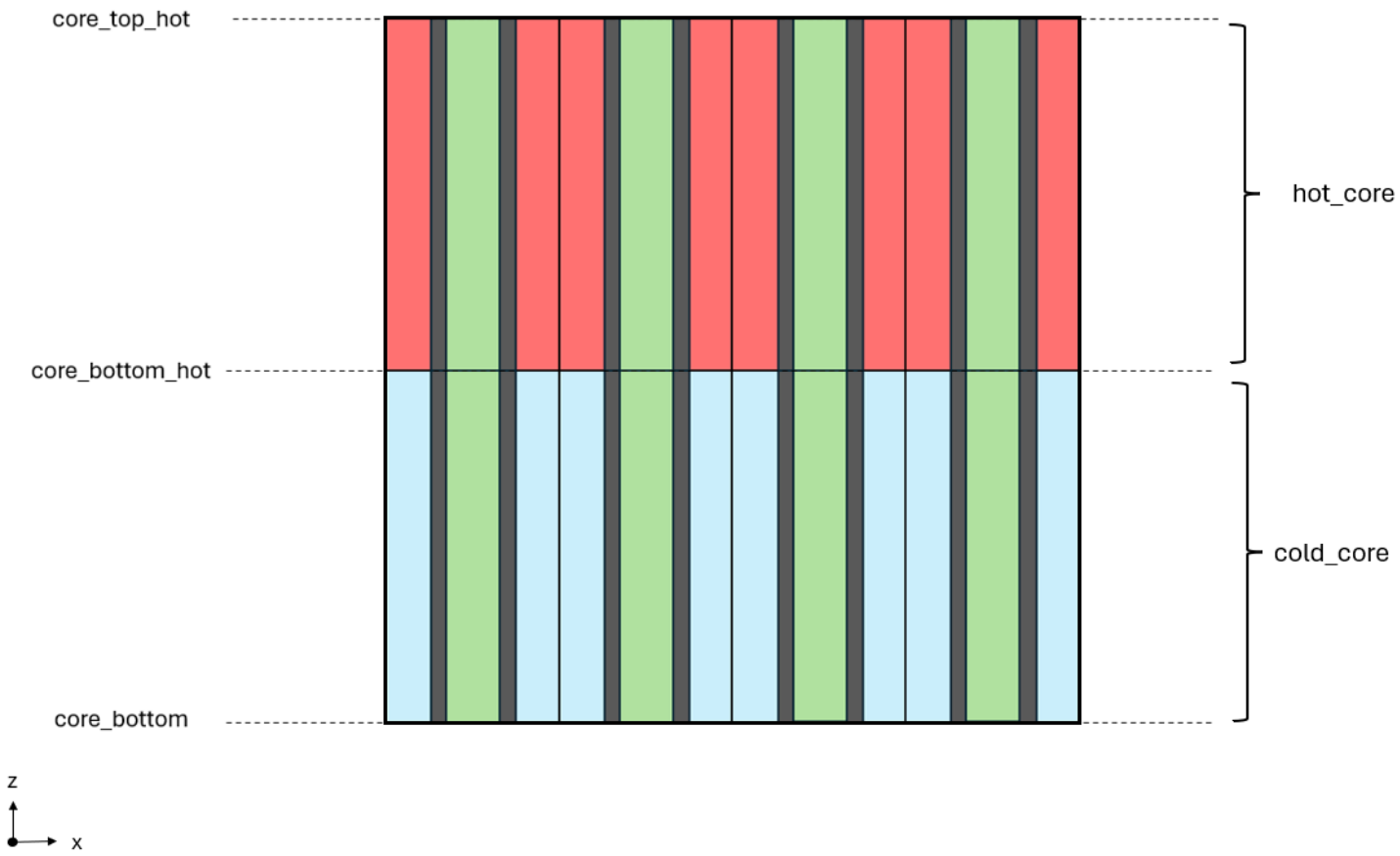


Figure 2.6: Graphic representation of the CompositeVolume. Each z-plane is shown in the figure, delineating the `hot_core` region in the upper part and the `cold_core` region in the lower part.

### 2.3.7 Root Universe Definition

In AGORA, the Root Universe serves as the foundational framework for the entire geometry model, encapsulating all components within a single cohesive computational domain. It ensures the simulation domain is finite, enabling a comprehensive modeling of the entire system, including both the physical geometry and the surrounding unoccupied space. Building upon the previous example, the reactor's geometric model can be finalized as follows:

```
main_universe = agora.Universe([cold_core, hot_core, vessel])
inside_shape = agora.Shape(-outer_vessel & +bottom_plane & -upper_plane)
v0 = agora.CompositeVolume(inside_shape, main_universe)
outside_shape = agora.Shape(+outer_vessel | -bottom_plane | +upper_plane)
outside_volume = agora.Volume.outside(outside_shape)
root_universe = agora.Universe([v0, outside_volume])
```

In this example, the `main_universe` contains the essential physical regions of the reactor, such as the cold core, hot core, and vessel. The `inside_shape` defines the spatial limits of this physical domain, which is constrained by the reactor vessel and

bounded vertically between the `bottom_plane` and `upper_plane`. The combination of `inside_shape` with `main_universe` through the Composite Volume `v0` ensures that the geometry is properly defined. The `outside_shape`, in contrast, specifies the unoccupied space surrounding the reactor geometry. This shape is constructed as the union of three distinct regions: the space above the upper plane, the space below the bottom plane, and the space outside the vessel boundary. Using the `.outside` method, this region is assigned to `outside_volume`, ensuring that all unoccupied space surrounding the physical reactor is accurately represented in the simulation. Finally, the `root_universe` combines the physical domain `v0` and the surrounding unoccupied space `outside_volume`, encapsulating the entire system within a single cohesive framework.
This facilitates the code's implementation of boundary conditions required for the simulation and enables the geometry navigator to accurately model particle interactions throughout the entire domain under analysis.

### 2.3.8 Correctly define the geometry

When defining a geometry model using the AGORA module, it is crucial to pay particular attention to several key aspects to ensure that the calculations are both accurate and reliable.

#### Absence of Gaps in the Geometry

One of the most fundamental aspects of defining a CSG geometry is to avoid the presence of unintended gaps or voids between volumes. Such gaps can lead to significant errors in the simulation results. If a particle enters an undefined area (a gap), the Monte Carlo code might be unable to handle it correctly, potentially leading to physically inaccurate results or even causing the simulation to stop unexpectedly. Gaps in the geometry can result in particles being lost, meaning they are no longer tracked accurately, thereby reducing the precision and reliability of the simulation outcomes.

#### Absence of Overlapping Volumes

Another critical aspect is ensuring that no overlapping volumes exist within the geometry. When two or more volumes overlap, the Monte Carlo code may struggle to correctly determine the particle's position and identify the material through which it is traveling. This can introduce errors in the particle transport calculations, leading to incorrect predictions of particle interactions. Moreover, overlapping volumes can cause issues in estimating absorbed doses, particle fluxes, and other parameters of interest within the simulations, potentially compromising the entire study.

# Chapter 3

# Methodology for Verification and preliminary Validation

In the realm of Monte Carlo codes, especially those pertinent to physical systems and nuclear simulations, the Geometry Navigation Engine (GNE) emerges as a cornerstone of paramount importance. This sophisticated component shoulders the responsibility of managing the spatial relationships and interactions between particles and the geometrical representation of the system under scrutiny. The significance of verifying and preliminarily validating this component cannot be overstated, as it is fundamental to ensuring the integrity and reliability of the simulation outcomes. Let us delve into a comprehensive examination of why this is the case.

## 3.1 The importance of the navigation engine

In Monte Carlo simulations, the accurate representation and navigation of the geometry under study are foundational to the success of particle tracking, directly influencing the reliability of the simulation results.
The GNE plays a central role in ensuring that particles interact correctly with materials and boundaries within the modeled domain.
The primary task of this component is to ensure the ability to accurately and reliably address three critical questions:

1. **In which composition does the particle reside?**
   By identifying the material in which the particle is currently located, the code applies the correct cross-sections for interactions such as fission, scattering, or absorption, ensuring the fidelity of calculations.

2. **What is the distance to the next geometrical boundary?**
   The GNE is responsible for calculating the distance to the nearest boundary, a crucial parameter for determining the particle's trajectory, especially in complex geometries. Accurate distance calculations are vital to maintaining the consistency and precision of the simulation, as errors in this step could lead to significant deviations in the particle's predicted path.

3. **Which is the next volume that the particle will traverse?**
   After crossing a boundary, the GNE must determine the next volume the particle will enter identifying the material composition, enabling continuous and accurate tracking as the particle moves through the domain.

Any inaccuracy made by the GNE in geometry representation, boundary detection, or material assignment can significantly undermine the reliability and accuracy of simulation results. Such inaccuracies can lead to flawed predictions of key outputs, including particle flux distributions or eigenvalue solutions, which are critical for applications ranging from safety assessments and system design to operational analysis. For instance, in Monte Carlo transport simulations, any lapse in handling particle interactions with the geometry can skew predictions, directly affecting conclusions about the system's behavior.
As contemporary simulations increasingly involve intricate and detailed geometries, the GNE must adeptly navigate these complexities to maintain accuracy and reliability. This requirement becomes particularly important in scientific research and regulatory contexts, where reproducibility and consistency of results are vital for verification and validation of models and supporting critical decisions.

## 3.2 Description of the methodology

To verify the correct functionality of AGORA, a fundamental step is establishing equivalence between geometries represented by different Monte Carlo GNEs. This process is essential to ensure the accuracy and reliability of simulations when comparing results across diverse computational tools.
A stringent approach would be to consider two geometries equivalent if their respective engines assign the same cell number to each point in space. However, this criterion is overly restrictive and impractical for comprehensive verification [6].
It is important to recall that particle tracking is predominantly influenced by material interactions rather than the geometrical structure itself and its decomposition. Moreover, an essential assumption is that the system can be represented as piecewise homogeneous, composed of discrete *Volumes* (or cells), each defined by a specific material composition (i.e. a list of nuclides and their concentrations).
Therefore, it is sufficient to relax the definition of equivalent geometry models to those that assign the same material composition to each point in the geometry space under study.
To clarify this concept, the Figure 3.1 displays the same geometry model as represented by two different Monte Carlo codes and their respective GNE. The image shows the geometry's decomposition into its respective volumes, with each volume represented by a different color.

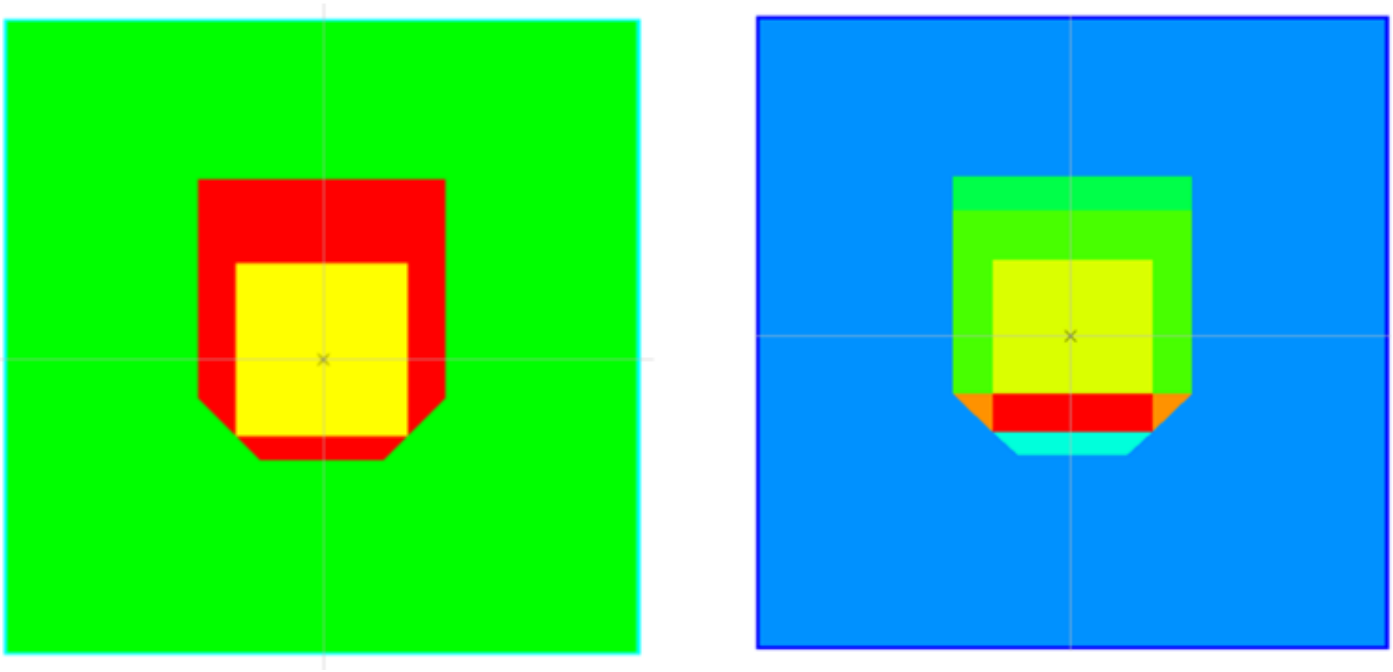

Figure 3.1: Decomposition into volumes of the same geometry model: AGORA model for TRIPOLI-5® (left) and ROOT model for TRIPOLI-4® (right) trough the T4G visualization tool, each volume is represented by a specific color in each software.

It is clear that the key factor for the equivalence of geometries lies not in the specific subdivision of the geometry model into volumes, but rather in the material composition of those volumes. Indeed, the same geometry could be modeled in entirely different ways depending on the software used, as shown in Figure 3.1. Furthermore, even within the same software, an identical geometric model could be constructed in different ways based on the programmer's approach.
Based on these assumption, we proceeded to implement a series of verification techniques and related preliminary validation tests designed to serve two key purposes: first, to ensure the **internal consistency** of the AGORA module within its host code TRIPOLI-5®, and second, to facilitate **external comparisons** between the geometric models produced by AGORA and those generated by other widely used Monte Carlo codes.

### 3.2.1 Internal Consistency

To verify internal consistency and assess the performance of AGORA, two configurations of TRIPOLI-5® were thoroughly examined:

- **TRIPOLI-5® with AGORA**
- **TRIPOLI-5® with ROOT**

As previously mentioned, at the initial stages of development, TRIPOLI-5® relied solely on the ROOT geometry package. This choice was driven by its existing integration with TRIPOLI-4®, making it easier the incorporation also into the new code. The ROOT Navigator has long been recognized for its reliability and extensive use within the simulation community, serving as a robust and trusted standard for geometry navigation. To ensure AGORA's functionality and seamless integration into TRIPOLI-5®, we compared simulation results from TRIPOLI-5® with AGORA to those obtained using TRIPOLI-5® with the ROOT geometry package. This comparison was essential for validating the correct integration of the new GNE within its host code to ensure compatibility.

### 3.2.2 External Comparisons

External comparisons were performed to analyze the results generated by TRIPOLI-5® with AGORA in relation to those produced by other Monte Carlo codes, used as benchmarks. The primary objective of these comparisons was to assess the fidelity of the geometric models developed by AGORA, as well as the performance and efficiency of its navigation algorithms within the simulation framework. By assessing the consistency and accuracy of AGORA's results against established codes, the external comparison aims to identify any discrepancies, offering a focused evaluation of potential issues.
The analysis compared the performance of **TRIPOLI-5® with AGORA** against the following configurations:

- **TRIPOLI-4® with ROOT**
- **TRIPOLI-4® with Native Navigation Engine**
- **OpenMC** [7]

The choice of the first two configurations was driven by the need to compare the behavior of TRIPOLI-5® with AGORA to the neutron transport Monte Carlo code currently used at the CEA, TRIPOLI-4®, both in its version with the native navigator and with the ROOT geometry engine.
As for the last configuration, the inclusion of OpenMC in our comparative analysis was driven by its shared reliance on CSG modeling principles with AGORA.
OpenMC, an open-source Monte Carlo code developed by the Computational Reactor Physics Group at the Massachusetts Institute of Technology, is widely recognized for its reliability and accuracy in transport simulations. Its use of CSG principles provides a robust foundation for benchmarking AGORA's geometric modeling and navigation performance. This alignment makes OpenMC a logical and valuable inclusion in our comparative analysis.

## 3.3 Verification Techniques

To enable external comparisons and ensure internal consistency, we developed a suite of Python [8] based verification tools. Python was chosen for its flexibility,

extensive library ecosystem, and seamless integration with the application programming interfaces (APIs) of the Monte Carlo codes and GNEs analyzed in this study. These tools are designed as a series of interconnected scripts executed sequentially, where the output of one serves as the input for the next. This modular architecture facilitates progressive refinement and allows for additional diagnostic checks whenever necessary, ensuring a systematic and thorough verification process.
The verification framework is built to assess two critical aspects of the geometry engine's functionality: its ability to accurately identify materials within the geometric domain and its capability to track particle trajectories with precision. These objectives are addressed through two complementary methodologies:

#### Particle Location Verification

This method, developed specifically to address the question *"In which composition does the particle lie?"*, involves generating random discrete points within the geometric domain. The geometry engine is tasked with identifying the material at each point, and the results are compared to expected outcomes. This method focuses solely on assessing material information at individual locations and does not require particle tracking.

#### Particle Tracking Verification

The second method is designed to address the questions *"What is the distance to the next geometrical boundary?"* and *"What material will the particle encounter next?"*. In this approach, random segments are traced through the geometry model under study. Each segment is characterized by a starting point, direction, and length. By analyzing the paths of these segments and their interactions with the geometric boundaries, we aim to ensure that material transitions are accurately associated to each point along the segment's path.

## 3.4 Selected reactor models

The verification techniques were initially conducted specifically on simple geometries in order to refine the developed tools. Having achieved a sufficiently mature level with these tools, we proceeded to more complex geometry models.
These reactor models were selected to provide a thorough and comprehensive evaluation of the geometry engine's capability to handle a wide range of reactor configurations. From the earliest historical designs to the most complex and modern reactor systems, these models encompass a variety of reactor types and complexities. The choice of these particular models is intentional, as they represent key benchmarks widely used in the reactor physics community. The reactors selected for this study are as follows:

- **Chicago Pile 1 (CP-1)** [9]: The first nuclear reactor, built in 1942, CP-1 was a graphite-moderated, uranium-fueled experimental reactor. This extremely simplified model serves as an historical benchmark for nuclear reactor physics.

- **Hoogenboom-Martin Benchmark** [10]: A hypothetical large pressurized water reactor core, designed as a test case for large-scale reactor simulations. Its complex geometry offers a challenging scenario for the verification of geometry engines.

- **Special Power Excursion Reactor Test III (SPERT-III)** [11]: A compact pressurized water research reactor, used to study power excursions. The SPERT-III model is notable for its intricate geometry and is often used in nuclear reactor physics studies.

While a detailed description of these reactors configurations may be unnecessary for the scope of this thesis, it is important to highlight the aspects pertinent to the following discussion. In particular, Table 3.1 provides a detailed breakdown of the AGORA geometry models used for these reactors. The table outlines key characteristics of each configuration, including the number of surfaces, the number of volumes (both simple and composite) the number of universes and the number of lattices.
These parameters reflect the complexity of the geometric representation and the structure of the selected reactor models, offering valuable insights into the challenges posed by each case.
Moreover, the following figures illustrate the radial and axial cross-sections of these geometric models in AGORA, visualized using the T4G tool, in their constituent volumes decomposition mode.

| **Geometry Model** | $\boldsymbol{N}_{\text{surf}}$ | $\boldsymbol{N_v}$ | $\boldsymbol{N_u}$ | $\boldsymbol{N_l}$ |
|---|---|---|---|---|
| `Chicago Pile 1` | 20 | 11 | 5 | 0 |
| `Hoogenboom-Martin` | 23 | 30 | 9 | 4 |
| `SPERT-III` | 210 | 227 | 64 | 16 |

Table 3.1: Features of the AGORA geometry models for the selected reactors: number of surfaces ($\boldsymbol{N}_{\text{surf}}$), number of volumes ($\boldsymbol{N_v}$), number of universes ($\boldsymbol{N_u}$), and number of lattices ($\boldsymbol{N_l}$) in the different configurations.

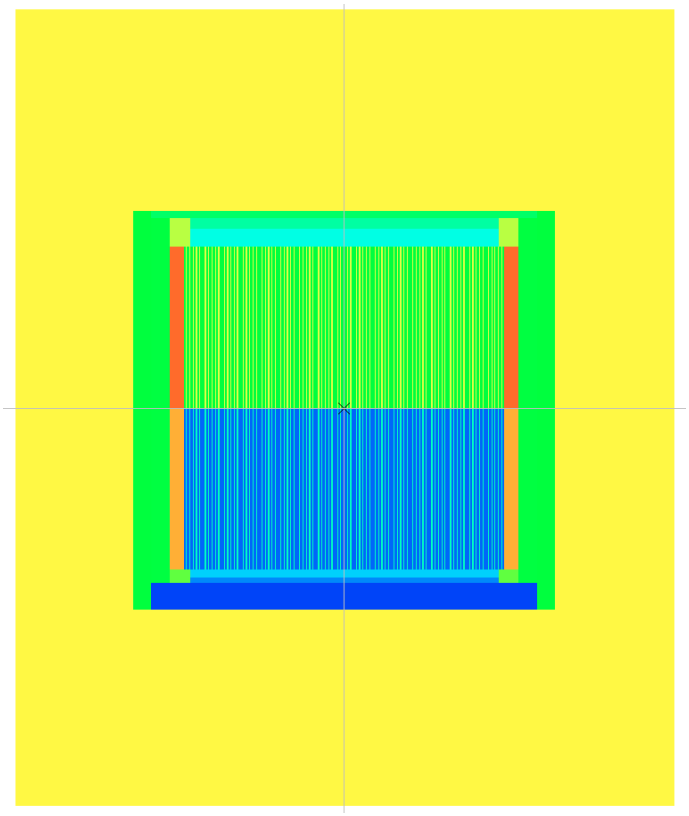

Figure 3.2: Hoogenboom-Martin Benchmark: Axial section of the AGORA model trough the T4G visualization tool.

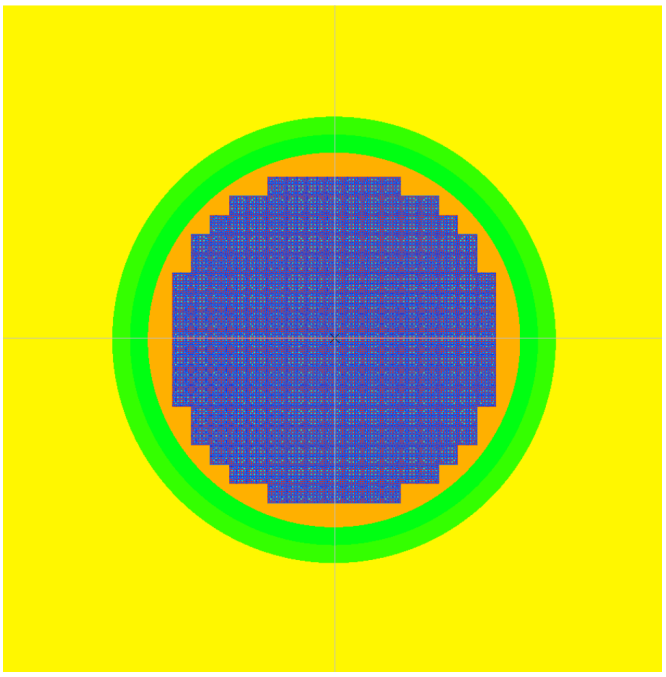

Figure 3.3: Hoogenboom-Martin Benchmark: Radial section of the AGORA model trough the T4G visualization tool.

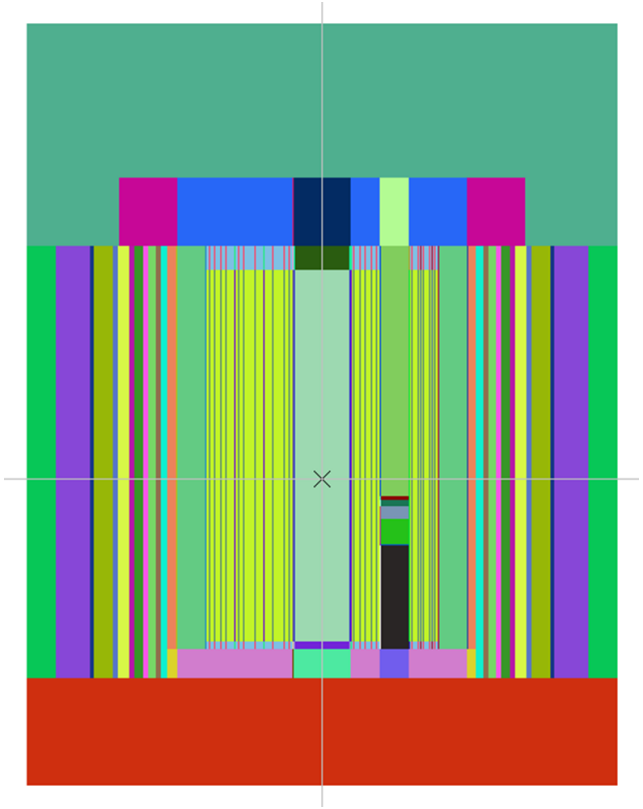

Figure 3.4: SPERT-III: Axial section of the AGORA model trough the T4G visualization tool.

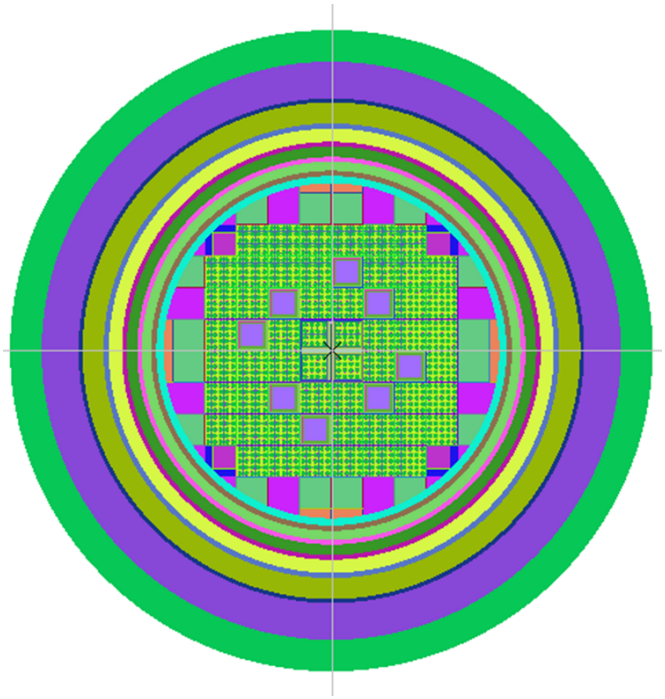

Figure 3.5: SPERT-III: Radial section of the AGORA model trough the T4G visualization tool.

# Chapter 4

# Particle Location Verification Through Points Analysis

The first phase of the verification process of the new geometry navigator engine AGORA focuses on particle localization. Specifically, we have developed a comprehensive procedure, along with an associated tool, designed to prove the ability of AGORA in answering the first fundamental question:

*In which material composition does the particle reside?*

This tool is composed by interconnected Python scripts, executed sequentially, where the output of one script serves as the input for the next one. The modular approach enhances the tool's flexibility by enabling progressive refinement and additional diagnostic checks if needed.
The procedure begins with the generation of a set of random points uniformly generated within a user-defined bounding box, encompassing the geometry model under investigation.
These points form the foundation for querying the geometry navigation engines of the Monte Carlo codes, which are individually interrogated to identify the material composition associated with each location. Each point is processed independently, with dedicated scripts interacting directly with the navigation engines to retrieve the material corresponding at each location.
The results obtained from AGORA are then compared with those generated by the other geometry navigator engines used as benchmark. This comparison adheres to the principle of equivalence, wherein AGORA is expected to produce results equivalent with those produced by the other engines. If discrepancies are identified, the developed tool is able to isolate and export the specific points and the corresponding materials where deviations occur, enabling targeted analysis and facilitating the resolution of potential issues.
To ensure consistency and interoperability, the tool uses `JSON` [12] files as a standardized input and output format. This file format was specifically chosen due to its readable and interpretable structure, making it accessible for manual inspection and debugging when necessary. Moreover, given the nature of the data (lists of points, materials, and mappings) the `JSON` format hierarchical organization proved to be a natural and practical choice for representing such information. Its widespread

support across programming environments further enhances its utility, facilitating seamless integration with the developed Python tool and the other software components used in this study.
The workflow of the procedure is structured as follows:

1. **Point Generation:** A set of random points is generated within the geometry model under investigation, ensuring uniform distribution across the defined domain. This step provides comprehensive coverage of the geometry model, enabling a robust comparison.

2. **Navigator Querying:** The generated points are converted into formats compatible with each geometry navigator, the navigator engine is queried using the random points to retrieve the material information at the specified random locations.

3. **Material Comparison:** The retrieved material data is systematically compared across AGORA and the other navigators. Any inconsistencies, if present, are flagged for closer examination, highlighting potential issues in material identification or particle localization.

4. **Boundary Proximity Test:** Points flagged with inconsistencies undergo further analysis in AGORA using a script to identify boundary proximity, directional material variations, and potential ambiguities, refining the accuracy of the comparison.

Incorporating OpenMC into this comparative framework required a slight modification to the methodology, as its geometry navigator engine cannot be queried directly without altering the source code. This adaptation, which is explained in detail later in this chapter, ensures seamless integration with the developed tool while maintaining the same methodological rigor. In Figure 4.1, a schematic description of the procedure is presented.
The following sections provide a detailed description of each component of the particle location verification tool, accompanied by results demonstrating its application to the test geometry models outlined in 3.4.

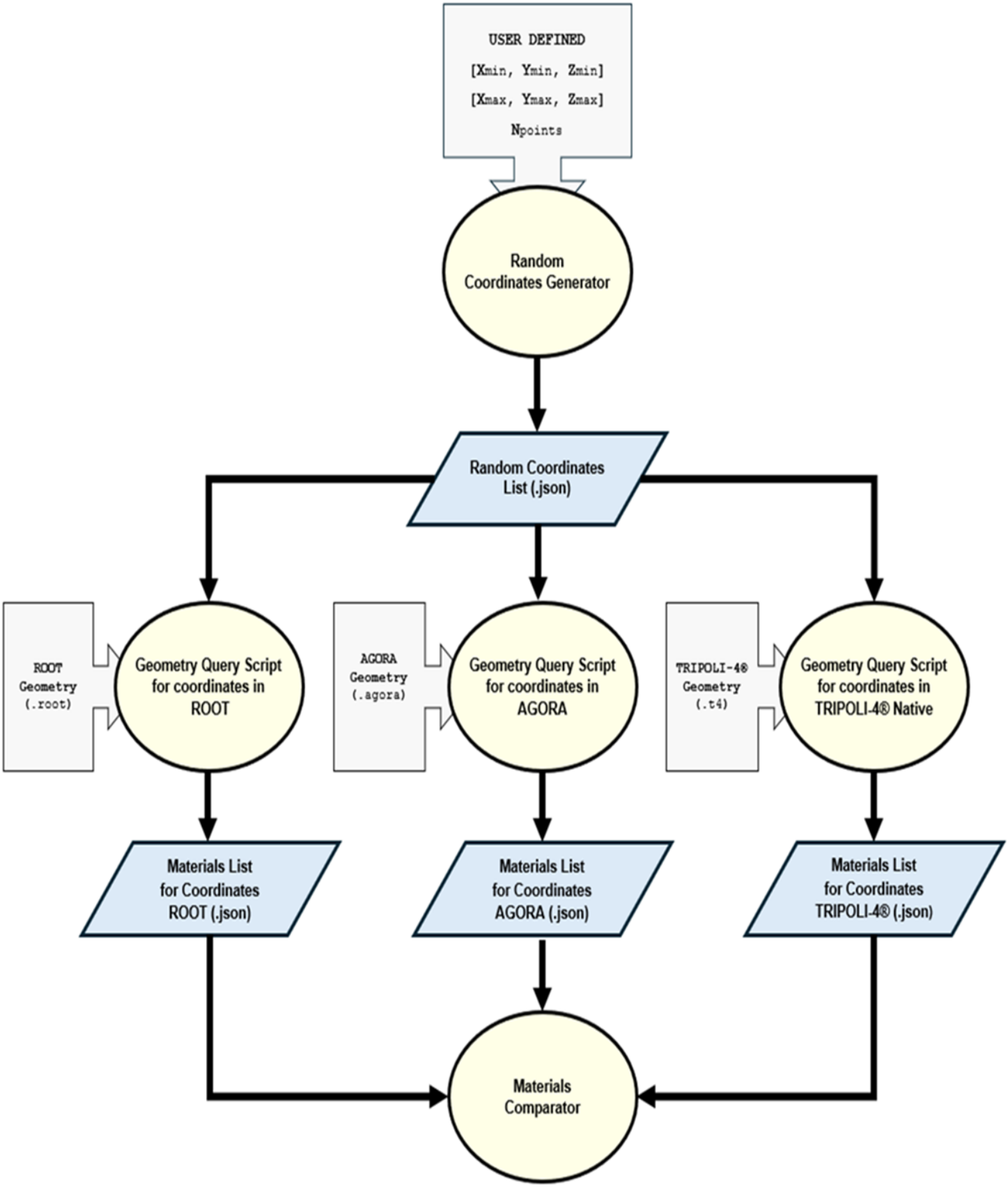


Figure 4.1: Schematic description of the particle location verification tool.

## 4.1 Random Coordinates Generator

The random coordinates generator script is the first element of the verification framework, designed to allow material query analyses within a three-dimensional space. Its role is to generate a set of random points uniformly distributed within a user specified volume. These points act as probes, enabling the interrogation of geometry navigator engines involved in this study and the evaluation of the material identification capabilities of AGORA.
The script ensures comprehensive spatial coverage by using an uniform probability density function for coordinates generation, meaning every location within the user defined bounds has an equal likelihood of being sampled. This unbiased distribution is critical for analyzing geometries as it ensures that no region of the model is over or under represented.

### Inputs

The script requires the following inputs to function:

- **Space boundaries:** Minimum and maximum coordinates (`x`, `y`, `z`) of the 3D space.
- **Number of points:** The desired quantity of random points to be generated.

### Output

Upon execution, the following output is generated:

- **Space coordinates:** A list (`.json`) of randomly generated coordinates in the defined 3D space.

The script operates by accepting user-defined inputs that specify the minimum and maximum coordinates of the three-dimensional space, effectively defining the box that encloses the geometry model under investigation. Additionally, the user must provide the desired number of random points to be generated uniformly in the domain.
Each coordinate is sampled from a uniform distribution across the specified range. This means that each value for $x$, $y$, and $z$ has an equal probability of being chosen within the user-defined bounds.
For each coordinate a random value is drawn from the uniform distribution $\mathcal{U}(c_{\min}, c_{\max})$, where the probability density function is defined as:

$$f(\xi) = \begin{cases} \frac{1}{c_{\max} - c_{\min}} & \text{if } c_{\min} \leq \xi \leq c_{\max} \\ 0 & \text{otherwise} \end{cases}$$

This process guarantees that the resulting dataset of points is evenly distributed throughout the space, providing a robust foundation for subsequent material analyses.
Once the points are generated, they are saved in a `JSON` file. This output file contains the generated points as a list of coordinates, which can then be directly used as input for subsequent material query scripts.

## 4.2 Geometry Query Script for AGORA

The geometry query script for AGORA is a crucial element of the verification framework. It is designed to analyze geometries by identifying the materials present at each randomly generated point within the geometry model under examination. The script operates by querying AGORA to navigate the geometry model using the points generated by the random coordinates generator script. For each point, AGORA identifies the material at that location, enabling detailed analysis of the material distribution within the geometry model.

#### Inputs

The script requires the following inputs to function:

- **Geometry file:** The AGORA file (`.agora`) containing the geometry model to be analyzed.
- **Random points file:** The `JSON` file containing the previously generated random points.

#### Output

Upon execution, the following output is generated:

- **Materials list for coordinates:** A list (`.json`) of materials encountered at each random point in the AGORA geometry.

The script begins by loading the random points from the `JSON` file together with the AGORA geometry file (`.agora`). The AGORA geometry navigator, which is responsible for moving through the model and identifying the material at each specified location, is then initialized. To guide the navigation, a direction vector is initialized too, steering the navigator along the geometry.
The core of the script involves iterating through each random point. For each point, the navigator is re-initialized at the new point's coordinates and in the specified direction. The navigator queries the geometry model and identifies the material present at each point, the process is repeated for all the coordinates present in the `JSON` list.
Once all the coordinates have been processed, the material data is saved into a new `JSON` file containing a list of the materials encountered at each random point. This output format ensures that the data is stored in a standardized manner, making it easier to integrate with other scripts and tools within this verification framework.

## 4.3 Geometry Query Scripts for ROOT and TRIPOLI-4®

The geometry query scripts for ROOT and TRIPOLI-4® are integral components of the verification framework, designed to analyze geometry models defined in these

respective formats by identifying the materials encountered at each randomly generated point within the model under study.
These scripts broaden the verification framework by enabling material identification across different geometry formats, allowing them to serve as benchmarks for a direct comparison with the material mapping provided by AGORA.

### Inputs

The scripts require the following inputs to function:

- **Geometry file:** The file containing the geometry model to be analyzed. For ROOT, this is a `(.root)` file, while for TRIPOLI-4® this is a `(.t4)` file.
- **Random points file:** The `JSON` file containing the previously uniformly distributed points.

### Output

Upon execution, each script produces its respective output, namely:

- A `JSON` file containing a list of materials encountered at each random point in the ROOT geometry.
- A `JSON` file containing a list of materials encountered at each random point in the TRIPOLI-4® geometry.

### ROOT Geometry Script

The `ROOT` geometry script begins by importing the necessary libraries, including the `ROOT` navigator for handling the geometry file and the required modules for numerical operations and data processing. The user provides the `.root` file via the command line, which is loaded into `ROOT`'s `TGeoManager` class. This class is responsible for managing and navigating the geometry, accessing volumes, shapes, and materials defined within the model. The script then loads the `JSON` file containing the list of random points and iterates over them. For each point, the `InitTrack` method is invoked, initializing the geometry navigator at the specified coordinates with a defined direction vector. This method allows the `ROOT` navigator to identify the material at the precise location by directly querying the geometry model. Throughout execution, the script provides periodic updates to inform users of its progress. After all points have been processed, the material information encountered at each coordinate is saved into a new `JSON` file for subsequent analysis.

### TRIPOLI-4® Native Script

The TRIPOLI-4® script is designed to process the random points, utilizing the native geometry navigator of the code to accurately allocate them within the geometry model under investigation. The process begins by reading the random points from the `JSON` file and converting them into a `(.text)` format, which is the required input

format for the TRIPOLI-4® native navigator.
The script constructs and runs a command that processes the generated points using the `.t4` file containing the geometry model under analysis. This command triggers the TRIPOLI-4® native navigator, which performs the material identification at each point present in the list and generates an output `(.text)` file containing the results. The script then process this `(.text)` file, extracting the material names in association with the corresponding random points.
Finally, the material information encountered at each coordinate is saved into a `JSON` file, ensuring that the output format is consistent with those produced by the other scripts in this verification framework.

## 4.4 Integration of OpenMC into the Tool

Due to the limitations of directly querying OpenMC at random points, for this part of the tool we employed an alternative method based on particle tracks recorded during simulations. This method aligns with the structure and purpose of other scripts in the framework while adapting to the specifics of OpenMC.

### Procedure

The process begins by running an OpenMC simulation configured to generate the necessary data for analysis. The simulation requires three key input files:

- `geometry.xml`: Defines the geometry model of the simulation domain in OpenMC using the CSG paradigm. This file outlines the spatial arrangement of the regions within the geometry model representing them as combinations of simple geometric shapes.
- `materials.xml`: Contains detailed information about the material compositions used in the simulation. It associates each material with specific indices (IDs), allowing the code to map materials to the correct regions in the geometry. This file includes data on material properties such as densities, atomic compositions, and other relevant physical characteristics necessary for the simulation.
- `settings.xml`: Configures the simulation parameters, defining the mode of operation (e.g., fixed-source or criticality), the spatial and energy characteristics of the source, and other run settings like the number of histories, batches, and simulated particles. This file is essential for tailoring simulations to specific needs.

The simulation is set to fixed-source mode with a spatially sampled source located within a user defined box. The number of histories and track files is imposed to match the target number of random test points, ensuring sufficient data for analysis. The simulation output is stored in the `tracks.h5` file, which contains detailed

records of particle trajectories, including their positions, directions, energies, and the materials encountered during interactions.

### Geometry Query Script for OpenMC

The integration of OpenMC into the verification framework required a slight modification to the existing scheme. A dedicated script processes the collision points recorded in the `tracks.h5` file, using them as sample points akin to those generated by a random coordinates generator. These collision points represent the locations where particles interact with the geometry and serve as a basis for material mapping and comparison.

#### Inputs

The script requires the following inputs to function:

- **Tracks file:** The `tracks.h5` file generated by an OpenMC simulation, containing data about particle histories and interactions.
- **Materials file:** The `materials.xml` file, which provides a mapping between numerical material IDs and their names.

#### Outputs

Upon execution, the following outputs are generated:

- **Particle positions:** A `JSON` file containing the coordinates (`x`, `y`, `z`) of the collision points extracted from the `tracks.h5` file.
- **Materials list for coordinates:** A `JSON` file listing the names of the materials encountered at the extracted coordinates, translated from their numerical IDs.

The script begins by parsing the `tracks.h5` file to extract particle positions and their associated material IDs. For each particle position, the script translates the material IDs using the `materials.xml` file, mapping them to meaningful material names. This mapping ensures consistency with other datasets and facilitates direct comparison with material mappings from AGORA and OpenMC geometry models. Once the execution is complete, the script produces two output files, both in `JSON` format. The first output contains the particle positions, which serve as input for querying other geometry navigator engines in the framework, effectively replacing the random point generator script used in this tool. The second output file stores the material information encountered at each coordinate in the OpenMC model, ensuring that the data is formatted consistently with the outputs from other scripts within the verification framework.
This approach ensures seamless integration of OpenMC into the broader verification framework while maintaining compatibility with the other procedures. The modifications to the previous scheme are outlined schematically in Figure 4.2.

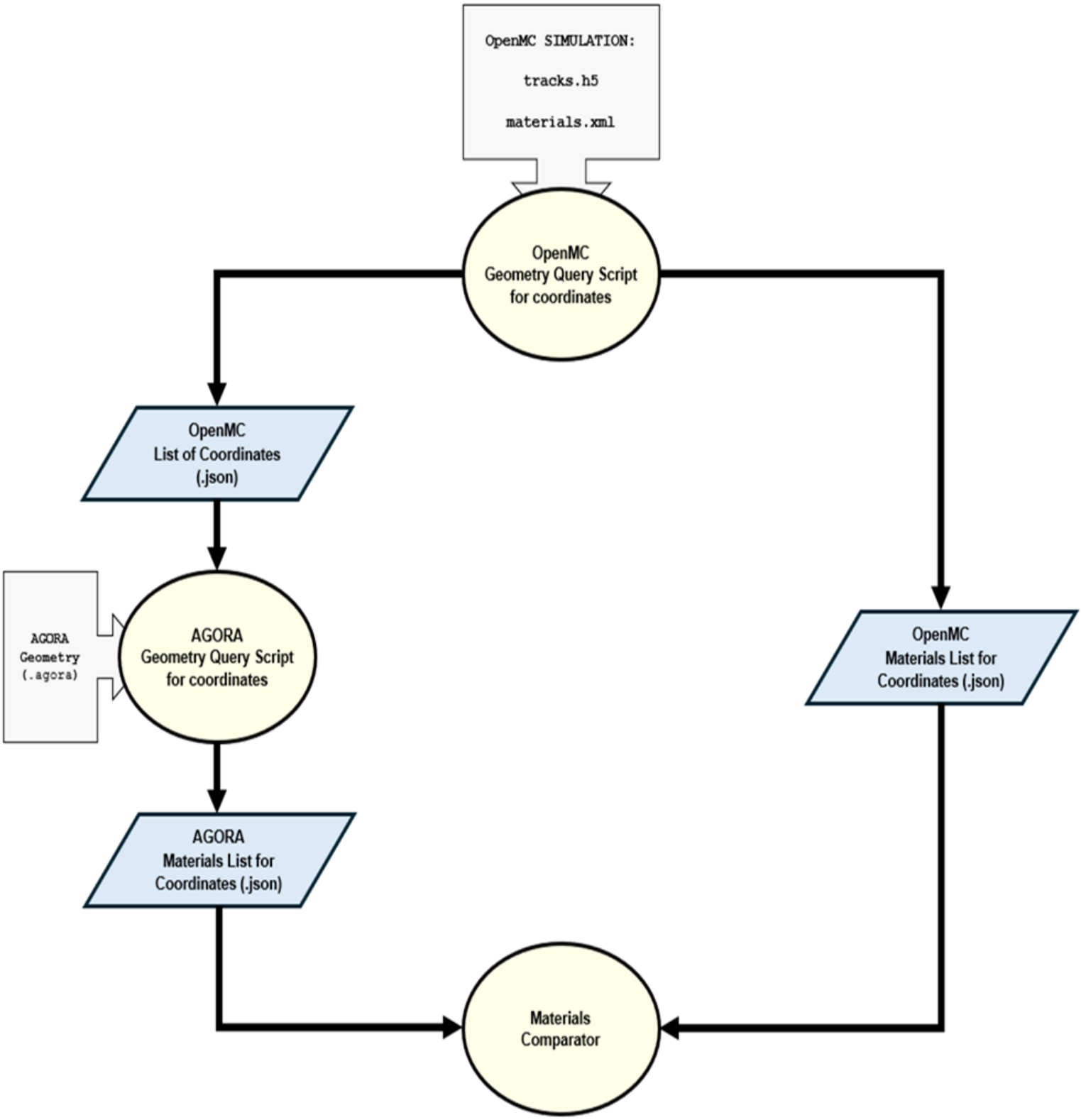


Figure 4.2: Integration of OpenMC in the particle location verification tool.

## 4.5 Materials Comparator Script

The materials comparator script is a core component of the particle localization verification framework. It is designed to verify the consistency of material assignments by comparing material specifications from two separate datasets generated by different software systems.

### Inputs

The script requires the following inputs:

- **Coordinate file:** A `JSON` file containing the coordinates of the points to be analyzed (e.g., generated by a random point generator or extracted from Monte Carlo simulation outputs).
- **Material files:** Two `JSON` files representing the material assignments output from two different software systems to be compared.

### Output

Upon execution, the script produces:

- **Differences file:** A `JSON` file documenting the materials identified at the discrepant points in both datasets, including numerical indices for easier cross-referencing.

The procedure begins with the script loading the necessary input files, including the coordinate file and two material files from separate software systems. These files are read and parsed into structured dictionaries to facilitate efficient analysis. This initial step ensures the data is accessible and prepared for the subsequent comparison. To streamline the comparison process, the script constructs a dictionary that maps material names to numerical indices for both datasets. This mapping is crucial as it ensures uniformity and simplifies the task of identifying discrepancies between the two datasets. By converting material names into consistent numerical representations, the script avoids potential complications arising from variations in naming conventions.
The heart of the script lies in its comparison loop. For each coordinate in the dataset, the script examines the material assigned to that point in both files. If the materials match, the point is marked as consistent, confirming that both systems assign the same material to that location. However, when a mismatch occurs, the script records the coordinate and the differing material indices from both datasets. This approach provides a precise account of any inconsistencies, allowing for targeted analysis.
After completing the comparison, points where discrepancies are detected are stored in a dedicated `JSON` file, capturing their coordinates for further examination. A detailed log of the materials assigned by each dataset at these points is saved, offering a comprehensive view of the inconsistencies ensuring a prompt detection of mismatches for further investigation.

Finally, the script concludes by summarizing its analysis. It reports the total number of points processed, the number of discrepancies found, and their percentage relative to the entire dataset.

## 4.6 Boundary Proximity Test

An additional test has been implemented in AGORA to identify points located near the boundaries between two bodies within the geometry. This boundary proximity test is designed to evaluate flagged points identified by the Materials Comparator, where mismatches occur between datasets. When a point lies very close to a boundary or directly on a surface separating two materials, different geometry navigation engines may assign distinct materials to the same point depending on their initialization direction. This behavior stems from the inherent sensitivity of geometry engines to positional nuances at boundaries.
By carefully assessing such cases, the test distinguishes true mismatches from artifacts caused by boundary effects, ensuring a more precise and reliable particle location verification of the geometry model under examination.

### Inputs

The script requires the following inputs:

- **Points file:** A `JSON` file containing the coordinates of points identified by the material comparator where discrepancies in material assignments have been detected.
- **Geometry file:** An AGORA file (.agora) containing the geometry data to be analyzed for each flagged point.

### Outputs

Upon execution, the script generates the following outputs:

- **Updated Points File:** A `JSON` file containing the coordinates $(x, y, z)$ of the points with updated data, including the minimum distance to the nearest boundary and the corresponding direction for each valid point.
- **Statistics:** Summary statistics (minimum, maximum, median, and mean distances) of the distances to the nearest boundary for valid points.

The boundary proximity test script begins by loading the necessary input files: the points file, which contains the flagged points to be analyzed, and the geometry file (e.g., `.agora`), which describes the geometry model. These files are parsed into structured dictionaries to enable efficient processing and analysis.
Once the input data are prepared, the AGORA geometry navigator is initialized. This navigator is essential for querying the geometry model and evaluating the material composition at each flagged point. The test is designed to examine each point in six different directions, represented by the vector $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z)$, where

each component corresponds to the directional components along the $x$, $y$, and $z$ axes. The analyzed directions are:

$$\begin{aligned} \mathbf{\Omega} &= (+1, 0, 0) \quad \mathbf{\Omega} = (-1, 0, 0) \\ \mathbf{\Omega} &= (0, +1, 0) \quad \mathbf{\Omega} = (0, -1, 0) \\ \mathbf{\Omega} &= (0, 0, +1) \quad \mathbf{\Omega} = (0, 0, -1) \end{aligned}$$

For each flagged point, the test initializes the navigator in the six directions querying the material composition at the coordinates location. If the material assignment differs depending on the direction, this suggests that the point lies on, or very close to, the boundary between two volumes. Such points are flagged as potentially unreliable, as different geometry navigation engines may assign differing materials to the same point based on their directional initialization. These points are excluded from further comparisons to avoid introducing artifacts into the results.
Conversely, if the material assignment remains consistent across all directions, the point is deemed valid for further analysis. For such points, the test calculates the minimum distance to the nearest boundary and records the direction in which this distance is measured. This ensures that only points sufficiently far from any boundaries in the geometry model are considered valid for further investigation.
The test concludes by storing the results for valid points. Subsequently, summary statistics are generated to provide insights into the maximum, minimum, mean, and median distances calculated for these points in all examined directions. This detailed examination supports a robust evaluation of flagged mismatches, distinguishing genuine discrepancies from artifacts caused by boundary effects.

## 4.7 Applications and Results

Based on the tool and methodology developed, we conducted a comprehensive series of verification tests employing the combinations of simulation codes and geometry engines described in 3. These tests were designed to evaluate the accuracy and robustness of AGORA in particle localization through the selected reactor configurations in 3.4.
The results of our verification tests are summarized in Table 4.1 for $N_p = 10^6$ random test points uniformly sampled across the entire geometry model under investigation.

For `Chicago Pile 1` case, perfect agreement was observed across all tested geometry engines and codes, with no discrepancies recorded for the sampled points.

Similarly, in the `Hoogenboom-Martin` case, AGORA demonstrated excellent accuracy, achieving 0% mismatches for all the configurations of codes for which the geometric model was available.

A slight discrepancy was noted in the `SPERT-III` case, where the ROOT models for this configuration exhibited a mismatch rate of 0.0175% compared to AGORA, while a perfect agreement was observed with the OpenMC model.

| **Verification Case** | **TRIPOLI-4® Native** | **TRIPOLI-4® ROOT** | **TRIPOLI-5® ROOT** | **OpenMC CSG** |
|---|---|---|---|---|
| `Chicago Pile 1` | 0% | 0% | 0% | 0% |
| `Hoogenboom-Martin` | - | 0% | 0% | 0% |
| `SPERT-III` | - | 0.0175% | 0.0175% | 0% |

Table 4.1: Summary of the results obtained for the verification on particle location for $N_p = 10^6$ random points: failing rate between TRIPOLI-5® with AGORA versus other codes and engines.

In general, the results demonstrate excellent agreement between AGORA and the tested simulation codes across all configurations. The isolated discrepancies observed in the `SPERT-III` model underscore the importance of cross verification with diverse geometry engines to identify and address minor variations.
For this configuration, the additional diagnostic test was conducted to verify whether changes in direction influenced material composition at any given point. No faulty points were detected during this test, confirming the absence of erroneous points and excluding proximity to boundaries or precision issues as the cause of the observed mismatches between the AGORA and ROOT-based geometries.
Preliminary investigations suggest that these differences stem from subtle variations in geometry definitions, particularly in the presence of overlapping volumes within the geometric model, which obscure the clarity of navigation as elaborated in 2.3.8. Nonetheless, preliminary tests which will be presented later in Chapter 6, revealed that the impact of these mismatches on the physical quantities calculated during simulations is almost negligible.
These findings highlight the robustness and precision of the developed particle localization verification process. They also demonstrate the potential for further refinement in geometry modeling to enhance compatibility across different simulation tools.

# Chapter 5

# Particle Tracking Verification Through Chord Analysis

The second phase of the verification process for the AGORA geometry navigator engine shifts from the point-based material identification approach to a more advanced methodology centered on segment-based analysis and the corresponding decomposition into chords.
Specifically, we have developed a comprehensive procedure, along with the associated tool, designed to tackle the two remaining fundamental questions:

*What is the distance to the next geometrical boundary?*
*Which is the next volume that the particle will traverse?*

To verify AGORA's correct response to these questions we developed a robust tool, comprising a suite of interconnected Python scripts, that performs a systematic analysis of randomly generated segments within a given geometry model. This analysis provides a comprehensive means to investigate geometrical models aiming to verify particle tracking. As in the case of the particle localization tool, Python was chosen for this task for the same reasons outlined in 3.3.
The central concept of this approach is the analysis of "chords", which are defined as the portions of a segment that lie, entirely or not, within a specific volume and the associated material composition. To clarify, when a segment crosses multiple volume boundaries, it is divided into distinct chords, each corresponding to the section of the segment contained within a single volume. Figure 5.1 illustrates this concept using a geometric model generated with AGORA, displayed through its decomposition into constituent volumes via the T4G visualization tool. To each volume is assigned a color that corresponds to the material it is composed of, making it easy to distinguish between different material compositions within the model. The red segments, which are not part of the original geometric model, have been added to visually clarify the process of chord decomposition in a clear and intuitive way. The segments $S_1$, $S_2$ and $S_3$ traverse the geometric model and cross the boundaries between adjacent volumes. At each boundary intersection, these segments are subdivided into smaller portions, the chords. Each chord is entirely contained within a single material and represents the section of the segment confined to an individual

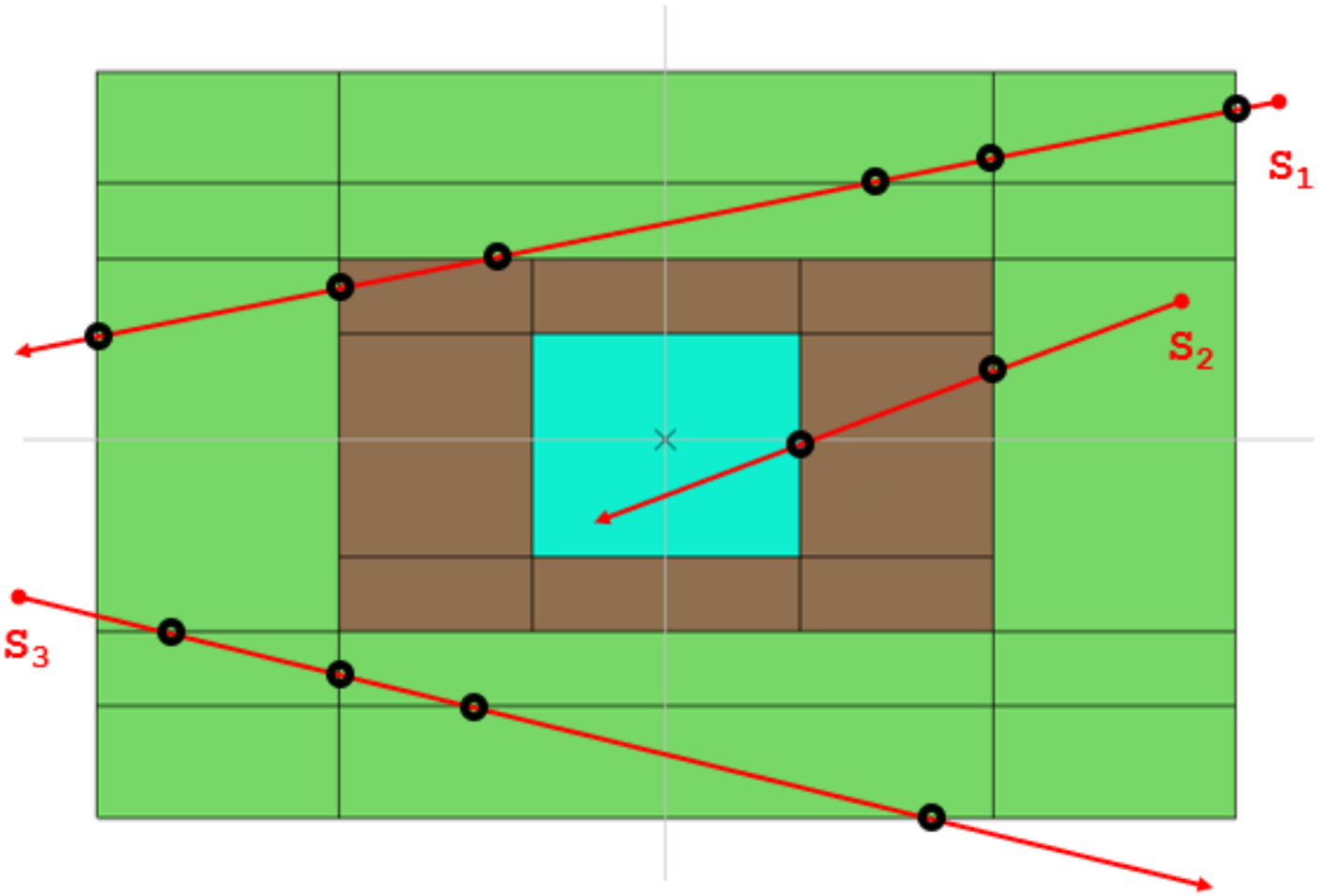


Figure 5.1: Representation of segment decomposition into their respective chords: AGORA simple geometry model trough the T4G visualization tool.

volume. The black dots in the figure indicate the intersection points where the segments cross the volume boundaries, defining the limits of the corresponding chords. Arrows at the segments' ends are used to highlight their direction as they cross the geometry model boundaries.

This chord-based methodology enhances the information gained from the verification of particle locations. While the point-based approach identifies discrepancies at isolated positions, it does not account for particle motion and, therefore, cannot fully verify the correctness of the particle tracking process. By following the segments and analyzing their decomposition through the geometric model, the chord-based approach provides a direct means of verifying particle tracking, offering a more robust and detailed assessment of the geometry navigator engine's accuracy in the identification of materials composition.

The developed tool is designed to facilitate the comparison of the same codes and geometry navigators introduced in 3 and, also in this case, its generic procedure has been modified to include OpenMC in the analysis.

The workflow of the procedure is structured as follows:

1. **Random Segment Generation:** Random segments are generated uniformly within the geometries of interest. These segments ensure thorough sampling across the entire domain.

2. **Navigator Querying:** After the segment data is converted into a format compatible with the geometry navigators under analysis, each segment is queried against the navigators to record the materials and distances associated with the chords comprising the segment.

3. **Chord Merging:** Consecutive chords that share the same material are aggregated to account for differences in how geometry engines subdivide the geometrical domain under investigation.

4. **Chord Comparison:** The aggregated chord data is compared across the different navigators and AGORA to assess consistency in material identification and distances traveled within each material.

The primary goal of this procedure is to ensure that the total distance traveled within each material for each chord is consistent in all geometry engines and AGORA. This demonstrates the geometry navigator engine module ability to accurately identify materials during particle tracking, verifying its reliability regardless of the geometric partitioning strategies employed by the others Monte Carlo codes and navigators used for the comparison. Figure 5.2 shows a schematic description of the procedure, in the following sections a detailed descriptions of each component of the particle tracking verification tool is provided, accompanied by results demonstrating its application to the test geometry models detailed in 3.4.

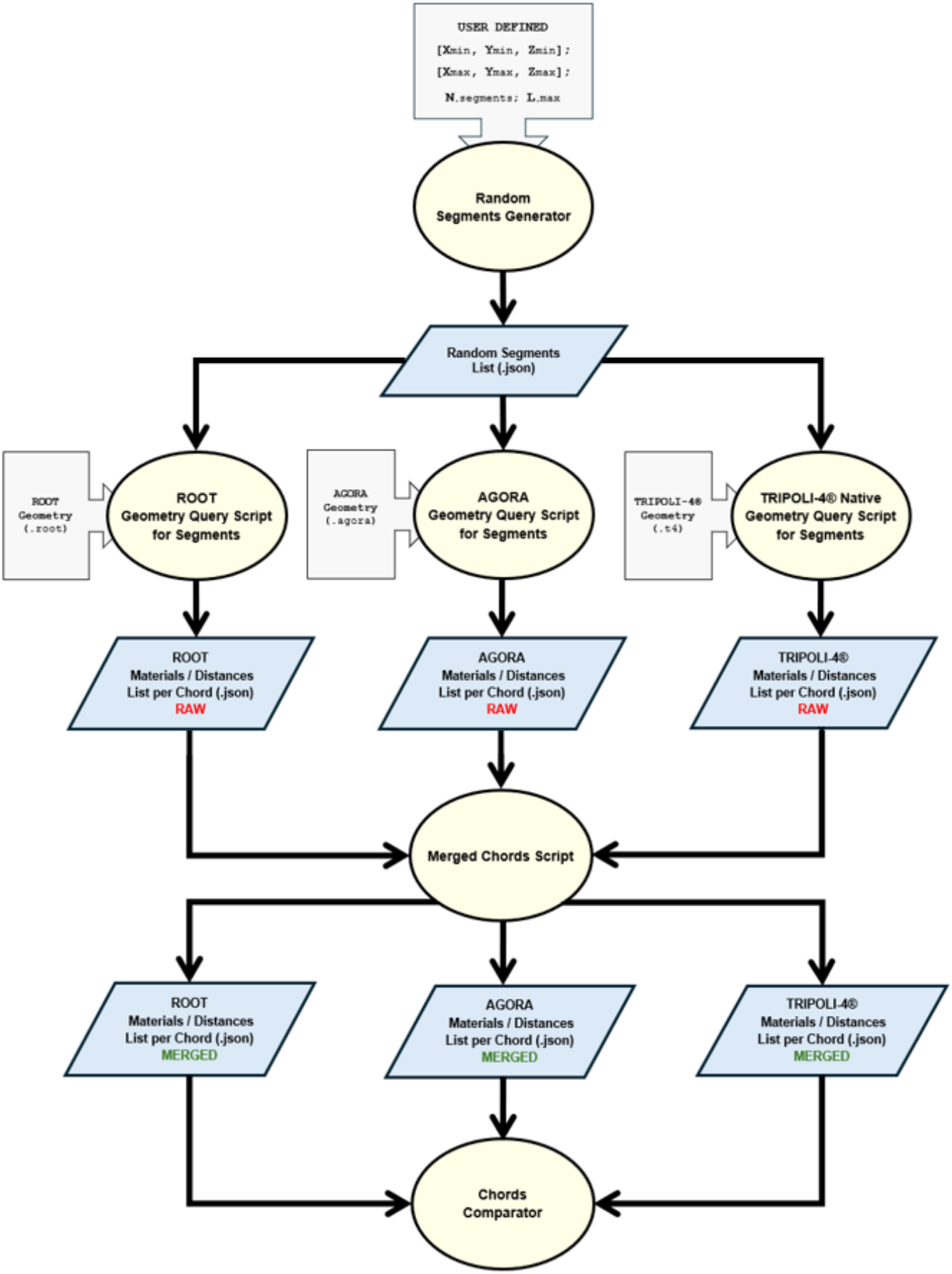


Figure 5.2: Schematic description of the particle tracking verification tool.

## 5.1 Random Segments Generator

The random segments generator script is the first starting component of the particle tracking verification tool, designed to extend material query analyses from random points to random geometric segments within a three-dimensional space.
Its purpose is to generate a set of random segments, each defined by a starting point, a direction, and a length, within user-specified boundaries. These segments act as probes, enabling the interrogation of the model under investigation by the respective geometry navigator engines.
The script ensures robust spatial coverage and flexibility by generating segments within user defined boundaries, allowing for a targeted or global exploration of the geometry under investigation. This capability is crucial because it enables the study of limited portions of the geometry model in cases where inconsistencies arise, focusing the analysis solely on the problematic regions of the model.

#### Inputs

The script requires the following inputs to function:

- **Space boundaries:** Minimum and maximum coordinates (`x`, `y`, `z`) defining the 3D space under investigation.
- **Number of segments:** The desired quantity of random segments to be generated.
- **Segment length range:** The maximum allowable length of the segments.

#### Output

Upon execution, the following output is generated:

- **Segment data:** A list (`.json`) of segments, each defined by a starting point, direction and corresponding length.

The script operates by first generating the starting point of each segment. The starting points are uniformly distributed within the user-defined space boundaries, ensuring that every region within the specified box has an equal likelihood of being sampled. This uniform distribution guarantees thorough coverage of the geometry under investigation.
Next, the script determines the direction of each segment.
This is achieved by generating a random vector $\mathbf{\Omega} = (\Omega_x, \Omega_y, \Omega_z)$, with each component independently sampled within the range [-1, 1]. This ensures that the directions are uniformly distributed across all three dimensions, eliminating any preferential orientation in space. The vector is then normalized, maintaining a uniform distribution of directions, which results in a unit direction vector for the segment.
The length of each segment is subsequently assigned by sampling randomly within the user-defined maximum, as the minimum length condition is set to be greater than a specified threshold value. This ensures that only meaningful segments are

considered, enhancing the quality of the analysis.
Finally, the script compiles each segment, represented by its starting point, direction and length, into a structured dataset. The output is stored in a `JSON` file, ensuring compatibility with the following components of the particle tracking verification tool.

## 5.2 Geometry Query Script for Segments in AGORA

The geometry query script for segments in AGORA is a fundamental component of the tool, enabling a comprehensive analysis of material composition within a specified AGORA geometry. It serves as a critical test for evaluating the functionality of the geometry navigator module, specifically its operability in particle tracking mode. Using randomly generated segments, the script assesses the navigator's ability to manage particle tracking and accurately identify material transitions within the geometry model.

#### Inputs

The script requires the following inputs to function:

- **Geometry file:** The AGORA file `(.agora)` containing the geometry model to be analyzed.
- **Segments file:** The `JSON` file containing the previously generated random segments.

#### Output

Upon execution, the following output is generated:

- **Chord data:** A `JSON` file documenting all the segments decomposed into chords, where each chord represents a portion of a segment that traverses a specific material with a corresponding distance. Each chord includes the material encountered and the distance traveled within that material.

The script begins by loading the AGORA geometry file `(.agora)` and the previously generated random segments from the `JSON` file. The geometry file is then processed by the AGORA geometry navigator module, which enables precise navigation through the model.
For each segment, the process starts by initializing the navigator at the segment's starting point. The navigator module uses the direction and length specified in the `JSON` file to trace the segment's path. If the starting point lies outside the geometry, the navigator proceeds along the segment until it either enters the geometry or reaches the segment's end, recording the traveled distance throughout.
A specific function of AGORA is used to determine how far the navigator can move within a current volume before encountering the next geometrical boundary. This function calculates the distance traveled before transitioning into a new volume,

updating the navigator's state as the segment crosses each boundary. This process allows the script to decompose the segment into individual chords, each corresponding to a specific material and the associated volume transition.
If the segment begins inside the geometry, the script records the initial material and tracks the distances traveled within that volume until either the segment exits the geometry or reaches its full length.
Additionally, the script features a functionality to apply a translation vector to the segments before querying the navigator. This capability accommodates comparisons of the same geometry model represented by two different geometry navigator engines, even in cases where differences in the coordinate reference systems are present.
After processing all segments, the script compiles the results into a `JSON` file. This output includes a detailed breakdown of the materials encountered and the corresponding distances traveled for each chord into which the segments have been decomposed. The structured data format ensures compatibility with other components of the tool, supporting further analyses and comparison of results.

## 5.3 Geometry Query Scripts for Segments in ROOT and TRIPOLI-4®

The geometry query scripts for segments in ROOT and TRIPOLI-4® are designed to analyze geometry models defined in the respective formats by tracing random segments through the geometry and documenting the materials encountered along each segment's path. By supporting different geometry formats, these scripts provide a versatile solution for analyzing material distribution and segment decomposition in to chords, enabling comparison in particle tracking across different simulation platforms. These two distinct components of the tool enable direct comparisons between the material mappings provided by AGORA geometries and those derived from the respective navigators, thereby enhancing the overall robustness of the particle tracking verification framework.

### Inputs

Each script require the following inputs to function:

- **Geometry file:** The file containing the geometry model to be analyzed. For ROOT, this is a `(.root)` file while for TRIPOLI-4® this is a `(.t4)` file.
- **Segments file:** The `JSON` file containing the previously generated random segments, with each segment defined by its starting position, direction, and length.

### Output

Upon execution, each script produces a structured output file:

- A `JSON` file documenting each segment's decomposition into chords for each code, including distances traveled within each chord and the materials encountered.

#### ROOT Geometry Script

The ROOT geometry script is designed to analyze material distribution within a geometry model defined in `ROOT` format. The user provides the `.root` geometry file and a `JSON` file listing the random segments. The script loads the `ROOT` geometry using the `TGeoManager` class and initializes the geometry navigator through its built-in navigation methods.
For each segment, the script sets the initial position and direction using the `InitTrack` method and navigates along the segment path. A specific method is used to determine how far the navigator can move within a volume before encountering a boundary, enabling the calculation of distances and tracking of material transitions as the segment progresses. The script records the distances traveled within each material until the segment exits the geometry or reaches its end.
Upon processing all segments, the script compiles the results into a `JSON` file, documenting each segment's journey through the geometry and its decomposition into chords. For each chord, the material encountered and the distance traveled, corresponding to the chord length, are recorded.

#### TRIPOLI-4® Geometry Script

The TRIPOLI-4® geometry script is designed to analyze the material distribution along the paths of randomly generated segments using the native geometry navigator of the TRIPOLI-4® code.
The user provides the `.t4` geometry file containing the model to be analyzed and the `JSON` file containing the list of random segments to be processed. The script first reads the `JSON` file, which contains the segment data, and converts this information into a format that is compatible with the TRIPOLI-4® native geometry navigator. This formatted data is written into an intermediate `.text` file, which is then passed along with the `.t4` geometry file to a specific TRIPOLI-4® processing tool.
The TRIPOLI-4® tool navigates through the segment path and performs the decomposition of the segments, returning a processed `.text` file with detailed chord data. The script reads this file to extract information about each chord, including the distances traveled and the materials encountered along the path.
The extracted data is then compiled into a structured `JSON` file consistently with the output of the other script in this verification tool.

## 5.4 Integration of OpenMC into the Tool

The procedure for integrating OpenMC into the geometry verification framework follows the same methodology outlined in the previous chapter, where OpenMC's simulation outputs are used to emulate the output produced by the random coordinates generator script.

Also in this case, the integration of OpenMC into the particle tracking verification leverages the track data recorded during a simulation, allowing inclusion in the analysis without altering the source code.

### Procedure

The integration process begins with configuring an OpenMC simulation to generate particle track data suitable for analysis. For this purpose, the simulation uses the same setup as detailed in 4.4, which includes three primary input files:

- `geometry.xml`
- `materials.xml`
- `settings.xml`

For further details on the configuration process, please refer to the previous chapter. However, in addition to the standard configuration already described, an optional configuration is available that can provide valuable support for this analysis. Specifically, this configuration allows for modifications to the `materials.xml` file, enabling the replacement of the standard material compositions with a "transparent" medium, such as Helium-4 at an extremely low density. By using this configuration, particles are able to travel along straight-line trajectories throughout the entire geometry, without experiencing any scattering or attenuation due to interactions with the materials. This ensures that the direction of the particles remains unchanged throughout their path, effectively emulating straight segments.
The output of the simulation is a `tracks.h5` file containing detailed records of particle histories, including their positions, directions, and material interactions. These data are processed for two main reasons: on one hand, to emulate the output of the random segments generator script, allowing querying of AGORA through these segments; on the other hand, to record the materials encountered during traversal by the OpenMC navigator, enabling a direct comparison with the material mapping provided by AGORA for the same geometry model.

### Geometry Query Script for Segments in OpenMC

A dedicated script processes the `tracks.h5` file to extract particle trajectories, segmenting them at points where direction changes occur. It then maps the material interactions along each segment. These points of interaction are crucial for dividing the paths into distinct segments and for conducting the following detailed analysis.

### Inputs

The script requires:

- **Tracks file:** The `tracks.h5` file generated by OpenMC, containing detailed particle trajectory data.
- **Materials file:** The `materials.xml` file, which serve to map material IDs to their corresponding names.

### Outputs

The outputs included are:

- **Segment paths file:** A JSON file documenting the start points, directions, and lengths of trajectory segments. This output is analogous to the random segment generator's output data.

- **Chords data:** A JSON file listing distances traveled through each material along the segments, enabling detailed chords comparison between OpenMC and AGORA geometries.

The script begins by parsing the `tracks.h5` file, which contains the trajectory data generated by OpenMC simulations. Once the data is loaded, the script iterates through the particle trajectories sequentially, processing each particle's path to create segments based on the direction of travel.
Initially, the direction of the particle's movement is monitored at each interaction point. As the particle moves through the geometry, it may change direction due to scattering events or others interactions with materials. The script identifies when these direction changes occur and uses them to delineate distinct segments. A new segment begins whenever the direction of travel differs from the previous one, ensuring that each segment corresponds to a straight path between two or more points.
For each segment, the script records both the starting and ending coordinate of the trajectory. These positions are then used to calculate the Euclidean length of the segment. The direction of movement is also stored, as it remains unchanged throughout the segment. Once the segments are established, the next task is to analyze the materials encountered along each segment to enable a comparison with AGORA. The script examines each segment in detail using the distance traveled through each material it encounters. If the particle crosses into a new material during its path, the script records the distance traveled within material before the transition. The material IDs encountered along the trajectory are cross-referenced with the `materials.xml` file to obtain the corresponding material names, ensuring consistency and enabling meaningful comparison.
Finally, the script generates two output in the form of JSON files, which document the results of the analysis. The first file contains geometric details about the segments, including their lengths and directions. This file will be used to emulate the output of the random segment generator script for subsequent querying of the AGORA geometry navigator module.
The second file lists the decomposition of segments into chords as "seen" by OpenMC, documenting the material encountered and the distance traveled within each material for every chord. This file will be compared with the results provided by AGORA to verify their consistency and equivalence.
Also in this case, the slight modification to the procedure enables meaningful comparisons and provides a comprehensive foundation for the verification of particle tracking across different simulation platforms. The modifications to the previous procedure are outlined schematically in Figure 5.3.

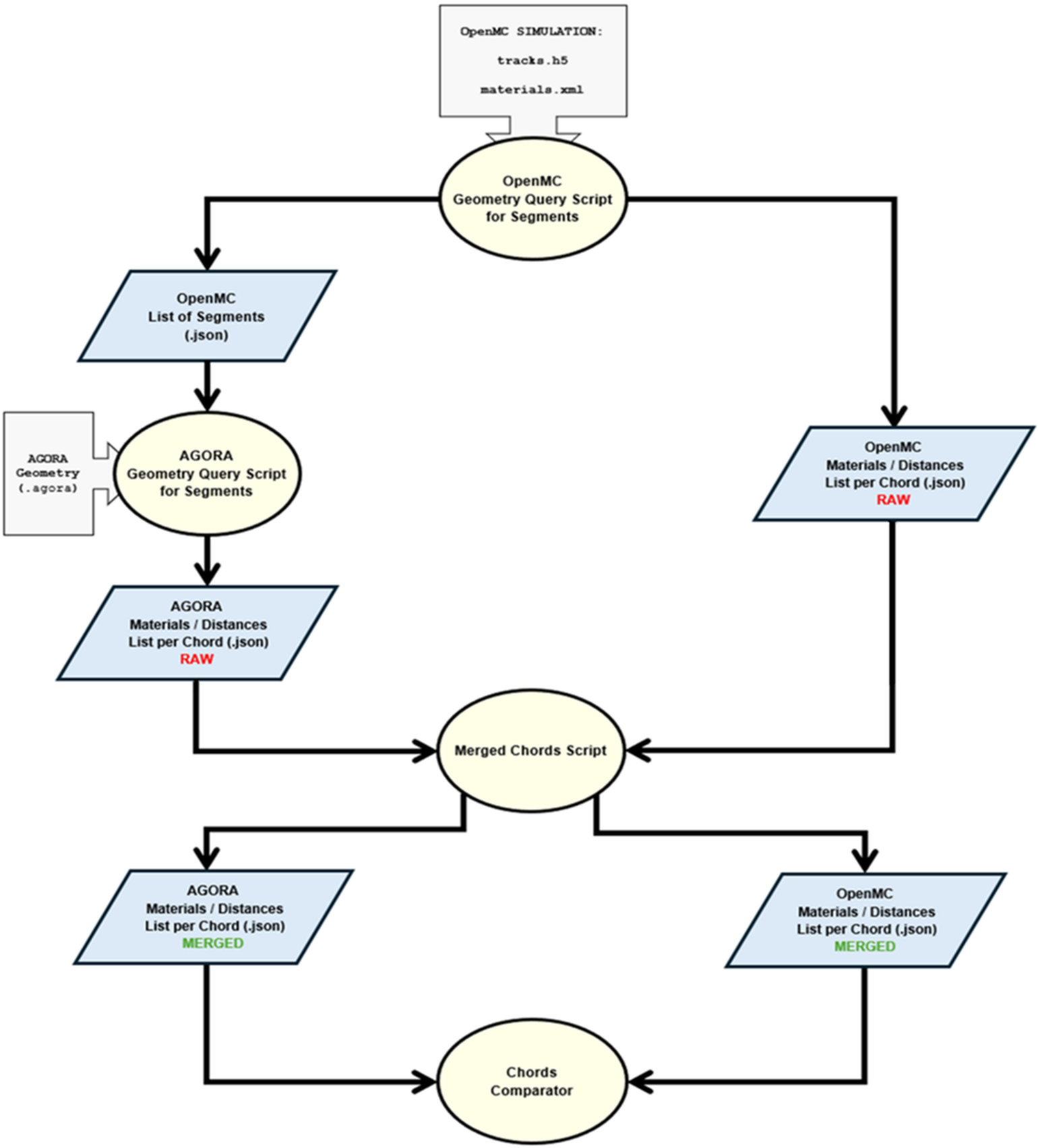


Figure 5.3: Integration of OpenMC in the particle tracking verification tool.

## 5.5 Merged Chords Script

The merged chords script its designed to process the chord data from the `JSON` file generated by each geometry query script previously explained. It operates by aggregating the lengths of consecutive chords that share the same material composition. This merging process is crucial for our verification, as the primary goal is to ensure the accurate identification of materials along each segment by AGORA, rather than focusing on the specific geometric subdivisions made by the geometry navigators involved in the comparison.

#### Input

The script requires the following input to function:

- **Input JSON file:** A `JSON` file containing detailed chord information derived from the segment query scripts of the respective geometry navigator.

#### Output

Upon execution, the following output is generated:

- **Merged chord data:** A `JSON` file containing the previously defined segments, where the distances of two or more consecutive chords within the same segment that share the same material are aggregated into a single entry.

The script begins by loading the input `JSON` file, which contains a list of segments. Each segment is characterized by an identifier, segment details, and a series of chords that capture the distances traveled and materials encountered along the segment's path. The main objective of the script is to merge consecutive chords within each segment that share the same material type. The script iterates through the list of chords for each segment. When consecutive chords are found to have the same material type, their distances are combined into a single entry. If a change in material is detected, the script starts a new entry for the next material type. This approach ensures that the material distribution along each segment is consistently represented, rather than reflecting the volume decomposition of the geometry model. Figures 5.4 illustrate this concept and the functionality of the script. The upper part of the figure presents the AGORA geometric model with the segment decomposition into chords, in the "raw" version (i.e., before applying the merged chords script). In this visualization, the chords are displayed without any merging, maintaining their original separation based on the geometry model's decomposition into volumes.

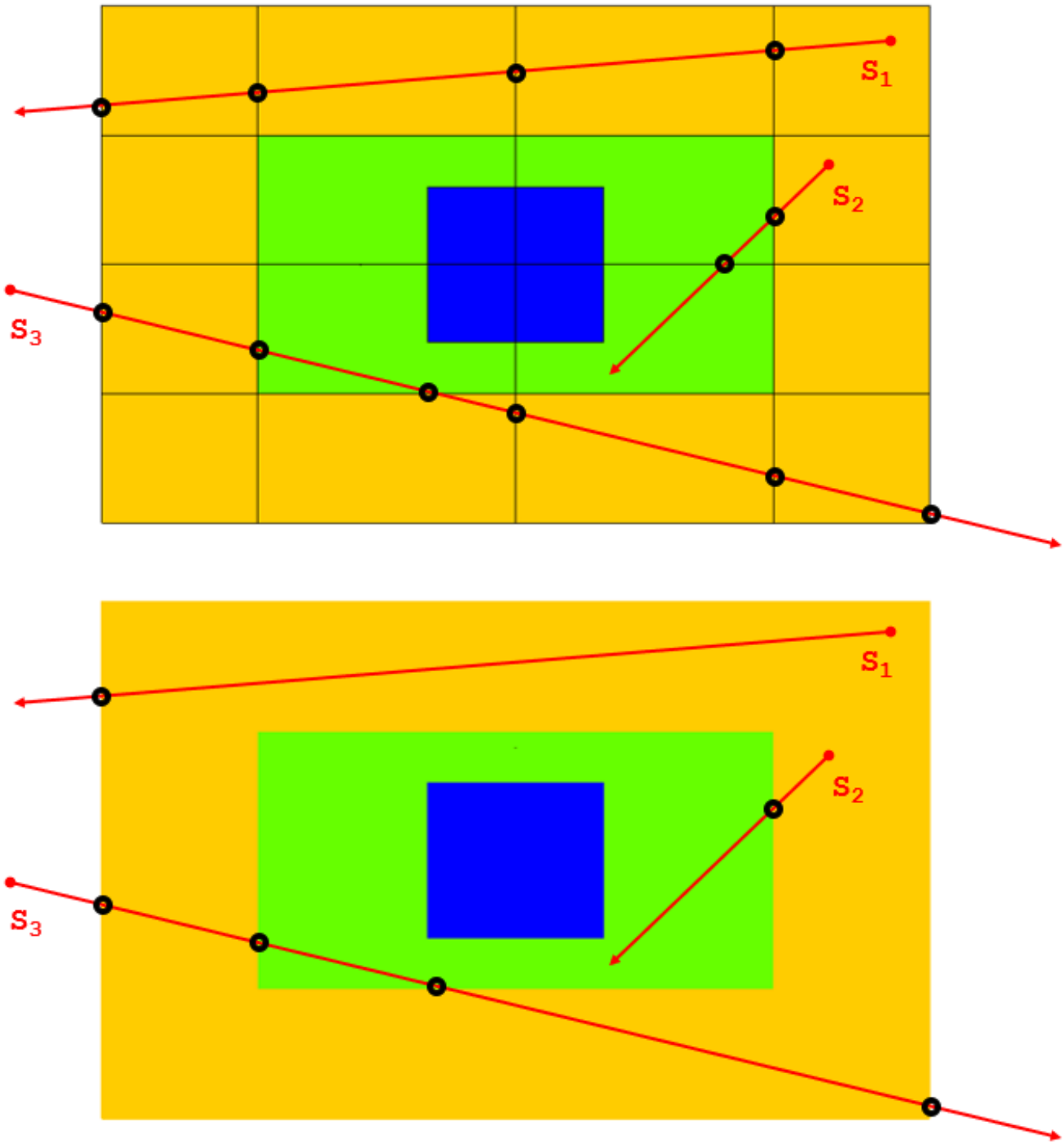


Figure 5.4: AGORA model: the upper part shows the raw chord decomposition, while the lower part shows the chords merged .

In the lower part, on the other hand, the same geometry model is shown with-

out the volume decomposition, as the option to visualize the constituent volumes has been disabled in the T4G visualization tool. Here, the materials that make up the model are distinguished by the same set of colors, but the individual volumes' boundaries are not shown. In the lower representation the same segments are now divided into chords, not based on the volume and the boundaries they previously traversed, but rather based on the material they pass through.
By aggregating contiguous chords lengths' that encounter the same material, we eliminate the variability introduced by different subdivision and geometry constructing methods, enabling direct and meaningful comparisons between results obtained by AGORA and the ones obtained from the other geometry navigators.
After processing all segments, the script provides statistics on the merging operation, including the number of chords before and after the process and the total number of merges performed. The final merged data is then saved to an output `JSON` file, ensuring it is ready for comparison.

## 5.6 Chords Comparator Script

The chords comparator script is the final component of the tool, it is designed to compare segments and their corresponding chord decompositions across two distinct datasets derived from their respective geometry navigators. Its primary goal is to identify any discrepancies in the distances traveled by particles and the materials encountered along their paths, ensuring that the datasets being compared align with a user-defined tolerance threshold.

#### Inputs

The script requires the following inputs:

- **Input JSON file 1:** A `JSON` file containing the first dataset of geometric segments and their corresponding chord decompositions for the geometry under analysis.
- **Input JSON file 2:** A `JSON` file containing the second dataset of geometric segments and their corresponding chord decompositions, derived from a different geometry navigator.
- **Relative tolerance:** A numerical value ($\epsilon$) defining the acceptable threshold for relative differences in chord lengths between the two datasets. If the relative difference exceeds this value, the whole segment is flagged as mismatched.

#### Output

Upon execution, the script produces:

- **Mismatch report:** A detailed output to the console, including the indices of mismatched segments, the total number of analyzed segments, the count of mismatches, and the percentage of errors relative to the total segments.

- **Flagged segments file:** A separate `JSON` file containing the coordinates, lengths, and materials of the segments flagged as mismatch for further analysis.

The procedure begins with the script loading the required input files. These include two `JSON` files representing the segments data derived from the different geometry navigator engines under comparison. Each file is parsed into structured dictionaries, where each segment is defined by an identifier, starting point, direction, length, and the series of chords into which is decomposed. Each chord is represented as a tuple containing the material encountered and the distance traveled through that material.
The script first maps material names in both datasets to numerical indices. This step ensures consistency by eliminating potential issues caused by differing naming conventions between the two navigators.
Before starting the comparison a relative tolerance ($\epsilon$) is applied to account for slight differences in the calculations of chord lengths between the two geometry navigators. This tolerance ensures that only meaningful discrepancies are reported, avoiding the detection of minor differences due to numerical precision or rounding errors criteria of the relative code/navigator engines.
Next, the script compares the segments between the two datasets. For each segment, it checks if the chord counts match. If there is a mismatch in the number of chords, the segment is flagged as a mismatch. If the number of chords is the same, the lengths of the corresponding chords are compared. Any relative difference between the lengths that exceeds the specified tolerance threshold is flagged as a discrepancy. Similarly, the materials associated with the chords are also compared and any mismatches, if present, are recorded. Once the comparison is completed, the script generates a detailed mismatch report in the console that includes the total number of segments analyzed, the number of mismatched segments, and the percentage of discrepancies.
After completing the comparison, the script stores the flagged segments in a dedicated `JSON` file. This file includes detailed information about the mismatches, such as the coordinates, chord lengths, and materials encountered in each dataset, providing a comprehensive overview of the discrepancies. This output facilitates further investigation by offering a clear and organized record of the inconsistencies, ensuring that any issues can be promptly identified and addressed.

## 5.7 Applications and Results

We have conducted an extensive series of verification tests using the same reactor configurations as described in 3.4, leveraging the particle tracking verification tool based on chord analysis. These tests were specifically designed to assess the accuracy and robustness of TRIPOLI-5® with AGORA in particle tracking, comparing its performance with other Monte Carlo codes and geometry navigator engines. For each test, the starting points of the segments were uniformly sampled across the entire geometry, ensuring that the spatial distribution was representative of the full model under analysis. Additionally, the segment directions were generated isotropically, resulting in a uniform angular coverage and eliminating any preferential ori-

entation in the tested directions.
Any chords resulting from the segment decomposition with lengths below an imposed threshold of $\Delta = 10^{-10}$ were excluded from subsequent analyses. This filtering ensures that only chords of significant length are considered, avoiding potential inaccuracies caused by numerical precision errors in very short chords. By focusing on meaningful chord lengths, the comparison reflects the performance of the geometry navigator engines in tracking only relevant particle trajectories.
The statistics of the number of chords per segment, both before and after the merging process, are summarized in Table 5.1 for the reactor configurations executed using TRIPOLI-5® with AGORA.

| **Verification Case** | $\langle N_c \rangle$ **(before merge)** | $\langle N_c^* \rangle$ **(after merge)** |
|---|---|---|
| `Chicago Pile 1` | 2.070 | 2.009 |
| `SPERT-III` | 54.513 | 32.086 |
| `Hoogenboom-Martin` | 233.000 | 162.327 |

Table 5.1: Average number of chords before merge $\langle N_c \rangle$ and after merge $\langle N_c^* \rangle$ per random segment using TRIPOLI-5® with AGORA for different verification cases.

These data offer insights into the segment decomposition and the merging process highlighting the complexity of the underlying geometry model. It is evident that, before merging, a segment is divided into a number of chords proportional to the volume boundaries it traverses in the geometry model. After merging, however, the same segment is divided into a number of chords proportional to the material transitions encountered along its path. A high average number of chords per segment, both before and after merging, serves as an indicator of a particularly complex and intricate geometry model.
The results presented in Table 5.2 offer a comprehensive comparison of the chord length calculations performed using TRIPOLI-5® with AGORA against other Monte Carlo codes and geometry navigator engines. Each verification case was executed using $N_s = 10^5$ random segments, allowing for a robust statistical analysis of the accuracy and consistency across the tested models. The failure rates between TRIPOLI-5® with AGORA and various other codes/engines are presented, accounting for distances excluded when they fall below the specified threshold $\Delta$.
All comparisons were performed with a relative tolerance of $\epsilon = 10^{-10}$, ensuring that even minute differences were captured during the chords lengths' analysis.

| **Verification Case** | **TRIPOLI-4® Native** | **TRIPOLI-4® ROOT** | **TRIPOLI-5® ROOT** | **OpenMC CSG** |
|---|---|---|---|---|
| `Chicago Pile 1` | 0% | 0% | 0% | 0% |
| `Hoogenboom-Martin` | - | 0% | 0% | 0% |
| `SPERT-III` | - | 0.326% | 0.326% | 0.032% |

Table 5.2: Summary of the results obtained for the verification on particle tracking for $N_s = 10^5$ random segments: failing rate between TRIPOLI-5® with AGORA versus other codes and engines.

For the `Hoogenboom-Martin` and `Chicago Pile 1` configurations, the results show perfect agreement among all geometry engines and codes tested with failing rates uniformly at 0%. This indicates that AGORA handled this configurations with consistent precision, achieving identical chord length calculations without discrepancies highlighting the capability of TRIPOLI-5® with AGORA to accurately handle this configurations without introducing any discrepancies in the particle tracking process.

Despite the overall agreement, the tests on chord lengths revealed certain discrepancies. Specifically, an apparent and systematic difference was observed between the AGORA model of the `SPERT-III` reactor when compared to the ROOT model, using both TRIPOLI-5® and TRIPOLI-4®, with a consistent failure rate of 0.326%. This divergence aligns with the findings based on point comparisons and discussed in 4.7, suggesting that these discrepancies may originate from fundamental differences in the geometric representation of the models.
Furthermore, for this verification case the tests highlighted a slight but consistent disagreement between the AGORA and OpenMC models. Notably, this small disagreement persisted even when the relative tolerance $\epsilon$ was reduced, indicating that the observed differences are unlikely to be due to numerical noise or minor variations in floating-point precision.

Instead, the persistency of the mismatches, even with relatively low failure rate of 0.032%, suggest the presence of residual overlapping volumes or inconsistencies in either one or both models, which has led us to investigate further in order to fully understand the underlying causes.

# Chapter 6

# SPERT-III Reactor Case: A Preliminary Validation Test Study Using AGORA and TRIPOLI-5®

Two verification tools were developed and applied to various geometry models in AGORA: a *point location* tool, which verifies whether points in the geometry are correctly assigned to specific materials composition, and a *particle tracking* tool, which aims to ensure accurate tracking of particle trajectories through the geometry model under study. Both tools have demonstrated robust performance across a variety of cases, except for the SPERT-III reactor model, where systematic discrepancies were observed between the AGORA when compared to the ROOT model, using both TRIPOLI-5® and TRIPOLI-4®. These discrepancies, already discussed in previous sections, raised questions about their potential impact on simulation results and emphasized the need for further investigation.

This effort builds upon a study titled *"Benchmarking of the SPERT-III E-core experiment with the Monte Carlo codes TRIPOLI-4®, TRIPOLI-5® and OpenMC"* [13], which involved a joint effort between CEA Paris-Saclay and the Nuclear Research Center Negev (NRCN, Israel). In this work, we contributed by providing the geometric model of the SPERT-III reactor in AGORA for use with TRIPOLI-5®.

The SPERT III E-core experiment, conducted in the 1960s, remains a valuable benchmark for validating neutronics and thermal–hydraulics codes under both steady-state and transient conditions. In this study, the computational results obtained with TRIPOLI-5® in combination with the AGORA geometry engine were compared to those produced by other Monte Carlo codes, such as TRIPOLI-4® and OpenMC, as well as to experimental data.

The analysis focused on key reactor physics parameters, including the neutron effective multiplication factor ($k_{\text{eff}}$), transient and control rod worth, reactivity coefficients, and kinetics parameters, using multiple nuclear data libraries.

While the primary goal of the paper was to develop and verify an OpenMC model of the SPERT-III reactor for stationary criticality calculations, the secondary goal was the verification of the TRIPOLI-5® stationary results using both the ROOT-based geometry initially conceived for TRIPOLI-4® [14] and the new AGORA-based geometry model.

## 6.1 SPERT-III Reactor Model in AGORA

The SPERT-III reactor is a pressurized light-water-moderated reactor, utilizing 4.8% enriched $UO_2$ fuel rods. The standard E-core configuration consists of 60 assemblies, which include 48 fuel assemblies with 25 (5 x 5) pin cells, four assemblies with 16 (4 x 4) pin cells, and eight control rods with fuel followers. Surrounding the core is a multi-layered stainless-steel vessel. At the center of the core, a transient cruciform boron-steel rod is positioned. This rod is rapidly ejected during experiments to insert the necessary reactivity and initiate the desired power excursion.

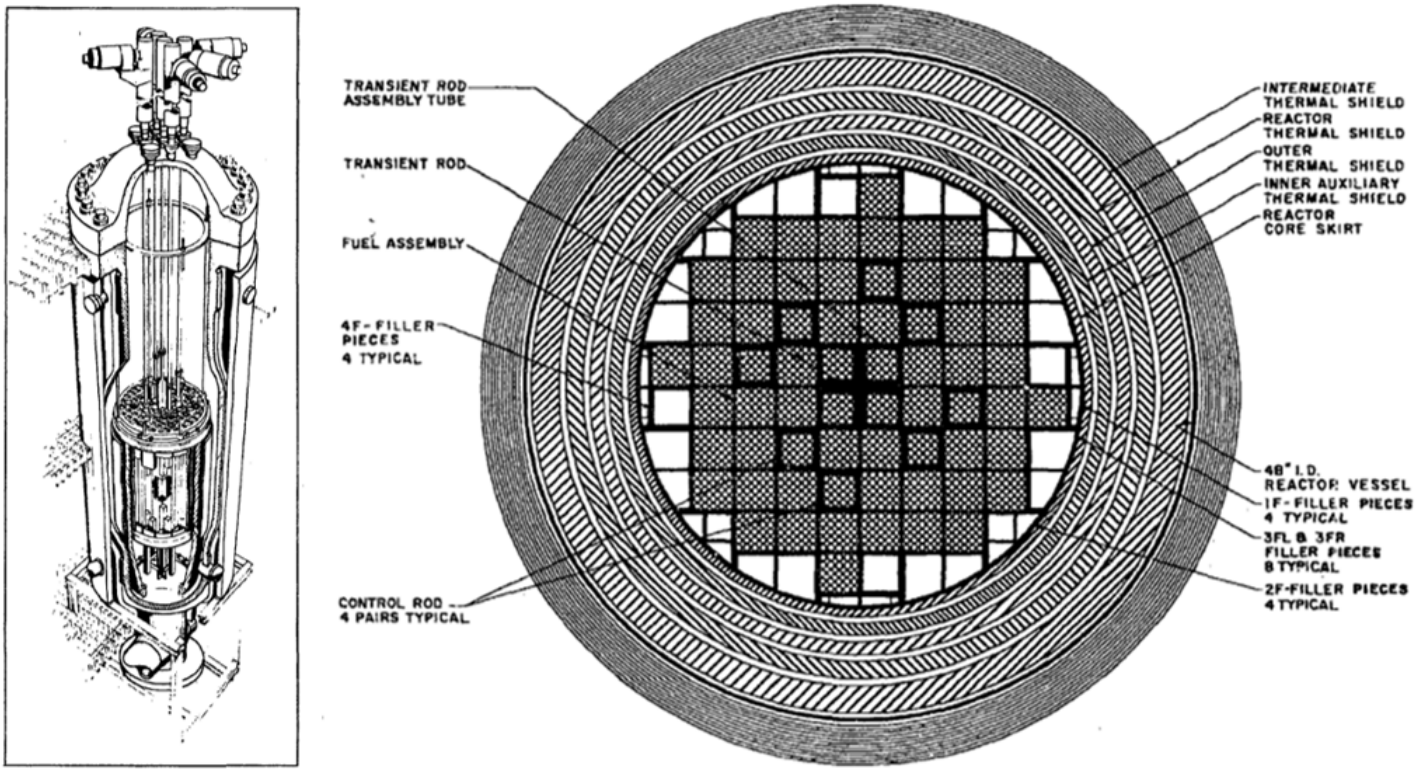


Figure 6.1: SPERT-III reactor: On the left, a general view of the reactor; on the right, a radial view of the reactor in the E-core configuration with a sectional view of its components.[11]

In the context of this study, we developed the geometric model of the SPERT-III reactor in AGORA to be used into the TRIPOLI-5® Monte Carlo code. As part of the modeling process, the various structural elements, such as the fuel assemblies, control rods, transient rod, and grids, were modeled to aiming to a correct representation of all reactor components. Specifically, the AGORA model of the SPERT-III reactor includes **210** surfaces, **227** volumes, **64** universes, and **42** lattices, all contributing to a detailed representation of the reactor's configuration.
Modeling this reactor geometry posed several challenges, particularly in transitioning from the original TRIPOLI-4® [14] model to AGORA, which employs a different methodology for handling and constructing geometries.

The process required converting the TRIPOLI-4® geometry into a CSG framework while ensuring that all geometric details were accurately preserved. This transition demanded a deep understanding of both the original model and the CSG approach to guarantee that the AGORA representation remained faithful despite the change in modeling framework.
In the following sections, a step-by-step breakdown of the SPERT-III reactor's main components is presented, illustrating their geometry and integration within the AGORA model.

### 6.1.1 Fuel Assemblies

The fuel used in the SPERT-III reactor consists of $UO_2$ with 4.8% enrichment. The fuel rods are made of 0.42 inch outer diameter pellets, which are housed in SS348 stainless steel tubes, with a 0.003 inch radial gas gap between the pellets and the inner wall of the cladding. To maintain the symmetry of the reactor lattice, the core includes two types of fuel assemblies: the 5 x 5 elements and the 4 x 4 elements.
The 5 x 5 fuel assemblies consist of 25 fuel rods arranged in a regular 5 x 5 array with a pitch of 0.585 inches. Cooling is facilitated by water circulation through holes in the SS348 assembly boxes of the 5 x 5 fuel assemblies. Although the precise geometry of the assembly box holes is unknown, a homogenization approach was adopted in the original TRIPOLI-4® native model, based on the total volume of the holes. This approximation has been retained in the AGORA model as well.
The 4 x 4 fuel assemblies are similar to the 5 x 5 assemblies, using the same fuel rods and pitch. These 4 x 4 assemblies are located around the transient rod and in the fuel follower section of the control rods. The fuel rods in both assembly types are surrounded by 348SS stainless-steel plugs, which complete the fuel rods between the active core and the grids. Figure 6.5 presents both the fuel assembly, as described in [11], and a radial view of the AGORA model for the 5 x 5 fuel assemblies.

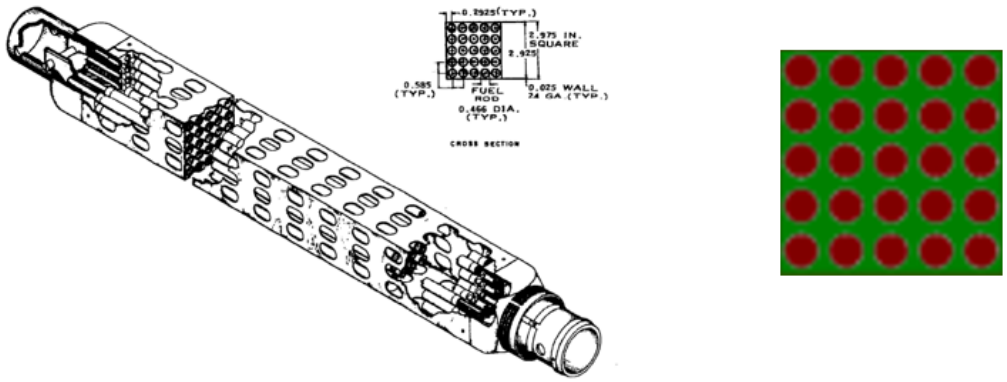


Figure 6.2: Left: Sectional view of the 5 x 5 fuel assembly. [11] Right: Radial view of the AGORA model for the 5 x 5 fuel assembly seen through the visualization tool of TRIPOLI-5®.

### 6.1.2 Control Rods

The SPERT-III reactor has four pairs of control rods, which are inserted from the top of the reactor. The specifications of these control rods are as follows:

- The fuel follower at the bottom consists of 4x4 fuel rod assemblies, protected by a 0.475 cm square zircaloy guide tube.
- The absorber part, located at the top, is made of moderator surrounded by a square casing of 0.472 cm made by 304 stainless-steel, enriched with 1.35% boron. This is further protected by a 0.475 cm zircaloy guide tube.
- The flux suppressor is located at the junction between the fuel follower and the absorber part. It consists of several components:
  1. An absorber grid surrounded by stainless steel with 1.35% boron content.
  2. An absorber grid with moderator in between.
  3. A stainless-steel plate with 3x3 moderator holes.
  4. An absorber grid with stainless-steel rods.
  5. An absorber grid with fuel rods in between.

Each of these components is depicted in Figure 6.3 in its full complexity. In particular is shown their decomposition visualized in materials composition mode, where each color represents a different material.

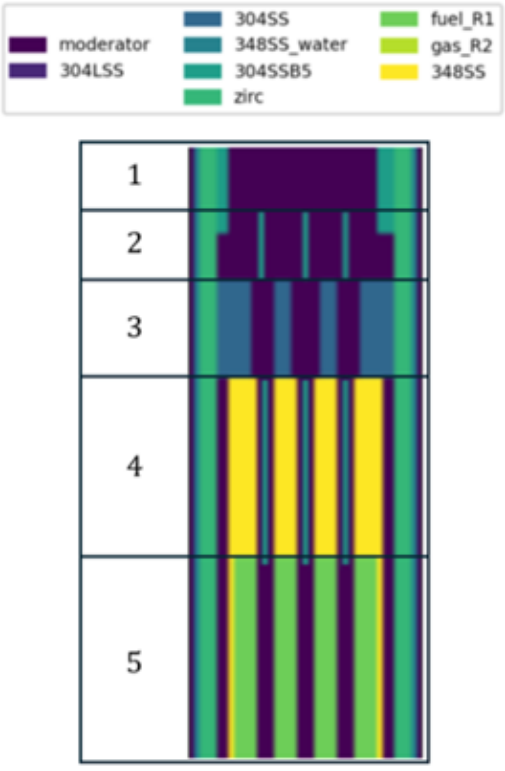


Figure 6.3: Visualization of the flux suppressor parts in materials composition mode.

While in Figure 6.4, the same portion is presented with its decomposition into constituent volumes, where each color represents a different volume in the geometric model.

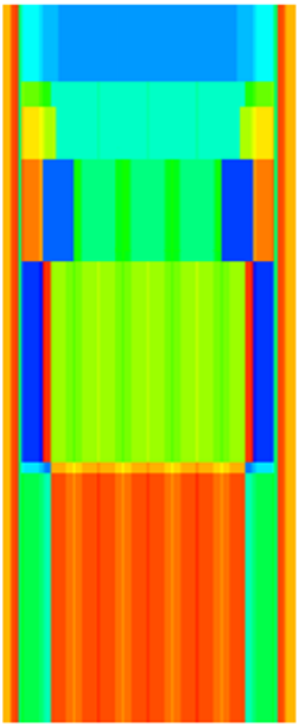

Figure 6.4: Visualization of the flux suppressor parts in volumes decomposition mode.

### 6.1.3 Transient Rod

The transient rod, designed in a cruciform shape, plays a crucial role in inserting reactivity into the core to initiate power excursions for experimental purposes. Under normal operating conditions, the upper section of the transient rod, made of stainless steel, resides within the core, while the lower section, constructed from 1.35% borated steel, is outside the core. To preserve the core's symmetry, the four fuel elements around the transient rod are of the 4 x 4 type. A Zircaloy-2 guide tube protects the transient rod and its surrounding fuel elements. Figure 6.5 presents a view of the transient rod as in [11] and a radial view of the AGORA model for the cruciform transient rod, including the $4 \times 4$ fuel assemblies.

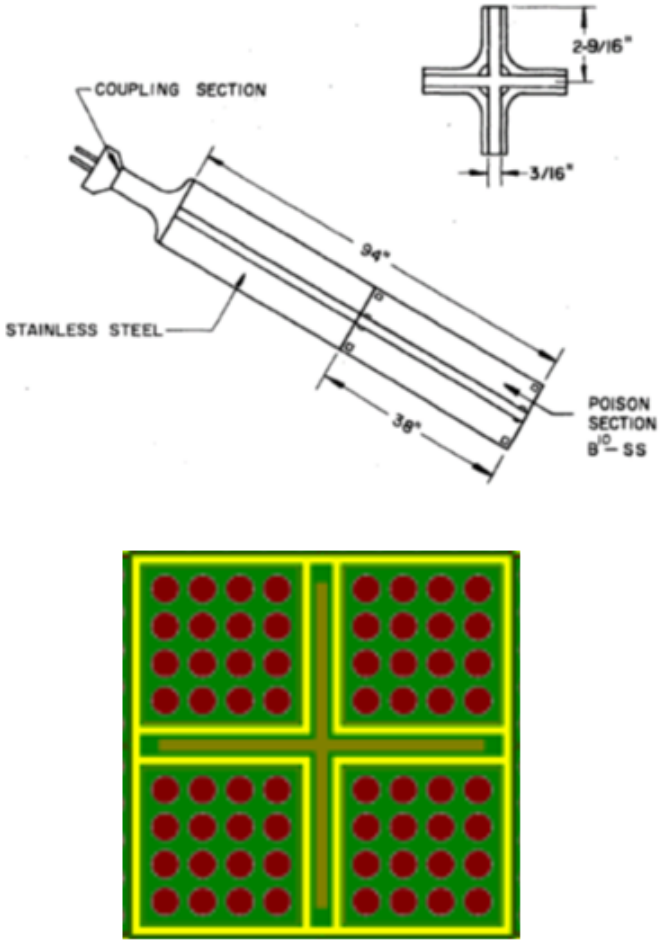


Figure 6.5: Top: View of the cruciform transient rod. [11] Bottom: Radial view of the AGORA model for the cruciform transient rod seen through the visualization tool of TRIPOLI-5®.

### 6.1.4 Top and Bottom Grids

The SPERT-III reactor is supported by upper and lower grids made from 304L stainless steel. These grids hold the fuel elements in place within the core. The lower grid has a thickness of 3 inches and a diameter of 31.97 inches. It contains holes corresponding to the fuel element locations, core fillers, and a cruciform hole for the transient rod. The upper grid, which is 7 inches thick and has a diameter of 42 inches, features holes for the fuel assemblies, transient rod, and control rods. The hole sizes differ: the top grid holes have a radius of 0.444 inches, while the bottom grid holes have a radius of 1.03 inches. The upper grid is surrounded by moderator above and around it, while the lower grid only has moderator below it. Figures 6.6 and 6.7 respectively show the upper and lower grids in the AGORA model of the reactor.

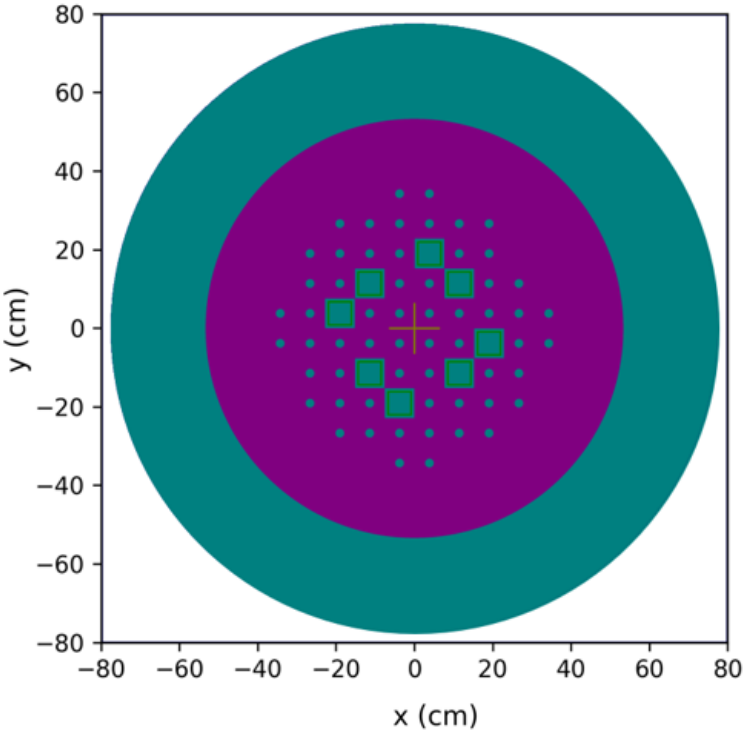


Figure 6.6: Radial view of the top grid seen through the visualization tool of TRIPOLI-5®.

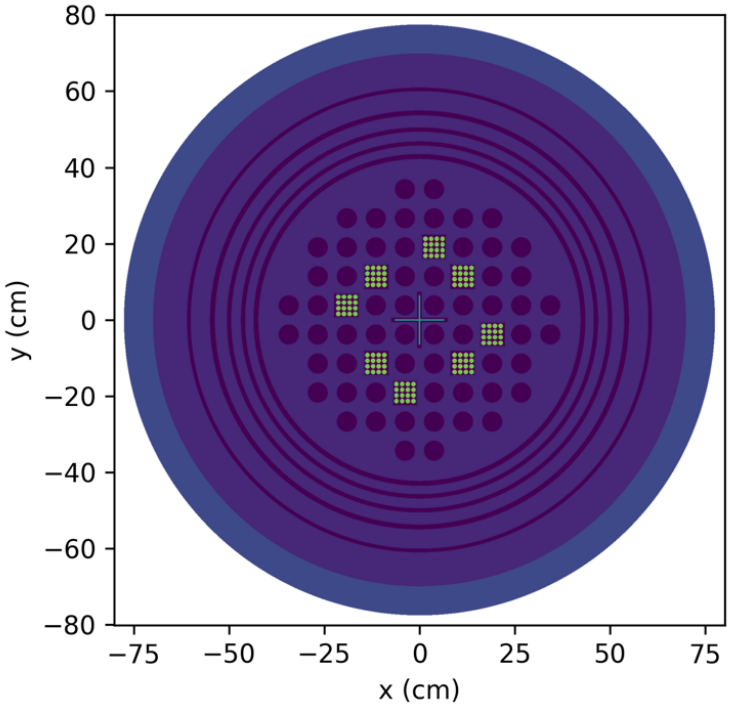


Figure 6.7: Radial view of the bottom grid seen through the visualization tool of TRIPOLI-5®.

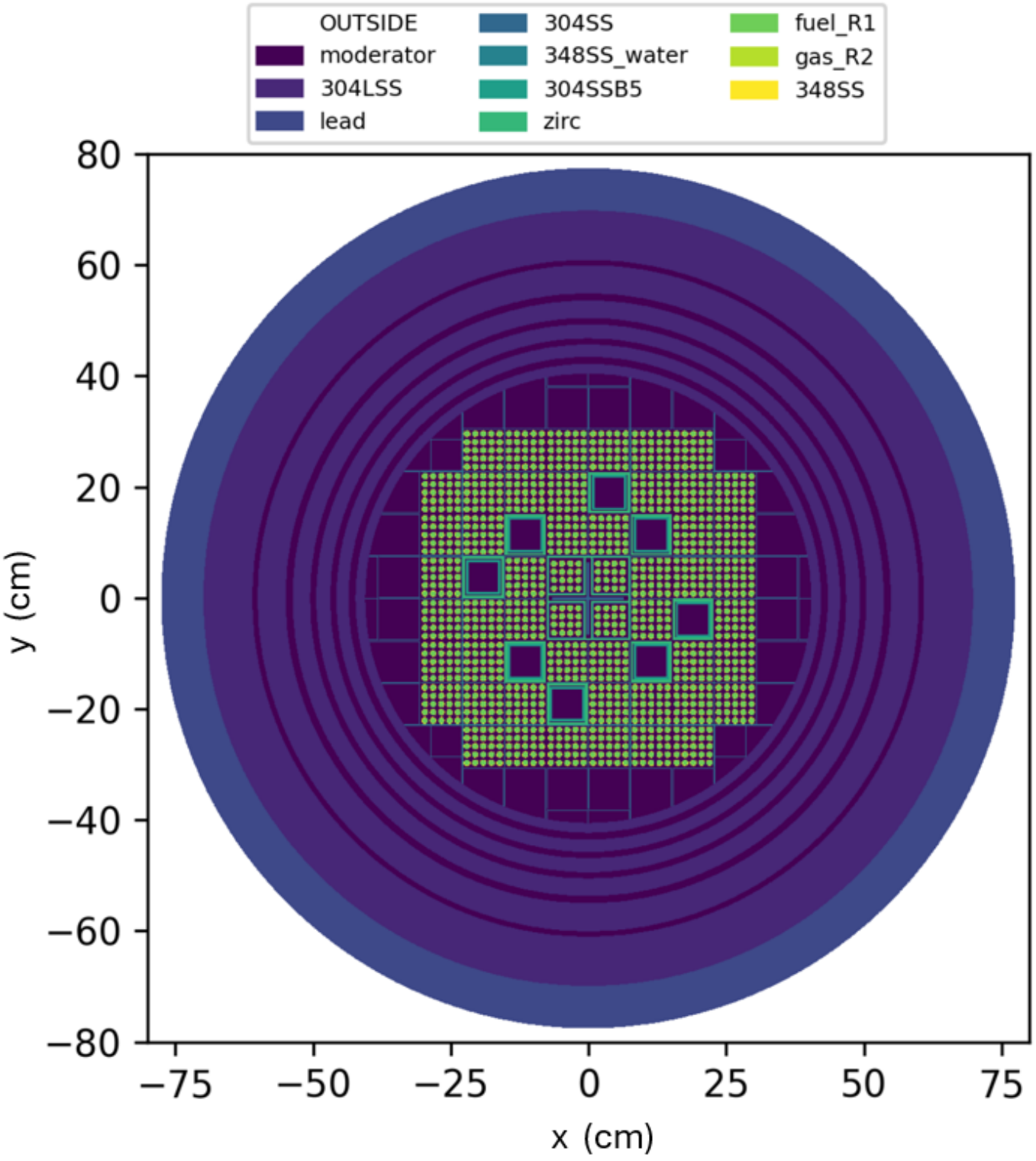


Figure 6.8: AGORA model of the SPERT-III reactor geometry seen through the visualization tool of TRIPOLI-5®: radial cut at z = 0.5

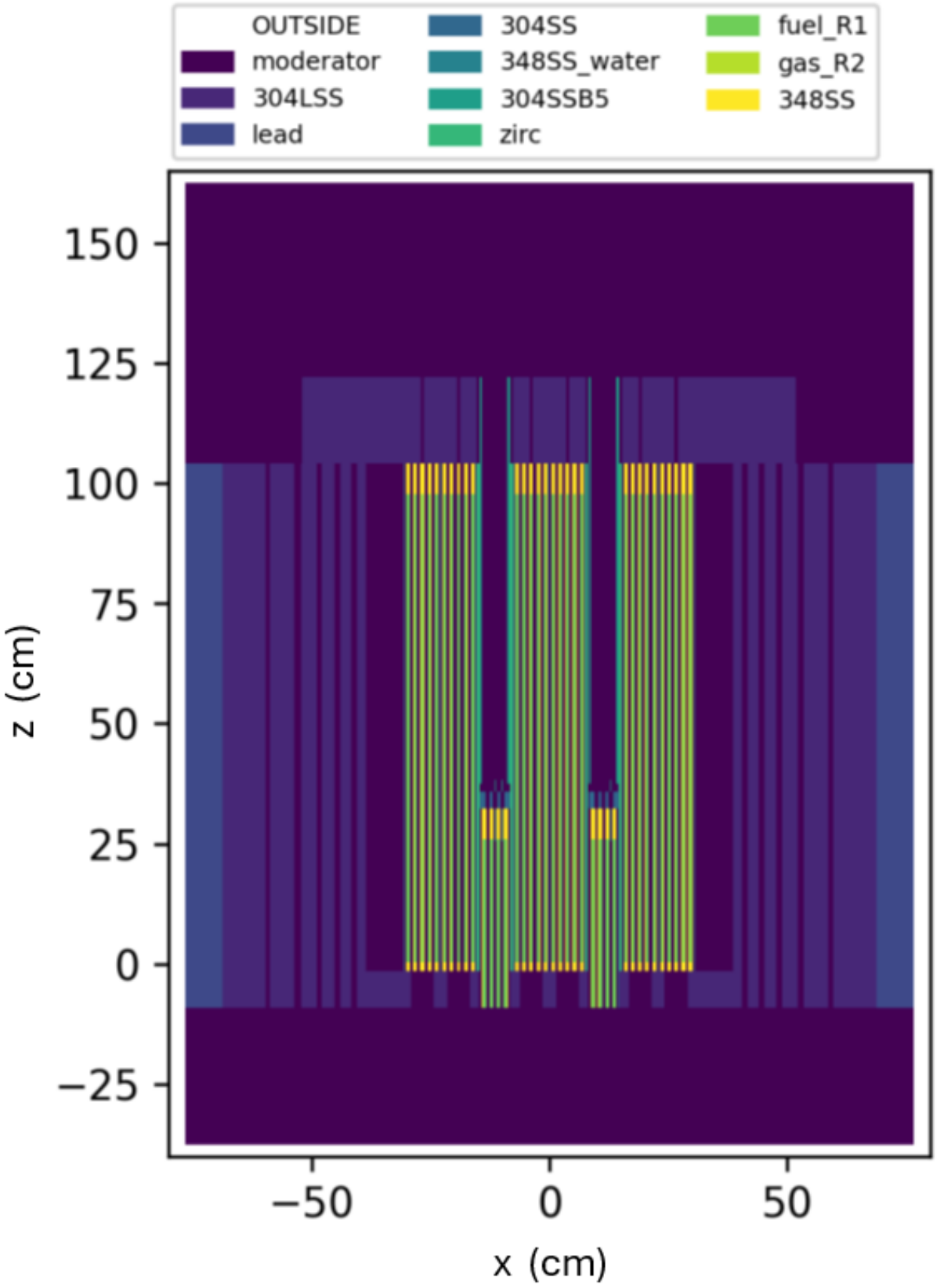


Figure 6.9: AGORA model of the SPERT-III reactor geometry seen through the visualization tool of TRIPOLI-5®: axial cut at y = 12.5

## 6.2 Key findings of the study

In this thesis, only the most significant results from the study are highlighted, specifically the Cold Zero Power $k_{\text{eff}}$ Calculation and the Transient Rod Worth Calculation. All results are presented with associated one-sigma error bars to quantify the uncertainty in the calculations.
For a comprehensive discussion on the physical models implemented, including the correction methods for energy shifts and neutron interactions, as well as the batch and cycle configurations used in the simulations, the reader is referred to the original work [13].

### Cold Zero Power (CZP) $k_{\text{eff}}$ Calculation

In this study, the critical Cold Zero Power (CZP) configuration of the SPERT-III reactor was investigated. In this setup, the control rods are positioned 14.6 inches above the bottom of the active fuel zone, measured from the top of the flux suppressor. Furthermore, the top of the poison section of the transient rod is placed at 0 inches from the bottom of the active fuel zone, meaning it is entirely outside the core. This configuration serves as a benchmark for evaluating the consistency and accuracy of different neutron transport codes.
The values of the effective multiplication factor $k_{\text{eff}}$ computed for the critical CZP configuration using TRIPOLI-4®, TRIPOLI-5®, and OpenMC are summarized in Table 6.1.

| **Code** | **CZP $k_{\text{eff}}$** | **Difference from TRIPOLI-4® [pcm]** |
|---|---|---|
| **ENDF/B-VIII.0** | | |
| TRIPOLI-4® | $1.00029 \pm 7 \times 10^{-5}$ | - |
| TRIPOLI-5® + ROOT | $0.99971 \pm 9 \times 10^{-5}$ | $58 \pm 11$ |
| TRIPOLI-5® + AGORA | $1.00018 \pm 10 \times 10^{-5}$ | $11 \pm 12$ |
| OpenMC | $0.99922 \pm 8 \times 10^{-5}$ | $107 \pm 11$ |
| **JEFF3.3** | | |
| TRIPOLI-4® | $1.00289 \pm 7 \times 10^{-5}$ | - |
| TRIPOLI-5® + ROOT | $1.00262 \pm 9 \times 10^{-5}$ | $27 \pm 11$ |
| TRIPOLI-5® + AGORA | $1.00312 \pm 10 \times 10^{-5}$ | $-23 \pm 12$ |
| OpenMC | $1.00235 \pm 8 \times 10^{-5}$ | $54 \pm 11$ |

Table 6.1: Reactivity results obtained for the CZP configuration using ENDF/B-VIII.0 and JEFF3.3 libraries.

The results obtained with TRIPOLI-5® using the AGORA geometry model are in very close agreement with those from the other Monte Carlo codes involved in the study. The difference between TRIPOLI-5® with AGORA and TRIPOLI-4® is only 11 ± 12 pcm for the ENDF/B-VIII.0 library, and -23 ± 12 pcm for the JEFF3.3 library, indicating a high level of consistency.

### Transient Rod Worth Calculation

The transient rod worth is a key parameter for assessing the impact of control rods in a reactor core. In the study, the control rods were maintained at the "critical" position for the CZP configuration, while the transient rod was progressively inserted from 6 cm to 20 cm, with 2 cm increments. At each insertion step, a criticality calculation was performed, and the corresponding effective multiplication factor $k_{\text{eff}}$ was estimated.
This procedure allowed for the determination of the integral rod worth, representing the total anti-reactivity worth of the transient rod, expressed in dollars \$. The total anti-reactivity values for the transient rod are reported in Table 6.2, comparing results from the different codes involved in the study and measurements from 1966, using different nuclear data libraries.

| | **Measurement 1966** | **TRIPOLI-4®** | **OpenMC** | **TRIPOLI-5® With ROOT** | **TRIPOLI-5® With AGORA** | **Nuclear data Library** |
|---|---|---|---|---|---|---|
| **Total anti-reactivity [\$]** | 4.6 | 4.33 ± 0.02 | 4.31 ± 0.01 | 4.33 ± 0.02 | 4.36 ± 0.02 | JEFF3.3 |
| | | 4.49 ± 0.02 | 4.48 ± 0.01 | 4.49 ± 0.02 | 4.48 ± 0.02 | ENDF/B-VIII.0 |

Table 6.2: Total anti-reactivity results for the transient rod in the CZP configuration using ENDF/B-VIII.0 and JEFF3.3 libraries.

In this case, a slight discrepancy is observed between results obtained with TRIPOLI-5® using the AGORA geometry model and the experimental measurements. This difference can be partially attributed to the inherent approximations made in modeling the transient rod, as some geometrical details were not clearly specified in the original references and had to be approximated. Despite these challenges, the anti-reactivity values calculated for TRIPOLI-5® with AGORA remain in close agreement with the results calculated by the other Monte Carlo codes involved in the study, and are within 10% of the measured values data of 1966.
The results presented for both simulations clearly demonstrate that the outcomes obtained with AGORA and TRIPOLI-5® are in close agreement with those of the other Monte Carlo codes tested in the comparison. Despite the discrepancies identified in both the particle tracking and particle location verification tools (albeit

small in magnitude but consistently observed) it has been demonstrated that their impact on the key physical parameters in the simulations is negligible. Specifically, fundamental quantities such as the effective multiplication factor $k_{\text{eff}}$ and the anti-reactivity values for the transient control rod exhibit strong agreement across different codes under study. This finding underscores the reliability and robustness of the AGORA module in geometry representation and navigation, confirming that the observed discrepancies in the verification tools do not compromise the accuracy of the primary quantities of interest in the simulation results. In the following chapter, it is proposed an extension of the analysis on chord lengths discussed in 5. This extension provides an additional approach for verifying and preliminary validate the performance AGORA by evaluating its alignment with an established physical property of random walks.

# Chapter 7

# Verification of an Invariant Property of Random Walks with AGORA

The analysis on chord lengths, explained in detail in Chapter 5, can be further extended to verify an invariant property of random walks, providing an additional means to prove the accuracy of the TRIPOLI-5® geometry navigator. This method leverages the intrinsic relationship between the geometric characteristics of a body and the statistical properties of random trajectories traversing it. By employing this framework, we can assess whether the geometry navigator accurately traces particle paths while respecting the invariant geometric relationships of random walks. Specifically, this technique is used to verify the correct application of Cauchy's mean chord formula, which relates the average chord length in a three-dimensional body to its volume and surface area.

## 7.1 Cauchy's Mean Chord Formula

Cauchy's mean chord formula establishes a relationship between the average chord length $\langle \ell \rangle$ through a three-dimensional body irradiated uniformly and isotropically and the geometrical quantities of the body, namely its volume $V$ and surface area $S$. The theoretical formula is expressed as:

$$\langle \ell \rangle_{\text{th}} = \frac{4V}{S}$$

where $\langle \ell \rangle_{\text{th}}$ represents the theoretical average chord length, $V$ is the volume, and $S$ is the surface area of the body.
Although the formula is primarily applicable to convex bodies [15], it can also be extended to non-convex bodies, provided that the geometry navigator accurately tracks the segments that traverse the volume. When a segment re-enters the body, the new segment length should be attributed to a new chord rather than being added to the previous one, this ensures consistency in the calculation of average chord lengths, as discussed in [16].

To facilitate this process and ensure adherence to the definition of chords, a specialized geometry was created for this study. Each volume within the geometry is filled with a distinct material, making it impossible for a segment to re-enter a volume without first traversing a different material. This design guarantees that any segment crossing back into a volume is treated as a new chord, aligning with the rule that a chord is defined as the uninterrupted traversal of a single material.
To ensure that the segments are generated uniformly and isotropically, the Random Segment Generator described in 5.1 has been modified. This type of distribution where the segments are both uniformly distributed and isotropic, is referred to as $\mu$-randomness.
This modification to the segment generation process is crucial to ensure that the conditions required for the application of Cauchy's formula are met. AGORA is then queried using the same script introduced in the section 5.2. The results, consisting of the decomposition of segments into chords, their lengths, and the materials encountered along each chord, are then processed by a newly developed script. This script computes the relevant statistical quantities, enabling a direct comparison with the theoretical values predicted by Cauchy's formula.
The procedure for carrying out this comparison is described in the following sections.

## 7.2 Segment Generation

The segment generation process explained in 5.1 has been carefully modified to ensure compliance with the conditions of $\mu$-randomness, a critical prerequisite for applying Cauchy's mean chord formula. This involves ensuring that segments are initiated from a uniformly distributed source and propagate isotropically through the geometric bodies under analysis. This is achieved by ensuring that all segments originate from the surface of a bounding box defined by the user. To define this bounding box, the user must provide three coordinates for the minimum boundary,

$$\mathbf{c}_{\min} = (x_{\min}, y_{\min}, z_{\min}),$$

and three coordinates for the maximum boundary,

$$\mathbf{c}_{\max} = (x_{\max}, y_{\max}, z_{\max}).$$

The face from which a segment originates is sampled based on probabilities proportional to the areas of the bounding box faces. For a box defined by $\mathbf{c}_{\min}$ and $\mathbf{c}_{\max}$, with dimensions

$$L_x = c_{\max,x} - c_{\min,x}, \quad L_y = c_{\max,y} - c_{\min,y}, \quad L_z = c_{\max,z} - c_{\min,z},$$

the probabilities associated with the six faces are:

$$P(i) \propto A_i,$$

where $A_i$ is the area of the face $i$, the associated probabilities are:

$$P(i) = \frac{A_i}{2 \cdot (L_x L_y + L_y L_z + L_x L_z)},$$

with $i \in \{0, 1, 2, 3, 4, 5\}$ corresponding to the faces of the bounding box.

The face index is determined using discrete sampling, with probabilities $P(i)$ as computed above.

The direction of each segment is the generated using a uniform distribution expressed in terms of spherical coordinates by the polar angle $\theta$ and azimuthal angle $\phi$.

The angle $\phi$ is uniformly distributed between 0 and $2\pi$, with the following probability density function and corresponding cumulative distribution:

$$f_\phi(\phi) = \frac{1}{2\pi}, \quad F_\phi(\phi) = \int_0^\phi f_\phi(t)\, dt = \int_0^\phi \frac{1}{2\pi}\, dt = \frac{\phi}{2\pi}, \quad \phi \in [0, 2\pi].$$

For $\theta$, isotropy in spherical coordinates requires:

$$f_\theta(\theta) = \sin\theta, \quad F_\theta(\theta) = \int_0^\theta f_\theta(t)\, dt = \int_0^\theta \sin t\, dt = 1 - \cos\theta, \quad \theta \in [0, \pi].$$

The cumulative distribution $F_\theta$ allows sampling of $\theta$ using the inverse transform method:

$$\theta = \arccos(1 - 2\,\xi_\theta),$$

where $\xi_\theta$ is a random number uniformly distributed in the interval $[0, 1]$.

Given $\theta$ and $\phi$ as sampled above, the corresponding Cartesian components of the unit direction vector $\mathbf{\Omega}$ are retrieved as:

$$\mathbf{\Omega} = \begin{bmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{bmatrix}.$$

Once the starting face $i$ is determined, the origin point $\mathbf{p}_0$ of the segment is uniformly sampled within the face, with coordinates:

$$\mathbf{p}_0 \sim U(\mathbf{p}_{\min}, \mathbf{p}_{\max}),$$

where $\mathbf{p}_{\min}$ and $\mathbf{p}_{\max}$ are the opposite vertices of the selected face.

Finally, the segment length $\lambda$ is determined by finding the first intersection point between the direction $\mathbf{\Omega}$ and the remaining faces of the cube.
For each face $j$ (with $j \neq i$), the distance is calculated as:

$$\lambda_j = \frac{c_{j,k} - p_{0,k}}{\Omega_k},$$

where $c_{j,k}$ is the coordinate of the plane of face $j$ along axis $k$, $p_{0,k}$ is the $k$-th coordinate of the starting point, and $\Omega_k$ is the $k$-th component of the direction

vector.
The final segment length is defined as:

$$\lambda = \min\left(\{\lambda_j \mid \lambda_j > 0, j \neq i\}\right)$$

## 7.3 Geometry Model

The procedure has been applied to a specially designed test geometry, created specifically for this verification framework. Although simple in structure, this geometry model integrates both convex and non-convex volumes. Figures 7.1 and 7.2 illustrates the geometric model, showcasing its composition of concentric cylindrical shells and the presence of a sphere that breaks the symmetry. The different volumes within the model are filled with distinct materials to allow for easy differentiation as follows:

- $v_1$ (innermost cylinder): Filled with FUEL.
- $v_2$ (first cylindrical shell): Filled with STEEL.
- $v_3$ (second cylindrical shell): Filled with MODERATOR.
- $v_4$ (sphere): Filled with LEAD.
- $v_5$ (outermost cylindrical shell): Filled with AIR.

The properties of the target bodies, including their materials, volumes, areas and theoretical mean chord lengths, are summarized in Table 7.1.

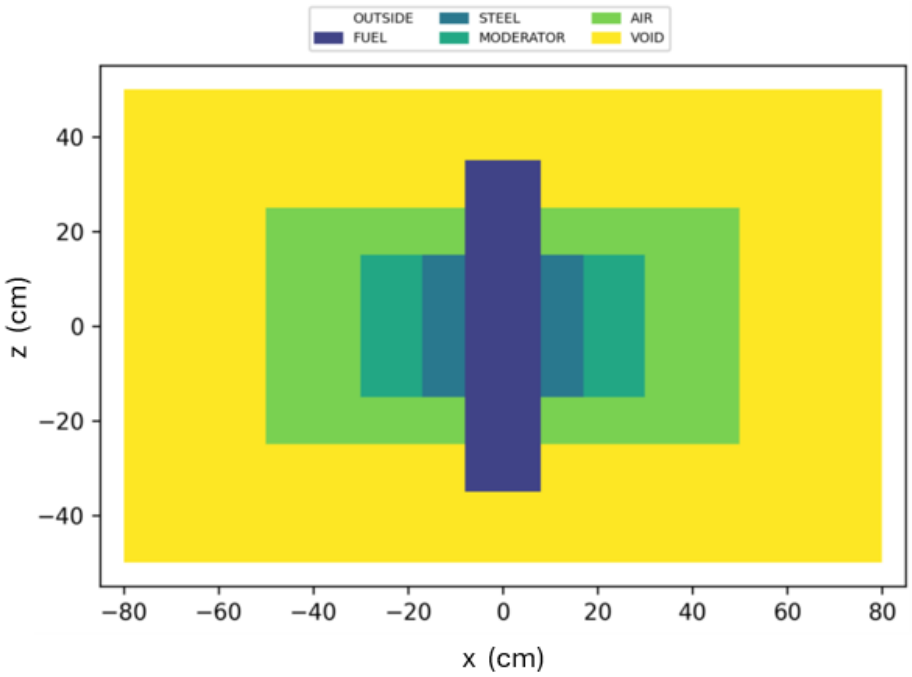


Figure 7.1: Model to test the Cauchy's formula: geometry seen through the visualization tool of TRIPOLI-5® from an AGORA file, section at $y = 0$.

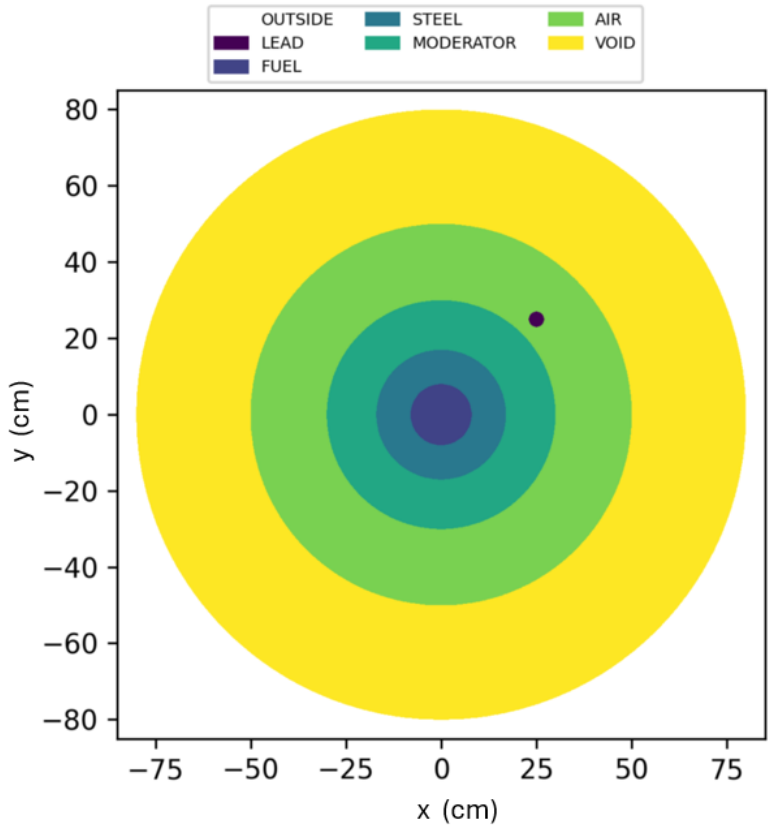


Figure 7.2: Model to test the Cauchy's formula: geometry seen through the visualization tool of TRIPOLI-5® from an AGORA file, section at $z = 0$.

For the sampling of the uniform and isotropic segments, we used a bounding box specifically designed to fully enclose the five target volumes. This fictitious sampling box was positioned in the space between the outermost cylindrical shell, $v_5$, and the external cylinder that encloses all of the five volumes. It is important to note that this yellow outermost cylinder, filled with VOID, does not contribute to the analysis, as it is not fully contained within the sampling bounding box.

| **Volume_ID** | **Material** | **Volume** | **Area** | $\langle \ell_{\text{th}} \rangle$ |
|---|---|---|---|---|
| $v_1$ | FUEL | 14074.335 | 3920.707 | 14.3589 |
| $v_2$ | STEEL | 21205.750 | 6126.105 | 13.8461 |
| $v_3$ | MODERATOR | 57585.393 | 12698.317 | 18.1395 |
| $v_4$ | LEAD | 33.510 | 50.265 | 2.6666 |
| $v_5$ | AIR | 303821.331 | 42976.987 | 28.2775 |

Table 7.1: Features of the test geometry: The theoretical mean chord length $\langle \ell_{\text{th}} \rangle$ is calculated as $\langle \ell_{\text{th}} \rangle = \frac{4V}{S}$, where $V$ is the volume and $S$ is the surface area.

This geometric model was then utilized as input in conjunction with the isotropic uniform segments, within the geometry query script for segments in AGORA, described in 5.2. The script, when executed, generates a `JSON` file as output which contains the decomposition of each segment into its corresponding chords. Each chord in the file includes information about the material encountered and the distance traversed within that material.

## 7.4 Results and Statistical Analysis

The generated `JSON` file is then processed by a dedicated script designed to compute the relevant metrics for this study. Specifically, it analyzes the data from the chord decomposition, calculating the following statistical quantities:

**Sample Average** ($\langle \ell \rangle$): The script calculates the sample average (mean) of the chord lengths within each volume. This is done by summing up all the distances (chord lengths) associated with a particular material and then dividing by the total number of distances recorded (number of chords) within that material. The formula is:

$$\langle \ell \rangle = \frac{\sum_{i=1}^{n} \ell_i}{n}$$

where $\ell_i$ is the distance of the $i$-th chord, and $n$ is the total number of chords that encounter that material.

**Variance**: The variance is calculated to measure the spread of the chord lengths around the mean value. It is computed by taking the difference between each calculated distance and the sample average, squaring this difference, and summing them up. The sum is then divided by $n - 1$ to obtain the variance as follows:

$$\sigma^2 = \frac{1}{n-1} \sum_{i=1}^{n} (\ell_i - \langle \ell \rangle)^2$$

where $\ell_i$ is the $i$-th chord length, and $\langle \ell \rangle$ is the sample average.

**Standard Deviation (SD)**: The standard deviation is the square root of the variance. It provides a measure of how much the chord lengths deviate from the sample average, and it is given by:

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (\ell_i - \langle \ell \rangle)^2}$$

**Relative Standard Deviation (RSD)**: The relative standard deviation is a normalized version of the standard deviation, calculated by dividing the standard deviation by the absolute value of the sample average:

$$\text{RSD} = \frac{\sigma}{|\langle \ell \rangle|}$$

**Student's Score ($t$)**: The Student's score is used to assess the agreement between the theoretical predictions and the Monte Carlo estimates. It is calculated as:

$$t(\ell) = \frac{|\langle \ell \rangle_{\text{th}} - \langle \ell \rangle|}{\sigma(\ell)}$$

where $\langle \ell \rangle_{\text{th}}$ is the theoretical mean chord length, $\langle \ell \rangle$ is the Monte Carlo estimate, and $\sigma(\ell)$ is the standard deviation of the chord lengths.

These quantities provide a measure of the dispersion and reliability of the computed chord lengths allowing to assess how consistent the results are and whether the sample of segments is representative of the expected values.
Finally, the Monte Carlo estimates of the mean chord lengths, computed from $M = 10^6$ random segments, were compared with the theoretical predictions $\langle \ell \rangle_{\text{th}}$ derived from Cauchy's formula as applied in 7.1. The results of the comparison, which incorporate the statistical quantities previously described, are presented in Table 7.2.

| **Volume_ID** | $\langle \ell_{\text{th}} \rangle$ | $\langle \ell \rangle \pm \sigma(\ell)$ | **RSD** | $t(\ell)$ |
|---|---|---|---|---|
| $v_1$ | 14.3589 | 14.3498 ± 0.0289 | 0.002 | 0.3136 |
| $v_2$ | 13.8461 | 13.8138 ± 0.0254 | 0.0018 | 1.2672 |
| $v_3$ | 18.1395 | 18.1048 ± 0.0234 | 0.0012 | 1.4822 |
| $v_4$ | 2.6666 | 2.6514 ± 0.0326 | 0.0122 | 0.4665 |
| $v_5$ | 28.2775 | 28.2916 ± 0.0217 | 0.0007 | 0.6615 |

Table 7.2: Average chord length in target volumes: analytical values and numerical results.

The comparison between the Monte Carlo-calculated chord lengths and the theoretical predictions demonstrated excellent agreement across all bodies, emphasizing the validity of Cauchy's formula for both convex and non-convex shapes. The results provided further confirmation of the reliability of the AGORA estimates. The

Students' Scores for all bodies, whether convex or non-convex, were consistently below 3, indicating that the differences between the Monte Carlo results and the theoretical values were minimal and statistically insignificant.
The strong statistical agreement with an invariant property of random walks underscores AGORA's accuracy in modeling complex structures, ensuring the correct preservation of geometric relationships between the constituent bodies.
This consistency demonstrates that AGORA can effectively track particles, leveraging an intrinsic property of their trajectories, and maintain the integrity of the underlying geometric relationships.
This method is proposed as a tool for verifying the accuracy of AGORA. The invariance of the average chord length ensures that the expected value is well defined, allowing for the assessment of the accuracy through simple statistical tests. These tests can be performed with arbitrary precision, providing a robust way to evaluate the performance of the geometry navigator engine module [17].

# Conclusions

During this thesis, two methodologies were developed for the verification of the new geometry navigator engine of TRIPOLI-5®. These methodologies were designed to verify two crucial and complementary aspects: particle location and particle tracking. Both techniques were implemented to ensure the correct functioning of AGORA, initially tested on simple geometries, before being applied to more complex and realistic reactor models and relevant benchmark cases.
The data related to the new geometry navigator engine of TRIPOLI-5® were compared with other Monte Carlo codes and geometry navigators to perform an external comparison and to ensure internal consistency. The geometries analyzed using these techniques yielded excellent results, demonstrating a remarkable level of agreement. Only in the case of the SPERT-III reactor did the two methodologies reveal minor yet consistent discrepancies. The impact of these differences was assessed through a dedicated study on this specific configuration, showing that even in the worst-case scenario of mismatch rate, whether in terms of points or chord analyses, the key results of interest and the fundamental physical parameters for the simulations remained highly accurate and in strong agreement with expectations, as well as with the results from the other codes under study.
As a continuation of the particle tracking method through chord analysis, the ability of AGORA to correctly compute an invariant property of random walks was verified. The strong agreement between the theoretical results predicted by Cauchy's rule for mean chord length and the results provided by AGORA, following a statistical approach, confirms the high accuracy and precision of the new geometry navigator engine of TRIPOLI-5® in navigating and correctly identifying all components and bodies present in the geometry under analysis.
During my internship, an additional tool was developed for the detection of voids and overlaps within an AGORA geometry model. This tool, of fundamental importance, will further ensure the correct definition of the AGORA CSG models, significantly aiding both the geometry construction and debugging phases. It will be essential in guaranteeing the integrity of geometries, thus ensuring the correct operation of the AGORA geometry navigator and of its host code TRIPOLI-5®.
Future developments will focus on the implementation of a geometry converter from OpenMC (.xml) to AGORA (.agora) geometry models. This will allow AGORA to leverage the vast library of CSG geometries available in OpenMC without requiring manual redefinition by the user. By simplifying the conversion of geometries between different codes, this tool will improve compatibility and make simulations more versatile and accessible across various platforms.

# Bibliography


[1] B. Bonin. *Neutronique*. E-den, Une monographie de la Direction de l'énergie nucléaire, CEA. ⟨cea-01152822⟩. CEA Saclay; Groupe Moniteur, 2013. ISBN: 9782281113716.

[2] E. Brun, F. Damian, C.M. Diop, E. Dumonteil, F.-X. Hugot, C. Jouanne, Y.-K. Lee, F. Malvagi, A. Mazzolo, O. Petit, J.C. Trama, T. Visonneau, and A. Zoia. "TRIPOLI-4®, CEA, EDF and AREVA reference Monte Carlo code". In: *Annals of Nuclear Energy* 82 (2015), pp. 151–160. DOI: 10.1016/j.anucene.2014.07.053.

[3] I. Antcheva et al. "ROOT — A C++ framework for petabyte data storage, statistical analysis and visualization". In: *Computer Physics Communications* 180.12 (2009), pp. 2499–2512. DOI: 10.1016/j.cpc.2009.08.005.

[4] D. Mancusi et al. "Overview of TRIPOLI-5®, a Monte Carlo code for HPC". In: *EPJ N - Nuclear Sciences & Technologies* 10 (2024). Status and advances of Monte Carlo codes for particle transport simulation, p. 26. DOI: 10.1051/epjn/2024028.

[5] Y.-K. Lee and F.-X. Hugot. "TRIPOLI-4® Monte Carlo Code Verification and Validation using T4G Tool". In: *31st International Conference on Nuclear Engineering (ICONE 31)*. 2024.

[6] D. Mancusi. "Automatic Conversion of MCNP Geometries to TRIPOLI-4®". In: *International Conference on Physics of Reactors (PHYSOR 2022)*. ANS - American Nuclear Society, 2022, pp. 2920–2929.

[7] P. K. Romano, N. E. Horelik, B. R. Herman, A. G. Nelson, B. Forget, and K. Smith. "OpenMC: A State-of-the-Art Monte Carlo Code for Research and Development". In: *Annals of Nuclear Energy* 82 (2015), pp. 90–97. DOI: 10.1016/j.anucene.2014.07.048.

[8] Python Software Foundation. *Python Programming Language*. 2023. URL: https://www.python.org/.

[9] United States Department of Energy. *The First Reactor*. Washington, D.C.: History Division, Office of the Executive Secretary, U.S. Department of Energy, 1982.

[10] J. Hoogenboom, W. Martin, and B. Petrovic. "The Monte Carlo performance benchmark test - aims, specifications and first results". In: *M&C 2011 International Conference on Mathematics and Computational Methods Applied to Nuclear Science and Engineering*. 2011.

[11] J. Dugone. *SPERT-III Reactor Facility: E-CORE Revision*. Tech. rep. IDO-17036. 1965.

[12] Ecma International. *ECMA-404: The JSON Data Interchange Format*. 2nd. Ecma International. Dec. 2017. URL: https://ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf.

[13] A. Dervaux, S. Kinast, M. Lemaire, V. Di Blasi, C. Larmier, D. Mancusi, and A. Zoia. "Benchmarking of the SPERT-III E-core experiment with the Monte Carlo codes TRIPOLI-4®, TRIPOLI-5® and OpenMC". In: *EPJ Web of Conferences* 302 (2024), p. 13011. DOI: 10.1051/epjconf/202330213011.

[14] A. Zoia and E. Brun. "Reactor physics analysis of the SPERT III E-core with TRIPOLI-4®". In: *Annals of Nuclear Energy* 90 (2016), pp. 71–82. DOI: 10.1016/j.anucene.2016.01.022.

[15] S. Blanco and R. Fournier. "An invariance property of diffusive random walks". In: *Europhysics Letters* 61.2 (2003), pp. 168–173. DOI: 10.1209/epl/i2003-00208-x.

[16] L. A. Santaló. *Integral Geometry and Geometric Probability*. Cambridge, UK: Cambridge University Press, 2004.

[17] F. Martelli, F. Tommasi, and A. Sassaroli. "Verification method of Monte Carlo codes for transport processes with arbitrary accuracy". In: *Scientific Reports* 11 (2021), p. 19486.