Alma Mater Studiorum · Università di Bologna

SCUOLA DI INGEGNERIA E ARCHITETTURA

DIPARTIMENTO DI INGEGNERIA INDUSTRIALE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ENERGETICA

TESI DI LAUREA

in
Radioprotezione M

INVESTIGATION OF NOVEL METHODOLOGIES FOR MCNP GEOMETRY PREPARATION: APPLICATION ON THE UPPER LAUNCHER BSM OF ITER

CANDIDATO:
Davide Laghi

RELATORE:
Chiar.mo Prof.
Marco Sumini

CORRELATORI:
Ph.D. Marco Fabbri (F4E)
Ph.D. Alfredo Portone (F4E)
Ing. Lorenzo Isolan

Anno Accademico 2017/2018

Sessione III

Contents

[1]A. Portone, personal communication, 2018.

List of Figures

List of Tables

Abstract

Il seguente elaborato è il risultato di 6 mesi di stage svolti presso Fusion For Energy (F4E) nell'ambito dell'analisi del trasporto neutronico con tecniche Monte Carlo. In particolare i temi che verranno trattati sono due e riguardano entrambi la creazione, gestione e utilizzo del codice MCNP per analisi nucleari.

Il primo argomento è quello degli errori geometrici che vengono generati quando avviene una conversione da formato CAD a Constructive Solid Geometry (CSG) e le loro relazioni con il fenomeno delle lost particles. Il passaggio a CSG tramite software è infatti inevitabile per la costruzione di modelli complessi come quelli che vengono usati per rappresentare i componenti di ITER e può generare zone della geometria che non vengono definite in modo corretto. Tali aree causano la perdita di particelle durante la simulazione Monte Carlo, andando ad intaccare l' integrità statistica della soluzione del trasporto. Per questo motivo è molto importante ridurre questo tipo di errori il più possibile, ed in quest'ottica il lavoro svolto è stato quello di trovare metodi standardizzati per identificare tali errori ed infine stimarne le dimensioni.

Se la prima parte della tesi è incentrata sui problemi derivanti dalla modellazione CSG, la seconda invece suggerisce un alternativa ad essa, che è l'uso di Mesh non Strutturate (UM), un approccio che sta alla base di CFD e FEM, ma che risulta innovativo nell'ambito di codici Monte Carlo. In particolare le UM sono state applicate ad una porzione dell' Upper Launcher (un componente di ITER) in modo da validare tale metodologia su modelli nucleari di alta complessità. L'approccio CSG tradizionale e quello con UM sono stati confrontati in termini di risorse computazionali richieste, velocità, precisione e accuratezza sia a livello di risultati globali che locali. Da ciò emerge che, nonostante esistano ancora alcuni limiti all'applicazione per le UM dovuti in parte anche alla sua novità, vari vantaggi possono essere attribuiti a questo tipo di approccio, tra cui un workflow più lineare, maggiore accuratezza nei risultati locali, e soprattutto la possibilità futura di usare la stessa mesh per diversi tipi di analisi (come quelle termiche o strutturali).

Abstract

The following dissertation is the result of a 6 months stage conducted at Fusion for Energy (F4E) in the neutron transport analysis field with a Monte Carlo approach. Two topics in particular will be discussed, both regarding the creation, management and usage of MCNP models for nuclear analysis purposes.

The first topic is about the geometrical errors that are generated during a conversion from CAD to Constructive Solid Geometry (CSG) format and their relationships with the lost particles phenomenon. The translation to CSG through software is indeed inevitable when building complex models like the ones used to represent ITER components and this can generate areas in the geometry that are not well defined. These areas cause a loss of particles during the Monte Carlo simulation, affecting the statistical integrity of the transport solution. This is the reason why it is key to reduce these errors as much as possible and so the focus of this work was to find a standard way to identify these errors and finally estimate their size.

If the first part of the thesis is focused on the problems deriving from CSG modelling, the second one, instead, suggest a possible alternative, which is the use of Unstructured Meshes (UM), an approach that is the basis of CFD and FEM analysis, but is highly innovative in the field of Monte Carlo codes. In particular UM have been applied to a portion of the Upper Launcher (an ITER component) in order to validate this methodology on high complexity nuclear models. The traditional CSG approach and UM one have been compared in terms of computational resources consumption, velocity, precision and accuracy both on global and local results. What emerged is that, even if there are still a few limits on the methodology due also to its novelty, many advantages can be found on the UM application. Among them there are a more linear workflow, better accuracy in local results, and more importantly the future possibility of using the same mesh for different types of analysis (like thermal or structural).

Chapter 1

General Introduction

The focus of the thesis was to improve the geometry modelling conditions in nuclear analysis investigating current problems and testing innovative solutions. This thesis is divided in three chapters: first, a general introduction discussing a few concepts that will be useful to understand the rest of the work, and then the two main topics which are the investigation of the lost particles phenomenon and the application of Unstructured Meshes (UM) to MCNP models.

1.1 F4E and the ITER project

All the work related to this thesis was carried out at Fusion For Energy (F4E), the domestic agency that is responsible for the European contribute to the ITER project. ITER, when completed, will be the biggest and most advanced prototype of a tokamak fusion reactor and it is currently under construction in Cadarache, France, where the first plasma date is set to be 2025. The partners participating to ITER are European Union (EU), USA, India, Korea, Japan and Russia, each one represented by its domestic agency. F4E, in particular, is responsible for the biggest contribution to the project (around 50%) because EU will be the one actually hosting the reactor. Since each partner contribution is "in kind",i.e. not to provide money, but to deliver reactor components instead, F4E is responsible for the preparation and coordination of the design, research and development (R&D) and fabrication of most of the high-technology components that are required to construct ITER. All of this is done both internally and involving skillful external companies through public tenders. Working in this environment was key to have access to the latest nuclear analysis technologies and to models that probably are the bigger and more complex in the whole nuclear community.

1.2 MCNP: a Monte Carlo code

1.2.1 Code's Basics of functioning

MCNP[7] (Monte Carlo N-Particle) was the code on which this study was based. This was done because it is the reference nuclear code used at Fusion for Energy and ITER project, but also because it is probably the most common (and better qualified) code used nowadays for studying the transport of particles and radiations. As several other nuclear analysis codes, MCNP is based on a Monte Carlo simulation approach, meaning that, differently from deterministic ones, its solution is based on statistics and probability. This is done because the transport of particles and radiations is extremely complex and highly depends from parameters like the cross sections that regulate the probabilities of different interactions of the particles with matter and from medium geometry. These cross section are derived experimentally as well as from simplified analytical models and depend on a bunch of factors like type of particle, material, energy, ecc.

The generic functioning of Monte Carlo code is as following: a particle is generated in the geometry with a certain position, direction and energy according to the source specifications that have been defined by the user and then the code starts to simulate what is called the particle history. The mean free path of the particle is computed and used to advance to the next interaction where, according to cross sections, the particle obtains a new direction of flight and energy. This process goes on until the particle is absorbed, exits from the geometry or it is cut off from the code because under the energy limit set by the user. During the history there could also be interactions that result in the creation of secondary particles (e.g. high energy neutrons generating photons in the tokamak machine): the histories of these particles can be simulated as well. When a sufficient numbers of histories is simulated in this way, thanks to the the Central Limit Theorem[3], it is possible to derive a general solution for the transport problem.

The great advantage of Monte Carlo codes is that they allow to simulate the transport on far more complex geometries than deterministic ones while the drawback is that it could require bigger computer resources and running time in order to simulate enough histories to have good statistical results. Indeed all the simulations involving complex models during the studies for this thesis were run on MARCONI super computer of the CINECA Computer Center in Bologna.

1.2.2 Constructive solid geometry CSG

An additional complexity that characterizes Monte Carlo codes is the use of Constructive Solid Geometry, also known with the acronym CSG. As it can be visualized in fig 1.1 Constructive Solid Geometry means that every complex shape has to be built starting from primitive solids like cubes, spheres, cones, ecc. that are combined using boolean operators. The reason for this is that when the code is simulating a particle crossing a surface it needs to compute the normal to that surface using its mathematical equation, and this is possible only up to second order surfaces.

This requirement considerably limited the complexity of nuclear transport modelling in the past since they had to be built by hand (even if they were still way less simplified than the ones used for deterministic codes). Luckily nowadays there are softwares like SuperMC[11] that provide a semi-automatic conversion from CAD to CSG format ready to be use in an MCNP input. This allowed to exponentially increase the complexity of nuclear transport models reaching peaks like the C-Model[5], the 90 degrees section of ITER's tokamak used as the reference model for all ITER nuclear analysis (a sectioned view of the C-model is represented in fig 1.2).

However, the CAD-CSG translation is not fully automated: in order to be able to convert complex CAD models, the analyst always has to perform additional manual operations like repairing, defeaturing and splitting. Defeaturing consists in slightly simplifying a CAD model removing all the smaller features like small rounds, short edges, etc. These features indeed have no significant impact on the analysis results while they increase dramatically the complexity of the conversion. Moreover special care has to be applied on removing all the splines in the CAD model since they are not allowed in MCNP geometries.

Even after defeaturing, a geometrically complex component is unlikely to be translated successfully without performing a splitting session first. Several cuts have to be performed on the CAD for two main reasons:

1. Without splits the software would try to convert the entire part in a unique cell, inevitably failing.

2. It has been observed that MCNP geometry loading time is driven by the cells which have

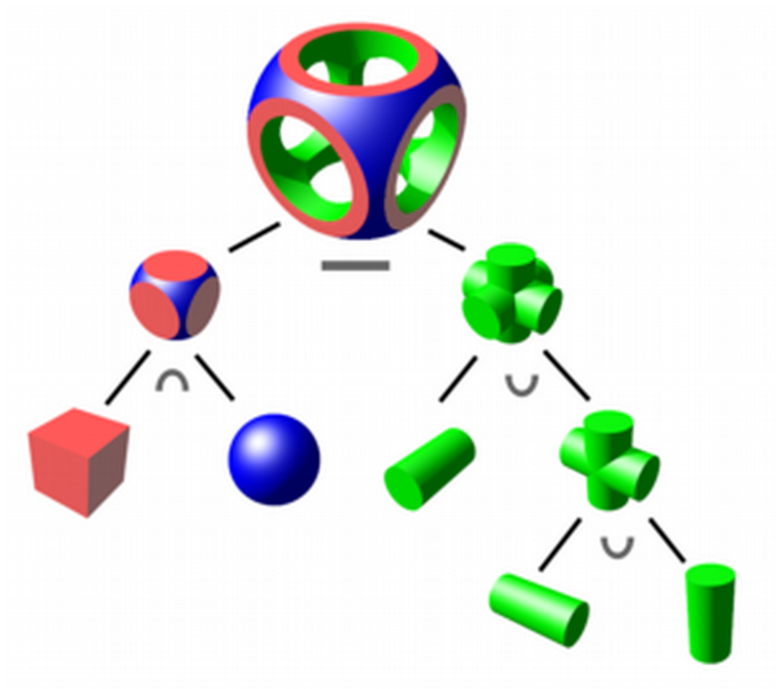


Figure 1.1: CSG: a complex cell is built using primitive solids and boolean operators

the higher number of "words" in their definition. This means that a higher number of simpler cells has to be preferred instead of fewer cells presenting a high number of "words".

Additionally, in order to not increase the number of words or the total number of surfaces in the model, it is good practice to split the CAD with surfaces already used by the geometry, without creating new ones.

These processes are quite time consuming and sometimes it happens that the conversion of complex or broken models generates geometrical errors, since the software is not perfect. In the following chapter of this thesis the topic of geometrical errors and their relationship with the lost particles phenomenon is addressed, while in the third chapter an alternative solution to the use of CSG is proposed and its results compared with the traditional workflow.

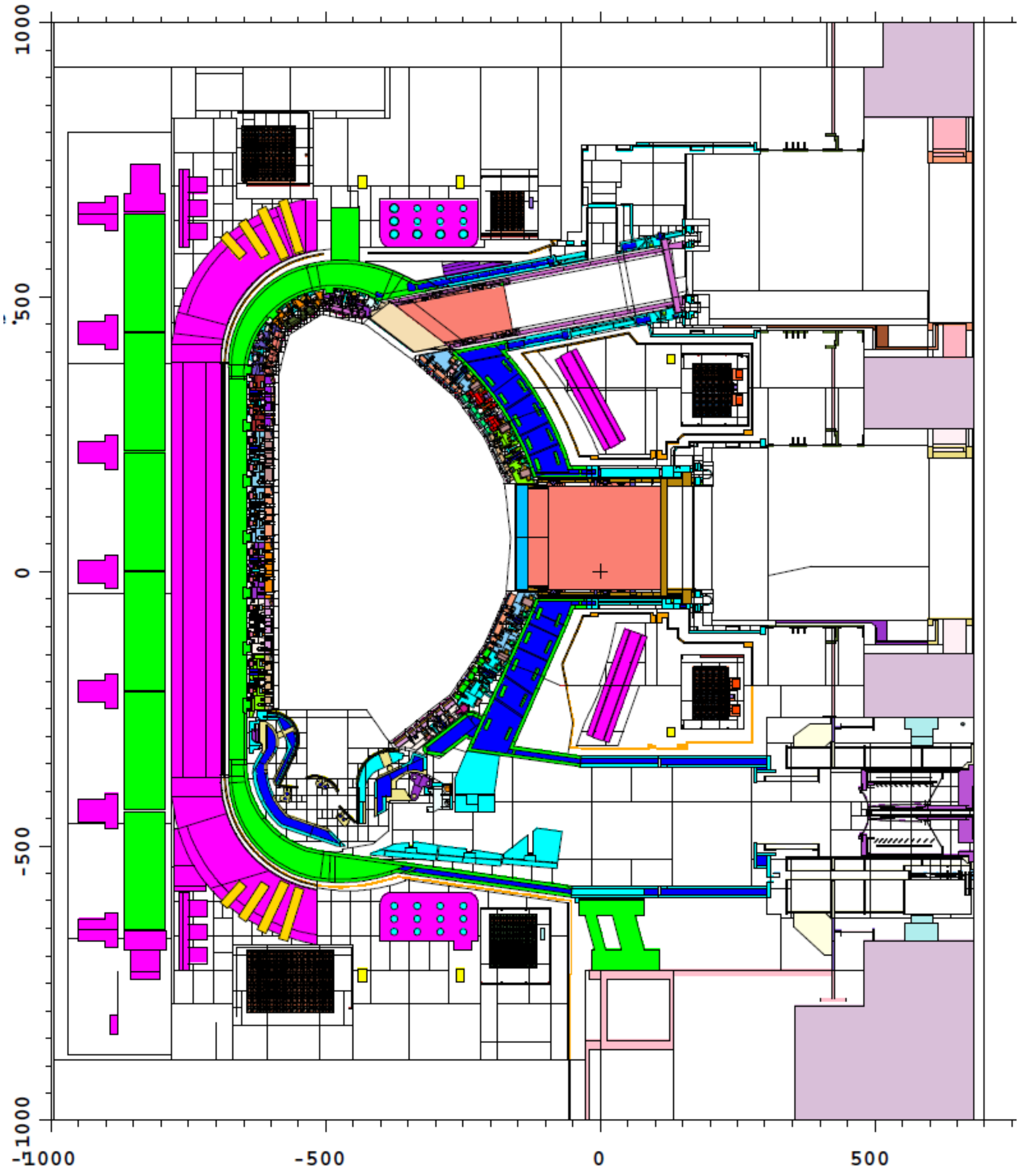


Figure 1.2: XZ section of the C-Model

Chapter 2

Understanding and Investigating the Relationships between Geometrical Errors and Lost Particles in MCNP

2.1 Introduction

2.1.1 The errors

There are actually only two possible types of errors that can be generated during a CAD-CSG conversion and it's possible to refer to them as the "intersecting geometry error" and the "undefined geometry error" .

The intersecting geometry error arises when two or more cells are wrongly defined in such a way that they overlap: this forces the code to associate multiple datasets to the same region of space while giving it no means to chose which one it should use, causing a particle crossing that region to be lost. Instead the undefined geometry error consists in a region of space that does not belong to any cell, meaning that the code has no datasets at all associated with that region: this again causes lost particles.

The nature of these errors and how they exactly work will be better examined in the following sections, but it is important to understand since the beginning why there is such an interest on lost particles. As explained in section 1.2.1 the MCNP solution is a statistical one, and it is based on computing the mean of all histories simulated. When a particle is lost, the history has been simulated, but then removed from the computing of the final solution. This can affect the statistical integrity of the solution, resulting in non-physical results. Indeed, in the MCNP manual[7], it is recommended that the number of lost particle during a simulation should not exceed 10, since even a single lost particle would already show that there is a glitch in the model geometry. This limit was thought for when the models were still made by hand, nowadays, in complex simulations as the one performed on ITER components, this is often exceeded. The lost particle phenomenon is currently tolerated, since its negative effect is considered to be less significant compared to the better accuracy that can be achieved by using more detailed models.

However, even if tolerated, it is still key to keep the number of lost particles as low as possible and great efforts are spent usually to debug models from geometrical errors causing them. This is the reason why it was important to investigate and understand more in detail how the particles are lost and which are the key parameters to take into account. From that, the final goal of this study was to identify a standard procedure for the debugging process, and to come up with a way to estimate the size of these errors.

2.1.2 The spherical source

The preferred way to debug a model for lost particles is to flood the geometry with particles hoping that some of them will get lost and reveal the geometrical errors that have to be corrected. This lost particles test provides also an idea of the lost particles rate (the number of lost particles divided by the number of histories simulated), which is often used as the parameter on which a threshold is set dividing acceptable from unacceptable models. At the moment though, analysts in the fusion community use different sources and set-ups to perform this test and these differences can change the lost particles rate (from now on LPR) even if the tests are performed on the same model. In other words there is not a standard procedure to follow that helps to characterize the quality of a model in terms of objective and unchanging parameters, leaving an opening for this work.

With this standardisation need in mind, the starting point for the study was to analyse the source suggested by MCNP manual when debugging for lost particles that hereafter is described:

- Spherical surface source
- Use of the VOID card
- NRM=-1 (inward direction)
- Default settings for VEC and DIR

The use of the VOID card means that the particles will not have interactions with matter and will travel in a straight line after being generated. This also allows to simulate a greater number of histories in a relative small time to properly flood the geometry with particles. Of course the source will have inward direction since there is no interest on everything outside it, while in order to better understand what's intended with default DIR and VEC here is reported a passage directly from the MCNP 6.2 manual[7](pag. 3-125):

"The source variables SUR, VEC, NRM, and DIR are used to determine the initial direction of source-particle flight. The direction of flight is sampled with respect to the reference vector VEC, which can itself be sampled from a distribution. The polar angle is the sampled value of the variable DIR. The azimuthal angle is sampled uniformly in the range from 0^o to 360^o [...] If VEC is not specified for a distribution on a surface (SUR=0), the vector normal to the surface, with the sign determined by the sign of NRM, is used by default. If DIR is not specified for a distribution on a surface, the cosine distribution [$p(DIR) = 2 \times DIR, 0 < DIR < 1$] is used by default".

This means that the flux through a general surface A from a surface source S will be:

$$\int_A \int_{\mathrm{S}} \int_\alpha \int_\phi S_0 \frac{2\cos\alpha}{2\pi} \sin\alpha \; d\alpha d\phi dS dA \tag{2.1}$$

where α and ϕ are respectively the polar and azimuthal angle in the VEC reference system, while $S_0 = \dfrac{nps}{S}$ will be the source strength.

The use here of a cosine distribution[6] grants that the particles fluence inside the sphere will be isotropic and uniform. An analytical proof of that and further explanation about expression (2.1) can be found in appendix A of this report. This is a key passage, because it means that the orientation or the position in space of the geometrical errors will not affect the number of particles that get lost since the fluence is uniform and isotropic.

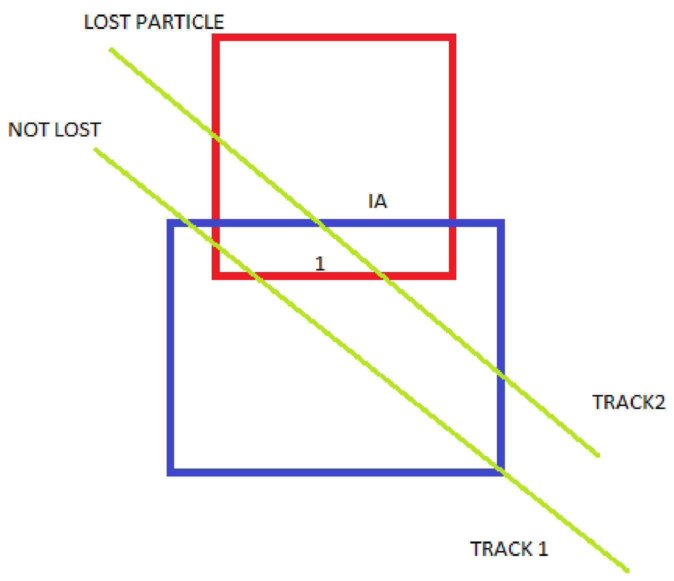


Figure 2.1: Visual explanation of IA dependency

2.2 Derivation of the relationships between lost particles and geometrical errors

2.2.1 The Intersecting Geometry Error

As previously explained an intersection error consists in wrongly defined cells that overlap and generate an intersection zone as visualized in figure (2.1). From the very first tests it was clear that changing the volume of this intersection could have no impact on the particle lost ratio. This provides an insight of how MCNP works: when a particle enters a cell, the code memorizes the surfaces delimiting that cell and his cross sections and until the particle crosses one of these surfaces the code will 'think' and act as the particle was still in that cell. When the particle finally reaches one of the delimiting surfaces the code will look for another cell that has the very same surface as a delimiter: if it finds it everything is fine and the particle enters in the new cell, if it does not (due to a geometrical error) the particle get lost. This means that only the particles crossing the intersection area between the two cells will get lost.

In figure (2.1) is shown a clarifying example: in track 1 the particle crosses some not properly defined surfaces, but it enters and exit from the blue cell through well-defined ones, causing the code to simulate it entirely as it was always in the blue cell. In track 2 instead the particle enters in the red cell through a well defined surface, but exits from an erroneous one and so gets lost. It's useful to notice that in this example both particles crosses surface 1, which is an erroneous one, but only particle 2 gets lost, and this happens because for a particle to get lost it must cross the intersection area (IA) first[1]. That explains for example why increasing the interference volume but leaving untouched the IA produce no change in the lost particle rate, while increasing the IA results in a growth of it.

Another important consequence of this is that a complete overlap of two cells (e.g. a sphere completely contained by a cube) would generate an intersecting error that would be undetectable with the lost particle method, because in that case particles would always enter

[1]The particle will get lost only when it reaches one of the ill-defined surfaces, and not when it crosses the IA, but if it didn't cross the IA before it means that it has entered and will exit through well defined surfaces, even if crossing ill-defined ones.

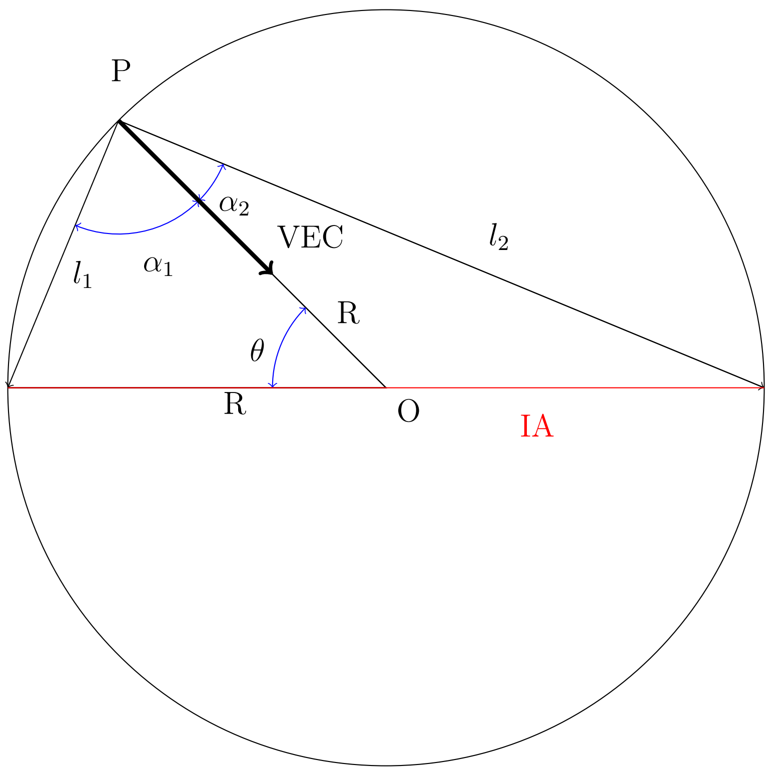


Figure 2.2: Maximum circular IA that can fit the source sphere

and exit only from the well defined surfaces bounding the bigger cell.

The other two parameters whose influence on the LPR had to be checked were the source sphere size and the error position in space. For a fixed IA, increasing the sphere source radius resulted in a drop of LPR value, while changing the IA position (with a fixed source sphere radius) appeared to not affect the LPR, in accordance with the assumption of a uniform and isotropic fluence inside the source sphere.

Heuristic derivation of the intersecting error law

There is a simple geometry configuration where is possible to semi-analytically derive a law relating the size of the IA, the source sphere radius and the LPR. Let's consider a circle built on a diameter of the source sphere as in figure (2.2). To integrate the cosine distribution of emission properly we would have to integrate α_1 on ϕ which is the azimuthal angle around VEC, but since in this way α_1 would be a function of ϕ, this would be too complicated to solve. So here an assumption is made: both α_1 and α_2 are considered at the same time and the integral over ϕ is calculated only on π instead of 2π, furthermore it is assumed that the sum of emission probabilities calculated for α_1 and α_2 remains constant for every ϕ. In other words this would mean that for example if we increase ϕ, the reduction in emission probability caused by the reduction of α_1 is exactly compensated by the augment due to the growing of α_2. If this assumption holds it means that the probability for a particle generated in P to hit the IA would be:

$$\int_0^{\pi}\left[\int_0^{\alpha_1}\frac{\cos\alpha}{\pi}\sin\alpha d\alpha d\phi+\int_0^{\alpha_2}\frac{\cos\alpha}{\pi}\sin\alpha d\alpha d\phi\right]=\frac{1}{2}\left[\int_0^{\alpha_1}\sin 2\alpha d\alpha+\int_0^{\alpha_2}\sin 2\alpha d\alpha\right]=$$
$$=\frac{1}{4}\left[-\cos 2\alpha\right|_0^{\alpha_1}+\frac{1}{4}\left[-\cos 2\alpha\right|_0^{\alpha_2}=\frac{1}{4}[1+(-\cos 2\alpha_1)+1+(-\cos 2\alpha_2)]=\frac{1}{2}(\sin^2\alpha_1+\sin^2\alpha_2) \tag{2.2}$$

where te trigonometric formula $\cos 2x = 1 - \sin^2 x$ was applied.

Using the sine law in triangles:

$$\begin{cases} \dfrac{\sin\alpha_1}{R} = \dfrac{\sin\theta}{l_1} \quad \Rightarrow \quad \sin\alpha_1 = \dfrac{R\sin\theta}{l_1} \\ \dfrac{\sin\alpha_2}{R} = \dfrac{\sin(\pi-\theta)}{l_2} \quad \Rightarrow \quad \sin\alpha_2 = \dfrac{R\sin\theta}{l_2} \end{cases} \tag{2.3}$$

Using the Carnot theorem:

$$\begin{cases} l_1^2 = R^2 + R^2 - 2R^2\cos\theta = 2R^2(1-\cos\theta) \\ l_2^2 = 2R^2(1-\cos(\pi-\theta)) = 2R^2(1+\cos\theta) \end{cases} \tag{2.4}$$

Inserting equations (2.4) and (2.3) in the master equation (2.2) results in:

$$\frac{1}{2}\frac{R^2\sin^2\theta[2R^2(1-\cos\theta+1+\cos\theta)]}{4R^4(1-\cos^2\theta)} = \frac{1}{2} \tag{2.5}$$

That means that each particle generated on the spherical source will always have a 50% chance to hit the considered IA independently from its position, that is, the expected lost particle rate (from now on ELPR) will be 0.5.

Heuristically it could be supposed that if this result is true, the general lost particle rate could be found relating a generic IA and the maximum IA previously discussed, something like:

$$\boxed{ELPR = \frac{1}{2} \times \frac{IA}{IA_{(\max)}} = \frac{1}{2} \times \frac{IA}{\pi R^2}} \tag{2.6}$$

This law perfectly fits the results discussed in the next section, but surely a more formal derivation should be produced.

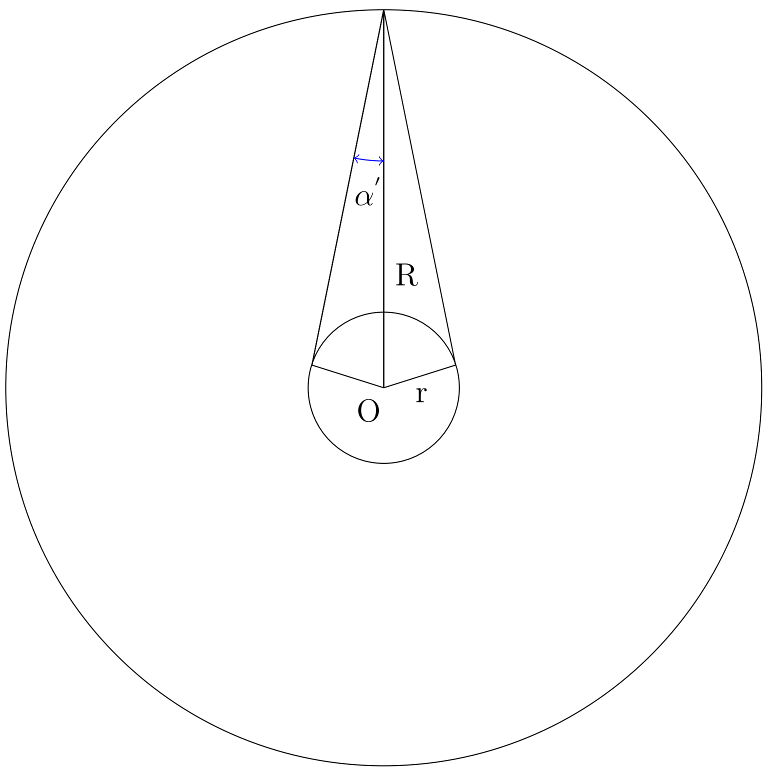


Figure 2.3: Scheme of the simplified void geometry

2.2.2 The Undefined Geometry Error

As previously stated an undefined geometry error is not created by the overlap of existing cells, but from the absence of them. The main difference between intersecting and undefined error is that in the former a particle will get lost only if crossing the IA first, instead, for the latter, is the total surface bounding the void region that matters since a particle crossing it would be immediately lost. That means for instance that an undefined geometry error can be always detected, even if completely contained by other cells.

Heuristic derivation of the undefined error law

Using the cosine distribution it is possible to derive a relationship between the void volume surface, the source sphere size and the LPR for a simplified case as shown in figure (2.3). In this case, where a spherical void error placed in the source origin is considered, the probability for a particle to get lost will be:

$$\int_0^{\alpha'}\int_0^{2\pi} S_0 \frac{2\cos\alpha}{2\pi}\sin\alpha d\alpha d\phi = S_0\int_0^{\alpha'}\sin(2\alpha)d\alpha d\phi =$$
$$\frac{S_0}{2}\left[-\cos 2\alpha\right|_0^{\alpha'} = \frac{S_0}{2}[1+(-\cos 2\alpha')] = S_0\sin^2\alpha' \quad (2.7)$$

where the relation $\cos 2x = 1 - \sin^2 x$ was used and the integration over ϕ was trivial thanks to α' remaining constant in this configuration.

This means that the probability for a particle to get lost only depends on α', which, thanks to the simplified geometry that is being considered, will be the same for every source particle and only depends on the size of the 2 spheres since $\sin\alpha = \frac{r}{R}$. In order to find the expected lost particle rate the probability found in eq.(2.7) will have to be integrated on the source's surface and then divided it by the total number of simulated particle:

$$ELPR = \frac{N_p}{4\pi R^2}\frac{r^2}{R^2}\cdot 4\pi R^2 \cdot \frac{1}{N_p} = \frac{r^2}{R^2} \tag{2.8}$$

A more general law could be extrapolated combining this result with an approach similar to the one proposed in section 2.1 for the intersecting error law based on the following considerations:

- The maximum undefined volume that can be generated in the source sphere is as big as the sphere itself.
- For such an error the probability for a particle to get lost will trivially be 1.
- It is reasonable to assume that the law will be a function of the ratio between the maximum error surface and the target error surface.

The following is the combination of the previous considerations:

$$\boxed{ELPR = 1 \times \frac{\text{Void Surface}}{4\pi R^2}} \tag{2.9}$$

which in the simplified case discussed before reduces indeed to:

$$ELPR = \frac{4\pi r^2}{4\pi R^2} = \frac{r^2}{R^2} \tag{2.10}$$

Law (2.9) perfectly fits the MCNP tests described in the next subsection, but will surely need a more formal derivation.

2.3 MCNP validation tests

In the previous section semi-analytical relationships were found relating errors size, LPR and source size. In this section are reported the results of all the different experimental test that were performed in order to validate them.

If not otherwise specified each MCNP test was conducted creating a block of 99 slightly different examples where one or more parameters were progressively changed. In order to this, an ad hoc library was created in python that contained classes useful to create cubes, spheres and sources in terms of MCNP input form and to perform basic operations on them like translating them in space or drilling holes in them. Moreover, also the simulations running and post-processing was automatised using python scripts. The default source sphere size had a 250 cm radius while the default particle histories number was 10^6 for each run.

2.3.1 MCNP test: Intersecting Geometry Error

As anticipated, the law derived at the end of previous subsection perfectly fit all the implemented tests' results. Hereafter those tests are briefly described:

1. Using the default source, a rectangular IA placed in the sphere's origin was progressively increased in size.
2. Considering a fixed rectangular IA ($100cm^2$), the radius of the spherical source was progressively increased from $16cm$ to $114cm$.
3. Considering the default source and a fixed IA size ($400cm^2$), the IA radial position was progressively increased from $y = 1.7cm$ to $y = 168.3cm$. This particular test was conducted simulating 10^8 particles in order to achieve a better statistical precision.

A summary of the results is visualized in fig (2.4).

As it can be observed in figure (2.4c) and (2.4d) translating the IA causes the LPR to oscillate around the predicted value (indicated by the black line). This behavior is due to the statistical nature of MCNP solution, but anyway the percentage error, computed as the difference between expected and real LPR divided by the expected one, always remains below 1%.

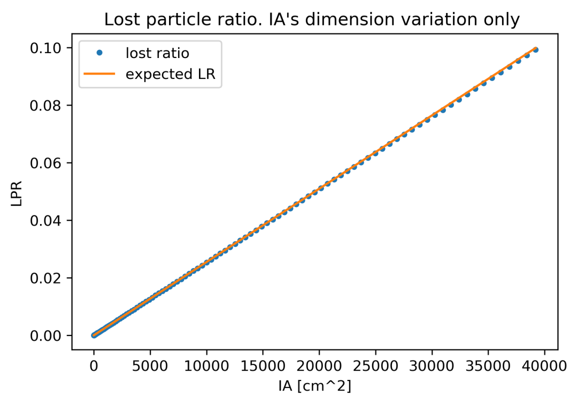
2.3.2 MCNP test: Undefined Geometry Error

As anticipated, the law derived at the end of previous subsection perfectly fits all the implemented tests' results. Hereafter those tests are briefly described:

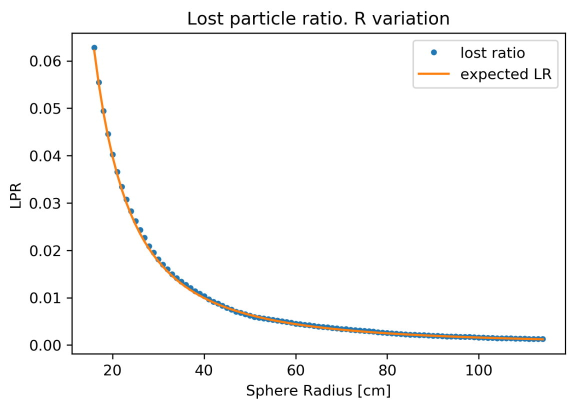
1. The radius of a spherical void error placed in the origin of the default source sphere was progressively increased from $0.2cm$ to $19.8cm$.
2. The same setup was reproduced but with a cubic void error instead of a spherical one.
3. Considering a fixed spherical void error size (radius=$10cm$), its radial position was progressively translated from $y = 2cm$ to $y = 198cm$. This test was run with 10^7 particles in order to achieve a better statistical result.

A summary of these results is visualized in fig (2.5).

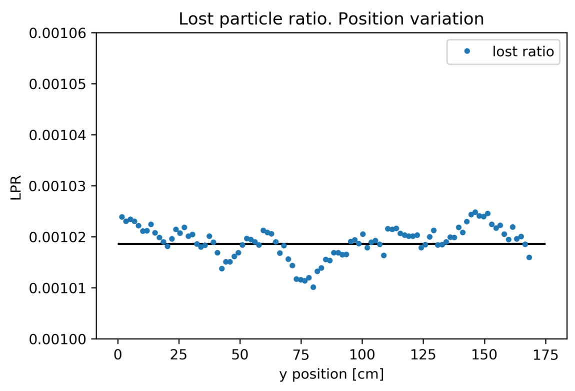
As it can be observed in figure (2.5c) and (2.5d) translating the void error causes the LPR to oscillate around the predicted value (indicated by the black line). This behavior is due to the statistical nature of MCNP solution, but anyway the percentage error, computed as the difference between expected and real LPR divided by the expected one, alway remains below 1.5%.



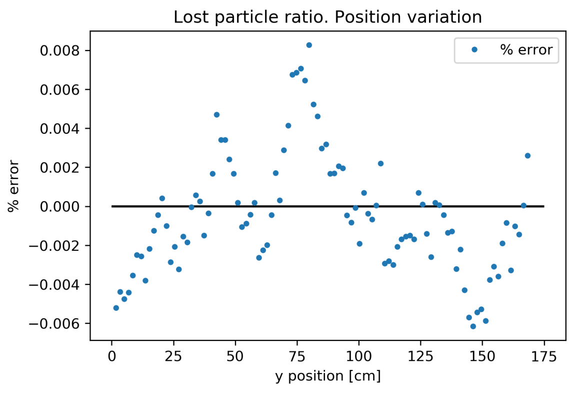
(a) *Fixed source and IA's position, changing IA's size*



(b) *Fixed IA's position and size, changing source radius*

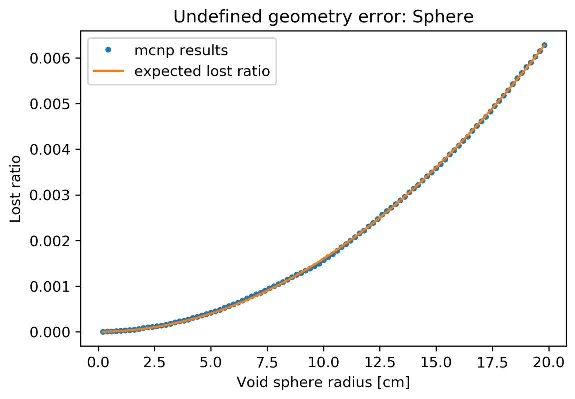


(c) *Fixed source and IA's size, translating on radial direction*

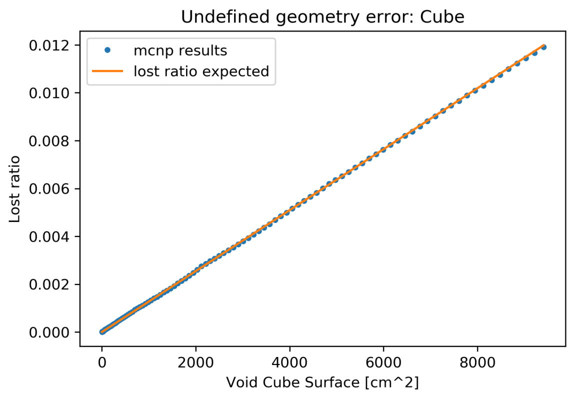


(d) *Relative error*

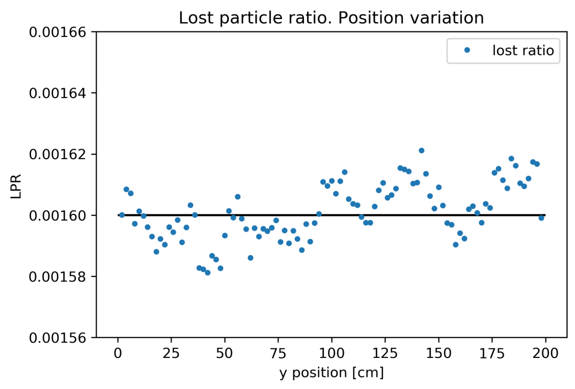
Figure 2.4: Summary of the results obtained with MCNP on intersecting geometry error tests



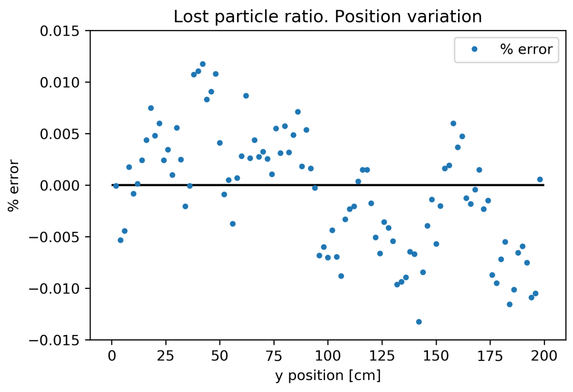
(a) *Default source sphere, changing void sphere size*



(b) *Default source sphere, changing void volume size*



(c) *Default source and fixed error size, translating on radial direction*



(d) *Relative error*

Figure 2.5: Summary of the results obtained with MCNP on undefined geometry error tests

2.3.3 Overall test results considerations

The test results allow to state quite confidently that the laws derived in the previous section are able to accurately relate LPR, source size and errors size. Additionally they confirm that these relationships are independent from the position or orientation in space of the errors. In fact, the deviations between numerical results and analytical predictions always remain under a few percentage points and it is possible to attribute this deviation to the statistical error associated with the MCNP solution. A more thorough investigation on the size of this deviation and its driving parameters will be conducted in a successive section of this chapter.

2.4 Spherical Source VS Parallelepipedal Source

It can be demonstrated that a cosine distribution allows to create a uniform and isotropic fluence not only using a spherical source, but also using a generic closed surface. This could be particularly helpful when trying to debug geometries that have a preponderant direction (eg. like a pipe) because it allows to fit those shapes in sources with a smaller surface area. For this reason an additional feasibility study on the use of a parallelepipedal source was conducted (from now on P-source). The creation of a parallelepipedal source appears to be a bit more complicated compared to the spherical one, indeed the input lines needed for his implementation are more than 50, while for a spherical source they were just a couple. In Appendix B of this report a more detailed explanation on the SDEF card required for the generation of such a source is provided.

2.4.1 Validation of P-source laws

Undefined Geometry Error

The first test conducted was to check if the undefined geometry error law derived for the spherical source was expandable also to a P-source, meaning:

$$ELPR = \frac{\text{Void Surface}}{\text{P-surface}} = \frac{\text{Void Surface}}{2ab + 2bc + 2ac} \tag{2.11}$$

where a, b, c are the parallelepiped dimensions.

The test was conducted simulating 10^6 particles with a spherical undefined error of radius equal to $1cm$ moved in 99 different positions inside a parallelepiped of dimensions $100cm \times 20cm \times 10cm$. From the results shown in fig.(2.7a,2.7b) it appears that the law is confirmed since the percentage error, except a few points, remains below 4% with a relatively small number of simulated particles.

Intersecting Geometry Error

Regarding the intersecting geometry error instead, the spherical source law resulted not directly applicable and a new derivation was necessary. It is always possible to consider a P-source as if it was created by a bigger sphere source like shown in fig(2.6): inside the region contained by the parallelepiped it is indistinctly possible to consider that the uniform and isotropic fluence is created by emitting N particles from the sphere or n from the parallelepiped. This is possible because the number of particle n that arrive on the parallelepiped surface are uniformly distributed thanks to the uniform fluence created by the sphere, and this means that if they are re-emitted by the parallelepiped with a cosine distribution we will have the same uniform and isotropic fluence inside the parallelepiped that would be created by the bigger sphere. Fortunately the spherical undefined geometry error law actually relates n and N:

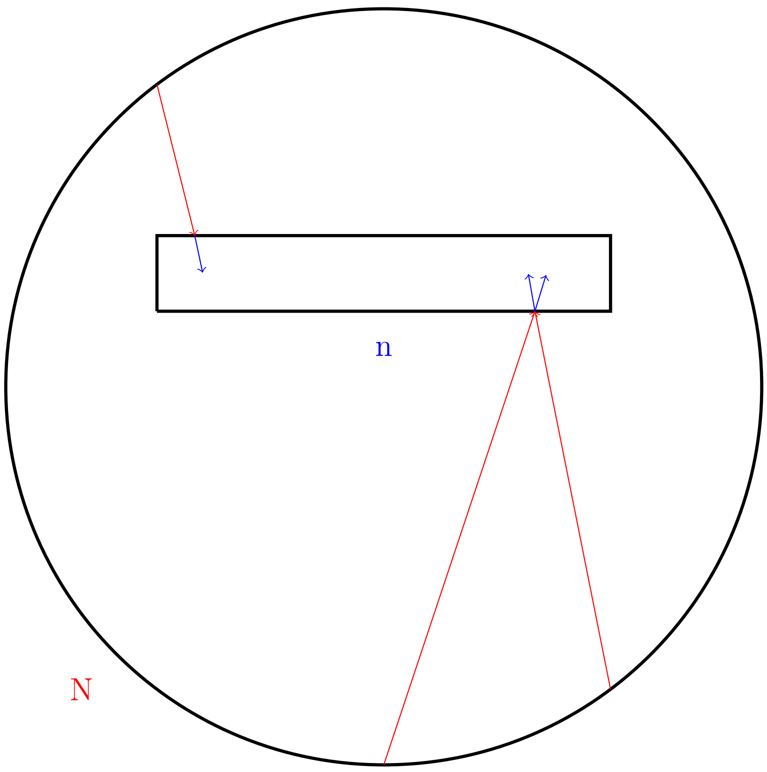


Figure 2.6: Spherical source generating a P-source

$$\text{ELPR} = \frac{n}{N} = \frac{\text{P-surface}}{\text{Sphere surface}} \quad \Rightarrow \quad N = n\frac{4\pi R^2}{\text{P-surface}} \tag{2.12}$$

where R is the sphere radius.

It only remains to apply the intersecting geometry error law for a spherical source:

$$\text{Lost Particle} = N\frac{1}{2}\frac{IA}{\pi R^2} = n\frac{4\pi R^2}{\text{P-surface}}\frac{1}{2}\frac{IA}{\pi R^2} \tag{2.13}$$

Dividing by n to get the expected lost particle rate it remains[2]:

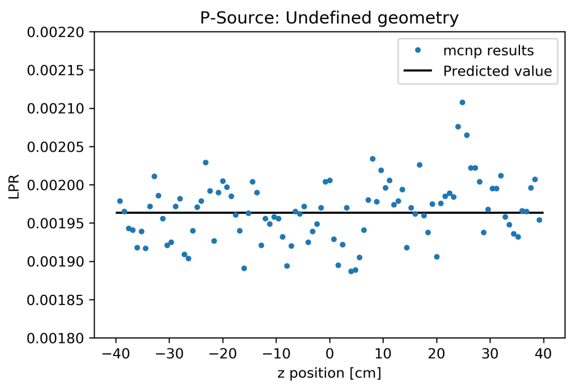
$$\text{ELPR} = 2 \cdot \frac{IA}{\text{P-surface}} \tag{2.14}$$

In order to verify these results a test was conducted simulating 10^6 particles with an intersecting geometry error presenting a rectangular IA shape of $64cm^2$ moved in 99 different positions inside a parallelepiped of dimensions $100cm \times 20cm \times 20cm$. From the results shown in fig. (2.7c,2.7d) it appears that the law is confirmed since the percentage error, except a few points, remains below 2% with a relatively small number of simulated particles.

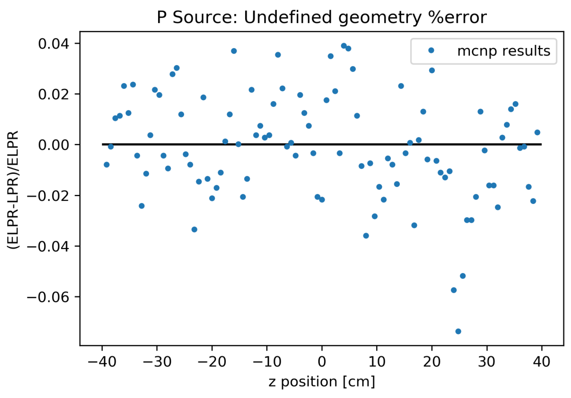
2.4.2 The advantage of a P-Source

The main advantage of a P-source is its augmented efficiency in finding smaller geometrical errors when the model shape has a predominant direction. This can be analytically demonstrated considering a model shape with two characteristic dimensions: a length L and a depth

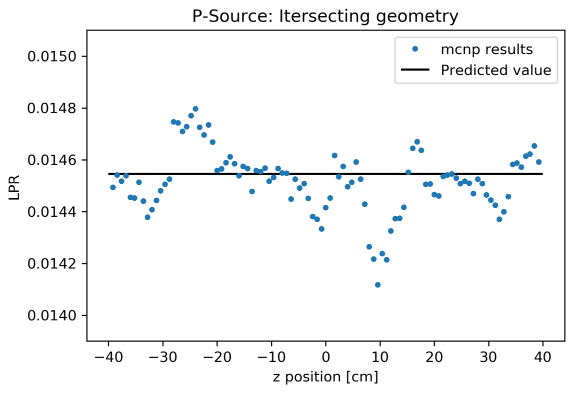
[2]equation (2.14) suggests that probably a more general law relating an intersecting volume error and generic source shape should be $\text{ELPR} = 2 \cdot \frac{IA}{\text{Source Surface}}$ that in the specific case of a sphere reduces indeed to $\text{ELPR} = \frac{1}{2} \cdot \frac{IA}{\pi R^2}$



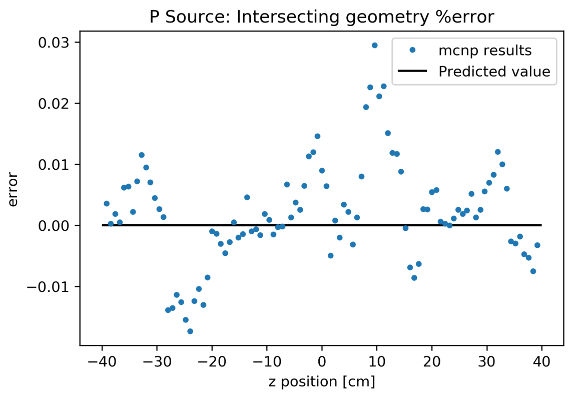
(a) *Fixed source and error size, changing error position*



(b) *Fixed source and error size, changing error position*



(c) *Fixed source and error size, changing error position*



(d) *Fixed source and error size, changing error position*

Figure 2.7: Validation test for the P-source

D such as his aspect ratio would be $\ell = \dfrac{D}{L}$. In this configuration the smaller parallelepiped that can include this shape will be of dimensions $D \times D \times L$, while the smaller sphere will have a radius equal to half the parallelepiped diagonal, that is $R = \sqrt{L^2 + 2D^2}/2$.

The sources surfaces will than be:

$$\begin{cases} A_p = (2D^2 + 4DL) \\ A_s = 4\pi \dfrac{L^2 + 2D^2}{4} \end{cases} \tag{2.15}$$

Using the laws derived in section (2.4.1) and (2.2.2) it is possible to calculate what will be the smaller undefined geometry error detectable once the source size and a LPR have been chosen:

$$\begin{cases} \mathrm{VSP_{min}} = A_p \cdot LPR & \text{min void surface using a P-source} \\ \mathrm{VSS_{min}} = A_s \cdot LPR & \text{min void surface using a Spherical source} \end{cases} \tag{2.16}$$

and the same thing can be done for the intersecting geometry error with the laws derived in sections (2.2.1) and (2.4.1):

$$\begin{cases} \mathrm{IAP_{min}} = \dfrac{1}{2} A_p \cdot LPR & \text{min IA using a P-source} \\ \mathrm{IAS_{min}} = \dfrac{1}{2} A_s \cdot LPR & \text{min IA surface using a Spherical source} \end{cases} \tag{2.17}$$

then defining η as the ratio between the minimum errors size detectable by a parallelepiped and a sphere it is possible to observe that:

$$\eta = \frac{\mathrm{VSP_{min}}}{\mathrm{VSS_{min}}} = \frac{\mathrm{IAP_{min}}}{\mathrm{IAS_{min}}} = \frac{(2D^2 + 4DL)}{\pi(L^2 + 2D^2)} \tag{2.18}$$

Dividing everything by L^2 the only variable that remains is the aspect ratio:

$$\eta = \frac{2\ell^2 + 4\ell}{\pi(1 + 2\ell^2)} \tag{2.19}$$

The graphic of this function is reported in fig (2.8): it can be seen that the P-source can detect a 40% smaller error if the shape model that is being checked for lost particle is cubic, while the detected error is more than 90% smaller in case of $\ell \simeq 0.1$.

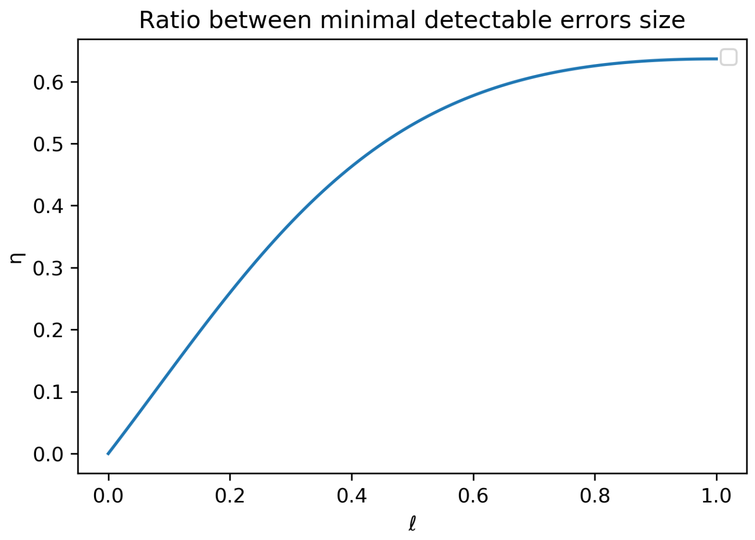


Figure 2.8: Ratio between the minimal error size that can be detected with a parallelepiped and a sphere

2.5 Convergence to the Expected Lost Particle value

Since the MCNP solution is a statistical one, once all the laws were derived it emerged the need to evaluate how the different sources were converging towards the ELPR, in other words to understand which were the parameters influencing the relative error between the MCNP obtained lost particle rate and the predicted theoretical one.

2.5.1 P-source vs Spherical Source

The first thing to check was if there were any differences in terms of convergence between a spherical and a parallelepipedal source. In order to clarify this a test was conducted inserting the same undefined geometry spherical error (radius 1cm) into two fixed size sources: a sphere with radius equal to $50cm$ and a parallelepiped of dimensions $100cm \times 20cm \times 10cm$. The test was composed of 18 examples where the number of simulated particle was increased progressively from 150000 to 147789188. Looking at the results shown in fig.(2.9) it appears that the source sphere had a slightly worse performance, but this was probably due to the fact that its surface was a bit bigger, meaning that there are no appreciable differences between how the two sources converge to the ELPR. Anyway this graphic and other less structured tests that were not included in this report suggested that the key convergence parameter was neither the source shape or size nor the nps value, but the absolute number of lost particles instead.

2.5.2 Lost particles dependency

This lost particles' dependency could be actually intuitive: the more particles are expected to be lost, the better will be the statistic result of a stochastic simulation. For example if ELP value is 1 and, once the simulation is run, the resulting number of lost particle is just one more than the expected, this will be already an error of 100%.

To verify this idea a more complex test was conducted and hereafter its creation is described:

- Two sets of absolute expected lost particles values were generated dividing the $[10 - 10^3]$ and $[10^3 - 10^5]$ ranges in 50 equally spaced points each.
- For each value of expected lost particles the simulation was run with NPS equal to 10^6, 10^7 and 10^8.
- For each NPS a fixed spherical source was considered with radius R equal to 100cm for 10^6 and 10^7, but only 50cm for 10^8. This was done to try to show that the source size was not a factor, anyway a more specific test about this topic was conducted successively.
- In each input a spherical undefined geometry error was inserted and its radius was calculated as[3]:

$$r = \sqrt{\frac{LP}{NPS}} R \qquad (2.20)$$

allowing to be sure to have the desired expected lost particles value in each example.

In fig.(2.10) are summarized the results obtained from these 300 examples, below are some considerations about it:

1. From figures (a) and (b) it seems confirmed that the relative error on a ELPR prediction does not depend directly on the lost particle rate, but on the predicted lost particles value instead. Indeed changing the NPS did not significantly affected the results.

[3]This is just a straightforward application of the undefined geometry error law

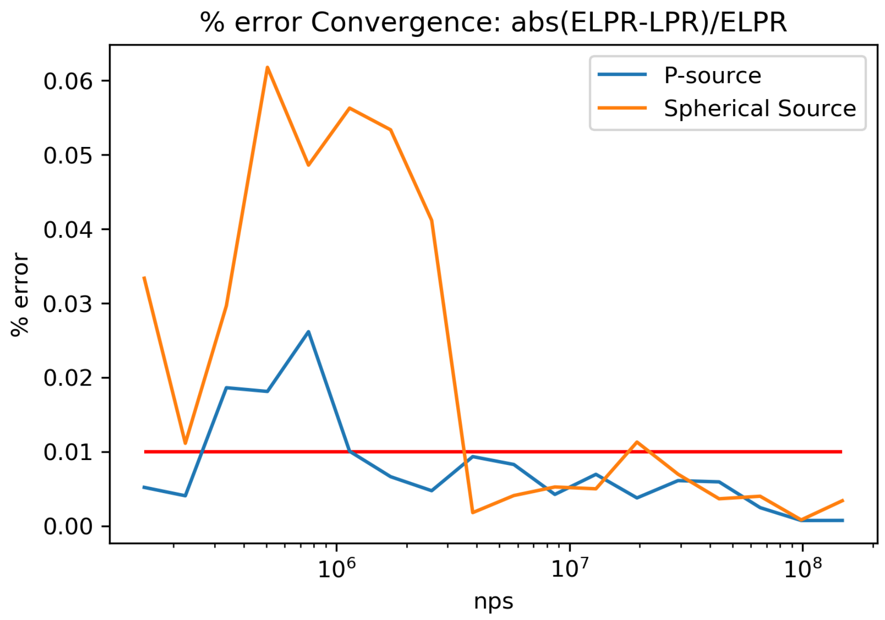


Figure 2.9: Convergence test result for a p-source and spherical source.

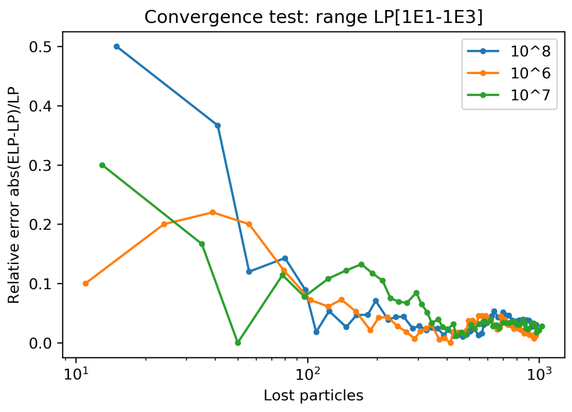
2. Source size appears to not have a massive effect on the convergence to the theoretical value since the NPS=10^8 results, which were obtained with a smaller source, are comparable with the other two.

3. Figure (c) suggests that is reasonable to expect a relative error on the LPR prediction below 10% if the expected lost particles value is around 10^2, while it can go under 1% once the same value is over 10^4. For higher numbers of lost particles the prediction rapidly converge to the theoretical solution, but to get these amount of lost particles the errors size or the NPS value would have to be so high that they would be unrealistic for normal application on lost particles debugging.

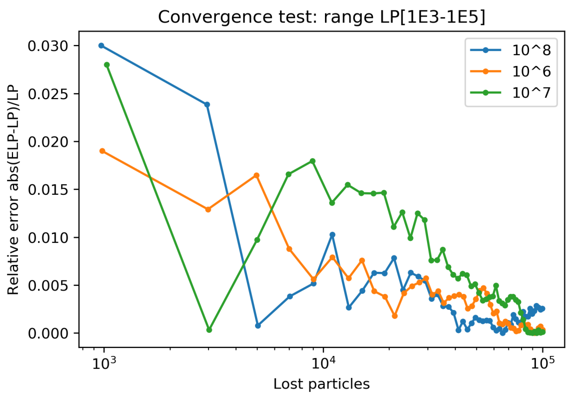
2.5.3 A more thorough check on source size influence on convergence

As previously anticipated an additional test was performed to better demonstrate that convergence does not depend on source size. This independence can be better understood with an example: two spherical sources with radius $R_1 > R_2$ are considered. If the same NPS value is used for both sources, source 2 will clearly generate a more uniform and isotropic fluence. However if inside the two sources is inserted an error whose size is calculated in such a way that the absolute number of expected lost particles is the same for both cases, this would mean that error 2 will be smaller than error 1. This explains why the source size does not affect the convergence: source 2 will actually generate a better fluence than source 1 at any given space scale, but source 2 will have to be confronted with a smaller error, meaning that it will have to operate at a smaller scale than source 1, causing in the end both sources to have the same precision. That is, the source size influence the prediction of the minimal error size detectable, while the absolute number of lost particle is an indicator of how much this prediction is reliable.

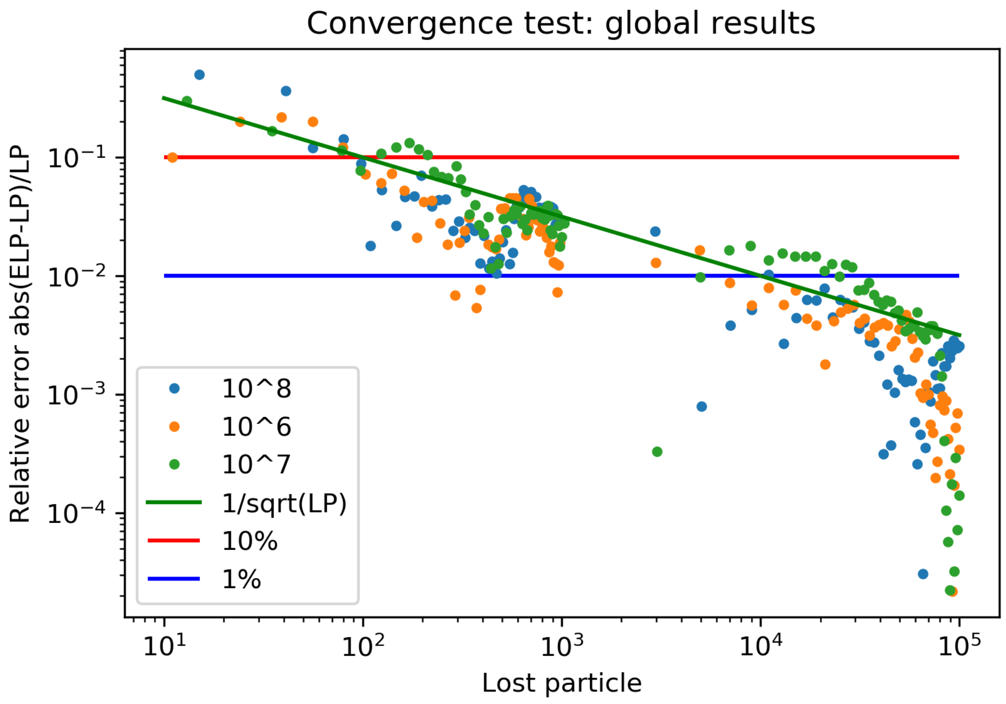
Two separate tests were constructed in similar fashion to the one presented in section 2.5.2 but this time the expected lost particles value were kept fixed (10^2 in the first test and 10^4 in the second), while the source radius was progressively increased. In fig.(2.11) are reported a further description and the results of the two tests: they suggest that, as expected, the relative error on the ELPR is roughly constant when the expected lost particles' value is constant, independently from the source size. Moreover it can be observed that the relative error value



(a) *Low LP value convergence, semilog graph*



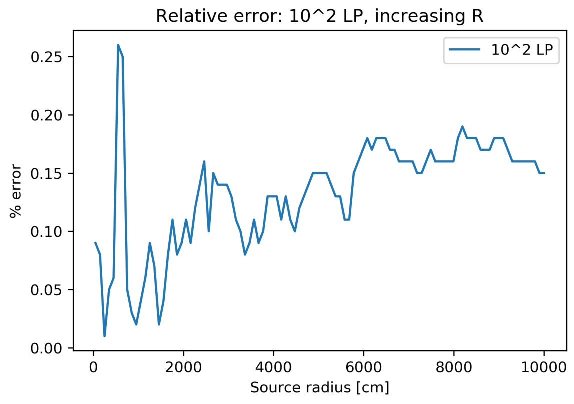
(b) *High LP value convergence, semilog graph*



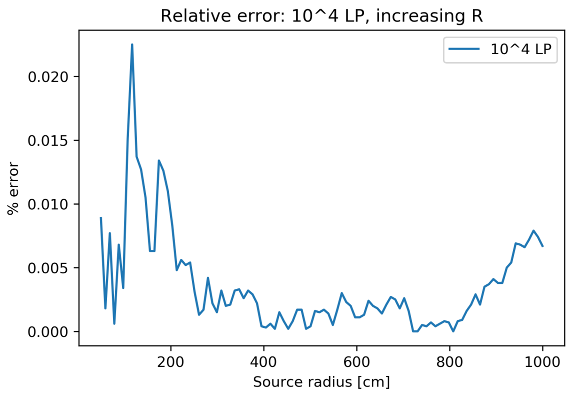
(c) *Global results, log-log graph*

Figure 2.10: Results summary of the convergence test based on lost particles value

is around 10^{-1} when the expected lost particle value is 10^2 and it is 10^{-2} when the same value is 10^4, confirming the results obtained in the previous section.



(a) $10^2 LP$, *source radius* $R \in [50cm, 10000cm]$, *100 points*



(b) $10^4 LP$, *source radius* $R \in [50cm, 1000cm]$, *100 points*

Figure 2.11: Results of tests conducted on the dependence between source size and convergence.

2.6 Conclusions

From the results contained in the previous sections it seems that 2 fairly easy relationships relating geometry error size, lost particle and source size exist:

$$\text{ELPR} = 2 \cdot \frac{IA}{\text{Source surface}} \qquad \text{Intersecting Geometry Error} \qquad (2.21)$$

$$\text{ELPR} = \frac{\text{Void Surface}}{\text{Source surface}} \qquad \text{Undefined Geometry Error} \qquad (2.22)$$

where these formulas remain valid for every shape and position of a geometry error inserted in a source with generic shape but able to generate an isotropic and uniform fluence inside its volume.

More useful is to rearrange these relationships to explicit what will be the smaller error size that would be statistically detected by a void run in MCNP once an acceptable LPR has been chosen:

$$\text{IA}_{min} = \frac{1}{2}(\text{Source surface}) \cdot \text{LPR} \qquad \text{Intersecting Geometry Error} \qquad (2.23)$$

$$\text{Void Surface}_{min} = (\text{Source surface}) \cdot \text{LPR} \qquad \text{Undefined Geometry Error} \qquad (2.24)$$

Hereafter are some important considerations:

1. All these tests were conducted for a single geometry error. In case of multiple errors, if they are reasonably small and far away from each other, the lost particles caused by one error should not have enough statistical impact to modify the fluence on each error's surface, that is the areas calculated with equations (2.23) and (2.24) will be the sum of all the geometry errors areas. Instead in case they were pretty close to each other an approximation could be done considering to have a void geometry error to include them all.

2. A parallelepipedal source can be more efficient than a spherical one, especially with low values of aspect ratio.

3. The intersecting geometry error is particularly problematic: the detection of it depends only from his IA, but its volume could be way bigger. The extreme case would be an intersection error completely contained by another cell: that would be undetectable.

4. The statistical error between calculated values and predicted ones seems to depend exclusively from the expected lost particles absolute number.

2.6.1 Possible standardized procedure for lost particle debugging

Based on these consideration a possible strategy to standardize the lost particle debug procedure would be to define a standard error size and a statistical error considered acceptable. Here below is an example of this approach if intersecting geometry errors are considered:

1. The maximum intersection area acceptable is set to $IA_{\max}$.

2. The LOST card is set equal to LP

3. The source geometry has to be chosen in such a way that its surface is minimized and it is able to completely contain the model that has to be checked.

4. The NPS value is set thanks to eq.(2.21) to:

$$NPS = \frac{LP}{2}\frac{\text{Source surface}}{IA_{\max}} \tag{2.25}$$

5. A void run is executed.

The void run can have only two possible outcomes, either the simulation is completed or it is interrupted. In the first case it would mean that the lost particle number was lower than LP, meaning that the geometrical error size was smaller than $IA_{\max}$. On the contrary an interruption would imply that size was bigger than $IA_{\max}$ and the model should be further debugged. Since the relative error between MCNP result and predicted value only depends on the absolute values of expected lost particles, the choice of LP is fundamental as it associates a statistical precision to the result. For example to chose LP=10^4 would mean that if the error's IA was about $IA_{\max}$ there would be roughly a 1% deviation between the number of lost particles found and the predicted one. Since the expected lost particles value is directly proportional to the geometry error's size, this would mean that a geometry error whose IA was 1% bigger than $IA_{\max}$ could still pass the void run test, while if his IA was 1% smaller it could fail it anyway. In other words from the chosen LP depends the tolerance accepted on the size of the geometry error that can be detected.

Numerical example

Thanks to this work it has been estimated that in the April 2018 release of the C-Model there was an equivalent IA of $17.6cm^2$. Starting from this, a way to choose an acceptable IA_{max} that a new part should have in order to be implemented in the C-model could be that the insertion/substitution should at least not increase the current IA value. This can be done assuming that the errors are equally distributed through the C-Model and then the IA_{max} admitted by a part should not be greater than the volume ratio between the part and the whole C-model multiplied by the current IA in the C-Model. If for example a port plug was considered, the following would be the IA_{max}:

$$\text{Volume ratio} \approx 0.0044 \tag{2.26}$$

$$IA_{max} = 0.0044 \cdot 17.6 = 0.078cm^2 \tag{2.27}$$

If then the lost card was set to 100, meaning that the tolerance on the error size would be 10%, the following would be the numbers of histories to simulate in order to pass the test with both a spherical source that could contain the whole C-Model, and one instead containing only the port plug:

$$\text{C-Model} \quad \Rightarrow \quad R = 20m \quad \rightarrow \quad NPS = \frac{100}{2} \cdot \frac{4\pi \cdot 2000^2}{0.078} \approx 3.2 \cdot 10^{10} \tag{2.28}$$

$$\text{Port plug} \quad \Rightarrow \quad R = 1.5m \quad \rightarrow \quad NPS = \frac{100}{2} \cdot \frac{4\pi \cdot 150^2}{0.078} \approx 1.8 \cdot 10^{8} \tag{2.29}$$

Chapter 3

Comparison between Unstructured Meshes and Fmeshses in neutronic analysis of the Electro-Cyclotron Upper Launcher Blanket Shield Module

3.1 Introduction

If the previous chapter was focused on problems that derive from CSG, this one instead suggests an alternative way to model MCNP geometry which is the application of Unstructured Meshes (UM). This approach is pretty new since it firstly appear in MCNP in 2009 with the release of MCNP6, but since then the UM capability has been significantly improved until it became fairly stable only with the last release of MCNP6.2 in 2018. The work continues on the path indicated by De Pietri's *et al.* work[4] which already investigated this feature applying it to a small section of the vacuum vessel of ITER tokamak using MCNP6.1. For this study instead, the new MCNP release was used and UM were applied on a more complex model in order to validate the UM feature on a proper ITER grade complexity analysis. The major upgrades made from version 6.1 to 6.2 of MCNP are the following:

- The possibility of inserting more than one mesh in the geometry was added.
- A bug causing the incorrect computation of the photon heating was fixed
- The geometry loading condition have been dramatically improved thanks to the creation of the .mcnpum file format. In all previous versions of MCNP6 the only accepted way by the code to load mesh informations was to attach a .inp file describing them. The problem was that this kind of format was not optimized for MCNP and it was serial. Together with version 6.2 was distributed a new utility called "convert" that is able indeed to convert a .inp file into a .mcnpum one. What happens is that the .inp file is preprocessed to extract all the informations needed by the code to run and everything is stored in the .mcnpum file. This is the same process carried out by the code every time the UM geometry had to be loaded in previous versions, but it is done in a highly parallelized way and just once. This means that the loading time for complex meshes can be reduced from hours to just tens of seconds when using a .mcnpum file instead of a .inp one.
- The .eeout file which is the standard MCNP output containing mesh results, has slightly changed its structure, meaning that all scripts interacting with this kind of file had to be adjusted.

The component chosen for the study was the Blanket Shield Module (also known as BSM) of the Electro-Cyclotron Upper Launcher. There will be four Upper Launchers installed in ITER tokamak upper port plugs (see fig 3.2) and their function will be to heat up the plasma and help to control its instabilities.

3.1.1 How are UM implemented?

Since now it has been repeated many times that Monte Carlo codes have to work with CSG, so how is it possible to switch to Unstructured Meshes? What happens actually is the creation of an hybrid model UM-CSG, where the mesh is inserted inside an ad hoc container called universe. The universe feature has been included in MCNP for a long time and it was created initially in order to group a set of cells, like for example the ones which constitute an assembly of a fission nuclear reactor core, in order to easily replicate or translate them. In the C-Model the universes have found a different application: ITER community refers to them as envelopes and they are universes used as placeholders for the different tokamak components. In this manner the C-Model is modular and its maintenance is highly simplified since a modification made on one of the so called envelope fillers (which is the collection of cells filling one envelope) will not affect the rest of the model integrity. A visualization of such a structure is reported in fig 3.1.

What is done when implementing UM is simply creating an *ad hoc* envelope were the mesh can be inserted and then the code is able to switch the way it computes the transport from CSG to UM when a particle enters it. It is very important for the envelope to not intersect in any way the mesh since this would cause lost particles.

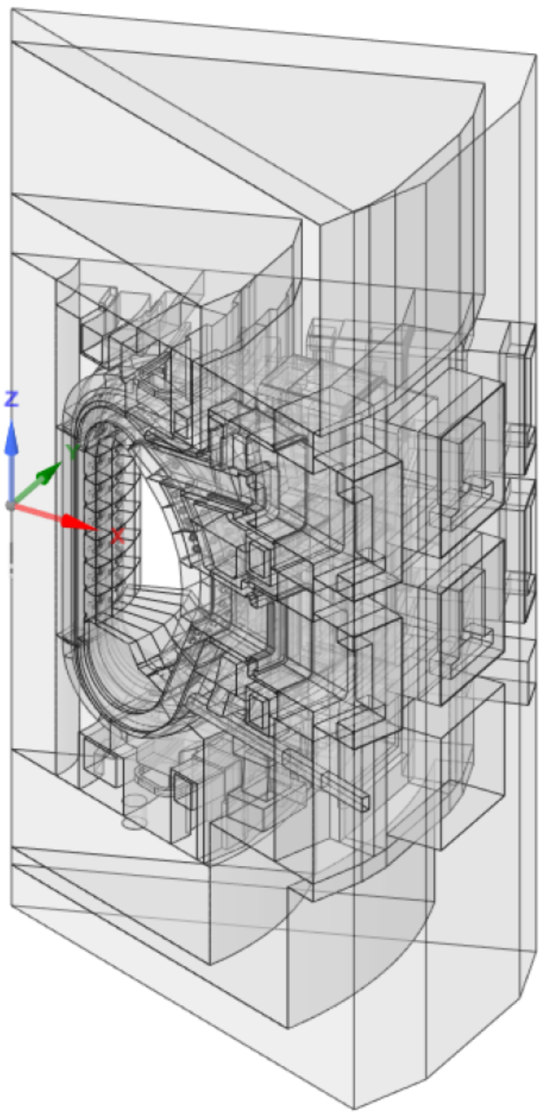


Figure 3.1: C-Model envelopes structure

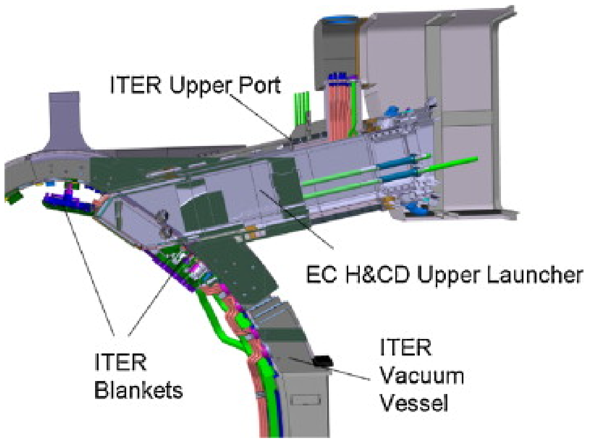


Figure 3.2: The upper launcher in the ITER tokamak

3.1.2 Why Unstructured Meshes?

The current standard strategy for carrying out nuclear analysis is the following:

1. *Model defeaturing*: using programs like SpaceClaim[2] the CAD model of the component to study is simplified in order to allow a smoother conversion to a Constructive Solid Geometry format (CSG).

2. *CSG conversion*: As already widely discussed, nowadays, with the aid of software like SuperMC, it is possible to automatically convert a CAD model to an MCNP-ready CSG model. However this process has a few limitation and, especially for large and complex models, is not errors free.

3. *Mesh Tally and results interpolation*: a structured Mesh Tally (MT) grid is superimposed on the model, the simulation is run and a radiation map is produced. As it is possible to see in fig(3.3) the main problem of a structured mesh is that it doesn't allow to perfectly "follow" the material borders, resulting in voxels that contain more than one material. This is a problem because it forces the user to make a decision on which approach should be used to calculate the results in each of this "border voxels". There are two strategies currently implemented: the implicit method, that tends to underestimate heating results, and the explicit one, that instead can lead to the arise of unphysical local peaks in the radiation map. These two procedures will be better described later on.

4. *Interpolation*: Once the radiation map is produced the results have to be interpolated on an unstructured mesh of the component in order to be properly visualized and used for other computational processes like thermo-fluid or mechanical analysis.

The complexity of this process and its intrinsic limit on reproducing faithfully the physical reality of the model near materials borders generate an interest on studying the feasibility of using unstructured meshes instead of structured ones:

1. Smaller need for defeaturing process since in this case its aim would only be to decrease the number of elements needed to mesh the component while there are no problems meshing for example splines or rounds.

2. No need for CSG conversion, since the mesh can be directly inserted in the model.

3. Better representation of the component's geometry.

4. Easier interpolation mesh to mesh, with the possibility in the future to use the same mesh for different analysis (like thermo-fluid ones) introducing no interpolation at all.

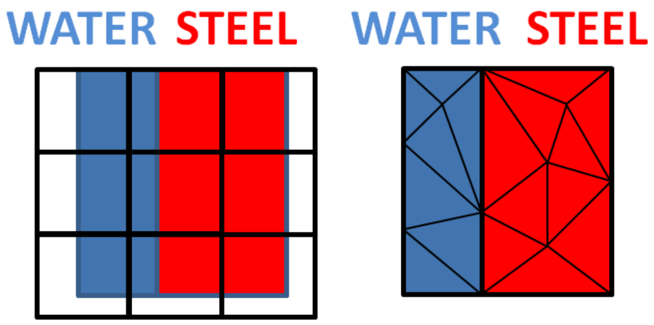


Figure 3.3: Structured Vs Unstructured mesh

It has to be specified at this point that the application of unstructured meshes is not free from limitations and problems, the hope though is that those are mainly due to the technique's novelty and computing power limits, both problems that could be resolved in the future.

3.2 Methodology

As introduced before, the focus of this work was the Upper Launcher's BSM which is displayed in fig(3.5). Since this was an experimental study the approach was to divide the BSM in smaller blocks and to start meshing them independently: this allowed to make tests on smaller and more manageable models and to better monitor the increasing complexity of the final one. Finally a compromise was found between meshing the bigger portion of the model possibly while still containing the estimation of preparing time and run time of the model. The final result is that three blocks were meshed reaching the plane showed in fig(3.5a), just after the bolts end. This was considered a sufficiently large portion of the BSM to analyze since, being the part that faces directly the plasma, it is the most critical one and requires the most precise data to design. Unfortunately the plane previously discussed cuts out the small mirrors circled in red in fig(3.5b), therefore the decision was to exclude those from the meshing process and just create a detailed CSG model of them to use in both the UM and CSG simulations. In fig(3.6) an exploded materials view of the final CAD model is displayed.

3.2.1 The MCNP model

In order to be able to compare structured and unstructured meshes analysis two simulations were run on two different models. The baseline for both was the C-model R180430[5], but, as an approximation, distant backscattering effects were neglected in order to speed up the calculation. To implement this, thanks to the C-model structure, all the envelope fillers far away from the upper port were switched off, while the blanket region left empty were set to zero importance. This caused all the particles entering in this region to die, resulting in a dramatic reduction of the simulation's run time. All of this is justified by the fact that the scope of this work is not to provide final design data for the upper launcher, but instead to compare the two methodologies already discussed on a ITER-complexity level analysis.

At this point, since the meshes in an MCNP input have to be inserted in a universe specifically created for them, from envelope 70 (the central upper port plug) were cut out the envelopes 701, 702, 703 and 704: the first 3 were filled with the 3 cuts of the meshed BSM while the last

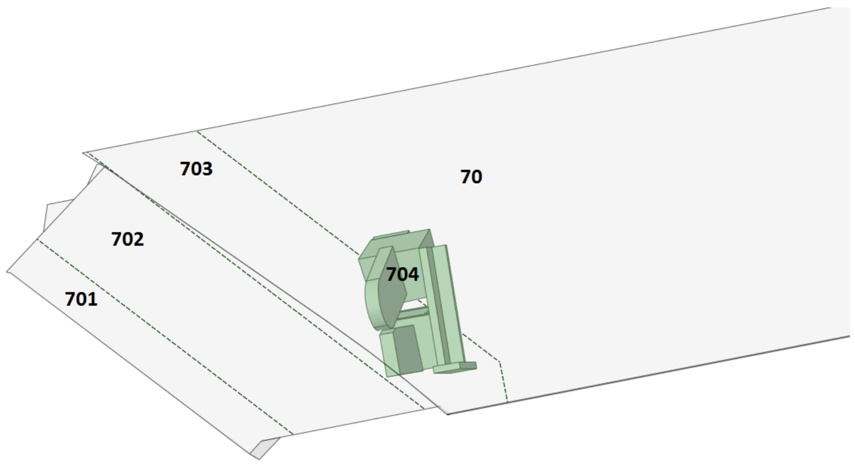
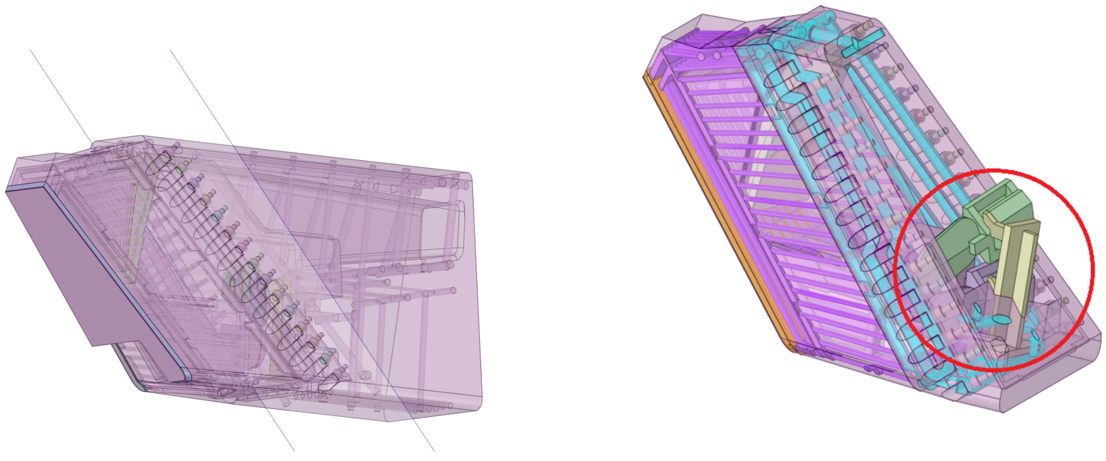


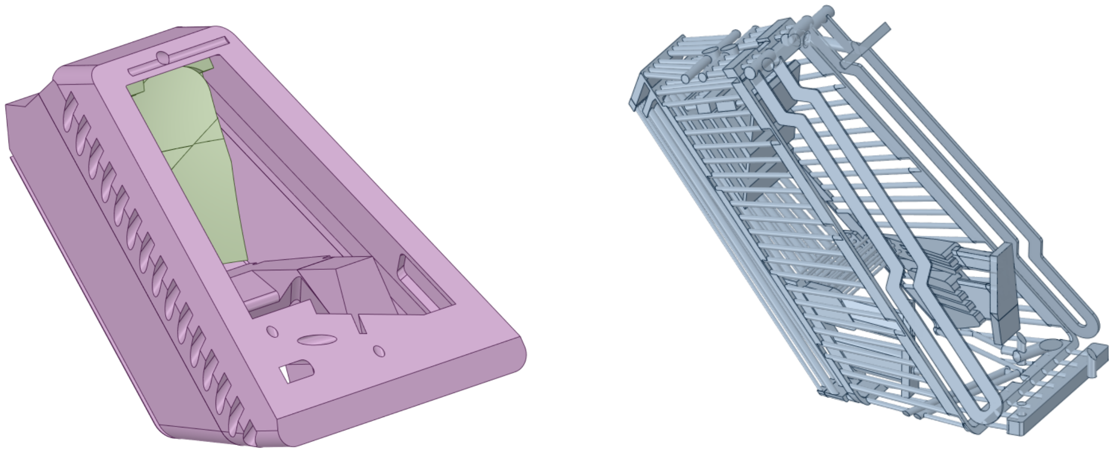
Figure 3.4: Visualization of the created envelopes.



(a) *Global model of the BSM* **(b)** *Meshed blocks and small mirrors (circled in red)*

Figure 3.5: The Bio Shield Module

one was created in order to host the small CSG model mirrors (fig 3.4). The BSM envelopes were obtained using the same planes used to create the three different parts in the first place, but since it is mandatory for the envelope to not crop in any way the UM geometry that it contains, all the meshes had to be translated 1mm in opportune directions to avoid coincidence. In addition to this, envelope 29 (Blanket module) had to be modified since a cutout was implemented to free a line of sight for the microwaves to reach the plasma as showed in fig(3.8). Moreover the dummy 70 filler was substituted with a semi-detailed model of the Upper Launcher. In figure (3.7) there is a visual representation of the final model built as described since here.


(a) *SS316L(N)-IG* **(b)** *Water*

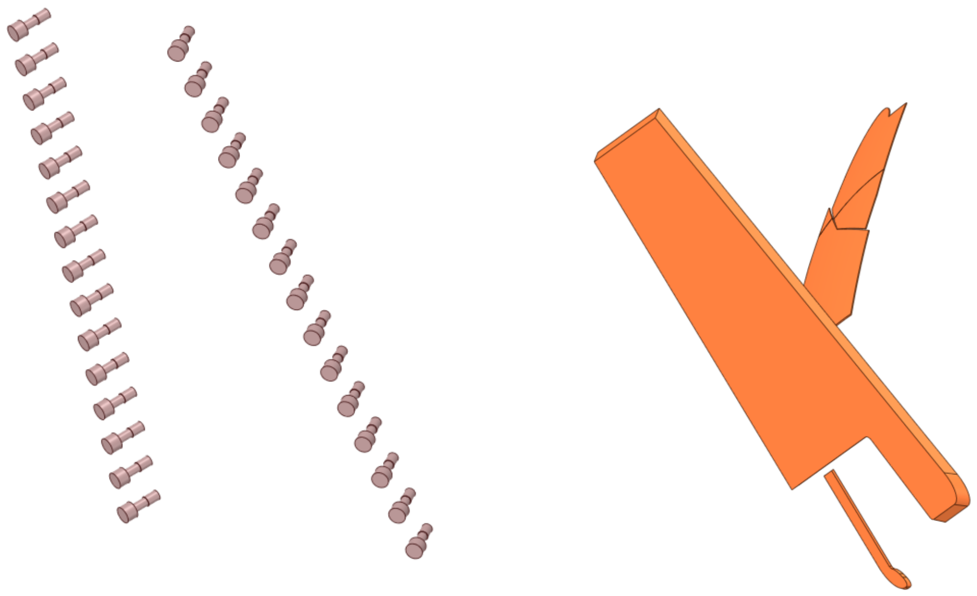
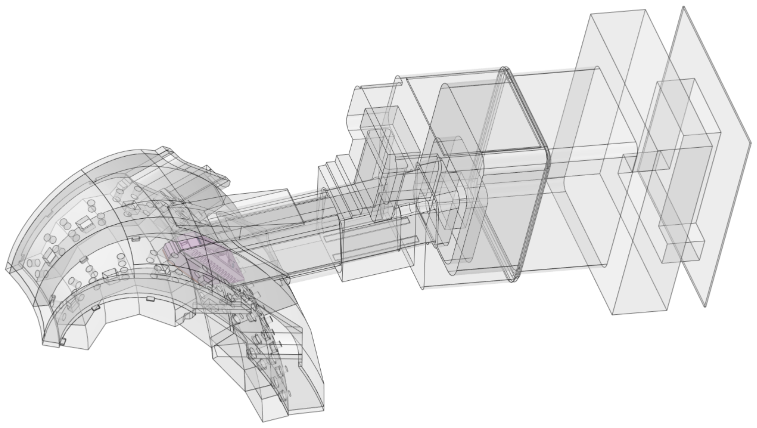
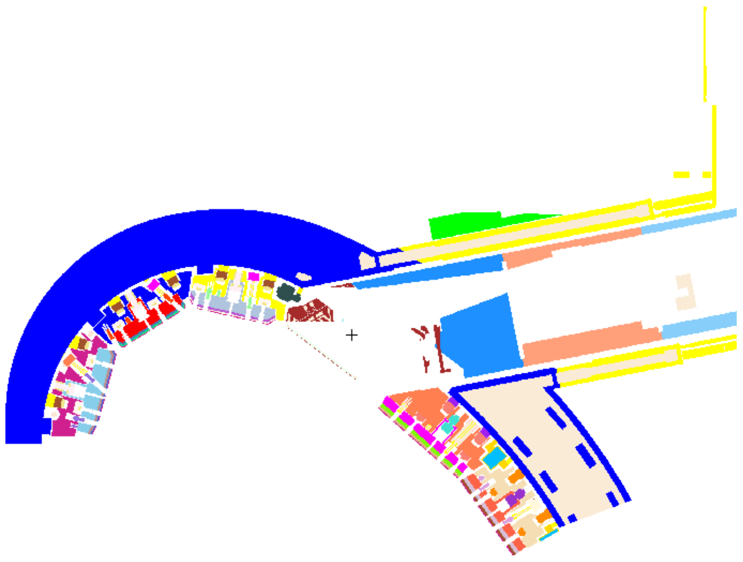

(c) *Steel 660* **(d)** *CuCrZr*

Figure 3.6: Exploded view of the meshed parts



(a) *Visualization of the envelopes that were filled. In pink it can be seen the BSM model.*



(b) *MCNP plotter view. Unfortunately due to a visualization bug the mesh is not entirely visible*

Figure 3.7: Global view of the model

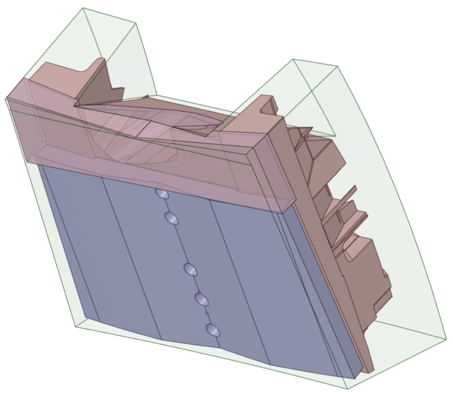
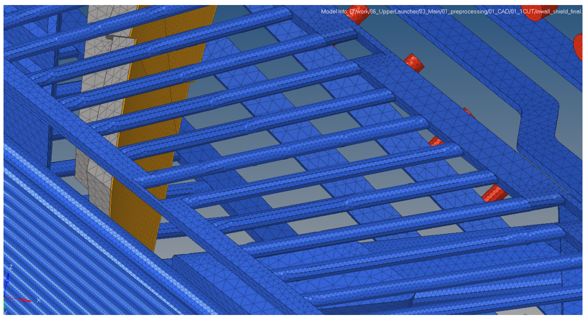


Figure 3.8: Blanket module with the cutout. In transparent green the original envelope 29, in pink the cut actually implemented and filled with void


(a) *Detail of the mesh*

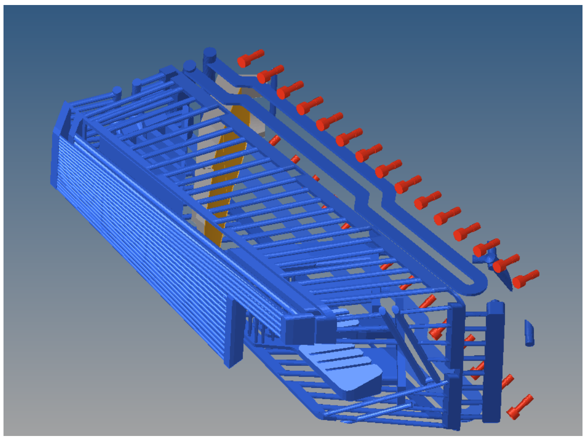

(b) *Visualization of the Mesh (No Steel)*

Figure 3.9: Visualization of the mesh

The Meshing Process

Three different meshes were built with the aid of HyperMesh[1] using only first order tetrahedra. This choice was derived from de Pietri's work[4] that shows how this kind of elements have the biggest advance in terms of simulation rate and memory usage. The total mesh was composed of 2 652 451 elements and presented a decreasing density of them while moving away from plasma. The meshing strategy was the following:

1. Cleaning and repairing of the geometry through SpaceClaim.
2. Mesh of the steel with the automatic tetrameshing tool of Hypermesh. This allowed to have a refining of the mesh only where needed, like near the channels, bolts or small rounds.
3. Extraction of the 2D faces of the steel mesh elements and projection of them to the neighboring materials in order to ensure connectivity.
4. 2D mesh of the remaining surfaces.
5. 3D tetramesh filling the 2D shell.

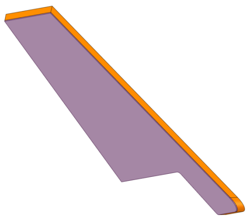
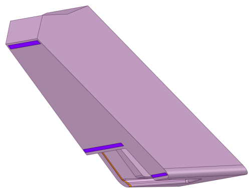
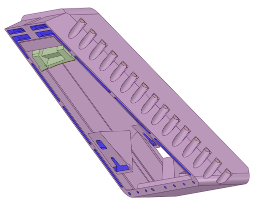
In fig(3.9) is showed a more detailed view of the mesh obtained.

At this point the mesh has been exported as a .inp file in Abaqus format, as requested by MCNP. It is important to notice that the procedure that leads to the creation of a working .inp file is really delicate, since even a small deviation from the procedure that was followed would prevent the geometry to load correctly when running MCNP. For this reason a more detailed pre-processing guideline to follow is described in section F.1. Moreover, with the new MCNP release (MCNP6.2), it is possible to create a preprocessed input file .mcnpum that already contains all the information needed by the code to start the simulation. This file is created preprocessing the previously modified .inp file with the “convert" MCNP's utility as described in the user guide for MCNP unstructured meshes [9]. In this way the conversion of the UM geometry can be done just once and with an highly parallelized procedure allowing to decrease the UM geometry loading time from tens of minutes/hours to seconds.

The CSG model

To have a fair comparison between UM and CSG a highly detailed CSG model of the BSM was produced. This required a repairing and defeaturing session on SpaceClaim of the initial CAD

Table 3.1: % volume deviation of the simplified CAD and of the Mesh with respect to the original one.

Universe	Block	N. Elements	Material	UM Deviation	CSG Deviation
701		801 597	SS316L(N)	0.00%	−0.31%
			CuCrZr	+0.79%	−0.33%
			Water	−2.65%	−0.44%
702		1 405 143	SS316L(N)	+0.54%	−0.15%
			CuCrZr	−0.24%	−0.24%
			Water	−2.07%	+0.36%
703		445 711	SS316L(N)	+0.31%	+0.04%
			S660 (Bolts)	−2.36%	+0.00%
			Water	−2.63%	+0.22%

model that was than converted to MCNP input format with the aid of SuperMC. Since a direct conversion of the model was impossible due to its complexity, a series of educated splitting of the geometry was performed to help with that and maintain low enough the number of “words" for each cell.

In table 3.1 are summarized the volume deviations of the CSG model and the UM one with respect to the original CAD model.

3.2.2 The Tally Section

In order to validate the UM results and to compare them with the CSG simulation ones, a series of tallies were implemented. Heating and flux were calculated both for neutrons and photons in the universes 70, 701, 702, 703 and 704. In addition the neutron and photon flux was calculated in all the blankets envelopes adjacent to the BSM (envelopes 23, 25, 26 and 29) to double check that there were no changes in the particles transport due to the use of UM.

As far as the tallying of the actual model is concerned the strategy was different depending on the utilized technique:

UM The tallying for an Unstructured Mesh in MCNP is pretty straightforward, the only thing to do is to insert the correct EMBEE card in order to generate the .eeout files for flux and heating for each mesh.

CSG In order to have a proper comparison 12 fmeshes were needed: 4 implicit calculating

Table 3.2: Different strategies for neutronic analysis

	UM	FMESH Implicit	FMESH Explicit
Transport	Simulated on geometry material	Simulated on geometry material	Simulated on geometry material
Flux	Computed on the geometry material	Computed on the geometry material + Average	Computed on the geometry material + Average
Heating	Computed on the geometry material	Computed on the geometry material + Average	Computed on user-defined material

neutron and photon flux+heating and then an additional two for each material to calculate the explicit heating of neutrons and photons.

In order to understand the difference between these two processes it is useful to go a bit more in depth on how an fmesh works. Since these mesh are superimposed on the model, the particles transport is simulated on the actual geometry, but since a single flux value has to be assigned to the voxel, it is clear that the result can be an average of fluxes computed on different materials. To get heating results instead, the flux has to be multiplied for energy dependent values contained in the material cross sections. There are two possible ways to do that:

Implicit: The heating is computed like the flux. This can cause under-estimation of the heating near borders of two different materials with distant nuclear properties. Indeed the resulting heating value associated with such a voxel will be the result of an average between highly different heating values since they strongly depend on the material cross section.

Explicit: The heating is computed on a specific material defined by the user. The analysis in this way is more conservative but an fmesh needs to be generated for each material in the model causing a more complex pre and post processing and additional running time.

In table 3.2 these different procedures are summarized.

All the fmeshes had the same structure:

- The dimension were 75cm x 90cm x 175cm. This allowed to cover entirely the BSM geometry while still keeping the mesh as small as possible.
- The steps on each dimension were respectively 100 x 120 x 233 for a total of 2 796 000 voxels. This was done to have more or less the same number of elements as in the UM simulation and to have almost cubic voxels (0.75cm x 0.75cm x 0.75cm = 0.422 cc).

In fig E.1, is reported a view of the fmesh position on the model.

All tallies were normalized to the standard ITER neutron power source of 400MW (500MW of fusion power). Since the flux results are given in #/cc per source particle this means that the flux had to be multiplied by the particle source rate:

$$\text{(Flux tally)} \cdot \frac{\text{Total Power/9(one sector)}}{\text{Neutron energy}} \cdot \text{(eV-J conv factor)} =$$
$$\frac{\#_p}{cc \cdot \#_{sp}} \frac{400/9 \cdot 10^6 \; J/s}{14.0791 \cdot 10^6 \; eV} \cdot 6.242 \cdot 10^{18} \frac{eV/\#_{sp}}{J} = [\text{(Flux tally)} \cdot 1.9730 \cdot 10^{19}] \frac{\#_p}{cc \cdot s} \quad (3.1)$$

while to convert from MeV per particle to W, the heating tallies were multiplied by:

$$\text{(Heat tally)} \cdot \text{(Source rate)} \cdot \text{(MeV-J conv factor)} =$$
$$\frac{MeV}{cc \cdot \#_{sp}} \cdot 1.9730 \cdot 10^{19} \frac{\#_{sp}}{s} \cdot 10^{19} \cdot 1.6022 \cdot 10^{-13} \frac{J}{MeV} = [\text{(Heat tally)} \cdot 3.1611 \cdot 10^{6}] \frac{W}{cc} \quad (3.2)$$

3.2.3 Simulations and Post-Processing

Unstructured Meshes Simulation

For UM simulation a series of variance reduction techniques were tried without success, since the analog run demonstrate to remain the one with better performances. In particular there were a normal directional bias toward the upper launcher and ADVANTG generated weight windows. The inefficiency reason of the directional bias is still unknown, while for ADVANTG the mediocre results were probably due to the peculiar model used for this work. In fact MCNP requires weight windows to be created in a region that includes the entire source. In order to do so, in this specific case, the majority of the domain on which ADVANTG was run resulted to be void, possibly interfering with the correct functioning of the code. Anyway the fact that a big section of the blanket was set to zero importance dramatically increased the simulation rate and allowed for a completely analog run with relatively small computer power.

In total 5e9 histories were simulated divided in three runs on MARCONI super computer:

- First run: 1e8 particles for 1h 11min on 20 nodes with 30 cores each (600 cores in total), the loading time for the geometry was roughly 22min.
- Second run: the first run was continued up to 2e9 histories in 5h and 36min on 20 nodes with 36 cores each (720 cores in total).
- Third run: the simulation was completed reaching 5e9 histories with an additional 5h 17min on 20 nodes with 42 cores each (840 cores in total).

This means that, excluding the geometry load time, on average the simulation rate was:

$$7426 \, \frac{histories}{CPU \cdot min}$$

The final .eeout files were converted in a single .vtk file with the aid of a python script in order to visualize the results on ParaView. This script had to be created since the native tool from MCNP (um post op), that is supposed to take care of this process, is currently bugged. Additional details on such a script can be found in appendix F.2. Finally the post processing analysis relied heavily on the use of Jupyter Notebooks.

CSG Simulation

The CSG simulation was effectuated in a single run of 5e9 particles that lasted 8h 24min (with a loading time of 24 min) on 20 nodes with 44 cores each (880 cores in total). This means that, on average, the simulation rate was roughly a factor 1.59 faster than the UM one:

$$11837 \, \frac{histories}{CPU \cdot min}$$

The post-processing though was more complicated since it involved to process and combine 12 different .meshtal files. Those were converted to the .vtk format using a C script and then interpolated through Paraview on the UM geometry to allow a confrontation between these 2 methods. The interpolation process was the following:

1. The cell values in the fmeshes were converted to point data using the Paraview filter “CellDataToPointData".
2. The point values were interpolated on the correspondent UM geometry using the Paraview filter “ResampleToDataset".
3. From the point data obtained in this way, the cell value for each element of the mesh was reconstructed using the Paraview filter “PointDataToCellData".

The datasets obtained in this way were than appended to their equivalent UM ones and exported in .csv format to be elaborated with the aid of Jupyter Notebook.

Computing resources

Loading and running time were already discussed in the previous sections, however there is an additional factor to take into account when discussing computing resources and it’s the RAM used to run the process. This is a key factor since currently in MARCONI the RAM is limited to 4GB for each core and can be increased only running with less CPUs on every node causing either longer simulation time or additional nodes allocation. To have an estimate of the necessary memory for both models, local runs on a single core were launched to manually check this value using windows task manager. It came to light that the UM simulation was lighter, requiring only 2.8GB versus the 3.3GB needed by the CSG one. A third model was evaluated consisting on the baseline of both models, that is, the common CSG fillers and cell tallies. The memory required by this baseline case was 1.5GB meaning that UM BSM and relative meshes weighted around 1.3GB while the CSG one plus the fmeshes (which were probably the heaviest part) were around 1.8GB, almost 40% more.

3.3 Results

3.3.1 Reference and Control Tallies

As explained in section 3.2.2 two kind of cell tallies were set up:

- Control Tallies: The flux in the blanket near the upper port was tallied to check that the use of UM do not change the particles transport.
- Reference Tallies: The total flux and heating in all the envelopes in the upper port plug were tallied to check how close the two simulations results were and to have a reference value for the integral result of structured and unstructured meshes.

The tables reporting the results of these tallies are table D.1 for the control ones and D.2 for the reference ones, but for readability purposes they have been moved in Appendix D. No substantial difference was observed between the two simulations on the control tallies as expected, while small differences were spotted on the reference ones and are highlighted in table 3.3. It can be noticed that in envelopes 70 (the rest of the upper port plug) and 704 (small mirrors) the difference are below 1% since the simulations were run with the exact same CSG model, there are instead bigger differences in the envelopes composing the BSM which are around $1 - 3\%$ with a peak difference of almost 10% in the photon heating of the first wall. This envelope is way smaller than the other two but it is important to notice that it is the most stressed zone of the BSM and it was the one with the higher refinement between all the 3 unstructured meshes. The bigger actor in the first wall photon heating is surely the thick CuCrZr layer and looking at the volume deviation it seems like there is roughly 1% more

Table 3.3: Reference Tallies deviations in upper port plug (UM-CSG)/CSG

Envelope	Neutron Flux	Photon Flux	Neutron Heating	Photon Heating
70	0.001220	-0.001523	-0.000766	-0.002668
701	0.014627	-0.014427	-0.015471	0.096309
702	0.010523	-0.013184	-0.021090	-0.012911
703	0.007734	-0.018967	0.025452	0.044690
704	0.012088	-0.009096	-0.002131	-0.009750
(701+702+703)	0.010944	-0.014835	-0.015902	0.007852

CuCrZr in the UM model compared to the CSG one. It is not sure though if this can be the only reason for such a spike in photon heating. Anyway if one looks at the total results for the BSM, that is the sum of envelopes 701, 702 and 703, the deviation remains more or less under 1.5% for all the tallies.

3.3.2 Global Results

Cleaning and Checking of the results

The first step of the post processing was to have a general look at the results to check that everything was in order. From this preliminary check two problems emerged from the UM simulation:

1. 10 mesh cells had an incredibly small volume compared to the other ones, in fact the biggest of them was still 7 order of magnitude smaller than the smallest cell in the rest of the mesh. This caused the result in these cells to be completely unreliable due to poor statistics and it was decided to eliminate them since they represented only the $6 \cdot 10^{-16}$% of the total volume.
2. A chunk of mesh cells (437) were erroneously calculated like they were CuCrZr instead of water as shown in fig(3.10). Since the chunk was peripheral and the volume affected was the 0.007% of the total one it was decided to not repeat the simulation and exclude these cells from the results analysis. This volume deviation indeed was way lower than the one already accepted between CAD and mesh volumes of the materials.

Error Analysis

The following thing to check were the statistical errors obtained in cell tallies, structured and unstructured meshes. Both the reference and control tallies had, as expected, a really low error oscillating between 0.01% and 0.02%. Among meshes results instead, the performance were different depending on utilized method and tallied value as showed in fig 3.11. As a general trend fluxes had lower errors then heating and neutron related values had lower errors then photon ones. All of this was expected since these results mirror the causality order of these

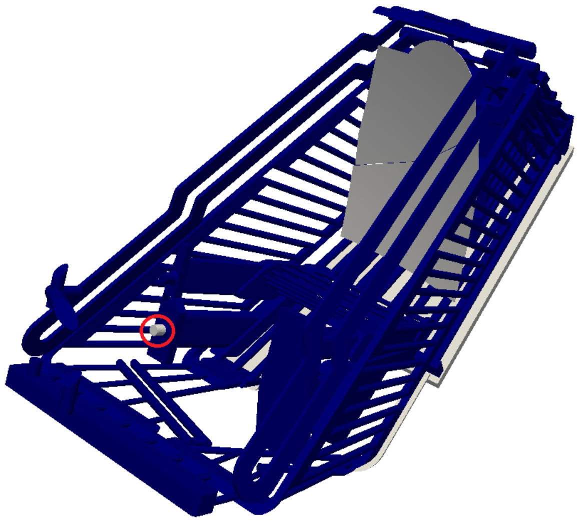


Figure 3.10: Circled in red the chunk of elements erroneously allocated to CuCrZr

Table 3.4: Comparison between reference cell tallies and integrated mesh results [W]

Value	Implicit			Explicit			UM		
	Reference	Result	% Dev	Reference	Result	% Dev	Reference	Result	% Dev
N heat	5.646e+4	5.237e+4	-7.241	5.646e+4	5.683e+4	0.654	5.5563e+4	5.5569e+4	-0.008
P heat	1.411e+5	1.273e+5	-9.784	1.411e+5	1.517e+5	7.517	1.4223e+5	14220e+5	-0.017

physical quantities: heating values depend on fluxes, while photons are generated by neutrons interacting with the material.

Next thing that can be noticed is that fmesh tallies had lower errors compared to unstructured ones with almost all the voxels below 10% error for every result. Nevertheless the difference was not too large, especially if the errors are considered in volume percentages where it can be seen that for UM 95% of the tallied BSM volume presented less than 10% error on every physical quantity.

Integral results: Implicit VS Explicit VS UM

In order to have an idea of the results reliability the values coming from the meshes were integrated on all the BSM geometry and compared to their corresponding reference tally since they are considered to be the most accurate. Indeed in each of these envelopes there is only void or material belonging to the BSM, meaning that every reference tally actually computed the heating of the BSM part they were containing. This means that cell tallies and integral mehses results should be exactly the same in ideal conditions: it's easy to understand why is that for an unstructured mesh, but this should be also the "aspiration" of a structured one once it is interpolated. The smaller the interpolation, or the bigger the fmesh refinement, the better will be the agreement between integral value and cell tally.

Hereafter are some consideration about the performances of these three methods which are also summarized in tab 3.4:

Implicit The under-estimation of the implicit fmesh was significant for both neutron and

Error	N flux	N heating	P flux	P heating
< 5%	96.02	46.57	63.38	33.68
< 10%	99.99	94.06	98.96	91.43
< 15%	99.99	98.71	99.93	98.96
< 20%	99.99	99.71	99.99	99.84

(a) *UM, percentage of cells*

Error	N flux	N heating	P flux	P heating
< 5%	98.98	63.67	85.21	59.92
< 10%	99.99	97.77	99.70	95.77
< 15%	99.99	99.67	99.99	99.69
< 20%	99.99	99.94	99.99	99.96

(b) *UM, percentage of volume*

Error	N flux	N heating	P flux	P heating
< 5%	100	88.51	98.89	86.21
< 10%	100	99.77	100	99.41
< 15%	100	99.98	100	99.953
< 20%	100	99.99	100	99.99

(c) *FMESH, percentage of voxels*

Figure 3.11: Percentage of cells/volume that presented a statistical error lower than defined thresholds for different nuclear properties results

photon heating reaching almost 10% on the photon one. This result was expected since almost all the voxels positioned on the BSM border will contain a considerable portion of void causing the calculated heating to be way under-estimated. This under-estimation can vary wildly depending on the fmesh resolution and on the complexity of the geometry.

Explicit There appeared to be good agreement between integral results and reference tally for the neutron heating, but a considerable difference instead (more than 7%) is observed for the photon one. This result is hard to explain with only the data provided so far and will need additional investigation in the following sections.

UM As expected there was almost a perfect agreement between mesh integral and reference tally results (deviation below 0.02%). This proves that indeed both the particles transport and heating are calculated in the unstructured mesh on the geometry material like they were a series of small cell tallies. The very small deviation registered should be attributed to the cells elimination discussed in section 3.3.2.

3.3.3 Local results

Cell volume comparison and interpolation

Before going into more details in local results a better understanding of the interpolation effect was needed. In particular it is probable that the ratio between structured and unstructured mesh cells and their relative position could have a big influence on interpolation efficacy. Previous studies have already shown how relative position between voxel and geometry can have huge effects on the results, while in this work the distribution of UM volume (plotted in fig 3.12) was analyzed.

The volume of each structured mesh cell was roughly 0.422cc while the average value for the unstructured ones was of 0.087cc meaning that on average the fmesh cells were almost 5 times bigger than UM ones, and it can be seen from the graph how actually the majority of UM cells had an even lower volume. This means that if some of the regions in space where under-sampled by the fmesh grid this could have caused huge local deviations between the methodologies and it had to be kept in mind while proceeding with the analysis.

Flux

As previously discussed the particles flux was calculated in the same way both for implicit and explicit fmeshes. In graph 3.13 is visualized the percent deviation between interpolated fmesh values and UM ones in each unstructured mesh cell. It shows that there was a quite good agreement between the different simulations since more than 97% of the cells had a deviation smaller than 10% for the neutrons flux and of 15% for the photons one. In figure 3.13(d) it is also presented the deviations of the average values of photon and neutron heating between implicit and UM for each material and it can be observed that generally this remains under 3% except for the bolts where it reaches a peak of around 5%. All of this shows that the transport of photons and neutrons is not affected significantly by small variations in the geometry or material. A peak difference in the bolts is reasonable since they are small objects, meaning that their mesh was more refined and a big surface/volume ratio. The last thing to notice is that it seems there was no clear over or under-estimation of the fluxes since the distributions reported in fig 3.13(a) and 3.13(b) seems to be more or less symmetric around a peak positioned relatively near to zero.

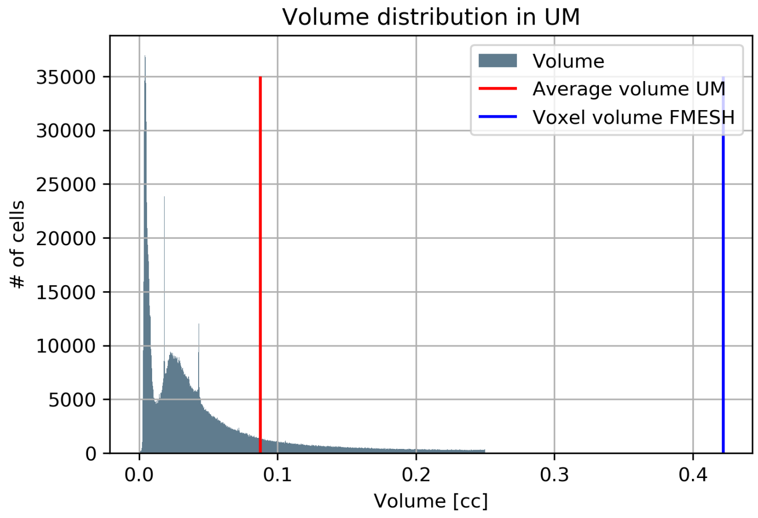
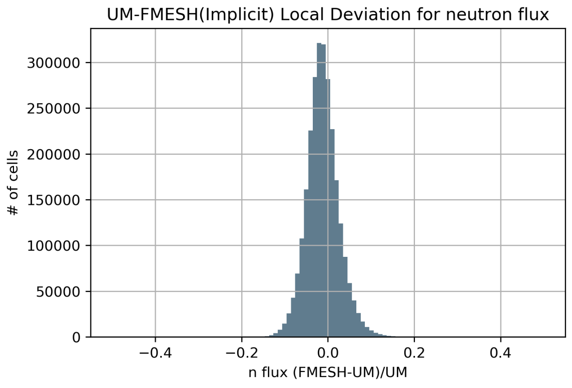
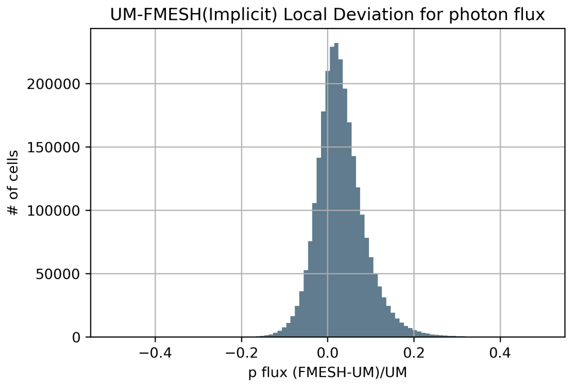


Figure 3.12: Volume distribution in unstructured mesh cells



(a) *Neutron flux, cell by cell*



(b) *Photon Flux, cell by cell*

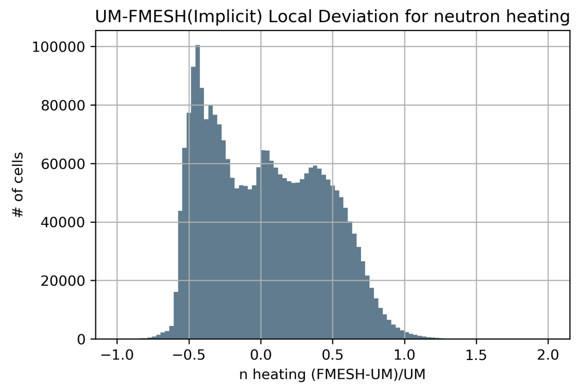
Deviation	N flux	P flux
$< 5\%$	0.810913	0.638466
$< 10\%$	0.984469	0.891206
$< 15\%$	0.998820	0.968422
$< 20\%$	0.999805	0.990224

(c) Ratio of cells under a fixed deviation

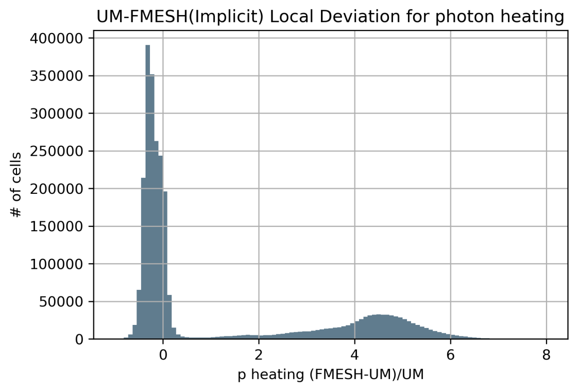
Material	avg N flux dev (%)	avg P flux dev(%)
Water	-2.6455	2.9636
S660	1.4662	5.5821
SS316L(N)	-0.8325	1.9345
CuCrZr	-3.2443	2.9223

(d) Percent deviation of the average values abs(CSG-UM)/UM

Figure 3.13: Local flux comparison results



(a) *Neutron flux, cell by cell*



(b) *Photon Flux, cell by cell*

Material	avg N flux dev (%)	avg P flux dev(%)
Water	-41.8031	424.3417
S660	-22.8798	-19.1959
SS316L(N)	18.9378	-17.1294
CuCrZr	41.3785	-32.2396

(c) Percent deviation of the average value abs(CSG-UM)/UM

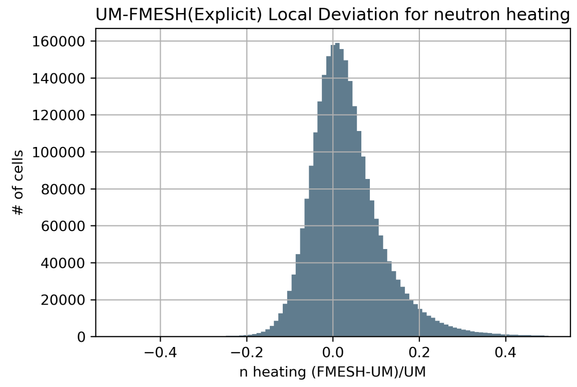
Figure 3.14: Local heating (implicit) comparison results

Heating: Implicit VS UM

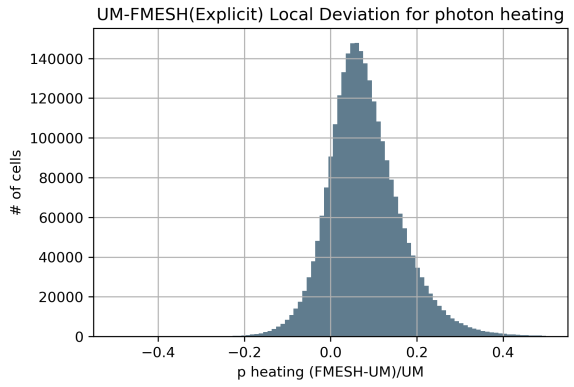
After the flux, heating local results were analyzed starting from comparing UM ones with the one obtained using the implicit method. From the beginning it emerged a huge discrepancy between them that is summarized in the graphs and tables of fig 3.14. This was probably due to the fact that the homogenization of different materials, especially if they have distant nuclear properties, often results in a gross approximation that strongly depends on geometry and mesh refinement. This idea is supported by fig 3.14 where was reported the deviation of the mean value for photon and neutron heating between implicit and UM results. Here a perfect example of the previously described discrepancy can be found looking at the water results, in fact in the model water is mainly distributed in relatively small channels encapsulated in steel or CuCrZr and the homogenization near the borders generates a huge underestimation of the neutron heating in the water. This happens because the contribute to it given by steel or CuCrZr is significantly lower than the water one, while the exact opposite is true for the photon heating which instead is way overestimated. Another example can be found looking at the bolts results: since they neighbor only with SS316 steel (which has similar neutronic properties) and void, both the neutron and photon heating were underestimated in this case because of the complete absence of heating in void.

Heating: Explicit VS UM

More coherent results were found when trying to compare cell by cell the heating values obtained with UM and the explicit method as summarized in fig 3.15. The general trend is to have an improvement in the fmesh performance compared to the implicit method and a way more symmetrical distribution of the deviation from the UM.



(a) *Neutron heating, cell by cell*



(b) *Photon heating, cell by cell*

Deviation	N flux	P flux
$< 5\%$	0.525070	0.337954
$< 10\%$	0.800960	0.629452
$< 15\%$	0.909014	0.815319
$< 20\%$	0.954842	0.915624
$< 25\%$	0.976474	0.962732

(c) Ratio of cells under a fixed deviation

Material	avg N flux dev (%)	avg P flux dev(%)
Water	-2.1996	6.6777
S660	7.1895	10.3849
SS316L(N)	3.5255	5.9061
CuCrZr	-1.1313	11.9450

(d) Percent deviation of the average value abs(CSG-UM)/UM

Figure 3.15: Local heating (explicit) comparison results

Table 3.5: Heating peaks on one cell [W/cc]

Material	n heating UM	n heating FMESH	p heating UM	p heating FMESH	n:(FMESH-UM)/UM	p:(FMESH-UM)/UM
CuCrZr	0.993882	0.844991	3.271116	2.681282	-0.149808	-0.180316
SS316L(N)	1.040252	0.975610	2.531820	2.388124	-0.062141	-0.056756
S660	0.217076	0.159666	0.747522	0.611654	-0.264471	-0.181758
Water	2.965323	2.471481	0.324553	0.267957	-0.166539	-0.174381

A series of considerations can be made:

1. Even if the agreement between UM and fmesh is better with explicit method there are still significant local variations between the result, indeed, as visualized in 3.15(c) only 63% of the cells present a photon heating deviation below 10%, while to include almost all the cells (around 96% of them) the percent deviation threshold have to be raised to 25%.

2. Fig 3.15(d) seems to suggest that there was an actual overestimation of the photon heating in the explicit fmesh compared to UM, in fact the deviation distribution appears to be symmetric, but around a peak that is shifted to around +7%. This is coherent with the result find in section 3.3.2 where it was observed a similar discrepancy between the explicit CSG integral result and correspondent reference tally.

Heating Peaks: Explicit Vs UM

At first, when the peak heating results for UM where checked, they presented values so high that they were considered unphysical. It was then decided to exclude from this analysis the cells that presented an error bigger than 30% on the photon heating. Neglecting these 238 cells seemed legit since they were considered unreliable and only represented the 0.00072 % of the BSM volume. After this operation the peak value for photon and neutron heating in each material was retrieved both for UM and explicit method: all the relative data is visualized in table 3.5.

The variations encountered were more acceptable compared to the implicit method ones, but still they were pretty big. For this reason an investigation was needed on the possible causes of this and it started with plotting only the cells with the higher values of heating thanks to the “threshold" Paraview filter. An example regarding the photon heating in CuCrZr is shown in fig 3.16. These cells, as expected, are all located in the first wall which is more exposed to the plasma, but the interesting thing is that it is clear how the cells majority present values ranging from 2.4 to 2.6 W/cc which is significantly lower then the maximum peak registered (3.27 W/cc), in fact just some isolated cells have higher values. This is additional proof that, even in high-flux regions, if the unstructured mesh cells are too small, they can lead to unphysical results probably caused by poor statistics. To test this idea, instead of considering the highest value in the dataset as the peak value, the 1000 cells with higher values were considered and the mean of those was taken as the peak value. In table 3.6 these new peaks are reported with an indication also of the ratio between 1000 and the total number of cells in each material. This was done to show that it is still possible to refer to these results as peak values since the

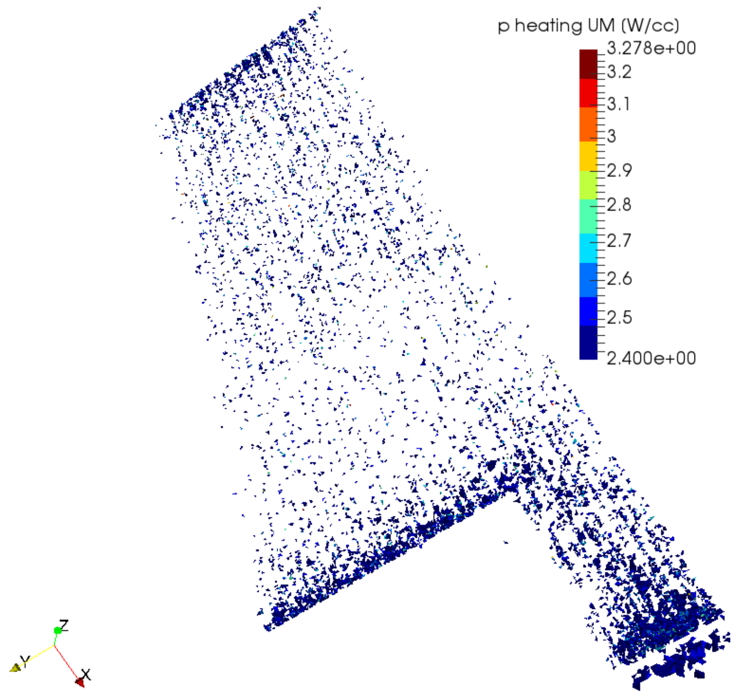


Figure 3.16: CuCrZr cells with a photon heating higher than 2.4 W/cc

Table 3.6: Heating peak, average on 1000 cell [W/cc]

Material	Cell Ratio (%)	n heating UM	n heating FMESH	p heating UM	p heating FMESH	n:(FMESH-UM)/UM (%)	p:(FMESH-UM)/UM (%)
CuCrZr	0.22	0.872	0.822	2.645	2.626	-5.73	-0.72
SS316L(N)	0.08	0.991	0.944	2.285	2.321	-4.74	1.58
S660	1.82	0.142	0.141	0.586	0.581	-0.70	-0.85
Water	0.12	2.728	2.438	0.272	0.258	-10.63	-5.15

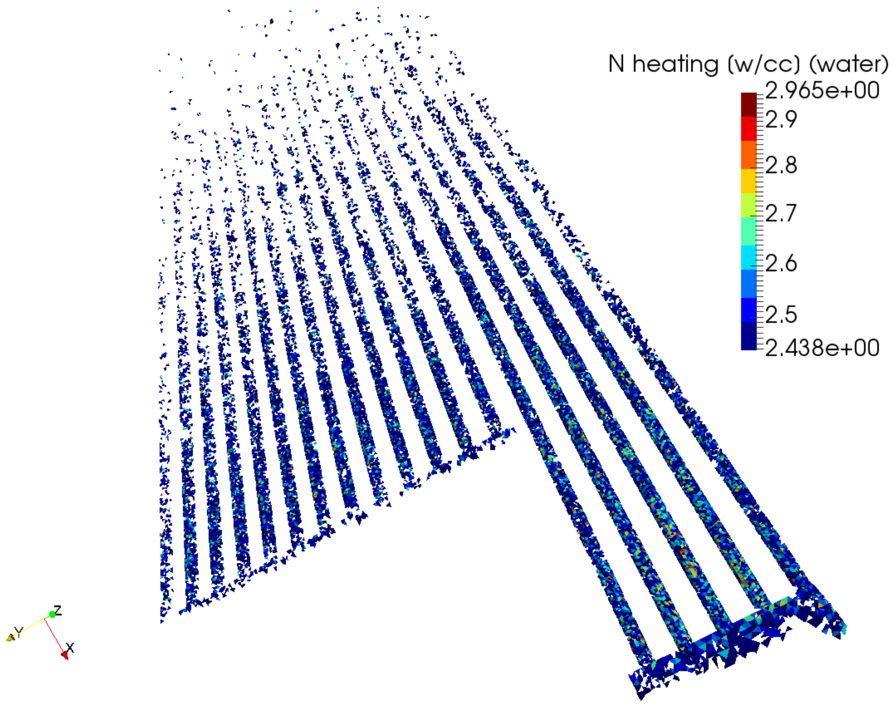


Figure 3.17: UM cells with higher neutron heating values in water

ratios in each material is around 1-2‰except for the bolts that have a fewer number of cells compared to the others. After this simple operation, the deviations are significantly reduced and they all remain under 5% except for the neutron heating in water which remains around 10%. To check that this last result was not still due to a bad statistics effect, through Paraview all the cells with a neutron heating higher than the CSG peak were plotted as reported in fig 3.17. It can be seen that there are actually a lot of cells with higher neutron heating values and it appears that a true underestimation was made by the CSG.

3.4 Conclusions and future work

From this work many considerations can be made, for clarity purpose they will be divided in macro-arguments:

Pre/Post-processing: The only true complexity during UM preprocessing is the meshing phase. This could require more or less time in dependence from a series of factors like CAD quality, meshing ability of the analyst and complexity of the part to mesh. However this process presents a lot of room for improvement and, at least in principle, can speed up considerably with experience. Moreover the time spent for meshing is comparable to the one needed for the defeaturing of CAD model and splitting in order to be able to convert the geometry in CSG form. Regarding the post-processing process instead, unstructured meshes have a clear advantage in terms of time and complexity: all the data are already there ready to be visualized and processed. This doesn't happen with fmeshes where an average analysis can easily requires the analyst to treat more than 10 meshes, interpolate them on different pieces of the geometry and then recombine everything.

Precision: The structured meshes presented a lower and more uniform error, while the UM one appeared to be highly dependent on the volume of the single cell as it could be expected. This means that additional care is required when post-processing UM results since also cells that are near the plasma can have bad statistics due to their dimensions. However the difference of spatial resolution achieved between the two methodologies, which was 5 times higher on average for UM, sufficiently explain this error difference. It has to be pointed out also that the high resolution in this case was not a choice, in fact it was the only way to faithfully model small and round elements like the water channels. However some improvement could have been done, like checking the cell dimensions during the meshing phase in order to avoid unphysical results due to tiny volumes or the adoption of 2nd order tetrahedra to reduce the mesh refinement while still maintaining the initial geometry shape.

Accuracy One thing is the statistical error of the obtained results, another is to understand how close are the obtained results to the physical reality, and regarding this the advantage of UM seems clear. This happens for two main reasons: the first is that UM are the only method in which flux and heating scoring are never averaged, the second is that, in order to have local results, the fmesh need interpolation which introduce an additional error. This conclusion is supported by the fact that the difference on local results is quite evident while is less pronounced when looking at the global results. Additionally, is possible to claim that UM global results were more accurate since their deviation with respect to their reference tally is significantly smaller than fmesh ones and the geometry was less simplified.

Integration: Interpolation is needed not only for visualization purposes, but it's fundamental when heating data has to be used as a thermal source for other kind of analysis which are always based on unstructured mesh. A possible advantage of the UM can be that the same exact mesh could be used in theory to calculate heating data and then directly applied as a source in thermal analysis for example. At this stage this path is still theoretical, but in principle it is already possible to apply in some applications: for example, thermal analysis often uses meshes only composed by tetrahedra and polihedra, it should be possible to develop a script able to automatically split the polihedra in tetras and than recompose them once the neutronic simulation is completed. This would mean not only better source data, but also a huge simplification in the integrated analysis workflow.

Computer Resources: As previously commented there was practically no difference in the loading time of the two models, while the simulation rate was around a factor 1.6 faster for the CSG+FMESH simulation if compared to the UM one. However, since the UM model was found to be lighter, this CSG advantage could be less effective than what it was in this case. Indeed if these simulations were to be run on the full C-model or using weight windows, the 4GB RAM limit could be easily exceeded causing a slow down of the simulation if the same number of nodes is used. This effect can be even more emphasized if the number of materials inside the model grows, since each one of them would require an additional fmesh.

To conclude, unstructured meshes in neutronic analysis applications still need some fine tuning and users still have to familiarize with them, but even if they still have some limitation the author believes that they are ready to be used for production in specific application. At the moment, the smaller the system, the more suitable UM are for the job, since intrinsically they work with a very high resolution in order to reproduce the model geometry with tetras. This could mean that UM can be easily used for specific analysis of sensitive components like mirrors or other diagnostic systems. For this kind of application an UM analysis would be way faster than a classic one when all the simulation life is considered from model production to post-processing, so, since UM are already mature enough and ready to be used, they should be preferred. Another line of work that could be pursued, even if still requires additional study, is the integration between nuclear and other kind of analysis using only one mesh: this would dramatically simplify the workflow and it is also the reason why UM capabilities were introduced in Monte Carlo codes. The starting point of this integration process would probably be the thermal analysis, since they mainly use meshes composed only by tetrahedra and polyhedra. The thermal analyst, indeed, could produce a tetra mesh suitable for its kind of analysis and then pass it to the nuclear analyst to compute the nuclear heating through MCNP using the UM approach. At this point, with the help of a script, the tetras could be collapsed into polyhedra providing in this way an appropriate local thermal source while avoiding the approximation that would be introduced by interpolation. All of this would let users familiarize with UM while still being productive, waiting for more studies and technical progresses that could expand the field of applications of the methodology.

Bibliography

[1] Altair. *Hypermesh.* 2015.

[2] ANSYS. *SpaceClaim.* 2018. URL: `http://www.spaceclaim.com/en/default.aspx`.

[3] R. Ash and C. Doleans-Dade. *Probability and Measure Theory.* Academic Press, 1999.

[4] M. De Pietri, M. Fabbri, D. Laghi, and R. Pampin. “A preliminary assessment of MCNP unstructured mesh integration in the ITER neutronics model”. In: *Fusion Engineering and Design* (2019). DOI: `10.1016/j.fusengdes.2019.01.058`. URL: `https://doi.org/10.1016/j.fusengdes.2019.01.058`.

[5] Leicht Dieter, Colling Bethany, Fabbri Marco, Juarez Rafael, Loughlin Michael, Pampin Raul, Polunovskiy Eduard, Serikove Arkady, Turner Andrew, and Bertalot Luciano. “The ITER tokamak neutronics reference model C-Model”. In: *Fusion Engineering and Design* 136 (2018).

[6] Greenwood John. “The correct and incorrect generation of a cosine distribution of scattered particles for Monte-Carlo modelling of vacuum systems”. In: *Vacuum* 67 (2002).

[7] Los Alamos National Laboratories. *MCNP USER’S MANUAL Code Version 6.2.* 2017.

[8] D. Laghi, M. Fabbri, R. Pampin, and A. Portone. “Understanding and Investigation of Relationships between Geometrical Errors and Lost Particles in MCNP”. In: *Submitted for poster presentation at ISFNT14 to be held in Budapest* (22-27 Sept. 2019).

[9] Roger L. Martz. *he MCNP6 Book On Unstructured Mesh Geometry: User’s Guide For MCNP 6.2 (LA-UR-17-22442).* 2017.

[10] R. Pampin, N. Casal, M. Fabbri, M. Gagliardi, and D. Laghi. “Estimation of Radiation Conditions in the ITER ECUL with State-of-the-Art Simulation Techniques”. In: *Submitted for poster presentation at ISFNT14 to be held in Budapest* (22-27 Sept. 2019).

[11] Wu Y., Son J., and et al. “CAD-Based Monte Carlo Program for Integrated Simulations of Nuclear System, SuperMC”. In: *Ann. Nuc. Energy* 82 (2015).

Appendices

Appendix A

Uniform and isotropic fluence inside a sphere using a cosine distribution: an analytical demonstration

A.0.1 Normalization

Let's assume the source differential angular distribution is:

$$S(\Omega) = S(\theta) = S_0 \frac{cos\theta}{2\pi} \quad \left[\frac{\#particles}{cm^2 \cdot steredians}\right] \tag{A.1}$$

where $S_0 = \dfrac{N_p}{4\pi R^2}$ is considered to be the source strength. Since:

$$d\Omega = \frac{dS}{r^2} = \frac{r^2 sin\theta d\theta d\phi}{r^2} = sin\theta d\theta d\phi \tag{A.2}$$

the un-normalized probability distribution function (pdf) will be:

$$p_{\mathrm{un}}(\theta, \phi) d\theta d\phi = S(\Omega) d\Omega = S_0 \frac{cos\theta}{2\pi} d\Omega = S_0 \frac{cos\theta}{2\pi} sen\theta d\theta d\phi \tag{A.3}$$

defining $\mu = cos\theta$ and remembering $d\mu = -sen\theta d\theta$, the previous equation can be rewritten as:

$$p_{\mathrm{un}}(\mu, \phi) d\mu d\phi = S_0 \mu (-d\mu) \frac{d\phi}{2\pi} = f(\mu) d\mu f(\phi) d\phi \tag{A.4}$$

that means:

$$\begin{cases} f(\phi) d\phi = \dfrac{1}{2\pi} d\phi \\ f(\mu) d\mu = S_0 \mu (-d\mu) \end{cases} \tag{A.5}$$

the distribution for ϕ is already normalized, we need to make sure the same stands for μ:

$$p(\mu) d\mu = \frac{-S_0 \mu d\mu}{-S_0 \int_1^0 \mu d\mu} = \frac{-\mu d\mu}{1/2} \tag{A.6}$$

Finally it is possible to recognize the $p(\mu) = -2\mu$ relation as it is described in MCNP manual. The minus sign comes from the fact that an increasing angle determines a decrease in cosine, as a matter of fact:

$$\int_0^{\pi/2} \sin\theta d\theta = \int_1^0 -d\mu = \int_0^1 d\mu \tag{A.7}$$

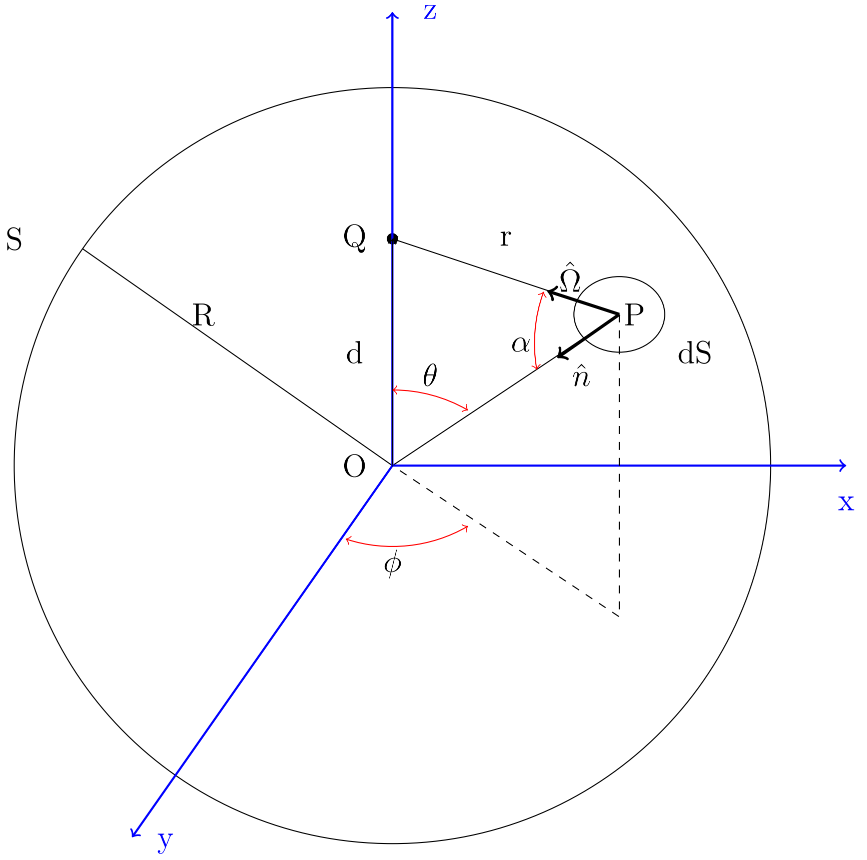


Figure A.1: Fluence's contribute of dS on point Q

A.0.2 Analytical demonstration[1]

The source's emissivity is considered to be:

$$E = \frac{S_0}{2\pi} 2\cos\alpha d\Omega \tag{A.8}$$

where S_0, the strength of the source, is $S_0 = NPS \cdot \dfrac{dS}{4\pi R^2}$ since the particles are sampled uniformly on the surface.

E represents the number of particle generated in dS that are emitted inside the differential solid angle $d\Omega$. To compute the fluence in Q the emissivity has to be integrated on all the source's surface, considering $\Omega = \dfrac{1}{r^2}dS$:

$$\varphi = \int\limits_S \frac{S_0}{\pi}\cos\alpha\frac{1}{r^2}dS = S_0\int\limits_0^{2\pi}\frac{1}{\pi}d\phi\int\limits_0^{\pi}\frac{\cos\alpha}{r^2}R^2\sin\theta d\theta = 2S_0R^2\int\limits_0^{\pi}\frac{\cos\alpha}{r^2}\sin\theta d\theta \tag{A.9}$$

The next step is to express α as a function of d, R and θ:

$$\cos\alpha = \hat{\Omega}\cdot\hat{n} = \frac{\vec{PQ}}{r}\cdot\hat{n} \tag{A.10}$$

where:

$$\vec{PQ} = -\vec{OP} + \vec{OQ} = [0,0,d] - [R\sin\theta\cos\phi, R\sin\theta\sin\phi, R\cos\theta] \tag{A.11}$$

$$\hat{n} = [-\sin\theta\cos\phi, -\sin\theta\sin\phi, -\cos\theta] \tag{A.12}$$

[1]A. Portone, personal communication, 2018.

Substituting equations A.11 and A.12 in equation A.10:

$$\cos\alpha = \frac{R\sin^2\theta\cos^2\phi + R\sin^2\theta\sin^2\phi + R\cos^2\theta - d\cos\theta}{r} = \frac{R - d\cos\theta}{r} \tag{A.13}$$

Using Carnot theorem also r can be expressed as a function of d, R and θ:

$$r^2 = R^2 + d^2 - 2Rd\cos\theta \tag{A.14}$$

Substituting all of that in equation A.9:

$$\varphi = 2S_0R^2 \int_0^\pi \frac{R - d\cos\theta}{r^3}\sin\theta d\theta = 2S_0R^2 \int_0^\pi \frac{R - d\cos\theta}{(R^2 + d^2 - 2Rd\cos\theta)^{3/2}}\sin\theta d\theta \tag{A.15}$$

The integral can be rewritten in a dimensionless form defining $\gamma = \dfrac{d}{R}$:

$$\varphi = 2S_0R^2 \int_0^\pi \frac{R(1 - \gamma\cos\theta)}{[R^2(1 + \gamma^2 - 2\gamma\cos\theta)]^{3/2}}\sin\theta d\theta = 2S_0 \int_0^\pi \frac{(1 - \gamma\cos\theta)}{(1 + \gamma^2 - 2\gamma\cos\theta)^{3/2}}\sin\theta d\theta \tag{A.16}$$

Introducing the usual $\mu = \cos\theta$:

$$\varphi = 2S_0 \int_{-1}^{1} \frac{(1 - \gamma\mu)}{(1 + \gamma^2 - 2\gamma\mu)^{3/2}} d\mu = 2S_0 \left[\frac{\mu - \gamma}{(1 - 2\gamma\mu + \gamma^2)^{1/2}}\right|_{-1}^{1} = 2S_0 \cdot 2 \tag{A.17}$$

Finally using the definition of S_0:

$$\varphi = 4 \cdot \frac{1}{4\pi R^2} = \boxed{\frac{1}{\pi R^2}} \tag{A.18}$$

Appendix B

SDEF card for a parallelepipedal source

Hereafter is reported an example of a SDEF card generating a working $100cm \times 20cm \times 10cm$ parallelepipedal source able to generate an isotropic and uniform fluence inside its volume:

```
sdef par=d1 y=fpar d20 z=fpar d30 x=fpar d40 vec=fpar d50 dir=d60
si1 l 1 1 1 1 1 1
sp1 0.03125 0.03125 0.15625 0.15625 0.3125 0.3125
c --Y ranges--
ds20 s 21 22 23 24 25 26
si21 -10.0 10.0
sp21 0 1
si22 -10.0 10.0
sp22 0 1
si23 l 10.0
sp23 1
si24 l -10.0
sp24 1
si25 -10.0 10.0
sp25 0 1
si26 -10.0 10.0
sp26 0 1
c --Z ranges--
ds30 s 31 32 33 34 35 36
si31 -5.0 5.0
sp31 0 1
si32 -5.0 5.0
sp32 0 1
si33 -5.0 5.0
sp33 0 1
si34 -5.0 5.0
sp34 0 1
si35 l 5.0
sp35 1
si36 l -5.0
sp36 1
c --X ranges--
ds40 s 41 42 43 44 45 46
si41 l 50.0
sp41 1
```

```
si42 l -50.0
sp42 1
si43 -50.0 50.0
sp43 0 1
si44 -50.0 50.0
sp44 0 1
si45 -50.0 50.0
sp45 0 1
si46 -50.0 50.0
sp46 0 1
c --VEC--
ds50 s 51 52 53 54 55 56
si51 l -1 0 0
sp51 1
si52 l 1 0 0
sp52 1
si53 l 0 -1 0
sp53 1
si54 l 0 1 0
sp54 1
si55 l 0 0 -1
sp55 1
si56 l 0 0 1
sp56 1
c --DIR--
si60 0 1 $just inwards direction
sp60 -21 1 $cosine distribution
```

Breaking it down:

- First a generic plane surface is defined as a function of PAR. All the source parameters depend on which of the 6 parallelepiped's faces they refer to. In order to do so a dummy parameter, the type of particle, was introduced.
- Each of the six surfaces receive a probability of emission equal to the ratio between correspondent face surface and the total surface area of the parallelepiped.
- X,Y and Z ranges are specified in order to actually generate each face of the parallelepiped, the probability of emission inside each range is uniform.
- All the 6 VEC are specified to be normal to the faces and with inward direction.
- The polar angle emission probability is set to be a cosine distribution where $0 < \cos\alpha < 1$ in order to emit particles only in the inward directions.

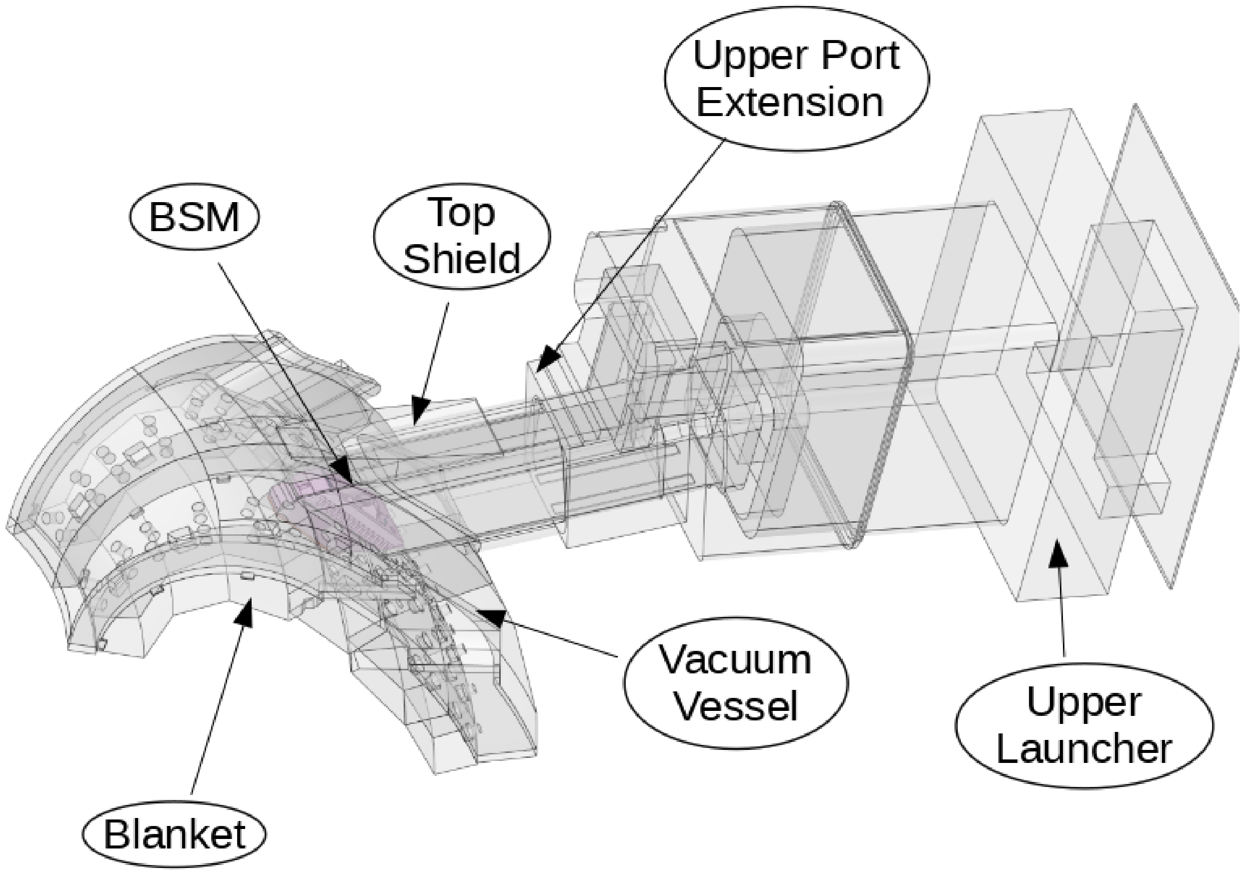
Appendix C

Used fillers

Hereafter is the list of envelopes that were left filled:

- '22' Blanket rows 7-8-9 [Blanket]
- '23' Blanket rows 7-8-9 [Blanket]
- '24' Blanket rows 7-8-9 [Blanket]
- '25' Blanket row 10 [Blanket]
- '26' Blanket row 10 [Blanket]
- '27' Blanket row 11 [Blanket]
- '28' Blanket row 11 [Blanket]
- '29' Blanket row 11 **(MODIFIED)** [Blanket]
- '30' Blanket row 11 [Blanket]
- '31' Blanket row 11 [Blanket]
- '32' Blanket row 12 [Blanket]
- '33' Blanket row 12 [Blanket]
- '34' Blanket row 12 [Blanket]
- '35' Blanket row 12 [Blanket]
- '36' Blanket row 12 [Blanket]
- '70' EC-upper launcher, **NEW FILLER** [Upper Launcher]
- '701' BSM 1st cut **(NEW ENVELOPE)** [BSM]
- '702' BSM 2nd cut **(NEW ENVELOPE)** [BSM]
- '703' BSM 3rd cut **(NEW ENVELOPE)** [BSM]
- '704' small mirrors **(NEW ENVELOPE)** [BSM]
- '125' Shield above upper port [Top Shield]
- '128' Upper port extension and connecting duct (CENTRAL) [Upper Port Extension]

- '157' VV PS2, field joint [Vacuum Vessel]
- '158' VV PS2, field joint [Vacuum Vessel]
- '159' VV PS2 [Vacuum Vessel]
- '160' VV PS2 [Vacuum Vessel]



Appendix D

Additional data

Table D.1: Control Tallies in Blanket, fluxes are in $\left[\frac{\#}{cm^3}\right]$ while heating is in $[W]$

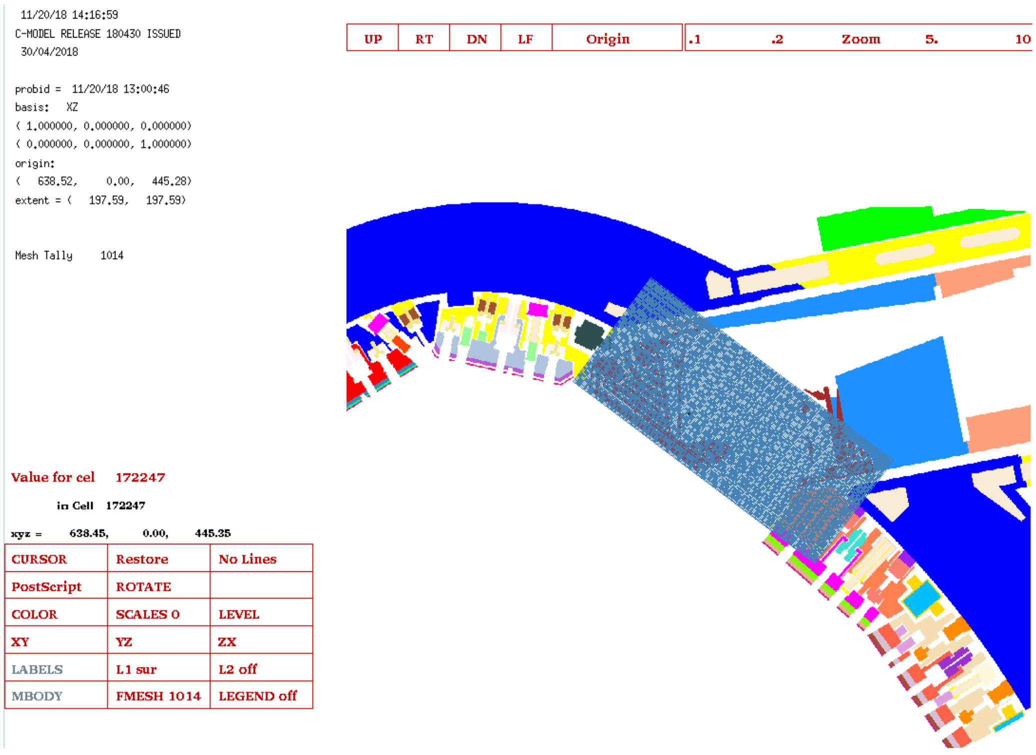
Envelope	N flux UM	N flux CSG	P flux UM	P flux CSG	N:(UM-CSG) /CSG	P:(UM-CSG) /CSG
23	6.200990e+19	6.199650e+19	3.687380e+19	3.688050e+19	0.000216	-0.000182
25	3.458390e+19	3.456450e+19	2.023990e+19	2.024940e+19	0.000561	-0.000469
26	3.447750e+19	3.445600e+19	2.015390e+19	2.016680e+19	0.000624	-0.000640
29	2.793380e+19	2.791690e+19	1.774680e+19	1.774980e+19	0.000605	-0.000169

Table D.2: Reference Tallies in Upper Launcher, fluxes are in $\left[\frac{\#}{cm^3}\right]$ while heating is in $[W]$

Envelope	**Neutron Flux CSG**	**Neutron Flux UM**	**Neutron Heating CSG**	**Neutron Heating UM**	**Photon Flux CSG**	**Photon Flux UM**	**Photon Heating CSG**	**Photon Heating UM**
70	1.770e+19	1.772e+19	24293	24274	8.360e+18	8.347e+18	61618	61454
701	4.885e+18	4.957e+18	10303	10144	2.753e+18	2.714e+18	15105	16560
702	8.992e+18	9.086e+18	41108	40241	5.614e+18	5.540e+18	103786	102446
703	4.4260e+18	4.460e+18	5049	5178	2.515e+18	2.467e+18	22226	23219
704	3.185e+17	3.224e+17	595	593	1.549e+17	1.535e+17	3308	3276
BSM	1.830e+19	1.850e+19	56461	55563	1.088e+19	1.072e+19	141118	142226

Appendix E

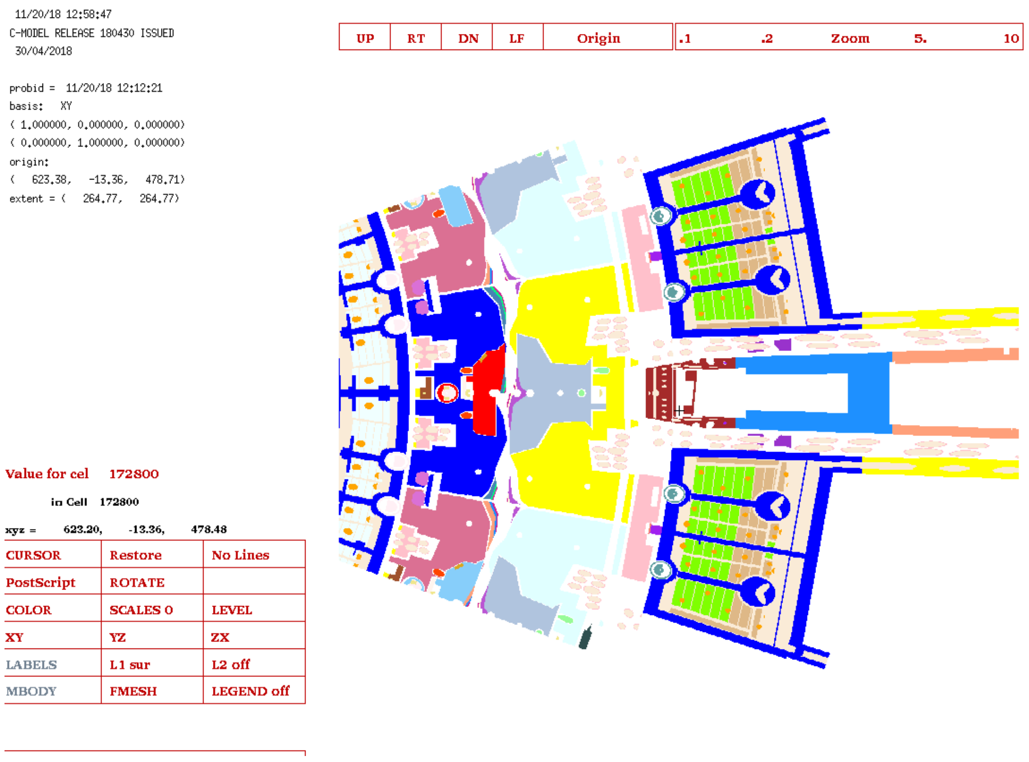
Additional figures



```
11/20/18 14:16:59
C-MODEL RELEASE 180430 ISSUED
 30/04/2018

probid =  11/20/18 13:00:46
basis:   XZ
( 1.000000, 0.000000, 0.000000)
( 0.000000, 0.000000, 1.000000)
origin:
(   638.52,     0.00,    445.28)
extent = (   197.59,   197.59)


Mesh Tally     1014
```

```
Value for cel    172247

       in Cell   172247

xyz =      638.45,      0.00,      445.35
```

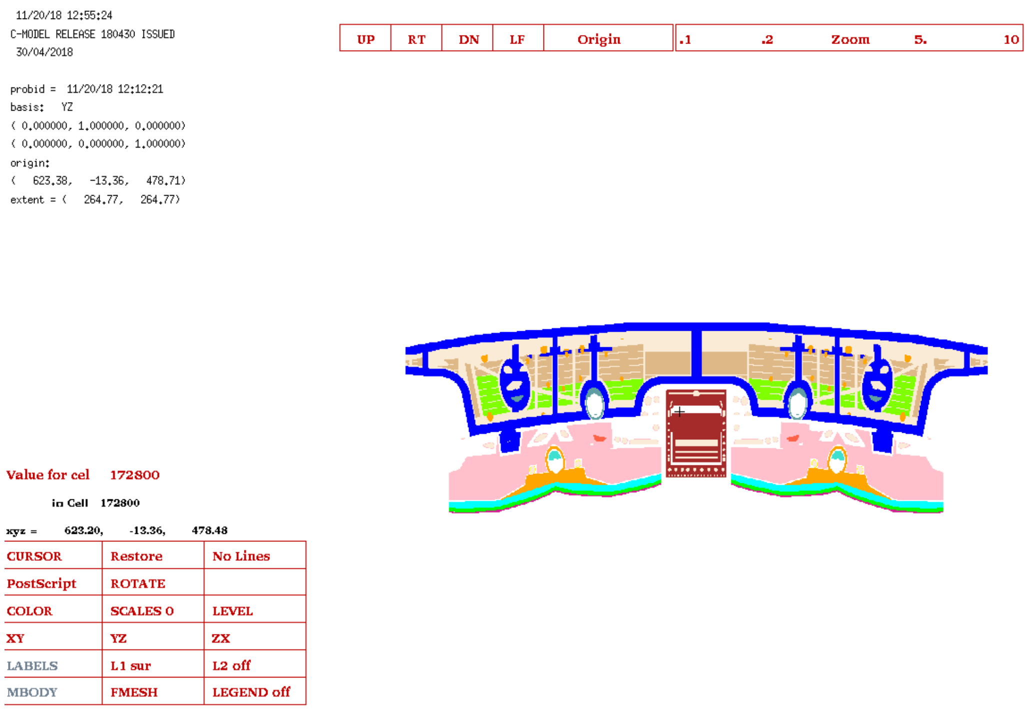
Figure E.1: FMESH structure and positioning



```
 11/20/18 12:58:47
C-MODEL RELEASE 180430 ISSUED
 30/04/2018

probid =  11/20/18 12:12:21
basis:   XY
( 1.000000, 0.000000, 0.000000)
( 0.000000, 1.000000, 0.000000)
origin:
(    623.38,    -13.36,    478.71)
extent = (    264.77,    264.77)
```

```
Value for cel     172800

        in Cell   172800

xyz =      623.20,      -13.36,      478.48
```

Figure E.2: XY view of the model



```
 11/20/18 12:55:24
C-MODEL RELEASE 180430 ISSUED
 30/04/2018

probid =  11/20/18 12:12:21
basis:   YZ
( 0.000000, 1.000000, 0.000000)
( 0.000000, 0.000000, 1.000000)
origin:
(    623.38,    -13.36,    478.71)
extent = (    264.77,    264.77)
```

```
Value for cel     172800

        in Cell   172800

xyz =      623.20,      -13.36,      478.48
```

Figure E.3: YZ view of the model

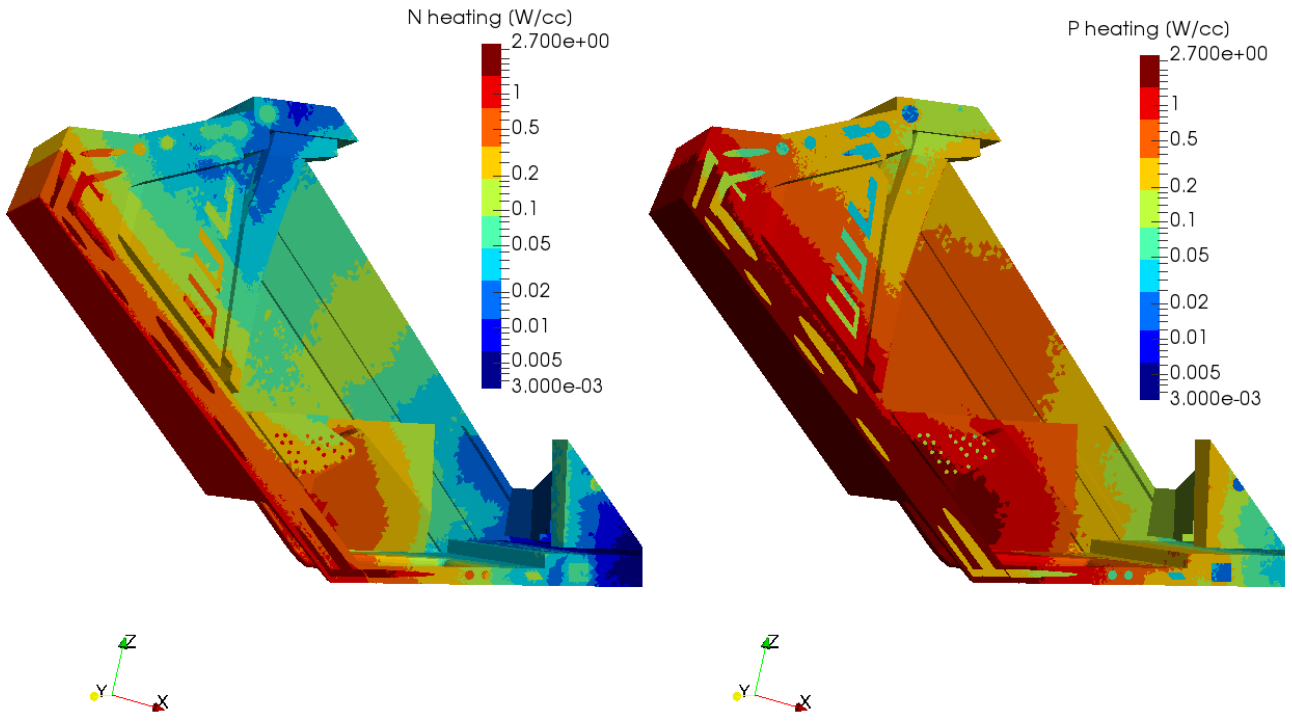


Figure E.4: Heating in Unstructured Meshes simulations, Y-section

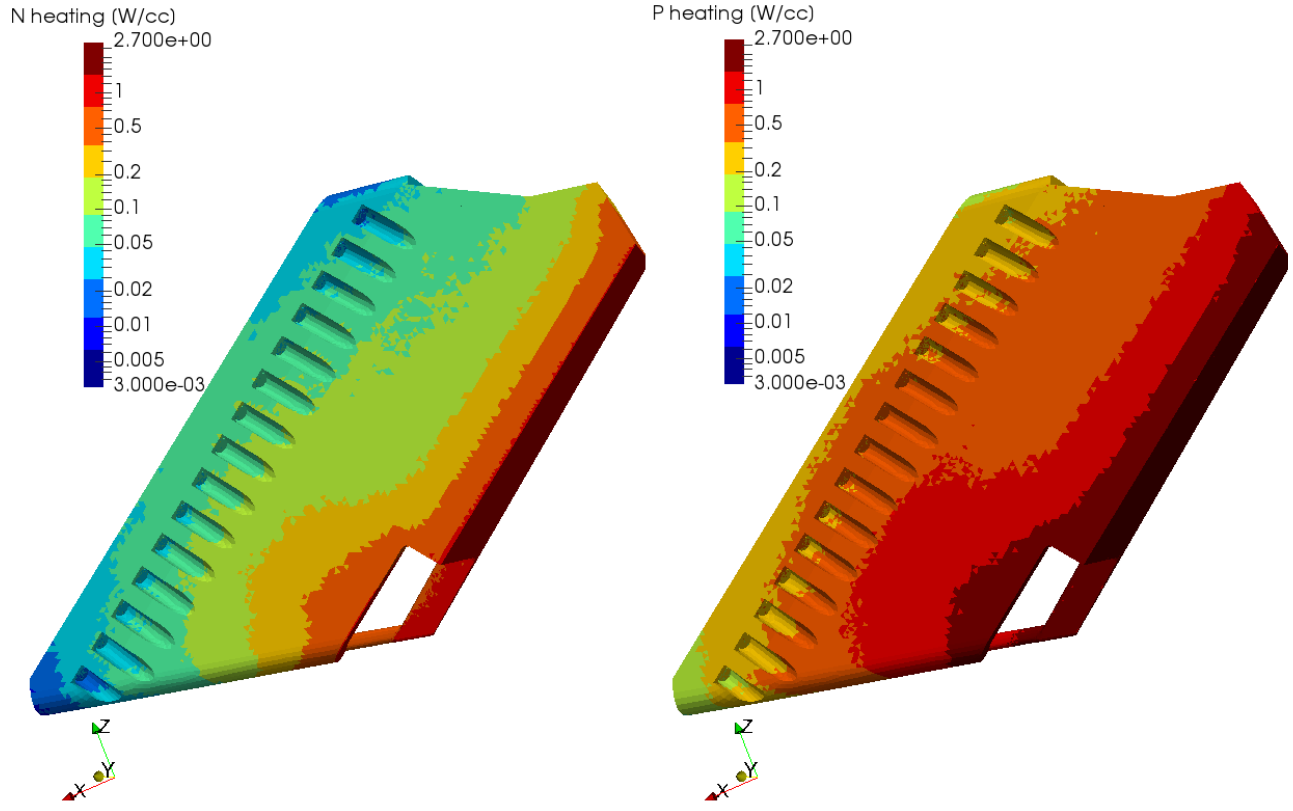


Figure E.5: Heating in Unstructured Meshes simulations, side view

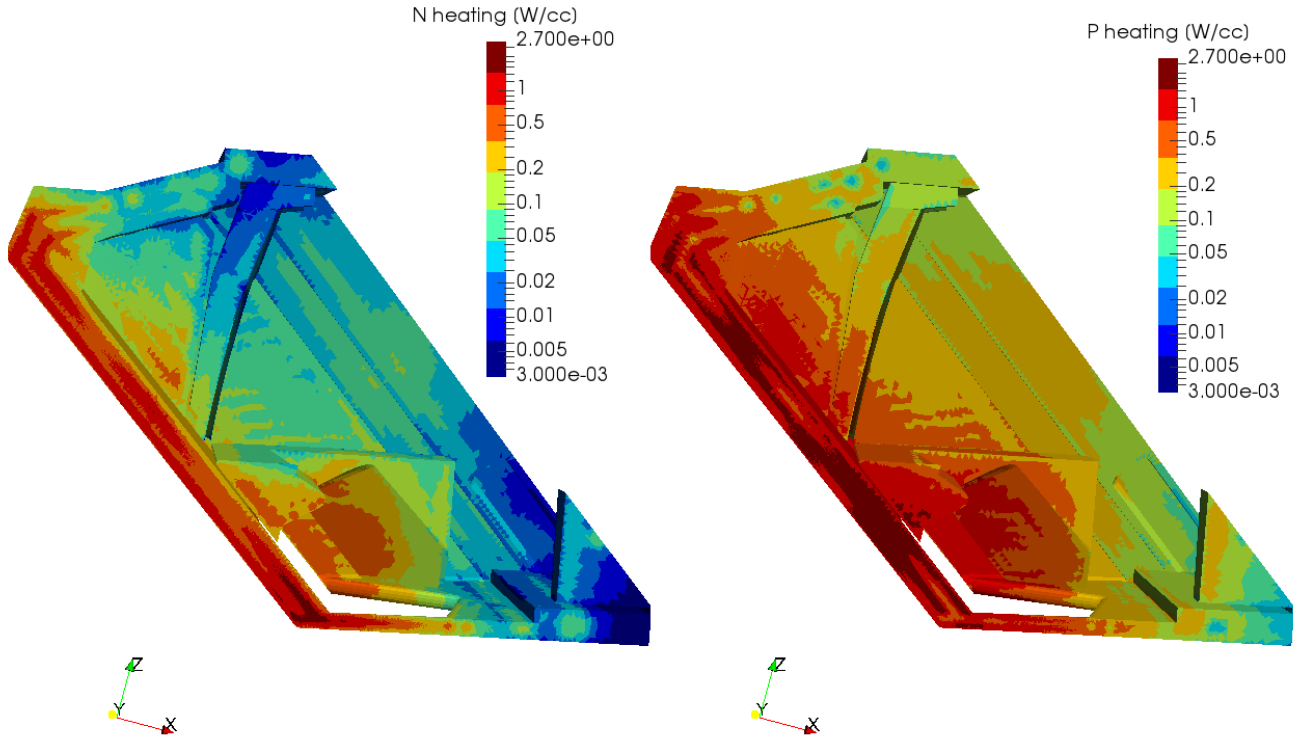
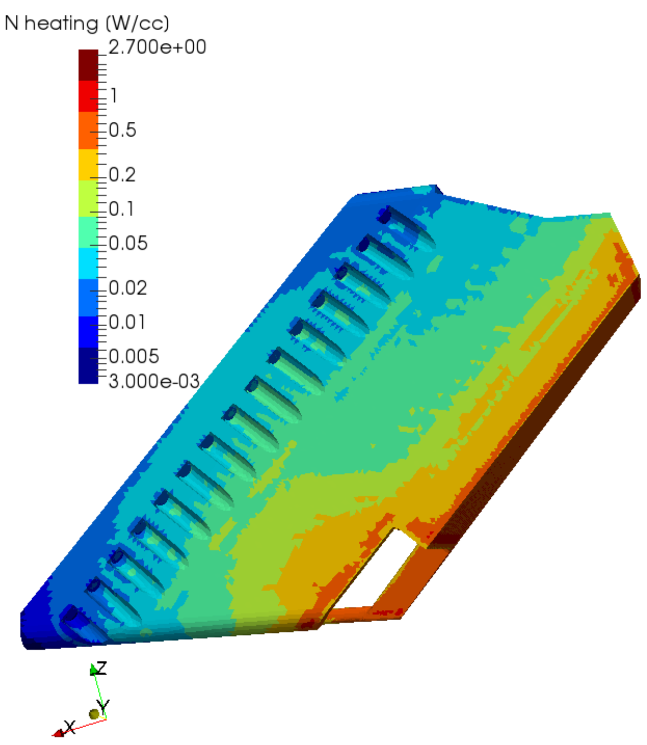


Figure E.6: Heating in Implicit fmesh simulations, Y-section



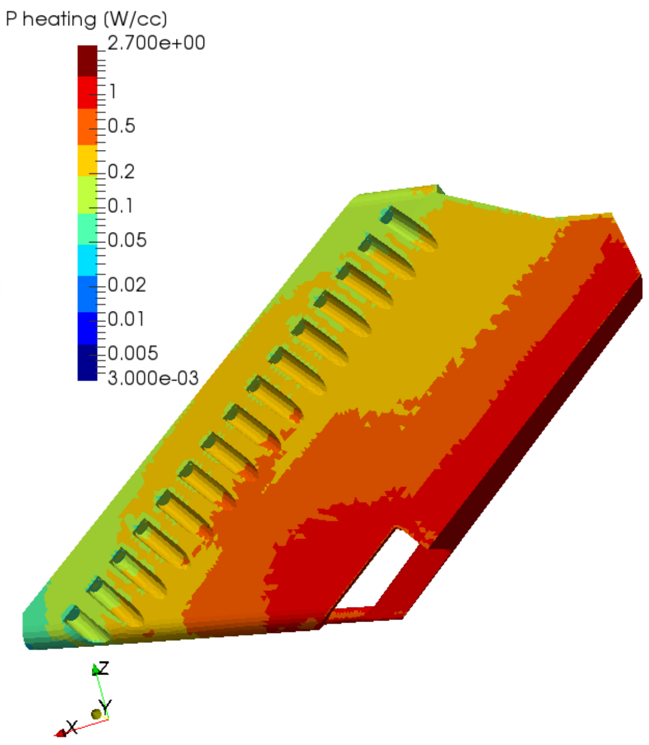
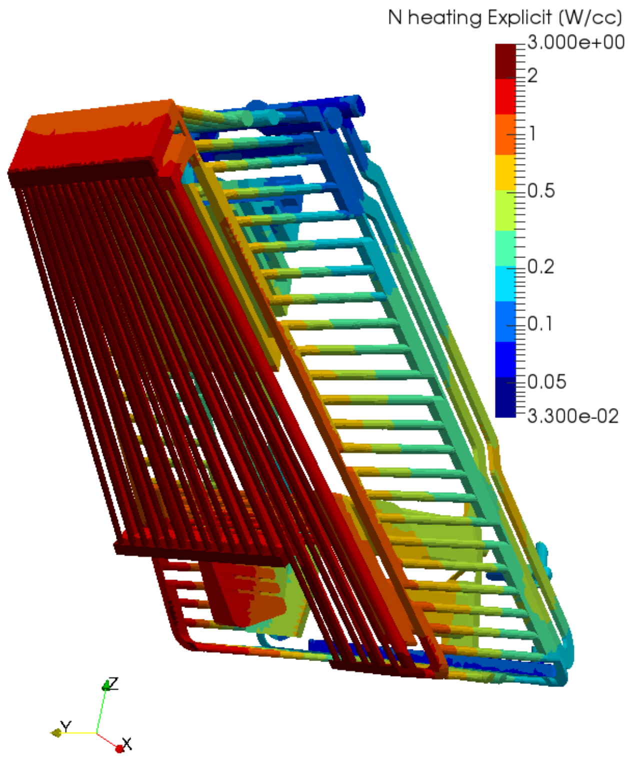


Figure E.7: Heating in Implicit fmesh simulations, side view



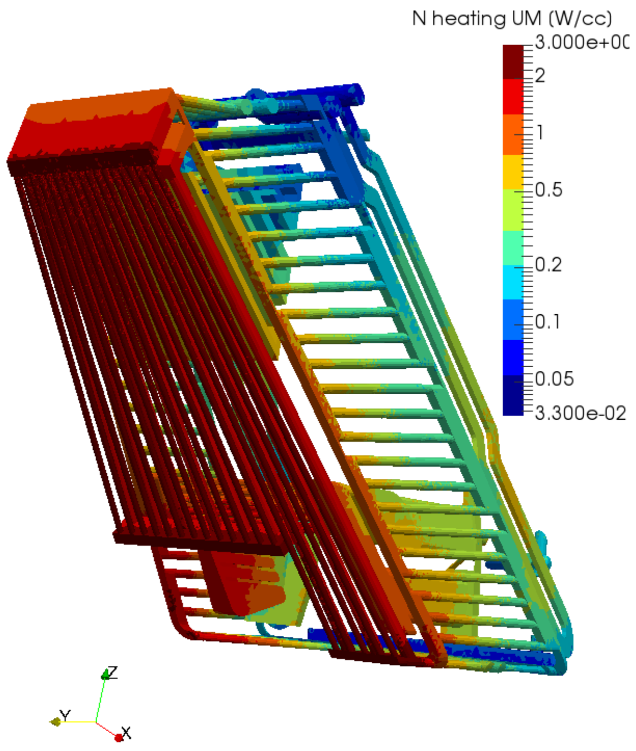


Figure E.8: Comparison between UM and Explicit results in water

Appendix F

UM pre/post processing workflow and guidelines

F.1 Pre-processing

In fig F.1 is reported a scheme summarizing the different pre-processing guidelines to follow both for UM and FMESH approach. The majority of these instructions have been already discussed in detail, but the points related to the generation of the .inp file need to be further explained.

The .inp file is the standard input file used by Abaqus and it contains all the information useful to define the mesh (that would be nodes, elements, etc.). MCNP uses this file format as its default UM geometry source, meaning that, when meshing, it has to be used either Abaqus or a meshing software that can export a .inp file. Alternatively, even a generic meshing software could be used, but an ad hoc script would have to be produced in order to translate the default mesh information file of that software into a .inp one. From all of this it is clear how using different meshing software can result in different procedures to follow, but, in the end, what matters is to obtain the correct structure of the .inp file as requested by MCNP. The following is the general structure that must be followed in order to describe a first order tetra mesh with only one part:

```
*Part, name=PART-1
*NODE
...
(List of nodes in .inp format starting from 1)
...
*ELEMENT, TYPE=C3D4
...
(list of mesh elements in .inp format starting from 1 )
...
*ELSET, ELSET=<elset 1 name>_material_statistics_001
...
(list of elements belonging to the elset)
...
*ELSET, ELSET=<elset 2 name>_material_statistics_002
...
(list of elements belonging to the elset)
...
*End Part
**
```

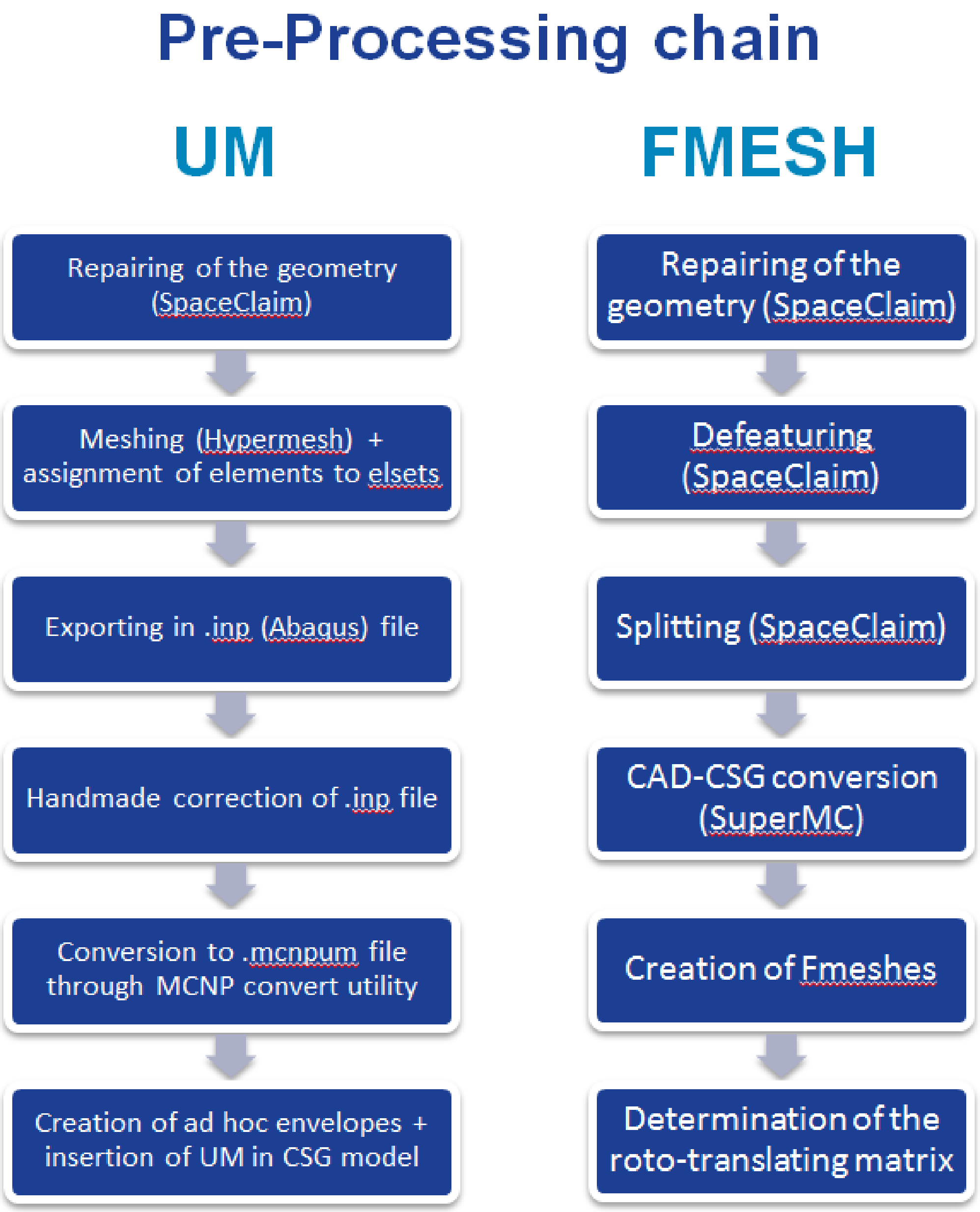

Pre-Processing chain
UM
FMESH
Repairing of the geometry (SpaceClaim)
Meshing (Hypermesh) + assignment of elements to elsets
Exporting in .inp (Abaqus) file
Handmade correction of .inp file
Conversion to .mcnpum file through MCNP convert utility
Creation of ad hoc envelopes + insertion of UM in CSG model
Repairing of the geometry (SpaceClaim)
Defeaturing (SpaceClaim)
Splitting (SpaceClaim)
CAD-CSG conversion (SuperMC)
Creation of Fmeshes
Determination of the roto-translating matrix

Figure F.1: Overview of the different pre-processing chains for UM and Fmesh approaches

```
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=PART-1-1, part=PART-1
*End Instance
*End Assembly
**HMNAME MATS          1 <material 1 name>_001    (number generated during exporting)
*MATERIAL, NAME=<material 1 name>_001
**HMNAME MATS          2 <material 2 name>_002     (number generated during exporting)
*MATERIAL, NAME=<material 2 name>_002
*****
```

If the meshing software is Hypermesh, to obtain this .inp these are the steps:

- An elset of elements for each material have to be created and renamed according to MCNP specifics (e.g. Channels_material_statistics_001)
- All the elements that do not have to be exported must be cancelled in order to avoid gaps in the elements numeration (this includes eventual 2D elements used during the meshing phase)
- All elements have to be renumbered starting from 1 using the Hypermesh tool "Renumber all".
- All the materials have to be created renaming them according to the corresponding elset (e.g. water_001)
- Export in Abaqus format

This will produce a naked structure containing nodes, elements, elsets and materials with the right format. At this point additional edit will be needed in order to mirror the final structure:

- Add the lines regarding the part and assembly definitions and instances
- Eliminate gaps between elements definition, this should be a unique list
- Add a space between "Element," and "Type" that will be probably missing.

In the following pages are reported an example both of the Hypermesh and of the MCNP versions of the .inp file.

.inp file: Hypermesh version (in red what needs to be cancelled)

```
*NODE
         1,  640.98812336299,  -20.10866271356,  487.91214050368
         2,  643.28151169247,  -22.13695520122,  489.24577549982
         3,  700.78715997404,  31.240273600544,  449.09417962591
        ...
*ELEMENT,TYPE=C3D4, ELSET=Body_SS316_material_statistics_001
         1,     56264,     56261,     55755,     55700
         2,     56378,     56379,     56264,     55755
         3,     56262,     56380,     55699,     56264
        ...
*ELEMENT,TYPE=C3D4, ELSET=Pipes_water_material_statistics_002
      1658,     156264,     156261,     155755,     155700
      ...
*ELEMENT,TYPE=C3D4, ELSET=Bolts_steel6608_material_statistics_003
     56284,     856264,     856261,     855755,     855700
      ...
*ELSET, ELSET=Body_SS316_material_statistics_001
         1,         2,         3,         4,         5,         6,         7,         8,
         9,        10,        11,        12,        13,        14,        15,        16,
        17,        18,        19,        20,        21,        22,        23,        24,
        ...
*ELSET, ELSET=Pipes_water_material_statistics_002
    302201,    302202,    302203,    302204,    302205,    302206,    302207,    302208,
    302209,    302210,    302211,    302212,    302213,    302214,    302215,    302216,
    302217,    302218,    302219,    302220,    302221,    302222,    302223,    302224,
    ...
*ELSET, ELSET=Bolts_steel6608_material_statistics_003
    369732,    369733,    369734,    369735,    369736,    369737,    369738,    369739,
    369740,    369741,    369742,    369743,    369744,    369745,    369746,    369747,
    369748,    369749,    369750,    369751,    369752,    369753,    369754,    369755,
    369756,    369757,    369758,    369759,    369760,    369761,    369762,    369763,
    ...
**HMNAME MATS          1 SS316_001     6
*MATERIAL, NAME=SS316_001
**HMNAME MATS          2 water_002    22
*MATERIAL, NAME=water_002
**HMNAME MATS          3 steel6608_003    52
*MATERIAL, NAME=steel6608_003
**HMNAME MATS          4 M312_004    20
*****
```

.inp file: MCNP version (in blue what has to be added)

```
*Part, name=PART-1
*NODE
         1,  640.98812336299,  -20.10866271356,  487.91214050368
         2,  643.28151169247,  -22.13695520122,  489.24577549982
         3,  700.78715997404,  31.240273600544,  449.09417962591
        ...
*ELEMENT,<space>TYPE=C3D4
         1,     56264,     56261,     55755,     55700
         2,     56378,     56379,     56264,     55755
         3,     56262,     56380,     55699,     56264
        ...
*ELSET, ELSET=Body_SS316_material_statistics_001
         1,         2,         3,         4,         5,         6,         7,         8,
         9,        10,        11,        12,        13,        14,        15,        16,
        17,        18,        19,        20,        21,        22,        23,        24,
        ...
*ELSET, ELSET=Pipes_water_material_statistics_002
    302201,    302202,    302203,    302204,    302205,    302206,    302207,    302208,
    302209,    302210,    302211,    302212,    302213,    302214,    302215,    302216,
    302217,    302218,    302219,    302220,    302221,    302222,    302223,    302224,
    ...
*ELSET, ELSET=Bolts_steel6608_material_statistics_003
    369732,    369733,    369734,    369735,    369736,    369737,    369738,    369739,
    369740,    369741,    369742,    369743,    369744,    369745,    369746,    369747,
    369748,    369749,    369750,    369751,    369752,    369753,    369754,    369755,
    369756,    369757,    369758,    369759,    369760,    369761,    369762,    369763,
    ...
*End Part
**
**
** ASSEMBLY
**
*Assembly, name=Assembly
**
*Instance, name=PART-1-1, part=PART-1
*End Instance
*End Assembly
**HMNAME MATS          1 SS316_001     6
*MATERIAL, NAME=SS316_001
**HMNAME MATS          2 water_002    22
*MATERIAL, NAME=water_002
**HMNAME MATS          3 steel6608_003    52
*MATERIAL, NAME=steel6608_003
**HMNAME MATS          4 M312_004    20
*****
```

F.2 Post-processing

As specified before, in order to convert the UM results from .eeout to .vtk format, a python script had to be generated. The script currently only works with first order tetrahedra and with MCNP6.2 version since the format of the .eeout file was changed from MCNP6.1. The python function asks for two parameters:

- “mode” → the possible modes are “single” which in case of multiple .eeout generates a single .vtk merging them all, or “multi” which instead creates a different .vtk for each .eeout file.
- “e” → is the list of .eeout file to convert

In addition to the conversion, the code also outputs a pandas DataFrame (results.pkl) that can be directly loaded in python for post-processing. The following is the actual code:

```
import re
import pandas as pd

def eeout_tovtk(mode,e):
#-------------------------------------------------
#=============== MULTI MODE =====================
#-------------------------------------------------
    if mode=='multi':
        #
        # === READING FILE ===
        #

        # -- General Variables --
        numTets=0
        numNodes=0
        particleList=[]
        nodesX=[]
        nodesY=[]
        nodesZ=[]

        #Identifiers
        idNodes='NUMBER OF NODES'
        idTets='NUMBER OF 1st TETS'
        idParticlesType='PARTICLE LIST'
        idNodeX='NODES X'
        idNodeY='NODES Y'
        idNodeZ='NODES Z'
        idElem='ELEMENT TYPE'
        idConn='CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED'
        idNeighbour='NEAREST NEIGHBOR DATA 1ST ORDER TETS'
        # Common patterns
        patNumber=re.compile('\d+')
        patNumberSci=re.compile('[-+]*\d+.\d+E[+-]\d+')
        patSpace=re.compile('\s+')
        patFlux=re.compile('d*4$')
        patHeat=re.compile('d*6$')
```

```
#special patterns
patTets=re.compile('\d\d+')

#Flags
readFlag=False

#pathfile=input('insert .eeout pathfile')
pathfile=e


# -- Reading mesh topology --
print('\nReading mesh Topology...')

# General infos
with open (pathfile,'r') as infile:
    for line in infile:
        if line.find(idNodes) !=-1:
            numNodes=patNumber.search(line).group()
        if line.find(idTets) !=-1:
            numTets=patTets.search(line).group()
            break


# Particle type
with open (pathfile,'r') as infile:
    for line in infile:
        if readFlag:
                particleList=(patNumber.findall(line))
                break

        if line.find(idParticlesType) !=-1:
                readFlag=True


with open (pathfile,'r') as infile:

    # Reading nodes
    readFlagX=False
    readFlagY=False
    readFlagZ=False

    for line in infile:

        # Reading nodes
        if readFlagX:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesX.append(float(a.group()))
        if readFlagY:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesY.append(float(a.group()))

        if readFlagZ:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesZ.append(float(a.group()))

        if line.find(idNodeX) !=-1:
            readFlagX=True
        if line.find(idNodeY) !=-1:
            readFlagY=True
            readFlagX=False
        if line.find(idNodeZ) !=-1:
            readFlagZ=True
            readFlagY=False

        #Reading element type
        if line.find(idElem) !=-1:
            readFlagZ=False



#Reading connectivity data
cells=[]
data=[]
with open (pathfile,'r') as infile:

    # Reading nodes
    readFlagCon=False

    for line in infile:
        if readFlagCon:
            data.append(line)
        if line.find(idConn) !=-1:
            readFlagCon=True
        if line.find(idNeighbour) !=-1:
            readFlagCon=False

del data[-1]
del data[-1]
for string in data:
    newline=string.strip()
    substrings=patSpace.split(newline)
    field = '4 '
    for substring in substrings:
        num=int(patNumber.search(substring).group())-1
        field=field+' '+'{:11d}'.format(num)
    cells.append(field)

#-- Reading edits --
print('Reading Edits...')
editParticles=[]
editUserNumber=[]
with open (pathfile,'r') as infile:
    editsDataFlag=False
    for line in infile:

        if editsDataFlag:
            strippedline=line.strip()
            array=patSpace.split(strippedline)
            editParticles.append(array[4])
            editUserNumber.append(array[1].strip('-'))
            editsDataFlag=False

        if line.find('EDIT DATA') !=-1:
            editsDataFlag=True
        if line.find('NODES X') !=-1:
            break

editSETS=[]
errorSETS=[]
patEditNumber=re.compile('\d+$')
n_tally=len(editUserNumber)
for editCounter,editNumber in enumerate(editUserNumber):
    with open (pathfile,'r') as infile:
        valuesFlag=False
        errorFlag=False
        rightTally=False
        valuesList=[]
        errorList=[]
        for line in infile:
            if valuesFlag and rightTally :
                strippedline=line.strip()
                values=patSpace.split(strippedline)
                for value in values:
                    valuesList.append(value)

            if errorFlag:
                strippedline=line.strip()
                errorValues=patSpace.split(strippedline)
                for value in errorValues:
                    errorList.append(value)
                if line.find('DATA OUTPUT PARTICLE')!= -1 or \
                 line.find('CENTROIDS X')!= -1:
                    del valuesList[0]
                    del valuesList[-37:]
                    editSETS.append(valuesList)
                    if n_tally==(editCounter+1):
                        del errorList[0]
                        del errorList[-9:]
                        errorSETS.append(errorList)
                        break
                    else:
                        del errorList[0]
                        del errorList[-20:]
                        errorSETS.append(errorList)
                        break

            if line.find('DATA OUTPUT PARTICLE') != -1:
                checkName=patEditNumber.search(line).group()
                rightTally=(editNumber==checkName)
            if line.find('DATA SETS RESULT TIME BIN') != -1:
                valuesFlag=True
            if line.find('DATA SETS RESULT SQR TIME BIN') != -1:
                valuesFlag=False
            if rightTally:
                if line.find('DATA SETS REL ERROR TIME BIN') != -1:
                    errorFlag=True

#--Read material--
materialFlag=False
materialsList=[]
fieldList=[]
with open (pathfile,'r') as infile:
    for line in infile:
        if line.find('CONNECTIVITY DATA') !=-1:
            break
        if materialFlag:
            strippedline=line.strip()
            matLine=patSpace.split(strippedline)
            for mat in matLine:
                materialsList.append(mat)
        if line.find('ELEMENT MATERIAL') !=-1:
            materialFlag=True

del materialsList[-6:]
fieldList.append(materialsList)
#--Read Density--
densityFlag=False
volumeFlag=False
densityList=[]
volumesList=[]
```

```
with open (pathfile,'r') as infile:
    for line in infile:
        if densityFlag:
            strippedline=line.strip()
            densityLine=patSpace.split(strippedline)
            for density in densityLine:
                densityList.append(density)
        if volumeFlag:
            strippedline=line.strip()
            volumeLine=patSpace.split(strippedline)
            for vol in volumeLine:
                volumesList.append(vol)
        if line.find('DENSITY') !=-1:
            densityFlag=True
        if line.find('VOLUMES') !=-1:
            volumeFlag=True
            densityFlag=False

del densityList[-8:]
fieldList.append(densityList)
fieldList.append(volumesList)
#
# === Writing output ===
#
for n_edit,editname in enumerate(editUserNumber):
    patDot=re.compile('.')
    title=patDot.split(pathfile)
    outpath=title[0]+'_'+editname+'.vtk'
    print('writing '+outpath+'...')
    with open(outpath,'w') as outfile:
        #-- Header --
        outfile.write('# vtk DataFile Version 3.0 \n'+ \
                      'Original file: '+pathfile+'\n'+ \
                      'ASCII \n \n')
        #-- DataSet --

        #NODES
        outfile.write('DATASET UNSTRUCTURED_GRID \n'+ \
                      'POINTS '+str(numNodes)+' float')
        for n, node in enumerate(nodesX):
            outfile.write( '\n'+'{:12.6f}'.format(node)+' '+ \
            '{:12.6f}'.format(nodesY[n])+' '+ \
            '{:12.6f}'.format(nodesZ[n]))
        #CELLS
        outfile.write('\n\nCELLS '+numTets+' '+str(int(numTets)*5)+'\n')
        for line in cells:
            outfile.write(line+'\n')
        #CELL Type
        outfile.write('\nCELL_TYPES '+numTets+'\n')
        for i in range(int(numTets)):
                outfile.write('10 \n')
            #CELL DATA
            checkflux=patFlux.search(editname)
            checkheat=patHeat.search(editname)
            if editParticles[n_edit]=='1':
                name='Neutron_'
            else:
                name='Photon_'
            if checkflux != None:
                name=name+'Flux_'+checkflux.group()
            if checkheat != None:
                name=name+'Heating_'+checkheat.group()
            outfile.write('\nCELL_DATA '+numTets+'\n'+ \
                          '\nSCALARS '+name+' float 1 \n'+ \
                          'LOOKUP_TABLE default \n')
            for item in editSETS[n_edit]:
                outfile.write(item+'\n')
            outfile.write('\nSCALARS '+name+'_ERROR'+' float 1 \n'+ \
                          'LOOKUP_TABLE default \n')
            for item in errorSETS[n_edit]:
                outfile.write(item+'\n')
            #FIELD DATA
            fieldDataNames=['Material','Density','Cell_Volume']
            for field,fieldDataName in enumerate(fieldDataNames):
                outfile.write('\nFIELD FieldData 1\n'+ \
                              fieldDataName+' 1 '+numTets+' float\n')
                for value in fieldList[field]:
                    outfile.write(value+'\n')

#--------------------------------------------------
#=============== SINGLE MODE =====================
#--------------------------------------------------
elif mode == 'single':
    #
    # === READING FILE ===
    #

    #Identifiers
    idNodes='NUMBER OF NODES'
    idTets='NUMBER OF 1st TETS'
    idParticlesType='PARTICLE LIST'
    idNodeX='NODES X'
    idNodeY='NODES Y'
    idNodeZ='NODES Z'
    idElem='ELEMENT TYPE'
    idConn='CONNECTIVITY DATA 1ST ORDER TETS ELEMENT ORDERED'
    idNeighbour='NEAREST NEIGHBOR DATA 1ST ORDER TETS'
    # Common patterns
    patNumber=re.compile('\d+')
    patNumberSci=re.compile('[-+]*\d+.\d+E[+-]\d+')
```

```
patSpace=re.compile('\s+')
patFlux=re.compile('d*4$')
patHeat=re.compile('d*6$')
#special patterns
patTets=re.compile('\d\d+')

#pathfile=input('insert .eeout pathfile')
pathfileList=e

numTetsMatrix=[]
numNodesMatrix=[]
particleListMatrix=[]
nodesXMatrix=[]
nodesYMatrix=[]
nodesZMatrix=[]
dataMatrix=[]
cellsMatrix=[]
editParticlesMatrix=[]
editUserNumberMatrix=[]
editSETSMatrix=[]
errorSETSMatrix=[]
fieldListMatrix=[]
nodes_to_add=-1
# -- Reading mesh topology --
print('\nReading mesh Topology...')
for iteration, pathfile in enumerate(pathfileList):
    # -- General Variables --
    numTets=0
    numNodes=0
    particleList=[]
    nodesX=[]
    nodesY=[]
    nodesZ=[]
    #Flags
    readFlag=False
    # General infos
    with open (pathfile,'r') as infile:
        for line in infile:
            if line.find(idNodes) !=-1:
                numNodes=patNumber.search(line).group()
            if line.find(idTets) !=-1:
                numTets=patTets.search(line).group()
                break
    numTetsMatrix.append(numTets)
    numNodesMatrix.append(numNodes)


    # Particle type
    with open (pathfile,'r') as infile:
        for line in infile:
```

```
        if readFlag:
                particleList=(patNumber.findall(line))
                break

        if line.find(idParticlesType) !=-1:
                readFlag=True
particleListMatrix.append(particleList)


with open (pathfile,'r') as infile:

    # Reading nodes
    readFlagX=False
    readFlagY=False
    readFlagZ=False

    for line in infile:

        # Reading nodes
        if readFlagX:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesX.append(float(a.group()))

        if readFlagY:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesY.append(float(a.group()))

        if readFlagZ:
            split=patSpace.split(line)
            for string in split:
                a=patNumberSci.search(string)
                if a != None:
                    nodesZ.append(float(a.group()))

        if line.find(idNodeX) !=-1:
            readFlagX=True
        if line.find(idNodeY) !=-1:
            readFlagY=True
            readFlagX=False
        if line.find(idNodeZ) !=-1:
            readFlagZ=True
            readFlagY=False

        #Reading element type
        if line.find(idElem) !=-1:
            readFlagZ=False
nodesXMatrix.append(nodesX)
nodesYMatrix.append(nodesY)
nodesZMatrix.append(nodesZ)



#Reading connectivity data
cells=[]
data=[]
with open (pathfile,'r') as infile:

    # Reading nodes
    readFlagCon=False

    for line in infile:
        if readFlagCon:
            data.append(line)
        if line.find(idConn) !=-1:
            readFlagCon=True
        if line.find(idNeighbour) !=-1:
            readFlagCon=False

del data[-1]
del data[-1]
for string in data:
    newline=string.strip()
    substrings=patSpace.split(newline)
    field = '4 '
    for substring in substrings:
        num=int(patNumber.search(substring).group())+nodes_to_add
        field=field+' '+'{:11d}'.format(num)
    cells.append(field)

dataMatrix.append(data)
cellsMatrix.append(cells)
nodes_to_add=nodes_to_add+int(numNodes)

#-- Reading edits --
print('Reading Edits in '+pathfile+'...')
editParticles=[]
editUserNumber=[]
with open (pathfile,'r') as infile:
    editsDataFlag=False
    for line in infile:

        if editsDataFlag:
            strippedline=line.strip()
            array=patSpace.split(strippedline)
```

```
                editParticles.append(array[4])
                editUserNumber.append(array[1].strip('-'))
                editsDataFlag=False

            if line.find('EDIT DATA') !=-1:
                editsDataFlag=True
            if line.find('NODES X') !=-1:
                break
    editParticlesMatrix.append(editParticles)
    editUserNumberMatrix.append(editUserNumber)

    editSETS=[]
    errorSETS=[]
    patEditNumber=re.compile('\d+$')
    n_tally=len(editUserNumber)
    for editCounter,editNumber in enumerate(editUserNumber):
        with open (pathfile,'r') as infile:
            valuesFlag=False
            errorFlag=False
            rightTally=False
            valuesList=[]
            errorList=[]
            for line in infile:
                if valuesFlag and rightTally :
                    strippedline=line.strip()
                    values=patSpace.split(strippedline)
                    for value in values:
                        valuesList.append(value)

                if errorFlag:
                    strippedline=line.strip()
                    errorValues=patSpace.split(strippedline)
                    for value in errorValues:
                        errorList.append(value)
                    if line.find('DATA OUTPUT PARTICLE')!= -1 or \
                     line.find('CENTROIDS X')!= -1:
                        del valuesList[0]
                        del valuesList[-37:]
                        editSETS.append(valuesList)
                        if n_tally==(editCounter+1):
                            del errorList[0]
                            del errorList[-9:]
                            errorSETS.append(errorList)
                            break
                        else:
                            del errorList[0]
                            del errorList[-20:]
                            errorSETS.append(errorList)
                            break
            if line.find('DATA OUTPUT PARTICLE') != -1:
                checkName=patEditNumber.search(line).group()
                rightTally=(editNumber==checkName)
            if line.find('DATA SETS RESULT TIME BIN') != -1:
                valuesFlag=True
            if line.find('DATA SETS RESULT SQR TIME BIN') != -1:
                valuesFlag=False
            if rightTally:
                if line.find('DATA SETS REL ERROR TIME BIN') != -1:
                    errorFlag=True
editSETSMatrix.append(editSETS)
errorSETSMatrix.append(errorSETS)

#--Read material--
materialFlag=False
materialsList=[]
fieldList=[]
with open (pathfile,'r') as infile:
    for line in infile:
        if line.find('CONNECTIVITY DATA') !=-1:
            break
        if materialFlag:
            strippedline=line.strip()
            matLine=patSpace.split(strippedline)
            for mat in matLine:
                materialsList.append(mat)
        if line.find('ELEMENT MATERIAL') !=-1:
            materialFlag=True

del materialsList[-6:]
fieldList.append(materialsList)
#--Read Density--
densityFlag=False
volumeFlag=False
densityList=[]
volumesList=[]
with open (pathfile,'r') as infile:
    for line in infile:
        if densityFlag:
            strippedline=line.strip()
            densityLine=patSpace.split(strippedline)
            for density in densityLine:
                densityList.append(density)
        if volumeFlag:
            strippedline=line.strip()
            volumeLine=patSpace.split(strippedline)
            for vol in volumeLine:
                volumesList.append(vol)
        if line.find('DENSITY') !=-1:
            densityFlag=True
            if line.find('VOLUMES') !=-1:
                volumeFlag=True
                densityFlag=False

        del densityList[-8:]
        fieldList.append(densityList)
        fieldList.append(volumesList)

        fieldListMatrix.append(fieldList)
#
# === Writing output ===
#
#Single VTK mode
results=pd.DataFrame()

totalTets=str(sum(list(map(int,numTetsMatrix))))
totalNodes=str(sum(list(map(int,numNodesMatrix))))

outpath='global.vtk'
print('writing '+outpath+'...')
with open(outpath,'w') as outfile:
    #-- Header --
    outfile.write('# vtk DataFile Version 3.0 \n'+ \
                  'Original file: '+pathfile+'\n'+ \
                  'ASCII \n \n')
    #-- DataSet --

    #NODES
    outfile.write('DATASET UNSTRUCTURED_GRID \n'+ \
                  'POINTS '+totalNodes+' float')
    for i, item in enumerate(nodesXMatrix): #for each nodesX set
        for n, node in enumerate(item):#for each x node value in the set
            outfile.write( '\n'+'{:12.6f}'.format(node)+' '+ \
            '{:12.6f}'.format((nodesYMatrix[i])[n])+' '+ \
            '{:12.6f}'.format((nodesZMatrix[i])[n]))
    #CELLS
    outfile.write('\n\nCELLS '+totalTets+' '+str(int(totalTets)*5)+'\n')
    for item in cellsMatrix:
        for line in item:
            outfile.write(line+'\n')
    #CELL Type
    outfile.write('\nCELL_TYPES '+totalTets+'\n')
    for i in range(int(totalTets)):
        outfile.write('10 \n')
    #CELL DATA
    outfile.write('\nCELL_DATA '+totalTets+'\n')
    for n_edit,editname in enumerate(editUserNumberMatrix[0]):
        checkflux=patFlux.search(editname)
        checkheat=patHeat.search(editname)
        if editParticles[n_edit]=='1':
                name='Neutron_'
            else:
                name='Photon_'
            if checkflux != None:
                name=name+'Flux_'+checkflux.group()
            if checkheat != None:
                name=name+'Heating_'+checkheat.group()
            outfile.write('\nSCALARS '+name+' float 1 \n'+ \
                          'LOOKUP_TABLE default \n')
            editsetList=[]
            for editset in editSETSMatrix:
                for item in editset[n_edit]:
                    editsetList.append(float(item))
                    outfile.write(item+'\n')
            results[name]=editsetList

            outfile.write('\nSCALARS '+name+'_ERROR'+' float 1 \n'+ \
                          'LOOKUP_TABLE default \n')
            errorList=[]
            for errorset in errorSETSMatrix:
                for item in errorset[n_edit]:
                    errorList.append(float(item))
                    outfile.write(item+'\n')
            results[name+'_ERROR']=errorList
        #FIELD DATA
        fieldDataNames=['Material','Density','Cell_Volume']
        for field,fieldDataName in enumerate(fieldDataNames):
            outfile.write('\nFIELD FieldData 1\n'+ \
                          fieldDataName+' 1 '+totalTets+' float\n')
            fieldList=[]
            for fieldlist in fieldListMatrix:
                for value in fieldlist[field]:
                    fieldList.append(float(value))
                    outfile.write(value+'\n')
            results[fieldDataName]=fieldList
        results.to_pickle('results.pkl')

return;
```

Acknowledgements

I would like to thank professor Marco Sumini and Mr. Alfredo Portone that brokered my stay at Fusion for Energy. Without them my work on this subject would have never even started. I particularly thank professor Sumini for all he taught us students in the nuclear field and Alfredo for the analytical demonstration that he donated me while I was giving up.

I am also grateful to Raul and Marco that supervised all my activities during my 6 months in Barcelona. They taught me something new every day and always helped me to overcome the obstacles I found during my research.

Last but not least I would like to thank my family that constantly supported me during these many years as a student. I hope their sacrifices are partially repaid today, at last.