31st International Conference Nuclear Energy for New Europe
NENE 2022 September 12-15 PORTOROŽ SLOVENIA

STOK - A Tool For Parametric Modeling Of Simple Tokamaks

Anže Gabrijel

Jožef Stefan Institute
Jamova cesta 39
SI-1000, Ljubljana,
Slovenia

and

Faculty of Electrical engineering
University of Ljubljana
Tržaška cesta 25
SI-1000, Ljubljana, Slovenia
anze.gabrijel@student.uni-lj.si

dr. Aljaž Čufar
Jožef Štefan Institute
Jamova cesta 39
1000, Ljubljana, Slovenia
aljaz.cufar@ijs.si

ABSTRACT

In fusion and fission reactor design and analyses, CAD (computer-aided design) models are usually the staple when it comes to geometry preparation. However, preparation of these models is often a time-consuming process requiring a significant amount of CAD designer time. Furthermore, this process is usually cyclical, where a design evolves through several iterations based on the results from various analyses. This begs the question; can such models be created using less user time and are such computer-generated models practical for use in neutronic simulations.

This paper focuses on describing the methods behind and usage of a parametric modelling tool called STOK, a Python-based program that allows parametric generation of simple tokamaks with rectangular cross-sections. These models are generated using sets of parameters, for instance, the vacuum vessel requires the user to define five parameters and when all the required parameters are defined a model is created using these parameters as constraints. Furthermore, intersections and other foreseen geometrical conflicts between components are automatically resolved, e.g., ports into the tokamak automatically introduce suitable openings into the shape of the vacuum vessel. Once the model is complete its components are exported in .step and .stl formats which can be directly used in neutronic simulations or can be converted using translators into a suitable format for use in simulations, e.g., through conversion into constructive solid geometry (CSG) commonly used in Monte Carlo codes for nuclear analyses. With these two methods we can perform cross-comparisons between different neutron transport codes like MCNP and Serpent for a set of significantly different geometries in an automated fashion, verify and benchmark different methods of geometry import into these codes, e.g., defining geometry with CSG in Serpent and importing it directly as .stl files to quantify the differences in both results and simulation efficiency, and work on automating reactor design optimization.

1 INTRODUCTION

Reactor design is very time consuming both in fusion and fission. In these fields CAD models are usually used for geometry preparation, but creating and adapting these models for use in analyses takes time, which could be used for analyses. As the design process is cyclical with multiple iterations where geometry changes are based on feedback from various analyses performed to test how well the design meets design requirements, e.g., structural, nuclear and thermohydraulic. The time spent for model preparation and adaptation for use in analyses can represent one of the bottlenecks in design optimization due to copious amounts of analyst time involved in these processes.

We approached the problem of expediting reactor modelling through automation with help from a Python [1] CAD-scripting library named CadQuery (CQ) [2]. Based on this library, we developed STOK [3], a Simple rectangular TOKamak generator that focuses on production of simple parametric reactor models that are often required for more fundamental analyses where simplicity of geometry is an advantage. A tool for generation of more realistic tokamak models is already available, i.e., Paramak [4].

This paper focuses on describing the methods behind and usage of a parametric modelling tool called STOK and presenting the benchmarks that were run for the purpose of validating this type of geometry generation for future projects which will be done on more complex reactor models of more up to date designs.

2 STOK

STOK is a tool created for speeding up geometry generation and, in the future, to test multi-parametric optimization algorithms for use in reactor design optimization. It is based in Python and is using an open-source CAD-scripting library called CadQuery to produce the geometry.

2.1 CadQuery

CQ is a Python library that gives the user the tools to create CAD models through scripting. Although it is written in Python its origin stems from Open CASCADE [5], which is a vast library of CAD design functions written in C++. Open CASCADE is complex, so CQ was made as more of a simplification while at the same time keeping much of the functionality of its parent library. Due to its ease of use this tool is very suitable for use in reactor geometry preparation.

2.2 How it works

In STOK components were generated, by first analysing their base geometry and finding the most basic shapes that they can be created from. We then created a basic shape that can be changed with some parameters, relevant to each object. From there, more complex components were created and parametrized to conform to the given larger design.

For example, the outer most layer of the tokamak wall (called layer 1 in Figures 1 and 2) was created by first deducing that this object can be created with two rectangular tori, one small and one large, and that the larger of the tori needs to have the inner radius equal to the radius of the central solenoid as well as conforming to the outer radius of the containment of the model.

A function was created that generates a torus with the following parameters; inner radius, outer radius, and the height of the torus.

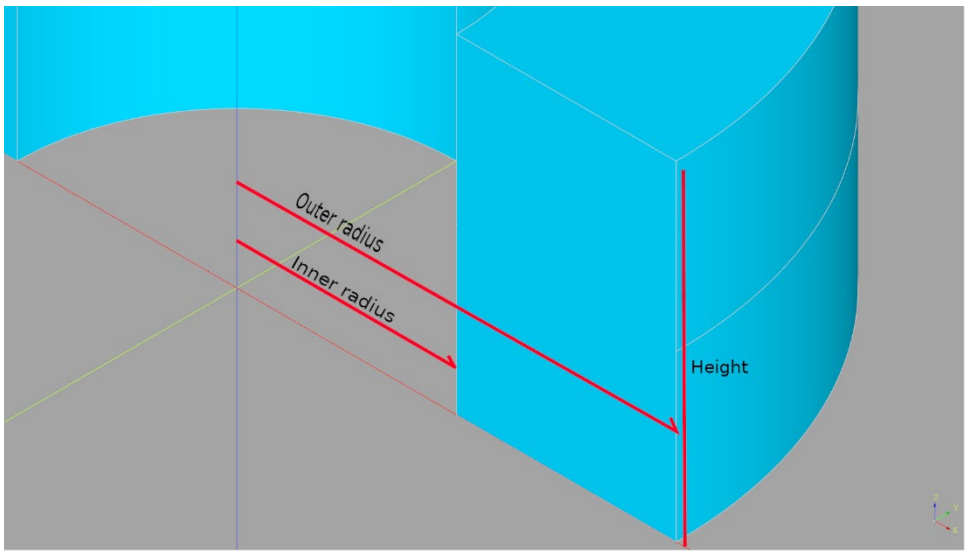

Figure 1: Outer torus with parameters.

This function was then used to generate both the larger and the smaller torus, where the parameters of the function conformed to the reactor model design. After we generate both tori, the smaller one is subtracted from the larger one resulting in a hollow torus that is used as the first layer of the reactor wall. The process is similar for other layers only differing in the values of the parameters used.

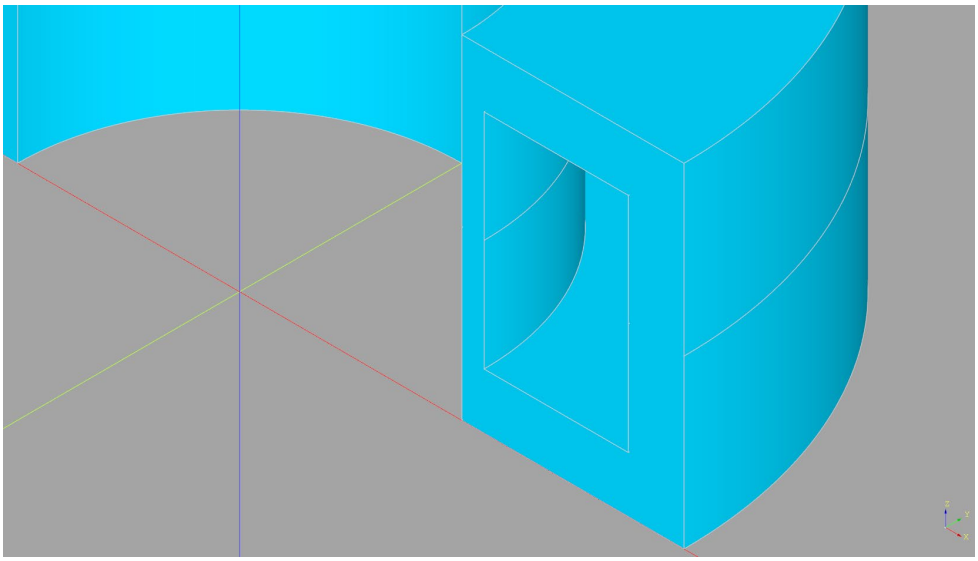

Figure 2: Torus with opening cut.

After all the reactor layers are complete, the port openings have to be added, which is modelled in a comparable way to the reactor layers. First a function is called that creates a box-like object and moves it to an appropriate radius. For the model presented in this paper this function is then used eight times to create eight rectangular boxes at of a certain length, width, and height. Each subsequent box is created with an offset from the previous one for forty-five degrees and moved outwards to the position of the outer radius of the first containment layer. When this is done, a subtraction is performed to remove the parts of reactor layers that overlap

with the boxes that represent the openings. The resulting reactor layer objects thus have holes in them to accommodate the reactor openings. Other parts of the geometry like central solenoid, detector cells, and transformer limbs are produced in the same fashion.

The most time intensive part of the reactor design is thus to find out and implement all the types of components that need to be included in a reactor model. Once this is done the models can be produced by simply combining these components and removing any overlaps with the use of an in-built function.

The last step of the model preparation process is then to export the geometry in a suitable format, e.g., into .stl for use in Serpent. Here we experienced some issues with the use of CQ generated .stl files in Serpent, the files seemed to be "leaky", meaning they had holes in the geometry definition of the .stl files and serpent therefore reported errors/inconsistencies in the geometry. We solved this using FreeCAD for conversion from CQ-generated standard CAD format .step into .stl. In the future we aim to utilize a different meshing library like Gmsh [6].

2.3 Use of STOK

STOK is used through a series of configuration files with which we control the parameters defining the dimensions of reactor components. For example; in Figure 3 is the configuration file used to define the transformer limbs and the spheres next to them. At the top of the file are the general parameters that are usually set once and can stay the same and under the line of hashtags (#) is the configuration specific to each limb individually.

```
# This is the transformer limb manipulation file, you may change any value but do NOT change the layout of the
file. If you do, STOK might not generate properly.
# Under here are the common values to all limbs
# Number of limbs
8

# Limb colour
(0,0,0,1)

# Distance between outer containment radius and the limb
1555.0

# Default sphere radius
500.0

# Offset for (port1-port2)/2 [degrees] (yes or no)
"yes"

##################################################################
# And under here is everything else
# Every new limb is to be set identically (as in the same format) as the first one
# 1:
# Limb length
2000.0

# Limb width
1000.0

# Limb radius (if left 0 the radius is the default value set in the common values section)
0.0

# Limb height (if left 0 it defaults to the containment height)
12000.0

# Limb name (optional if left 0 it will default to generic limb naming)
"Custom limb name"

# 2: ##################
```

Figure 3: Transformer limb configuration file

After we set all the configuration files the generator is run through Python. STOK can optionally insert itself into lines of existing Serpent input files, but that feature is still under development. The idea is that as STOK prepares the folder containing the files describing the

geometry defined by configuration files and it also injects the links to these files and defines the materials of each component directly into Serpent input file.

Each of these elements was created with the other in mind, meaning that if one of the parameters of the input changes, the others change as well. A nice example of this are the containment layers themselves. Let's say that the user originally defined the outer most containment with an outer radius that was too large. In STOK, by changing a single parameter, the whole model can be shrunk for the discrepant amount, saving the time of changing each layer individually.

This comes with the cost of longer base model development times, mainly because every reactor component has to be defined in parametric fashion and then also appropriately positioned relative to the other objects to ensure the model does not overlap and the inconsistencies are not created. However, the benefit of this method is in its ability to either make minor adjustments to the geometry or completely change every parameter of the reactor in a short amount of time. Furthermore, any needs for inclusion of models that are not parametric can be met with the use of functions where parametrically defined geometry can be injected into an existing Serpent input file.

At this point the main focus of the development was on the side of parameter definition, model quality and on making sure that the produced models reliably work with Serpent. We realised that the tool needs improvement in terms of ease of use and in terms of interfacing with other codes to streamline generation of larger number of models. These challenges will be addressed in the future revisions of the tool that will be more focused on interfacing with tools for automation of design optimization or for parametric analyses.

2.4 The model geometry

The model used in testing presented in this paper was that of a simple tokamak with rectangular vertical cross-section, featuring seven layers and one layer of empty space, a central solenoid spanning from the bottom of the outer containment layer to the top of the outer containment layer, eight transformer limbs positioned radially outwards from the centre of the solenoid each being accompanied by two spheres, that represent ex-vessel detectors. Openings were cut in each containment layer that represent ports. These ports are offset from the transformer limbs by twenty-two point five degrees respectively, as shown in Figure 4. The model is based on the simplified JET-like reactor model that has proven to be useful in code testing [7] and in analyses used to better understand the behaviour of detectors in complex simulations such as calibration of JET's neutron yield detectors [8].

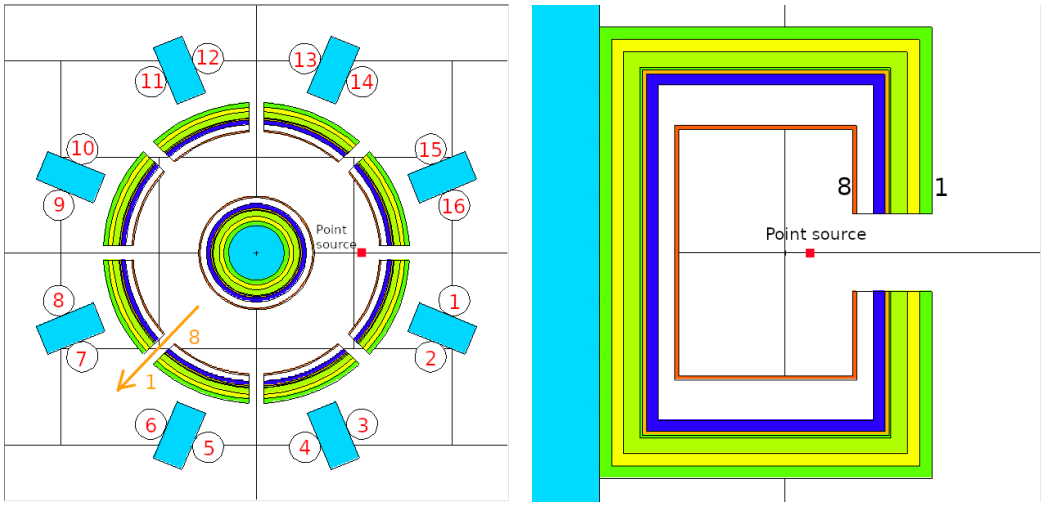

Figure 4a and b: Orientation and numbering of reactor geometry and slice views. Horizontal (left) and vertical (right) cross-section.

Furthermore, the whole reactor is surrounded by a cylindrical model of concrete walls around the reactor and a bounding box that represents the outer boundaries of the model. The walls and the bounding box are not shown in the figure.

These components were used to benchmark the reactor, the results of which can be seen in the next section. Although only the above-mentioned components were used in this first benchmarking, more components were designed and included in the code for later use in more complicated reactor models. Most notable of which are equatorial limiters, that are designed to fill the ports with limiter like structures and a rectangular plasma object that is meant to be used as a neutron source cell in neutron transport codes

3 STOK COMPUTATIONAL BENCHMARKING

3.1 Model complexity analysis

In the STL (.stl) format every object is described with a mesh of triangles that describes the surfaces of the geometry. The accuracy and fineness of this mesh can be fine-tuned depending on the needs of the analyses. Using a basic meshing algorithm one can adjust the maximum angle at which two adjacent edges of the triangle facet can be joined as well as adjusting the deviation between the triangle edge and the real surface edge. In our testing of the effect of the use of different meshing parameter values, we only changed the latter.

In addition to benchmarking the geometry, a comparison of pre-processing and computational time was performed. The geometric model was exported from STOK in meshes of increasing fineness via the STL file format (i.e., the deviation was decreased resulting in more triangles) and run in Serpent with the same settings applied throughout the four test cases. The maximum deviation was set to 1 mm, 0.1 mm, 0.01 mm and 0.001 mm respectively. The second parameter that could be changed, i.e., the maximum value between adjacent edges, was kept at a default value of 0.5235988 radians or 30°.

We observed a steady shift of pre-processing and calculation times between all levels of fineness, as seen in Table 1.

Table 1: Number of triangles to describe the geometry, pre-processing time to export .stl files and simulation time for 108 particles on a dual Intel Xeon Gold 6238R based CPU node.

Edge deviation	**Nr. of triangles**	**Pre-process time [sec] (single core)**	**Simulation time [hours:min:sec] (1 CPU node)**
1mm	104 568	57	1:40:26
0.1mm	880 816	76	2:23:52
0.01mm	8 279 384	435	4:25:26
0.001mm	8 763 424	623	12:37:54

Although these pre-process and simulation times vary significantly, the fluxes, observed in the spherical detectors around the model, are consistent for 0.1 mm, 0.01 mm and 0.001 mm, more detailed description can be seen in Figure 5. Of the tested cases only the 1 mm case shows significantly different results. The origin of this discrepancy is not clear as the effect of component sizes and shapes was not expected to be sensitive to modelling discrepancies of such size. Further investigation is needed to understand this potential issue.

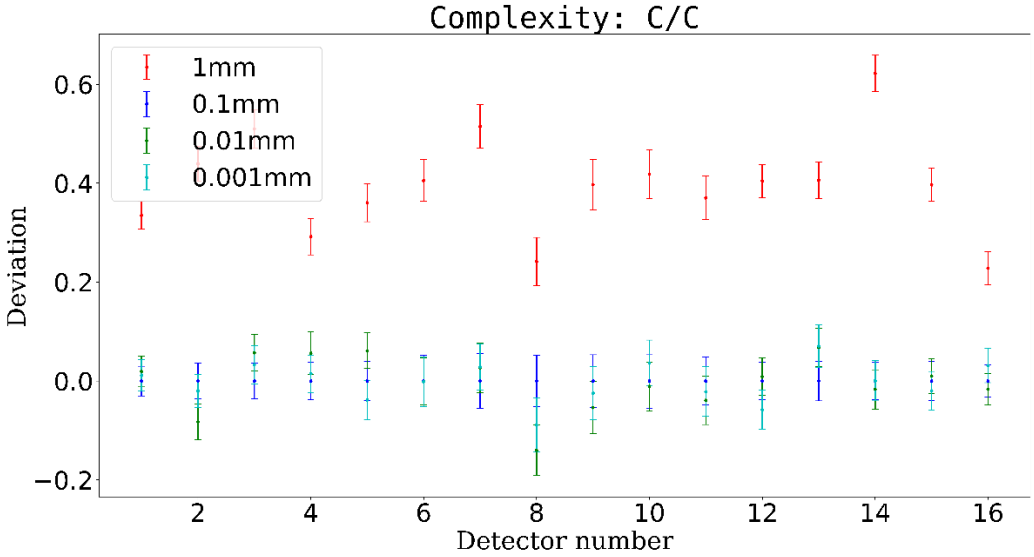

Figure 5: Complexity analysis.

As we can see, the complexity of the .stl file only affects processing and pre-processing time while the results are consistent once the geometry is described in sufficient detail

3.2 Computational benchmarking vs MCNP

Neutron fluxes were calculated using two neutron transport codes, Serpent 2.1.32 [9] and MCNP5 v1.60 [10]. Tallies used were, in MCNP 5, the F4 tally, which calculates track length estimates of the neutron fluence averaged over the cell of interest, and in Serpent a neutron flux detector. These tallies/detectors are made to calculate the same thing i.e., neutron flux.

In both transport codes, the same materials and material mixtures were used, furthermore to minimize a possibility for errors the same cross-section datafiles based on FENDL-3.1 were used. If a nucleus was missing from this library, i.e., some of the contents of the concrete walls, it was substituted for the same element in JEFF 3.3.

Three input files were created:

- MCNP: The first was an input file generated by the SuperMC [11]. Using its .step to CSG converter we were able to recreate the designed model in MCNP geometry.
- Serpent STL: The second of the input files was created in Serpent using the built in .stl import function with STOK generated geometry files. 0.1 mm deviation was used.
- Serpent CSG: The third input file was created by converting the input file of MCNP with a conversion script that converted the MCNP input to a Serpent CSG input. The script was provided by Serpent developers but is not an official conversion tool.

The neutron source used in these analyses was a point source of 14.1 MeV found inside the vacuum vessel as shown by a red square in Figures 4a and b

3.3 Benchmark results

To confirm that the geometry of the model is indeed consistent between CSG and STL, we first had to figure out that the volumes of the detectors are consistent between the two Serpent codes, so we compared them with the same volumes calculated by a CAD program, for this we chose SpaceClaim [12]. The results between SpaceClaim and Serpent were comparable for reference we also did the same with MCNP which yielded comparable results (Figure 6).

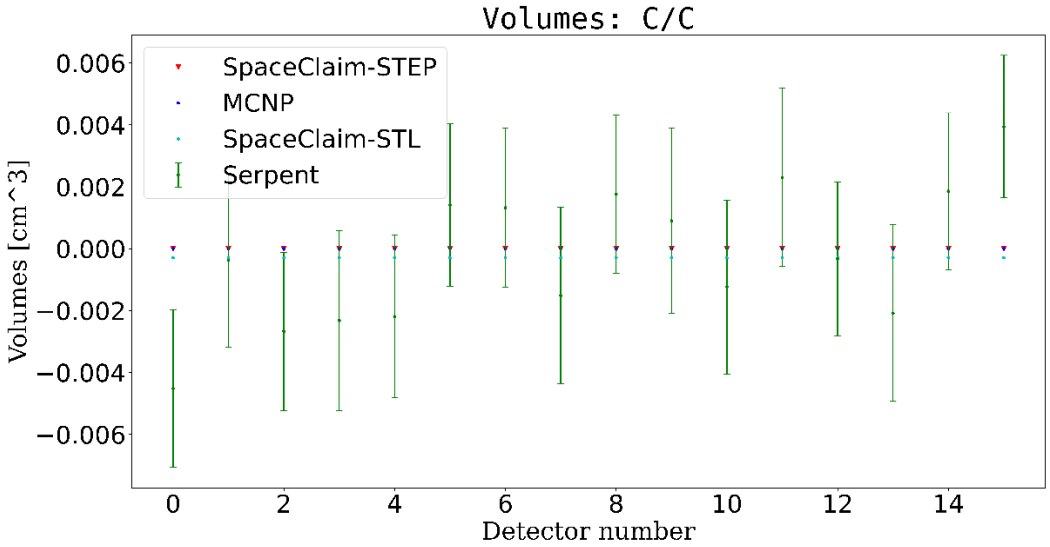

Figure 6: Detector cell volume comparison between SpaceClaim, MCNP and Serpent

The Serpent code does deviate from the norm at the first and last detector but the discrepancy is negligible.

So far, the results of the three code comparisons were inconsistent with differences between the three codes ranging up to 20% for detector positions furthest from the source. This clearly shows inconsistencies in either the geometry generated or in the configurations of the three codes so further investigation is required.

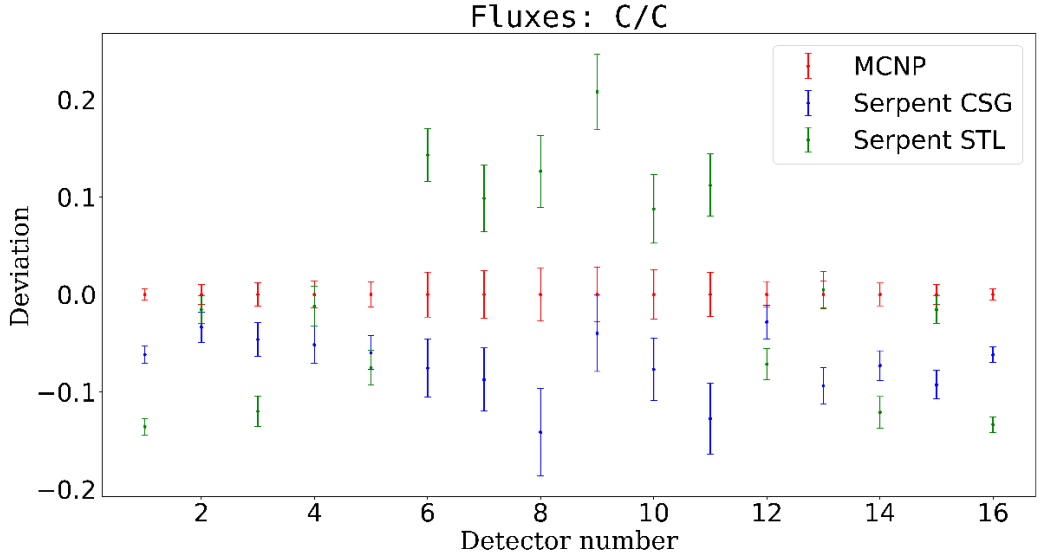

Figure 7: Flux comparison between MCNP, Serpent STL and Serpent CSG

CONCLUSIONS AND FURTHER WORK

Parametric modelling can represent a faster way to produce models for some applications in fusion and fission reactor design but it cannot solve all difficulties with modelling and is not suitable for all modelling needs. However, once a parametric model is created, it can be significantly faster at making small adjustments to the geometry and this process can even be automated. The downside of such modelling is that the time it takes to create such model or a family of models is typically longer than creating a single model in a traditional way.

In this paper we described how a tool called STOK produces simple tokamak models for use in neutronics simulations. We show that increasing the resolution beyond a threshold where a geometry is captured sufficiently does not change the results. However, the computational time required for pre-processing and neutron transport is significantly increased for more complex models featuring larger geometry files with more triangles. At the same time, we observed that decreasing it over a certain edge length of the triangles yields significantly different results in regards to neutron flux. These results are likely less correct so care needs to be taken to make sure the geometry description is sufficiently detailed.

We also observed that the calculated neutron flux shows inconsistencies with MCNP. Further work is needed to find the origin of these discrepancies and try to resolve them

ACKNOWLEDGMENTS

The authors acknowledge the financial support from the Slovenian Research Agency (research project Z2-3201, research program No. P2-0073).

REFERENCES

[1] Python Software Foundation (2001) Python 3.8 source code (Version 3.8.10) [Source code]. https://www.python.org/downloads/release/python-3810/

[2] (2014) CadQuery source code (Version 2.1) [Source code]. https://github.com/CadQuery/cadquery

[3] STOK: A. Gabrijel (2020) STOK source code (Version 0.7) [Source code] https://github.com/A-Gabrijel/STOK

[4] Paramak: J. Shimwell et al., John Billingsley, Rémi Delaporte-Mathurin, Declan Morbey, Matthew Bluteau, Patrick Shriwise, Andrew Davis, The Paramak, https://f1000research.com/articles/10-27.

[5] (2011-2022) OPEN CASCADE SAS https://dev.opencascade.org/

[6] C. Geuzaine and J.-F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[7] A. Čufar et al., (2017) The Analysis of the External Neutron Monitor Responses in a Simplified JET-Like Tokamak Using ADVANTG, Fusion Science and Technology, 71:2, 162-176, DOI: 10.13182/FST16-113

[8] L. Snoj et al., (2013) Calculations to support JET neutron yield calibration: Modelling of the JET remote handling system, Nuclear Engineering and Design, Volume 261, 244-250, ISSN: 0029-5493

[9] Serpent: J. Leppänen et al., The serpent Monte Carlo code: status, development and applications in 2013, Ann. Nucl. Energy 82 (2015), 142-150

[10] MCNP5 1.60: F. Brown et al, Verification of MCNP5-1.60, LA-UR-10-05611, Los Alamos National Laboratory (2010).

[11] SuperMC: Y. Wu, FDS Team, CAD-based interface programs for fusion neutron transport simulation, Fusion Engineering and Design 84 (2009), 1987-1992.

[12] Ansys SpaceClaim, https://www.ansys.com/products/3d-design/ansys-spaceclaim