Automation of Creating High Fidelity TRISO Particle Fuel Element in MCNP Models

Ben Impson, Zachary Crouch, Braden Goddard, Zeyun Wu

Department of Mechanical and Nuclear Engineering, Virginia Commonwealth University,
401 West Main Street, Richmond, Virginia, 23284,
impsonba@vcu.edu, crouchzc@vcu.edu, bgoddard@vcu.edu, zwu@vcu.edu

[leave space for DOI, which will be inserted by ANS]

INTRODUCTION

Pebble Bed Reactors (PBRs) are a type of high temperature reactor whose fuel and design are compatible with molten salt and gas cooled reactors, both of which are part of the generation IV systems currently being developed under the DOE missions [1]. The concept of a PBR was first conceived by Dr. Farrington Daniels of Oak Ridge in 1947 with spheres of graphite imbedded with fissile material being placed in a configuration to sustain a nuclear chain reaction with coolant passing around the spheres. However, it was not until 1966 that a prototype reactor would be constructed to validate Dr. Daniels' conception. This prototype reactor operated for 21 years until it was shut down due to operational problems.

This first prototype PBR was initially fueled using carbide BISO fuel, however, during the last 5 years of its operation, TRi-structural ISOtropic (TRISO) fuel was used. TRISO fuel is a nuclear fuel composed of tiny (~1 mm in diameter) particles of fissile materials, most frequently uranium, coated in layers of cladding materials, usually composed of pyrocarbon and silicon carbide as shown in Figure 1. Thousands of these TRISO particles are then placed into a larger pebble, with the space between and around them being filled with graphite [2]. The considerable amount of layering between the fissile fuel and the outside of each pebble is one of the primary benefits of using TRISO fuel over traditional oxide fuel pellets. The presence of pyrocarbon and graphite keeps fission products contained, preventing them from leaking into the coolant. The other considerable advantage over traditional fuel is its high durability in the face of high pressures and temperatures. While the Zircaloy rods used in pressurized water reactors (PWRs) can only withstand temperatures up to 647 K (374 °C), TRISO fuel is designed to survive upwards of 1800 °C [3]. This makes TRISO fuel ideal for high temperature gas-cooled PBRs. TRISO fuel will also not corrode in the same way Zircaloy will. For reactors like PBRs, this is extremely important.

From the computational perspective, the high level heterogeneous geometric setting in both the TRISO fuel particle and fuel element (e.g., the fuel pebble) in PBR renders great challenges for modeling and simulation of such reactors.

Previous work has explored the effect of randomly packed pebble beds within the reactor, as well as the effect of randomly distributing the particles throughout the pebble, although those prior works primarily focused on molten-salt reactors, not the high temperature gas cooled reactors these pebbles are intended for in this study [4][5]. This research builds on this previous work and endeavors to simplify the process of creating complex input files for radiation transport codes. This paper summaries a few practical difficulties arisen in the course of high fidelity modeling the TRISO fuel pebbles and introduces a computer program based automation technique as a solution to facilitate and accelerate the reactor modeling procedure.

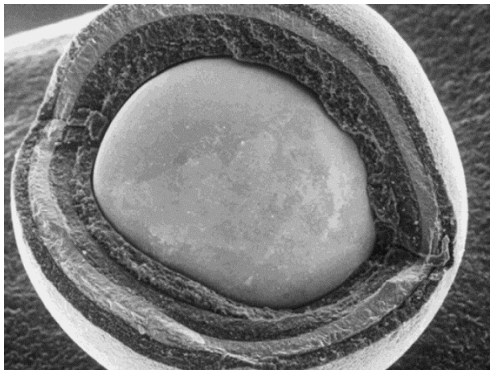

Fig 1. Image of an intentionally broken TRISO Particle revealing the cladding coatings and inner fissile kernel, approximately 1 mm in diameter. [2]

HIGH FIDELITY TRISO FUEL MODEL

Mapping a Pebble

In order to develop nondestructive assay (NDA) techniques to determine the amount of fissile material remaining within the pebble, it is necessary to determine where the TRISO particles are located throughout the pebble and how their dispersions effects NDA techniques. This can be accomplished by using the radiation transport simulations code Monte Carlo N-particle version 6.2 (MCNP6.2) [6]. While making a repeating lattice structure in MCNP6.2 is relatively straightforward (see Figure 2), this simplistic approach creates features that are not present in real pebbles.

For instance, particles at the edge of the fuel region of the pebble would be cut, resulting in fractional particles. Particles are also perfectly aligned in a lattice grid while in real pebbles the particles are randomly distributed. These challenges can be corrected by manually changing geometry parameters in the MCNP6.2 input deck, although this manual process is time consuming. To reduce manual monotonous work, a computer script was written in MATLAB that removes cut particles and will allow for a more randomized distribution of particles. For future expanded applications, the script was then adapted into Python.

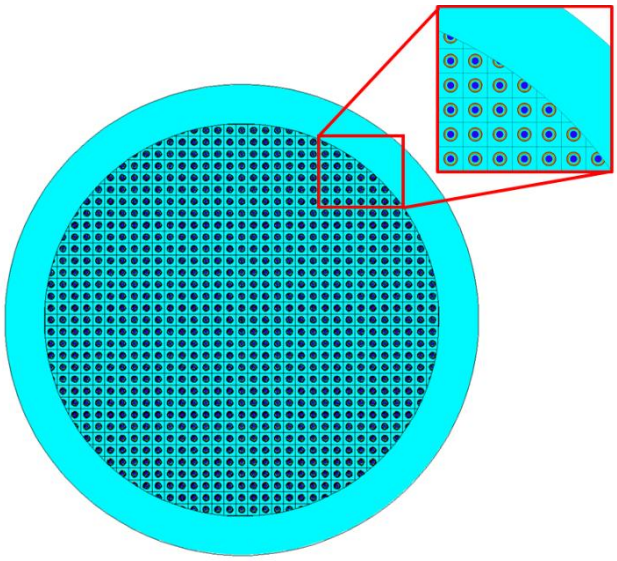

Fig 2. Cross-sectional image of a MCNP6.2 model of a pebble with a simple lattice of TRISO particles showing cut particles at the edge of the lattice.

Validation of MATLAB Script

An Excel spreadsheet was manually created that would make the lattice distribution section of an MCNP6.2 input that did not contain any cut particles. This spreadsheet served as both validation of the MATLAB script as well as a template for the output of the MATLAB script to follow. The script assigns the number '1' for lattice positions with particles and '2' for lattice positions without particles (formally cut particles). The script consists of three nested 'for' loops, each one tied to x, y, and z coordinates within the pebble.

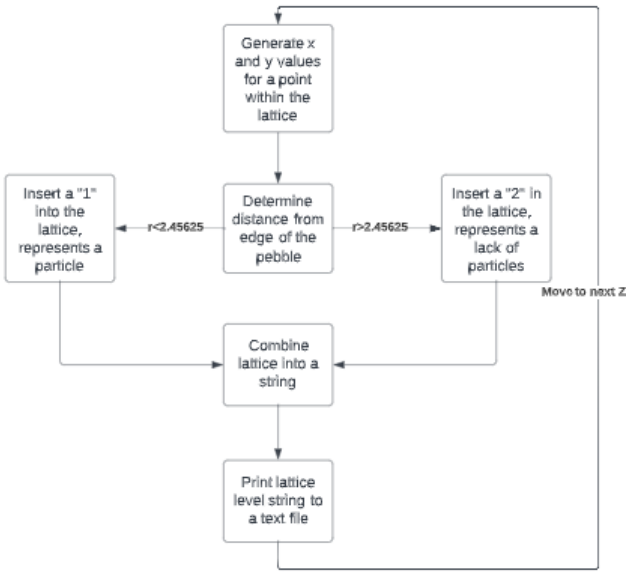

Fig. 3. Program Flow Chart

The MCNP6.2 model has a lattice range from -17 to 17 in each direction, however MATLAB would not accept negative numbers as indices. As such, the script instead ranges from 0 to 35, compensating for the perturbation by subtracting 17 from each x, y, and z value. These corrected values were then used to calculate the distance from that position to the edge of the pebble. If the radius was greater than 2.45625 cm, there would be a '2' placed in the lattice to represent a lack of a particle. Otherwise, a '1' is placed in lattice. The eventual output is 35 lattices, each representing a different level on the z plane. The script exports these lattice value to a text file (Figure 3), where they are separated by a 'C' to allow for better readability. The script also allows rapid changes to be made to the model itself. As the parameters, such as the dimensions or how the lattice is broken down, of the pebble are functionally inputs of the MATLAB script. If the size of the pebble or the number of TRISO particles changes, it is easy and quick to input the new parameters into the MATLAB script and generate a new MCNP6.2 input file. Each run of the script takes approximately 30 seconds with the largest variation resulting from larger and larger lattices, allowing for rapid adjustments and improvements to be made.

Randomly Distributed Particles

The process of adapting the script into Python was aided by the use of the ChatGPT AI system. While it was not flawless in its advice, it allowed for much faster progress than attempting to write a program almost completely from scratch. Once the original MATLAB output had been fully recreated using Python, the parameters were modified to create semi-randomly distributed particles. Python was selected for this task due to its ease of use and acquisition.

While the Serpent Monte Carlo program is capable of creating a randomly packed pebble [7], it is export controlled and more difficult to acquire for some nationalities.

The pitch between the particles in the lattice was reduced to the diameter of the particles, 0.0855 cm, and the dimensions of the lattice were increased to be 58 by 58 by 58 cells, resulting in the original diameter of the pebble. The parameters for particle determination were not changed, which caused an extremely large number of particles to be created, far more than the 19,000 that would exist in a normal pebble. The inflation in particles allowed for a number of them to be removed from the lattice to reach 19,000. By using a random number generator and an If-loop in Python, these removals could result in randomly distributed particles. The original run of the program creates approximately 100,000 particles, so removing 81% results in an accurate pebble. When run on an average laptop, this script takes approximately 15 seconds. Figure 3 shows an MCNP model cross sectional view of a pebble with the particles randomly distributed within the lattice.

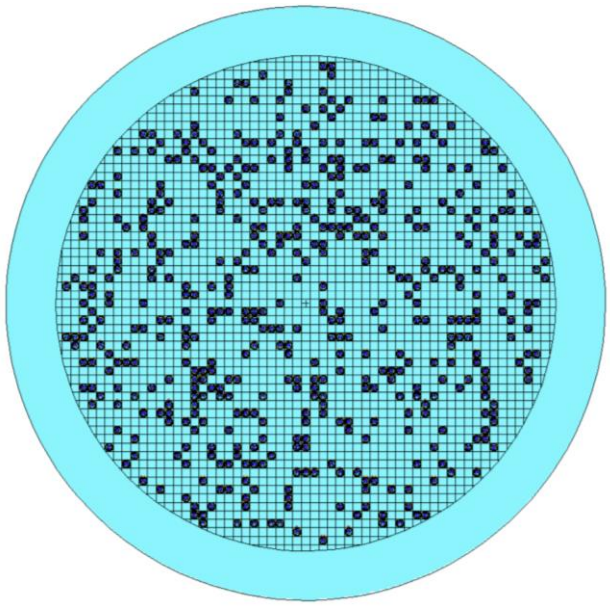

Fig 3. MCNP model cross sectional view of a pebble with the particles randomly distributed within the lattice as created by the Python script.

RESULTS

This process has resulted in a script that allows for rapid remodeling of a TRISO particle filled pebble. Rather than needing to manually create spreadsheets, a simple MATLAB script can complete the task in seconds. It demonstrates that simple programs can generate complex outputs. Those outputs can then be used for further examination in MCNP6.2 simulations for eventual assessment of NDA techniques. This is an essential step in moving towards the end goal of precisely determining when a pebble no longer contains sufficient fissile material to be reinserted into the core of a PBR.

It also demonstrated the benefit of researchers learning multiple programming languages. While most programming languages are useful, some are more applicable to certain tasks than others. For instance, in this research, it would have taken far longer for the random distribution to be developed in MATLAB rather than Python.

FUTURE DEVELOPMENT

The next step in this process will be to run burnup calculations in MCNP6.2 on different distributions of particles. The effect of different particle locations on the effective multiplication factor is important for determining the remaining fuel in a pebble. Both a random and homogenous model pebble were created in the process of this research, but these are not the limits of the geometries. There are plans in place to create models where the particles are either concentrated in the center of the pebble, or are only along the edge of the particle. Once these are created, the effective multiplication factor will be compared for all four models.

ACKNOWLEDGEMENTS

This work is performed with the support of U.S. Department of Energy's Nuclear Energy University Program (NEUP) with the Award No. DE-NE0009304.

REFERENCES

1. Generation IV Systems, Gen IV International Forum, Retrieved January 3, 2023, from https://www.gen-4.org/gif/jcms/c_40465/generation-iv-systems
2. United States Department of Energy, Office of Nuclear Energy, "TRISO Particles: The Most Robust Nuclear Fuel on Earth", July 9, 2019, Retrieved December 20, 2022, from https://www.energy.gov/ne/articles/triso-particles-most-robust-nuclear-fuel-earth.
3. Loza, P. and Staden, M. V., "Submittal of Xe-100 Topical Report: TRISO-X Pebble Fuel Qualification Methodology", X-Energy, Revision 2, (2021)
4. Li, Z., Cao, L., Wu, H., He, Q., and Shen, W., "On the improvements in neutronics analysis of the unit cell for the pebble-bed fluoride-salt-cooled high-temperature reactor," Progress in Nuclear Energy (2016)
5. Auwerda, G. J., Kloosterman, J. L., Lathouwers, D., and Hagen, T. H. J. J. van der, "Effects of random pebble distribution on the multiplication factor in HTR pebble bed reactors," Annals of Nuclear Energy (2010)
6. C.J. Werner, J.S. Bull, C.J. Solomon, et all., "MCNP6.2 Release Notes," LA-UR-18-20808 (2018)
7. V. Rintala, H. Suikkanen, J. Leppänen, R. Kyrki-Rajamäki, "Modeling of realistic pebble bed reactor

geometries using the Serpent Monte Carlo code,”
Annals of Nuclear Energy, vol. 77, pp. 223-230 (2015)