Providing a Graphical Tool for Modeling Reactor Cores

Patrick O’Leary, a Jacob Becker, a Robert O’Bara, a David Thompson, a Rajeev Jain, b Vijay Mahadevan b

a Kitware Inc., 28 Corporate Drive, Clifton Park, NY 12065, bob.obara@kitware.com
b Argonne National Laboratory, 9700 S. Cass Avenue, Argonne, IL 60439

INTRODUCTION

Complex, coupled physical systems that occur in various domain sciences present several serious challenges in implementing an efficient and seamless simulation workflow. These challenges impact the productivity of scientific research and the overall time it takes to verify simulation results. To overcome these computational modeling barriers, specifically in nuclear reactor analysis, one must rely on flexible and scalable tools that simplify modeling complexity.

The geometry model of the reactor core can consist of a large number of volumetric regions and surface boundaries. For example, an advanced burn test reactor (ABTR) core can have over twenty thousand volumes and over a hundred thousand surface boundaries. Many simulations require a discretized representation of the geometry that can be made with a variety of meshing tools such as CUBIT [1] and MeshKit[2]. CUBIT is a hexahedral mesher that is developed by Sandia National Laboratories, and MeshKit is part of the Scalable Interfaces for Geometry and Mesh-based Applications (SIGMA) toolkit [3].

This summary presents a modeling/visualization tool that aims to simplify the preprocessing workflow tasks for various reactor models (light-water reactors, or LWRs, and sodium-cooled fast reactors, or SFRs) [4,5]. The techniques employed by this tool provide the abilities to model complex cores, simplifies the process of assigning material and boundary specifications, and interactively visualize their three-dimensional (3D) geometry.

With the above motivations to simplify computational workflows, we present the development of innovative techniques within an open-source tool: Reactor Geometry (and mesh) Generator (RGG) [6]. RGG simplifies the traditional reactor modeling process and the mesh generation of the core. The tool uses Qt for the two-dimensional (2D) user interface components, and it uses the Visualization Toolkit (VTK) [7] to visualize the core in 3D.

Modeling Reactor Core Geometry

Nuclear reactor core geometry can have a two-level hierarchy of lattices. The first level of the hierarchy corresponds to fuel assemblies. It is formed as a lattice of cylindrical pins. In the second level, assemblies of various types are arranged in a lattice to form the reactor core. These pins typically contain fuel, absorption material for controlling the nuclear chain reaction, or instrumentation [8]. The surrounding materials can function as coolant, as a neutron energy moderator, or simply as supporting structure. Fuel assemblies, pins, and material definitions can be identical between similar reactor core models.

Two layout strategies are commonly used to place pins within a reactor assembly or to place assemblies within a reactor core. The first strategy employs a rectangular lattice. It is used primarily in water-cooled reactors. (Fig. 1) The second strategy is a hexagonal-based approach. It is used mainly in the design of sodium and gas-cooled reactors. (See Fig. 2)

Fig. 1: (a) 2D schematic view of the full PWR core model. (b) 2D schematic view of one of the fuel assemblies shown in blue in (a).

Fig. 2: Geometric representation of the ABTR core, which depicts a core composed from a number of complex assemblies.

Designing Pins

The first stage involves designing the individual pin cells that will be used in the reactor assemblies. These pins represent fuel, control rods, and various instrumentation. The pin cell editor models a pin cell as a collection of segments, which can either be cylinders or conical frustums, as Fig. 3 demonstrates.

By default, the editor constrains neighboring segments to have the same mating radii. If the end user changes one
segment’s radius, all constrained radii are automatically updated. In addition, the pin editor allows the end user to define contiguous layers throughout the pins. Each layer is assigned its own material. Each material is presented as a radial percentage, which allows the actual radius of the material to be updated when the overall pin radius is modified. In Fig. 3(b), 80% of the pin is composed of fuel, while the outer 20% is cladding.

Fig. 3: (a) Pin cell editor modeling a fuel pin cell that is composed of a cylindrical segment and a frustum segment. (b) 3D view of the current pin defined by the pin cell editor in (a). (c) Cutaway view of the same pin, which shows how materials vary along the pin’s major axis.

Designing Ducts
The second stage involves designing the ducts, which represent the volumes of assemblies that are not occupied by reactor pins. Each assembly consists of a set of ducts that will later contain the pins. As with pins, ducts can be composed of several segments. Unlike pins, each segment of a duct can have the same overall cross-section, yet they can have different layers and materials. Fig. 4 depicts the RGG duct editor as it computes a duct with two segments.

In MeshKit, the overall height of the ducts and pins must be the same across all the core assemblies. To enforce this requirement, the end user can set the overall core length, which is then enforced through the workflow. RGG hides this by performing auto splitting, which ensures that seams are matched. Similar to the pin cell editor, the duct editor provides end users with the ability to specify the duct structure of each assembly.

Fig. 4: (a) Duct editor, which shows a duct composed of two segments. The blue area represents the coolant material (in this case sodium), the pink represents the lower reflector material, and the green represents the outer duct material. (b) 3D view of the duct that is defined by the duct editor in (a). (c) Cross-sectional view of the duct.

Designing Assemblies.
In the third stage, the pins and ducts are available for placement within an assembly lattice, as shown in Fig. 5. In addition to specifying general information such as the size of the pin lattice, the stage specifies whether pins should be automatically centered within the lattice or offset by an amount defined as the pitch. The assembly editor provides both a 3D view of the assembly and an editable 2D schematic view. The end user can copy pins within the lattice by selecting and dragging pins.

Fig. 5: One of several reactor assemblies used in the design of the ABTR core, which is defined in RGG. (a) Assembly editor. (b) 2D schematic view. (c) 3D view.

Core Layout
In the fourth stage, the complete core is defined by placing the different assemblies into the core’s lattice. Similar to the assembly editor, the core editor view provides a 3D view of the core and editable 2D views of the lattice. Fig. 6 depicts the core design process in RGG.

As depicted in the schematic views of Fig. 6(a) and Fig. 5(b), nuclear reactor cores and assemblies exhibit a fair degree of symmetry. To exploit this, the RGG user interface provides various options for smart placement to help the end user positions pins within an assembly or assemblies within a core. Fig. 7 shows the various layout patterns the interface provides for hexagonal cores and assemblies: column, row,
diagonal, ring, and circle. In addition, the interface provides both ring and circle options for relative placement. These options are based on a user-specified center.

As shown in Fig. 8, when placing pins within the assembly, the interface can detect interference between an existing pin and one that is being placed. In this example, a control pin (CP) is copied from an existing one. The greyed-out cells indicate invalid placements for the pin due to the existing pin and the outer boundary of the assembly.

**PERFORMANCE RESULTS**

Explicitly modeling the geometry of every individual pin and duct in a reactor core would consume a large amount of system memory, and it would adversely impact rendering performance on the graphical processing unit (GPU). As a result, RGG places only a small number of unique geometric structures within the assembly lattice that are then “instanced.” It uses the same process when placing the assemblies within the core lattice.

![Fig. 8: Smart layout tools. The schematic views highlight invalid placements of pins in an assembly.](image)

Using VTK’s capabilities in glyph mapping, RGG sends only the unique geometry of the pins and ducts to the GPU. This information is then instanced on the GPU. Next, the appropriate linear transformation is applied. The transformation maps the native coordinate system of the geometry to the coordinate system of the placed instance. This technique allows RGG to model complex reactor cores while maintaining high framerates.

**INTEGRATION WITH ADVANCED SIMULATION WORKFLOWS**

RGG has been successfully interfaced with reactor modeling and meshing workflows based on MeshKit. MeshKit provides a command line tool, AssyGen, which generates solid models and meshes of the assemblies that are defined within the core. This is achieved by generating journal files that are then processed externally by CUBIT. Finally, CoreGen, which is another command line tool that is found in MeshKit, creates a mesh of the core by instancing the appropriate assembly meshes and stitching the meshes together where they come in contact. The RGG tool was extended to allow information to be specified such as the size of the mesh and details of the boundary layer. Fig. 9 shows an example of RGG being used with MeshKit and CUBIT.
Fig. 9: An example of the RGG tool being used in a MeshKit-based workflow. a) 3D geometry view showing a preview of boundary layer. (b) Mesh view with the final mesh, which consists of boundary layers.

This functionality is also currently being refactored so that it can be used in other workflows such as those based on the NEAMS Workbench framework [9]. In order to accomplish this, the Simulation Modeling Toolkit (SMTK)[10], is being extended to support the RGG workflow. SMTK is an open source toolkit that provides access to geometric models, meshes, and simulation information as well as operations that can modify these resources. In terms of modeling functionality, SMTK interfaces with both discrete and parametric modeling environments such as OpenCASCADE[11].

Fig. 10: SMTK Reactor Core Modeling plugin being used in CMB’s Model Builder Application.

These “sessions” can then be accessed as plugin modules within the application. A reactor modeling session is being developed that captures the functionality provided by the RGG tool including reusable GUI elements for visualizing and editing various reactor core components. Fig. 10 shows the current version of the SMTK reactor core plugin being used inside of Computational Modeling Building (CMB)[12] application.

REFERENCES