iMC GUI: An Integrated Environment for Input Authoring and CSG Geometry Visualization for Monte Carlo Simulations

Sangjin Lee [a], Yonghee Kim [a*]
[a]*Department of Nuclear & Quantum Engineering, Korea Advanced Institute of Science and Technology (KAIST), 291 Daehak-ro, Yuseong-gu, Daejeon 34141, Republic of Korea*
[*]*Corresponding author: yongheekim@kaist.ac.kr*

Keywords : Monte Carlo simulation, particle transport, constructive solid geometry, graphical user interface

1. Introduction

Constructive solid geometry (CSG) is the standard geometry representation in many Monte Carlo transport codes because it is compact, expressive, and well suited to repeated structures. At the same time, these same advantages can make a large model difficult to understand when the only source of truth is a text input. To address this problem, several research groups have developed graphical user interfaces (GUIs) for geometry viewing, including Gxsview [1] and its incorporation into PHIG-3D [2], SimpleGeo [3], and the plotting tools developed by the OpenMC team [4]. Across these efforts, the same practical need appears repeatedly: users need a workflow in which CSG geometry can be authored and checked visually with minimal friction, and that workflow must support repeated structures such as nested universes and lattices, which are essential in many particle transport geometries.

The iMC team at KAIST is developing a GUI in which input authoring and geometry visualization are integrated into a single interactive workspace. Unlike tools that are used primarily as separate visualizers for preconfigured input files, the iMC GUI is designed to support input construction itself, so that changes made by the user are reflected in real-time in both 2-D and 3-D geometry views. The GUI is implemented in C++ using Qt 6 and VTK [5], and the CSG definitions and geometry routines are taken directly from the Fortran iMC core library to minimize duplication and maintain consistency with the simulation code. This paper focuses on that integrated authoring and visualization workflow.

2. Interactive Input Authoring

The iMC workflow is organized around a project folder that contains one or more input folders. Each input folder stores the standard XML inputs for iMC (variables.xml, settings.xml, materials.xml, geometry.xml, and tallies.xml). In the GUI, these files are not presented as a raw XML syntax tree, and instead, the user sees a cleaner navigation tree with component-specific labels such as Universe, Lattice, Cell, and Material, and edits the selected item through a property panel that shows only the fields relevant to that item (Figure 1).

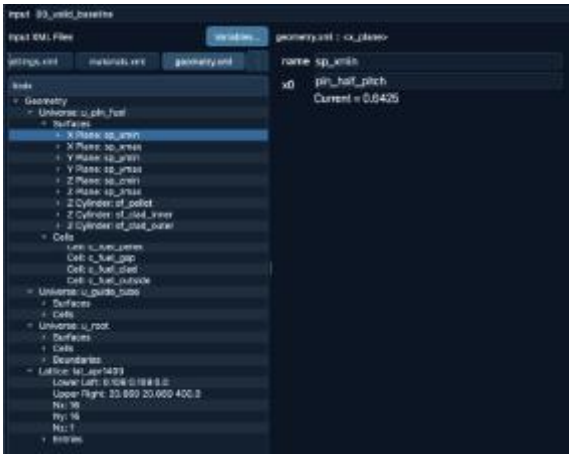

Figure 1. Input navigation tree and property editing panel for a sample input.

A variables window allows the user to add custom parameters, and built-in constants such as pi and e are also available. The numeric fields in the various input files are filled with an expression string that can store either a literal value, a variable, or a math expression mixing literals and variables, and the final scalar value is computed and stored.

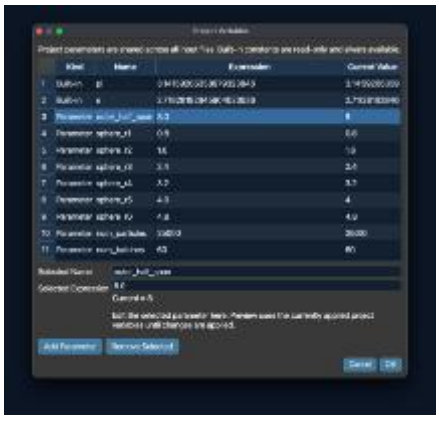

Figure 2. Variables window for a sample input.

Figure 2 illustrates the project-wide variable workflow. The window shows shared values across all input files in one place, distinguishes built-in constants from user-defined variables, and displays both the authored expression and the currently evaluated value. This makes shared dimensions and run-control values much easier to manage than treating variables as a hidden side file.

On the geometry side, the authoring model is designed specifically for repeated structures that are common in transport calculations. A cell_universe defines a reusable local geometry. A universe_lattice places those universes on a regular 3-D grid, with nz=1 used for nominally 2-D lattices. Each lattice node places the child universe at the node center by default, and the user may then add an extra translation and rotation for that entry. This makes it practical to define one reusable universe and place it many times while still handling more complex repeated features such as guide tubes that span multiple lattice cells.

To make that workflow usable, lattice editing is done in a dedicated lattice editor. The editor presents a large clickable node grid and per-node transform controls, which makes repeated-structure editing much clearer than working directly in raw XML.

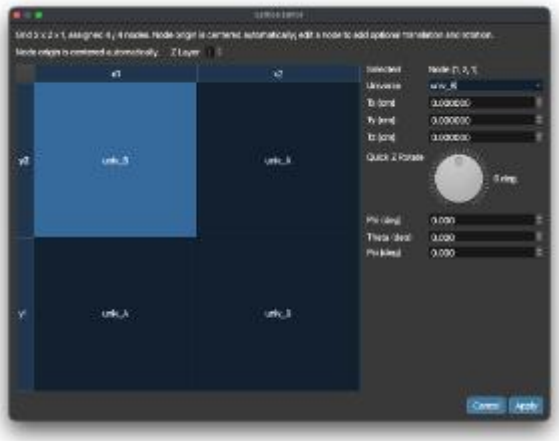

Figure . Lattice editor for a sample input with a 2x2 lattice.

Figure 3 shows the repeated-structure authoring workflow in a small sample case. The 2x2 checkerboard pattern makes the alternating placement of two pin cell universes, univ_A and univ_B, easy to see at a glance, while the panel on the right shows the selected node's universe choice together with its additional translation and rotation (none in this case). Even in this simplified example, the same rule is visible: the reusable local geometries are defined once, then placed through the lattice and adjusted only where an extra local transform is needed.

3. 2-D Geometry Visualization

The 2-D geometry visualization mode enables fast, exact geometry visualization on planes of interest. It begins by tracing rays across the selected geometry plane, and it uses the same Fortran geometry engine as the transport simulation. Starting from the left edge of the selected plane, rays are simulated to traverse the geometry, assuming void materials until the other side is reached. For each image row, the core finds contiguous intervals that remain in the same cell, instead of testing every pixel independently. Those row segments are then returned to the GUI and converted into the displayed image.

This approach is much faster than a naive point-by-point containment loop, and it is well suited to quick, interactive scrolling through the model. The GUI renders the image asynchronously, keeps a cache of recently visited positions, and reuses the cached geometry information when the user moves the slider back and forth. The same cached image can also be recolored immediately when the user changes the coloring method, for example from coloring by material to coloring by cell.

Because the 2-D view is computed directly from the geometry equations and drawn with correct physical proportions, the CSG definition is displayed exactly.

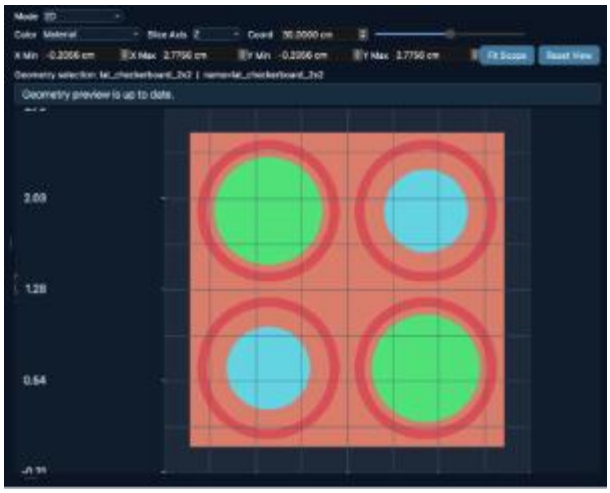

Figure . A 2-D plane generated for the sample 2x2 lattice input.

Figure 4 is a focused screenshot of only the geometry-viewer region for the 2-D image computed for the sample 2x2 lattice input mentioned in Section 2.

Figure 5 shows the same 2-D viewing method in the full GUI workspace for a sample input with concentric spheres of various sizes. The colored rings are the intersections of the spherical shells with the selected Z plane, and the surrounding square region is the finite outer box that bounds the model. This figure also shows the nearby controls used to choose the viewed plane,

move the displayed coordinate, and adjust the visible x/y range.

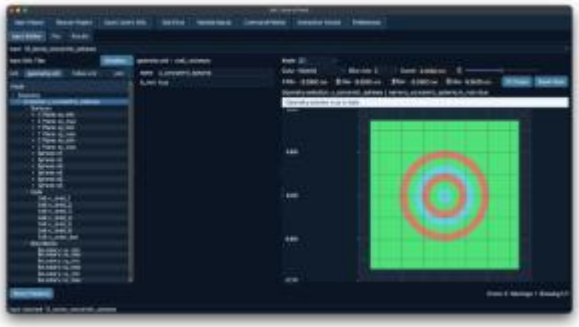

Figure . The GUI window showing the geometry viewer displaying a 2-D plane generated for the concentric spheres sample input.

4. 3D Geometry Visualization

The 3-D mode complements the 2-D view by showing the same geometry in a form that can be rotated, panned, and zoomed in real time. Its purpose is still visual geometry inspection, but it allows the user to understand overall shape, depth, and repeated placement in ways that are not possible in the 2-D geometry view alone.

The current 3-D mode does not display the analytic geometry directly. Instead, it builds an approximate polygonal surface representation from the same CSG definition so that the model can be explored interactively. First, the selected geometry region, such as the root universe, a selected universe, or a lattice region, is collected into a temporary structured description that includes any nested universes and lattice transforms. Next, the supported surfaces are converted into VTK classes that represent analytic surfaces, such as vtkPlane for planes and vtkQuadric for spheres, cylinders, cones, and general quadrics. The iMC code then builds the corresponding Boolean CSG logic from the cell expression.

After that, vtkSampleFunction, which is a VTK class for sampling an implicit field on a 3-D grid, samples the resulting CSG field inside a finite 3-D box covering the selected region. Then vtkFlyingEdges3D, which is a VTK class for extracting a polygonal surface from that sampled field, generates the displayed surface. In practical terms, the 3-D view shows a surface made of polygons generated from the underlying CSG description. It is therefore not a fully exact analytic renderer, but it is a stable and useful real-time representation of the authored model.

VTK provides the analytic surface classes, the sampling step, and the surface-extraction step, but the actual CSG meaning is completed by iMC. The iMC code interprets the signed-surface cell expression, builds the Boolean CSG logic, applies nested-universe and lattice transforms, and defines the finite region that is sampled for display.

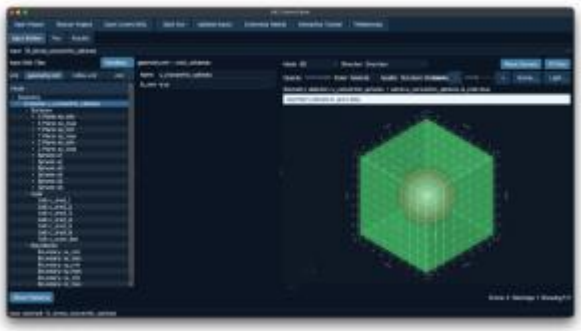

Figure . The GUI window showing the geometry viewer displaying a 3-D view for the concentric spheres sample input.

Figure 6 shows the same example in the full GUI window, now in the 3-D overview direction. The nested shell surfaces appear inside the finite bounding box, while the bounds grid and the small orientation marker provide spatial reference during rotation. Because the entire workspace is visible, the figure also shows the practical 3-D workflow around the viewer: direction selection, polygon resolution, zoom, fit/reset camera, and scene or lighting controls remain available without leaving the geometry workspace.

5. Conclusions

The current iMC GUI combines structured input authoring with immediate geometry feedback in one workspace. Its main contribution is the tight connection between a Fortran-owned single source of truth for the input and geometry model and a Qt/VTK frontend that keeps authoring, validation, GUI visualization, and CLI behavior aligned.

For practical transport modeling, the most important results are that repeated structures can be authored directly, the 2-D mode provides exact ray-tracing-based geometry checking, and the 3-D mode provides an interactive approximate polygon-based representation of the same CSG model. Together, these give the user both precise checking and broader spatial understanding.

At present, the GUI acts as an interactive input builder, visualizer, and validator. A natural next step is to extend it into a complete graphical interface for the full simulation workflow, including job launching, run control, live run monitoring, and viewing, postprocessing, and plotting of simulation results.

ACKNOWLEDGMENTS

This research was supported by the National Research Foundation of Korea (NRF) grant funded by the Korean Government (MSIP) (2021M2D2A2076383) and the Korea Energy Technology Evaluation and Planning (KETEP) grant funded by the Korean Government (MTIE) (RS-2024-00439210).

REFERENCES

[1] S. Ohnishi, "Gxsview: Geometry and cross section viewer for calculating radiation transport," *SoftwareX*, vol. 14, p. 100681, Jun. 2021, doi: 10.1016/j.softx.2021.100681.

[2] T. Sato *et al.*, "Recent improvements of the particle and heavy ion transport code system – PHITS version 3.33," *Journal of Nuclear Science and Technology*, vol. 61, no. 1, pp. 127–135, Jan. 2024, doi: 10.1080/00223131.2023.2275736.

[3] C. Theis *et al.*, "SimpleGeo – New Developments in the Interactive Creation and Debugging of Geometries for Monte Carlo Simulations," *Progress in Nuclear Science and Technology*, vol. 2, no. 0, pp. 587–590, Oct. 2011, doi: 10.15669/pnst.2.587.

[4] P. K. Romano and B. Forget, "The OpenMC Monte Carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274–281, Jan. 2013, doi: 10.1016/j.anucene.2012.06.040.

[5] W. J. Schroeder, K. Martin, W. E. Lorensen, L. S. Avila, K. W. Martin, and W. E. Lorensen, *The visualization toolkit: an object-oriented approach to 3D graphics ; [visualize data in 3D - medical, engineering or scientific ; build your own applications with C++, Tcl, Java or Python ; includes source code for VTK (supports UNIX, Windows and Mac)]*, 4. ed. Clifton Park, NY: Kitware, Inc, 2006.