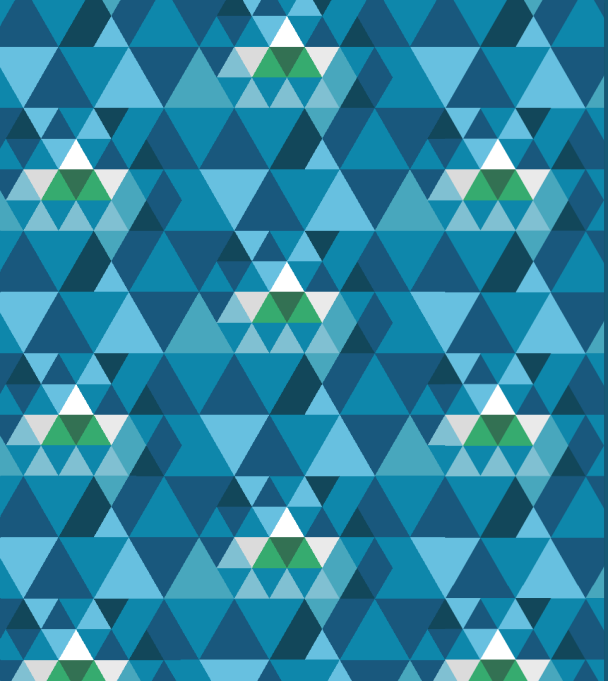


On the performance of the particle tracking routine in Serpent 2

11th International Serpent UGM

Garching, Germany, Aug. 29 – Sept 1, 2022

Jaakko Leppänen, Research Professor, VTT

Outline

Main topics:

- Universe-based geometry model
- Performance of the CSG geometry type
- Advanced geometry types
- Tracking modes in Serpent

Universe-based geometry model

Serpent features a universe-based geometry type:

- A universe is a geometry construct that fills the entire space
- Universes can be placed inside each other, to create multi-layered geometries

Universe types available in Serpent 2:

- CSG universe, constructed from Boolean combinations of elementary and derived surface types
- Nests, constructed of regions nested inside each other (“pin” is the most common subset of this type)
- Lattices: square, hexagonal, circular array, ...
- Explicit HTGR particle fuel / pebble bed geometry type
- Unstructured mesh-based geometry type (OpenFOAM mesh format)
- CAD-based geometry type (STL file format)
- Voronoi-tessellated stochastic geometry type (to become available in the next update)

All universe types share many of the same properties, and can be combined in an arbitrary way.

Basic functions: Where am I and how far away is the boundary?

VTT

One of the fundamental functions of the geometry routine is to figure out the position (x, y, z) where the particle is located at:

- Recursive routine that passes through the levels of the geometry until a material cell is found
- Different algorithms for different universe types
- CSG cell search loops over all cells in the universe and performs tests for the combination of the constituent surfaces
- Performance of the algorithm depends on the number of cells per universe and the complexity of the cells

Another important function is to figure out the distance to the nearest material boundary from (x, y, z) in the direction of motion (u, v, w):

- The CSG routine calculates the distance to all surfaces that make up the cell and picks the shortest (positive) distance
- Performance of the algorithm depends on the number of surfaces per cell

Basic functions: Where am I and how far away is the boundary?

Algorithm 1 CSG cell search routine (with intersections only)

1: **for** $m \leftarrow 1$ to M **do** ▷ Loop over candidate cells
2: **for** $n \leftarrow 1$ to N_m **do** ▷ Loop over intersection list
3: $c \leftarrow S_n(x, y, z)$ ▷ Call surface test routine
4: **if** $S_n(x, y, z)$ is flagged with '\' **then** ▷ Check complement flag
5: $c \leftarrow \backslash c$ ▷ Apply complement operator
6: **end if**
7: **if** $c = \mathrm{FALSE}$ **then** ▷ Check condition
8: Break loop ▷ Point is outside, test next cell
9: **end if**
10: **end for**
11: **if** $n = N_m$ **then** ▷ Check if all surfaces passed the test
12: **return** m ▷ Point is inside cell m
13: **end if**
14: **end for**
15: **return** ERROR ▷ Geometry error: no cell at (x, y, z)

Basic functions: Where am I and how far away is the boundary?

Algorithm 2 Cell search with multiple universes and lattices

1: $U \leftarrow U_0$ ▷ Start from root universe
2: **while** TRUE **do** ▷ Loop over universes
3: $m \leftarrow C(U, \boldsymbol{r})$ ▷ Call cell search routine for universe U (algorithm 1)
4: **if** cell m is a material cell **then**
5: **return** m ▷ Point is inside cell m
6: **else if** cell m is filled with universe U' **then**
7: $U \leftarrow U'$ ▷ Update universe
8: **else if** cell m is filled with lattice l **then**
9: $(U', \boldsymbol{r}_0) \leftarrow L(l, \boldsymbol{r})$ ▷ Get lattice cell universe and origin
10: $\boldsymbol{r} \leftarrow \boldsymbol{r} - \boldsymbol{r}_0$ ▷ Update coordinates
11: $U \leftarrow U'$ ▷ Update universe
12: **else**
13: **return** m_o ▷ Point is outside the geometry
14: **end if**
15: **end while**

How to speed up the CSG cell search routine?

Serpent uses a number of tricks to improve the efficiency of the basic CSG cell search routine:[1]

- Sorted search list for each universe that puts most likely cells on the top
- Sorted surface list for intersection cells that puts the surface most likely to fail the test on the top

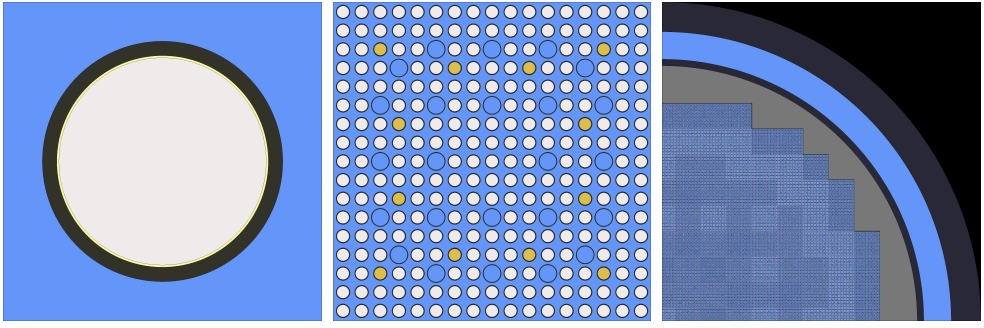
How can the user improve the efficiency?

- The best way is to sub-divide the geometry into separate universes, so that the number of cell tests at each level is minimized
- The same approach may not be optimal for other codes (MCNP), which is why geometries converted to Serpent format often run very slow

For a complex geometry, the way the cells are structure can have factor of 10 difference in running time!

[1] There are other tricks, such as neighbor search, that have been tried, but have not produce significant improvement in efficiency.

Tracking in lattices



Lattices are a very efficient way to represent regular structures:

- Automatic division into sub-universes
- Loop over cells replaced by a table look-up based on lattice indexes

Using lattices to define regular structures not only makes the input easier for the user, but also leads to optimal performance of the geometry routine.

Advanced geometry types

Some Serpent geometry types rely on a search mesh to speed-up the geometry routine:

- Explicit HTGR particle fuel / pebble bed geometry type
- Unstructured mesh-based geometry type (OpenFOAM mesh format)
- CAD-based geometry type (STL file format)
- Voronoi-tessellated stochastic geometry type (to become available in the next update)

The idea is similar to that in a regular lattice:

- Search mesh is a regular structure laid on top of an irregular geometry
- Each search mesh cell contains a pre-calculated short list of geometry objects that intersect its boundaries
- Loop over all objects is replaced by a table look-up and loop over this short list

Because of this approach, the running time is not strongly dependent on the complexity of the geometry (although memory footprint can be).

Advanced geometry types

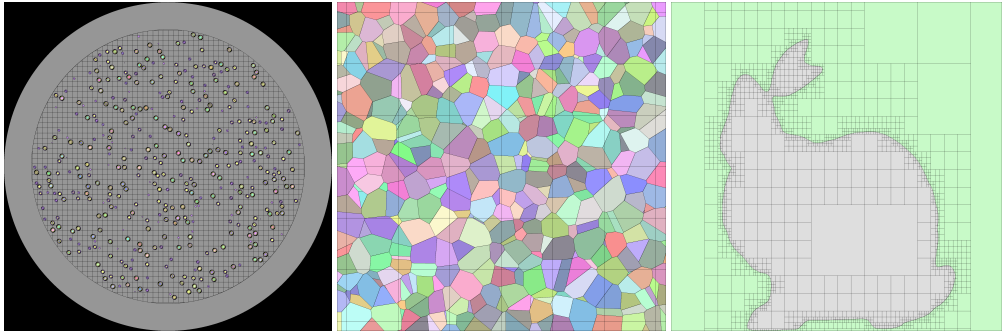


Figure 1: Left: Cartesian search mesh in an HTGR fuel pebble; Center: Cartesian search mesh in a Voronoi-tessellated geometry (available in the next update); Right: Adaptive search mesh in a CAD-based geometry.

Advanced geometry types

VTT

Exercise:

- Consider a geometry comprised of a 10x10x10 cubical lattice of spheres (e.g. fuel in moderator), surrounded by reflective boundary conditions
- Three options for creating the same geometry
  1) CSG geometry without a lattice (1000 spheres inside a cell)
  2) Using a regular lattice
  3) As an HTGR geometry type (coordinates read from a file)
- Try different variations of dimensions, number of spheres, and materials (e.g. criticality vs. external source calculation)
- The results should show that option 1 runs very slow, but because the search mesh used with the HTGR geometry type behaves similar to a lattice, there should be no major difference between 2 and 3
- The running time in case 3 should not significantly change if the same number of spheres are put in random positions

Advanced geometry types

The CAD-based geometry type also takes advantage of an adaptive search mesh. The performance can be comparable to a CSG geometry. For more information, see:

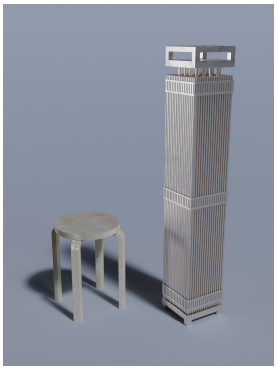
> Leppänen, J. "Methodology, Applications and Performance of the Cad-Based Geometry Type in the Serpent 2 Monte Carlo Code." Ann. Nucl. Energy, 176 (2022) 109259.

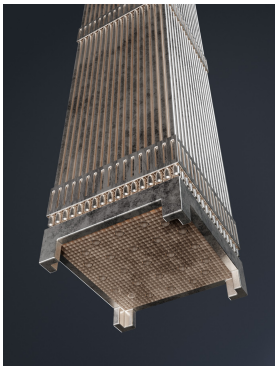
Notes on the CAD-based geometry type:

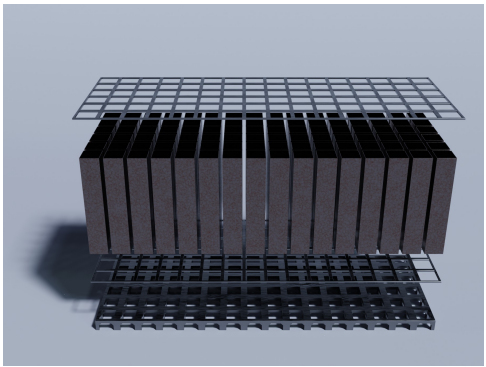
- The search mesh is defined by the user, and the input parameters can be have a significant effect on performance
- Aim for high “fill fraction” without excessive memory usage
- Structuring CAD components in in different levels may also speed up the calculation
- Don’t over-use the CAD-based geometry type – fuel rods in a regular lattice can be described using a CSG model combined with CAD-based solids for more complicated structures

Learning the basics of CAD modeling is not very difficult – FreeCAD is a good choice for creating Serpent models (open source, produces good STL files, lots of tutorials on YouTube).

Advanced geometry types







Tracking modes in Serpent

Serpent uses two tracking modes to transport particles through the geometry:

Surface-tracking

- Standard tracking mode in almost all Monte Carlo codes
- Particles are moved from one geometry boundary to the next, until a collision occurs before the intersection
- Performance depends on the cost of calculating surface distances and the number of boundary crossings between collisions
- Poor efficiency in geometries where the particle mean-free-path is long compared to dimensions

Delta-tracking

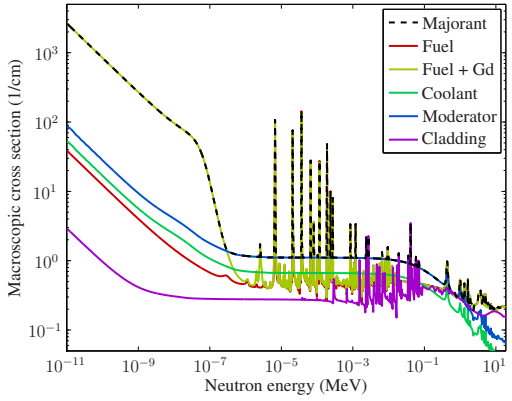
- Originally implemented in Serpent for the sake of simplicity, and because the method performs well in reactor geometries
- Rejection sampling technique, which allows moving particles directly to the next tentative collision site
- Performance depends on the efficiency of the rejection sampling algorithm
- Poor efficiency in geometries with localized zones of materials with large cross sections (typically strong absorbers)

Tracking modes in Serpent

Serpent uses the combination of surface- and delta-tracking:

- DT is usually faster in most geometries, but high rejection probability in some part of the geometry may lead to poor overall performance
- The routine automatically switches to ST when the local rejection probability is high
- The threshold is defined by the “set dt” input option
- Value of 0.9 was set as the default years ago, when Serpent was used for exclusively reactor physics

Question: How does the value of “set dt” affect the overall performance?



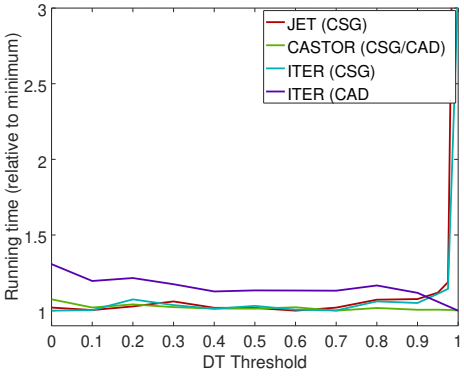
Tracking modes in Serpent

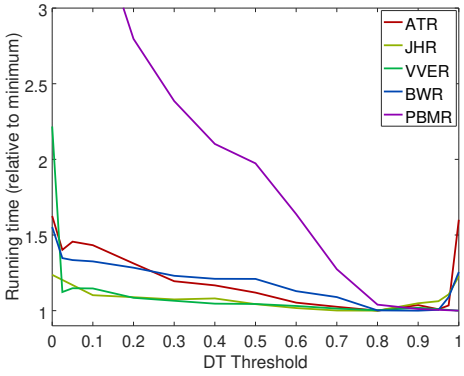
Test cases:

- JET fusion reactor, CSG model, neutron external source simulation
- CASTOR transport cask, hybrid CSG/CAD model, photon external source simulation
- ITER fusion reactor, CSG- and CAD-based models, neutron external source simulation
- ATR reactor core, neutron criticality source simulation
- JHR reactor core, neutron criticality source simulation
- VVER-440 fuel assembly, neutron criticality source simulation
- BWR fuel assembly with BA, neutron criticality source simulation
- PBMR fuel pebble, neutron criticality source simulation

Comparison of running time when “set dt” is varied between 0 and 1.

Tracking modes in Serpent





Final notes on delta-tracking:

- The track-length estimator (TLE) of flux cannot be used with delta-tracking (surface crossings not recorded)
- Serpent uses the collision flux estimator (CFE) for all standard detectors, even when DT is off (“set dt 0”)
- TLE is available as a separate detector for super-imposed surfaces (“dtl” detector type)
- Efficiency of CFE depends on how frequently the (virtual) collisions are scored (“set cfe”), especially in void regions

For more insight, see:

Leppänen, J. "On the use of delta-tracking and the collision flux estimator in the Serpent 2 Monte Carlo particle transport code." Ann. Nucl. Energy, 105 (2017) 161-167.

Summary and conclusions

Main takeaways from this presentation:

- Understanding how the tracking routine works can help avoid computational bottlenecks when constructing complicated geometry models
- Converting geometry models from other codes to Serpent CSG format almost always leads to sub-optimal performance
- The performance of advanced geometry types is often comparable to CSG; the main drawback is larger memory footprint
- Don’t be afraid to use CAD-components as part of the geometry model
- Understanding the peculiarities of delta-tracking may help when optimizing the calculation

Thank you for your attention!

Jaakko.Leppanen@vtt.fi

https://serpent.vtt.fi/serpent