Aalto University
School of Science
and Technology

Monte Carlo method in particle transport simulations
Lecture 2 – Constructive Solid Geometry model (CSG)

Jaakko Leppänen

Department of Applied Physics
Aalto University, School of Science
Jaakko.Leppanen@aalto.fi

Sep. 29, 2020

Topics of this lecture

Lecture topics:

- Basic building blocks of the CSG geometry: surfaces and cells
- Cell search routine
- Nests
- Universes and lattices
- Boundary conditions
- Geometry transformations

2nd programming exercise

Position and direction vectors

The Monte Carlo simulation consists of a large number of particle histories,[1] in which the random walk of an individual particle is followed, or tracked, through the geometry from its birth to eventual absorption or escape from the geometry.

From here on it is assumed that the tracking takes place in a three-dimensional Cartesian coordinate system. In vector notation, the position and direction of motion are defined by two vectors:

$$\boldsymbol{r} = x\hat{\boldsymbol{i}} + y\hat{\boldsymbol{j}} + z\hat{\boldsymbol{k}} \tag{1}$$

and

$$\hat{\boldsymbol{\Omega}} = u\hat{\boldsymbol{i}} + v\hat{\boldsymbol{j}} + w\hat{\boldsymbol{k}} \tag{2}$$

where $\hat{\boldsymbol{i}}$, $\hat{\boldsymbol{j}}$ and $\hat{\boldsymbol{k}}$ are the unit vectors defining the three-dimensional Cartesian coordinate system. Direction vector $\hat{\boldsymbol{\Omega}}$ is normalized to unity:

$$\hat{\boldsymbol{\Omega}} \cdot \hat{\boldsymbol{\Omega}} = 1 \tag{3}$$

or

$$u^2 + v^2 + w^2 = 1 \tag{4}$$

Coefficients u, v and w are the direction cosines, i.e. the cosines of the angle that vector $\hat{\boldsymbol{\Omega}}$ forms with the positive x-, y- and z-axis, respectively.

[1] The number of simulated histories is typically counted in millions, or in large-scale simulations, billions. This is several orders of magnitude less than the number of neutrons in an actual reactor, even at low-power operation.

Position and direction vectors

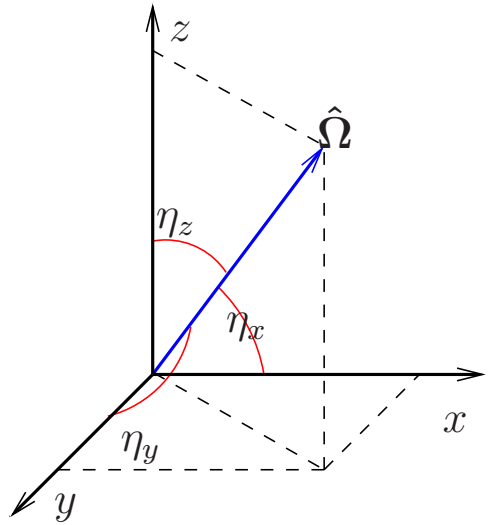


Figure 1: Direction vector $\hat{\mathbf{\Omega}}$ and direction cosines $u = \cos\eta_x$, $v = \cos\eta_y$ and $w = \cos\eta_z$ in the Cartesian coordinate system.

Surfaces

Monte Carlo transport codes are most typically based on the constructive solid geometry (CSG) type, in which the geometry is composed of homogeneous material cells, defined using combinations of elementary and derived surface types.

The most elemental building block is the surface, described using algebraic equations, typically of the quadratic type. The action that puts an arbitrary position $\boldsymbol{r}$ on one or the other side of a surface is based on a simple test carried out by substituting the coordinates into the surface equation:

$$S(\boldsymbol{r}) = S(x,y,z) \begin{cases} < 0 & \text{if the point is inside the surface} \\ = 0 & \text{if the point is on the surface} \\ > 0 & \text{if the point is outside the surface} \end{cases} \tag{5}$$

This surface test also fixes the concepts of “inside” and “outside” for each surface type, which is important when forming the cells from the surface combinations.

The general quadratic surface can be written in parametric form as:

$$S(x,y,z) = Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fzx + Gx + Hy + Iz + J \tag{6}$$

where $A, B, C, D, E, F, G, H, I$ and J are constants.[2]

[2] There are also non-quadratic surfaces, such as the torus.

Surfaces

Common examples of quadratic surfaces obtained from the parametrized quadratic equation include the plane perpendicular to x-axis at x_0:

$$S(x) = x - x_0 \tag{7}$$

sphere centered at (x_0, y_0, z_0) with radius r:

$$S(x, y, z) = (x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 - r^2 \tag{8}$$

and straight infinite cylinder parallel to z-axis centered at (x_0, y_0) with radius r:

$$S(x, y) = (x - x_0)^2 + (y - y_0)^2 - r^2 \tag{9}$$

Surface equation for general plane is obtained by dropping the second-order terms in Eq. (6):

$$S(x, y, z) = Gx + Hy + Iz + J \tag{10}$$

where G, H, I and J are constants. The plane can also be defined using three points: (x_1, y_1, z_1), (x_2, y_2, z_2) and (x_3, y_3, z_3), in which case the constants for Eq. (10) become:

$$\begin{aligned} G &= y_2 z_3 - y_3 z_2 - y_1(z_3 - z_2) + z_1(y_3 - y_2) \\ H &= z_2 x_3 - z_3 x_2 - z_1(x_3 - x_2) + x_1(z_3 - z_2) \\ I &= x_2 y_3 - x_3 y_2 - x_1(y_3 - y_2) + y_1(x_3 - x_2) \\ J &= -x_1(y_2 z_3 - y_3 z_2) + y_1(x_2 z_3 - x_3 z_2) - z_1(x_2 y_3 - x_3 y_2) \end{aligned} \tag{11}$$

The three points determine the inward surface direction according to the right-hand rule.

Surfaces

Surface equations are also used for determining the distance to the nearest material boundary in the direction of motion. The points where the particle path intersects the surface are obtained by solving Eq. (5) with condition:

$$S(\boldsymbol{r} + \delta\hat{\boldsymbol{\Omega}}) = S(x + \delta u, y + \delta v, z + \delta w) = 0 \tag{12}$$

i.e. by setting a point located at distance δ from position $\boldsymbol{r}$ in the direction of motion $\hat{\boldsymbol{\Omega}}$ on the surface, and solving for δ. When the equation has multiple solutions, the nearest point corresponds to the smallest positive value of δ. If all solutions are negative or no solution exists, the surface is away from the line-of-sight.

The distance to general quadratic surface (6) can be written as:

$$\delta = \frac{-L \pm \sqrt{L^2 - 4MK}}{2M} \tag{13}$$

where:

$$\begin{aligned} K &= Ax^2 + By^2 + Cz^2 + Dxy + Eyz + Fxz + Gx + Hy + Iz + J \\ L &= 2(Aux + Bvy + Cwz) + D(vx + uy) + E(wy + vz) + F(wx + uz) \\ &\quad + Gu + Hv + Iw \\ M &= Au^2 + Bv^2 + Cw^2 + Duv + Evw + Fuw \end{aligned}$$

Surfaces

For example, the distance to a plane perpendicular to x-axis at x_0 is obtained from

$$x + \delta u - x_0 = 0 \Longleftrightarrow \delta = \frac{x_0 - x}{u} \tag{14}$$

and the distance to a cylinder parallel to z-axis centered at (x_0, y_0) with radius r from:

$$\delta = \frac{-L \pm \sqrt{L^2 - 4MK}}{2M} \tag{15}$$

where:

$$\begin{aligned} K &= x^2 + y^2 - 2(x_0 x + y_0 y) + x_0^2 + y_0^2 - r^2 \\ L &= 2(ux + vy) - 2(x_0 u + y_0 v) \\ M &= u^2 + v^2 \end{aligned}$$

Distance to general plane (10) is given by:

$$\delta = -\frac{Gx + Hy + Iz + J}{Gu + Hv + Iw} \tag{16}$$

which corresponds to the general quadratic equation (13) with coefficients of second order terms set to zero.

Surfaces

Monte Carlo codes often provide additional derived surface types, which are formed by combinations of elementary surfaces. For example, a cuboid with boundaries $[x_1, x_2]$, $[y_1, y_2]$, $[z_1, z_2]$ consists of six planes perpendicular to the coordinate axes:

$$S := \begin{cases} S_1(x) = x - x_1 \\ S_2(x) = x - x_2 \\ S_3(y) = y - y_1 \\ S_4(y) = y - y_2 \\ S_5(z) = z - z_1 \\ S_6(z) = z - z_2 \end{cases} \tag{17}$$

Other derived surface types convenient for reactor modeling include truncated cylinders, and square and hexagonal prisms.

The surface test and calculation of distance to nearest intersection point for derived surfaces involve multiple solutions to Eqs. (5) and (12). From the algorithmic point of view these operations should be seen as functions that return the result regardless of the surface type (elementary or derived).

Surfaces

It should be noted, however, that in case of derived surface types, a positive distance returned by (12) may not correspond to intersection with the actual surface, but rather the continuation of one of its constituents.

These “false” crossings can be accounted for by additional checks in the distance function or writing the tracking routine in such way that they can be ignored altogether.

It is important to remember that computers operate on limited arithmetic precision and positions at or very close to surfaces may cause false results for the surface test and distance calculation!

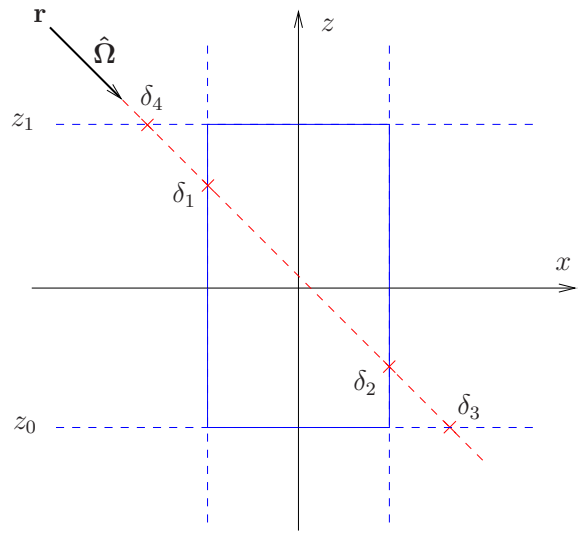


Figure 2: Particle path intersecting a truncated cylinder in the xz-plane demonstrating the false crossings. All surface distances in this example are positive and $\delta_4 < \delta_1 < \delta_2 < \delta_3$. Points 4 and 3 are not part of the boundary. The shortest actual distance is δ_1, even though the routine may return δ_4.

Surfaces

Third application for surface equations is the calculation of normal vectors, which are needed for various purposes, including:

- Calculating the surface flux estimator
- Applying reflective boundary conditions

The normal vector is given by the gradient of the surface equation:

$$\hat{\boldsymbol{n}}(\boldsymbol{r}) = \frac{\nabla S(\boldsymbol{r})}{|\nabla S(\boldsymbol{r})|}\,, \tag{18}$$

where

$$\nabla S(\boldsymbol{r}) = \nabla S(x,y,z) = \left(\frac{\partial S}{\partial x}\right)\hat{\boldsymbol{i}} + \left(\frac{\partial S}{\partial y}\right)\hat{\boldsymbol{j}} + \left(\frac{\partial S}{\partial z}\right)\hat{\boldsymbol{k}} \tag{19}$$

and

$$|\nabla S(\boldsymbol{r})| = |\nabla S(x,y,z)| = \sqrt{\left(\frac{\partial S}{\partial x}\right)^2 + \left(\frac{\partial S}{\partial y}\right)^2 + \left(\frac{\partial S}{\partial z}\right)^2} \tag{20}$$

For example, the normal vector on an infinite cylinder parallel to z-axis centered at (x_0, y_0) with radius r is given by:

$$S(x,y) = (x-x_0)^2 + (y-y_0)^2 - r^2 \Longrightarrow \hat{\boldsymbol{n}}(x,y) = \frac{(x-x_0)\hat{\boldsymbol{i}} + (y-y_0)\hat{\boldsymbol{j}}}{\sqrt{(x-x_0)^2 + (y-y_0)^2}} \tag{21}$$

Cells

The next building block in the CSG hierarchy is the cell, which is constructed from the combination of surfaces, and it defines a homogeneous material region. The construction is based on three operators:

Intersection: $S_1 \cap S_2$ – Point is inside the cell if it is inside *both* surface S_1 and S_2

Union: $S_1 \cup S_2$ – Point is inside the cell if it is inside *either* surface S_1 or S_2, or both

Complement: $\backslash S_1$ – Point is inside the cell if it is outside surface S_1

The intersection and union operator behave very similar to arithmetic multiplication and addition, respectively, and they share the properties of commutativity (order of operands is exchangeable):

$$\begin{aligned} S_1 \cup S_2 &= S_2 \cup S_1 \\ S_1 \cap S_2 &= S_2 \cap S_1 \end{aligned} \tag{22}$$

associativity (two or more similar operations can be grouped in an arbitrary manner):

$$\begin{aligned} (S_1 \cup S_2) \cup S_3 &= S_1 \cup (S_2 \cup S_3) \\ (S_1 \cap S_2) \cap S_3 &= S_1 \cap (S_2 \cap S_3) \end{aligned} \tag{23}$$

and distributivity (precedence of intersection over union):

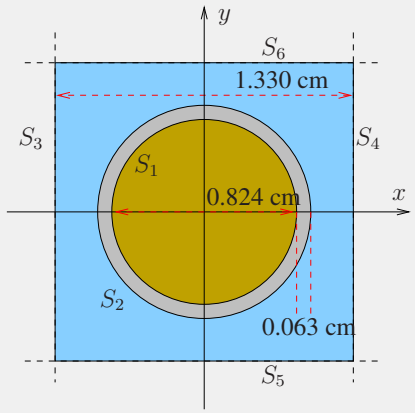
$$(S_1 \cup S_2) \cap S_3 = (S_1 \cap S_3) \cup (S_2 \cap S_3) \tag{24}$$

The use of the operators is best illustrated by an example.

Example 1: Cell definition in CSG geometry

Consider a 2D pin-cell model of a light water reactor fuel pin surrounded by coolant. The pin consist of a fuel pellet with an outer diameter of 0.824 cm, enclosed inside a 0.063 cm thick cladding. The square pitch of the unit cell is 1.330 cm. The surfaces are defined as:

$$
\begin{aligned}
S_1(x,y,z) &= x^2 + y^2 - 0.412^2 \\
S_2(x,y,z) &= x^2 + y^2 - 0.475^2 \\
S_3(x,y,z) &= x + 0.665 \\
S_4(x,y,z) &= x - 0.665 \\
S_5(x,y,z) &= y + 0.665 \\
S_6(x,y,z) &= y - 0.665
\end{aligned}
\tag{25}
$$



These surfaces are used to define four cells using intersections, unions and complements:

$$
\begin{array}{lcll}
C_1 & : & S_1 & : \text{ Fuel} \\
C_2 & : & (\backslash S_1) \cap S_2 & : \text{ Cladding} \\
C_3 & : & (\backslash S_2) \cap (\backslash S_3) \cap S_4 \cap (\backslash S_5) \cap S_6 & : \text{ Coolant} \\
C_4 & : & S_3 \cup (\backslash S_4) \cup S_5 \cup (\backslash S_6) & : \text{ Outside world}
\end{array}
\tag{26}
$$

The last cell, described as the “outside world”, is not a part of the actual geometry, but it needs to be defined in order to tell the geometry routine that the particle has escaped the system, or that boundary conditions need to be applied.

Cells

Testing cells that consist of only intersections and complements is straightforward:

- If all surfaces comprising the cell pass the surface test (taking into account the complement operators), the point is inside the cell
- If any of the surfaces fails the test, the point is outside the cell

The complement operator can be included by adding flags in the surface intersection list.

The problem with unions is that testing arbitrarily constructed cells with one or several unions becomes a non-trivial task, requiring either a tree-based or post-fix algorithm.

It is possible to construct geometries, even complicated ones, without using the union operator. This can be accomplished using derived surface types. In practice this means that any union operators are performed inside the surface functions instead of within the cell test algorithm.

Most geometries consist of not a single, but of multiple cells, and in order to determine which cell fills the space at an arbitrary position, the cell search routine must loop over all possibilities until the conditions for the constituent surfaces are matched.

Cells

Algorithm 1 Cell search 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)

Nests

Reactor geometries are often comprised of cells nested inside each other, e.g. fuel pins formed by cylindrical layers of fuel, gas gap, cladding and coolant.

This type of geometries are easy to define, and it may be computationally beneficial to handle them using a separate algorithm.

The cell search is performed by looping over the surfaces, starting from the innermost zone. Point (x, y, z) is inside the cell that first passes the surface test for its outer surface.

If all surfaces fail the test, the point is in the outermost zone (extending to infinity).

Nests are best treated as a special type of universe (see next slide).

Universes and lattices

Handling of complex geometries can be considerably simplified by dividing the model into multiple levels. In reactor geometries the natural division is:

1) Core-level
2) Assembly-level
3) Pin-level

The reactor core consists of a regular array of fuel assemblies, many of which are identical and interchangeable. The same applies to fuel pins inside assemblies.

Instead of modeling each pin and assembly explicitly, they can be described as separate objects in their own universe. The same object can then be copied into multiple locations by filling cells with an entire universe.

The use of lattices helps the construction of regular structures, such as the pin layout in fuel assemblies or the loading pattern of assemblies in the reactor core.

Structuring the geometry into multiple levels not only simplifies the input model, but may also lead to considerable increase in performance as the number of computationally expensive geometry operations is reduced.

Universes and lattices

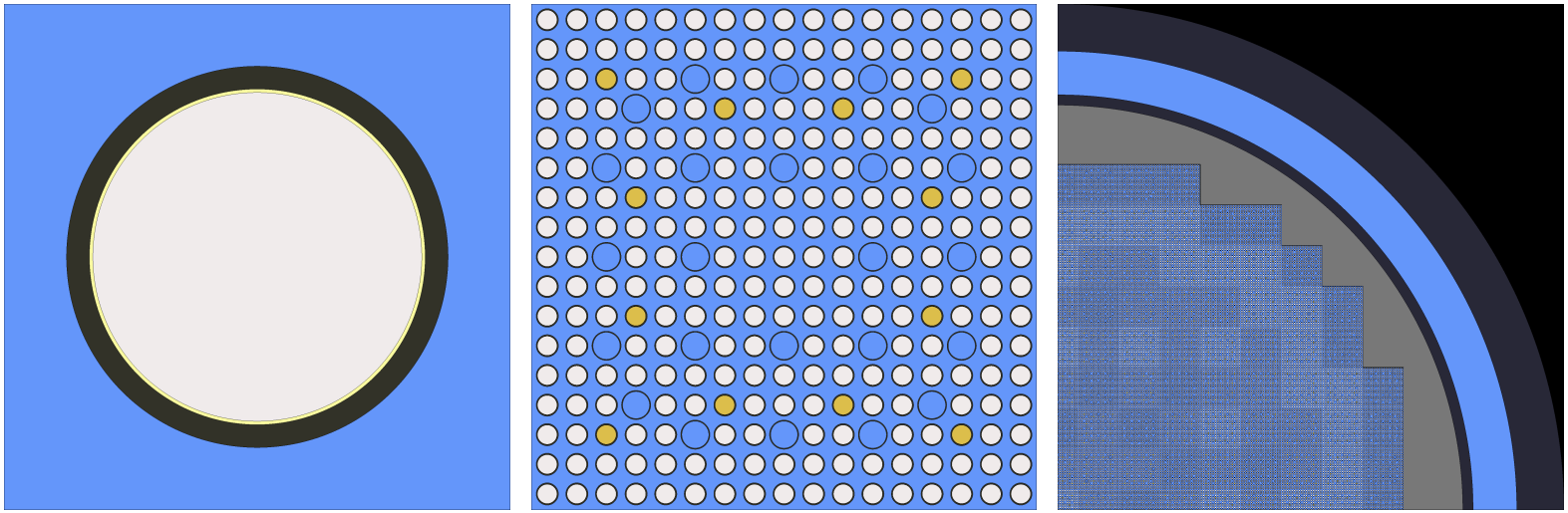


Figure 3: Pin-, assembly- and core-level geometries in a 1/4 core model of a PWR. Left: The first level is comprised of universes describing individual fuel pins, control rod guide tubes, etc. Center: The second level describes fuel assemblies as regular lattices of pins. Right: The last level is comprised of the reactor vessel, with an assembly lattice inside it. When the cell search routine finds itself inside a cell filled with an entire universe, it makes a coordinate transformation to the next level, and recursively locates the material cell at the position.

Universes and lattices

Most common lattice types are 2D square array and hexagonal “honeycomb”, but the concept can basically be extended to any regular (or even irregular) structure.

When the point is found inside a lattice, the geometry routine automatically makes a coordinate transformation between the levels and centers the universe with respect to the lattice cell.

This requires calculating the lattice indexes, which for a Cartesian type is written as:[3]

$$i = \text{floor}\left\{\frac{x}{p_x}\right\} \quad j = \text{floor}\left\{\frac{y}{p_y}\right\} \quad k = \text{floor}\left\{\frac{z}{p_z}\right\} \tag{27}$$

where i, j and k are the lattice indexes and p_x, p_y and p_z are the cell widths (pitches). The “floor” operation refers to truncation of the decimal part.

The universe filling the lattice cell is found from an input table using the lattice indexes. The coordinate transformation to the new origin is written as:

$$\begin{aligned} x' &= x - ip_x \\ y' &= y - jp_y \\ z' &= z - kp_z \end{aligned} \tag{28}$$

[3] It is assumed here that the lattice is centered at origin and has an odd number of cells in each direction.

Universes and lattices

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**

Boundary conditions

The geometry model is limited to an outer boundary, beyond which there are no more cells. What happens to a neutron crossing this boundary depends on the boundary conditions:

- Vacuum (or black) boundary condition – the neutron has escaped the geometry and the history is terminated
- Reflective (or mirror) boundary condition – the neutron is reflected back into the geometry at an angle symmetrical with respect to the surface normal
- Periodic boundary condition – the neutron is moved into a symmetry position on the other side of the geometry
- White boundary condition – the neutron is reflected back into the geometry isotropically (random direction)

Reflective and periodic boundary conditions can be used create infinite geometries in which the same structure is repeated over and over again. If the outer boundary is a square or hexagonal prism, the boundary conditions can also be invoked by a coordinate transformation.

With vacuum boundary conditions it is important that the geometry is non re-entrant, as the neutron is killed immediately after crossing the boundary without the possibility of returning to the geometry.

This topic is revisited later with particle tracking.

Boundary conditions

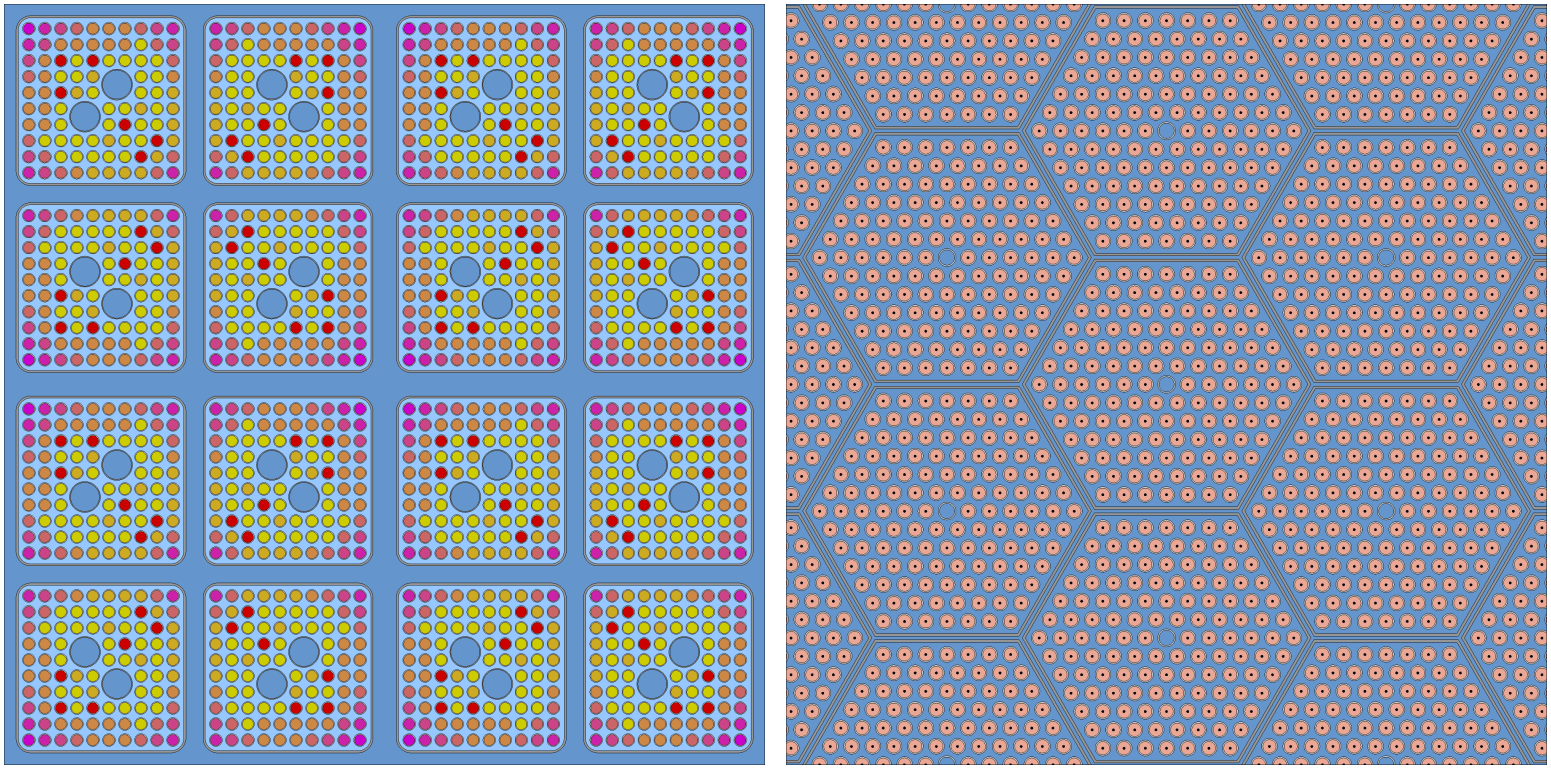


Figure 4: Infinite lattices of identical fuel assemblies generated using repeated boundary conditions. Left: BWR fuel lattice, with reflective boundary surrounding a single assembly. Right: VVER-440 fuel lattice with periodic boundary surrounding a single assembly.

Geometry transformations

Surfaces can be moved and rotated using geometry transformations, which considerably simplifies the input, for example, when the geometry contains cylinders that are not parallel to any coordinate axis.

These operations are based on Euclidean transformations, which in the general form can be written as:

$$\boldsymbol{r}' = \mathbf{A}\boldsymbol{r} + \mathbf{d} \tag{29}$$

where $\mathbf{A}$ is the transformation matrix and $\mathbf{d}$ is the displacement vector.

For the purpose of particle transport simulations, however, it is sufficient to consider only isometric transformations, which preserve the length of the vector, or more precisely, the distance between any two points in space. These transformations are also orthogonal, which means that the angle between any two vectors remains unchanged.

Isometric transformation matrices can be divided into rotations, for which $\det(\mathbf{A}) = 1$ and reflections, for which $\det(\mathbf{A}) = -1$.

Geometry transformations are applied to particle coordinates and direction cosines before calling the surface test or distance routines. In addition to surfaces, geometry transformations can be applied to entire universes.

Geometry transformations

Rotations with respect to the three Cartesian coordinate axes can be applied using three matrices:

$$\mathbf{R}_x(\varphi_x) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\varphi_x & -\sin\varphi_x \\ 0 & \sin\varphi_x & \cos\varphi_x \end{pmatrix} \tag{30}$$

$$\mathbf{R}_y(\varphi_y) = \begin{pmatrix} \cos\varphi_y & 0 & \sin\varphi_y \\ 0 & 1 & 0 \\ -\sin\varphi_y & 0 & \cos\varphi_y \end{pmatrix} \tag{31}$$

$$\mathbf{R}_z(\varphi_z) = \begin{pmatrix} \cos\varphi_z & -\sin\varphi_z & 0 \\ \sin\varphi_z & \cos\varphi_z & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{32}$$

where φ_x, φ_y and φ_z are the rotation angles.

Note that the transformation (29) can also be applied as:

$$\boldsymbol{r}' = \mathbf{A}(\boldsymbol{r} + \mathbf{d}) \tag{33}$$

i.e. by applying the displacement before the rotation.

2nd programming exercise

Mandatory tasks:

- Expand the input routine to read the structures needed for generating CSG geometries.
- Implement surface test and distance functions for elementary surface types: sphere, infinite cylinder, planes parallel to coordinate axes.
- Implement surface test and distance functions for derived surface types: truncated cylinder, cuboid, infinite square prism.
- Implement cell search routine capable of handling intersection and complement operators, without universes, lattices or repeated boundary conditions.
- Demonstrate the implemented routines by calculating the volumes of cells by sampling random points in the geometry. Consider two cases:[4]

  1) Fuel pin with dimensions given in the example on slide 13

  2) Hollow cylinder with inner diameter of 4 cm, inner height 7 cm, wall thickness 0.05 cm, filled with two materials separated by a horizontal layer at 3.5 cm from the bottom

  Cell volume is given by:

$$V_c = \frac{N_c}{N} V \tag{34}$$

  where N_c is the number of points inside the cell, N is the total number of sampled points and V is the volume of the "bounding box" where the points are sampled.

[4] For 2D geometries, "volume" refers to volume per unit length, i.e. the cross-sectional area.

2nd programming exercise

Bonus tasks (in recommended order)

1. Implement a routine that evaluates cell volumes by drawing lines over the geometry bounding box (ratio N_c/N in (34) is then given by the sum of segment lengths inside the cell divided by the sum of lengths inside the bounding box). Demonstrate the implementation by comparing the results and FOM's to the point sampling routine (+1 point).
2. Include universes and square lattices in the geometry routine, demonstrate the implementation with volume calculation (+2 points).
3. Expand the surface routines to additional types: general plane and hexagonal prism, demonstrate implementation with volume calculation (+1 point).
4. Implement a geometry plotting routine that illustrates 2D cross-sectional slices of the geometry either as an ASCII map, data read into visualization software or direct graphics file (+1 point).
5. Implement surface translations and rotations and demonstrate the implementation using the geometry plotter (+1 point).
6. Implement hexagonal lattices in the geometry routine and demonstrate the implementation using the geometry plotter (+2 points).
7. Expand the surface routines to elliptical torus. Demonstrate the implementation by calculating the volume using both methods: sampling points and lines. (+3 points).

2nd programming exercise

Milestones:

- Implementation of basic geometry routines needed later for particle tracking.
- Expanding input routine to read surface, cell and other geometry definitions.

Methods developed in two of the bonus exercises are needed to complete bonus exercises later on:

- Volume calculation by drawing lines over the geometry forms the basis of the so-called surface-tracking routine.
- Universe-based geometry routine is required for modeling heterogeneous reactor geometries.

Based on last year's statistics this exercise takes twice the time required to complete exercise 1!