

Flexible Geometry Treatment in PRAGMA Using NVIDIA® Ray Tracing Engine OptiX™

Jaek Im, Namjae Choi, Han Gyu Joo*

Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul, 08826, Korea

*Corresponding author: joohan@snu.ac.kr

1. Introduction

PRAGMA is a GPU-based continuous energy Monte Carlo (MC) code dedicated for power reactor analyses [1]. To meet the primary purpose, the geometry module of PRAGMA is based on lattice structures which typical LWRs have. However, eventually a flexible geometry module should be developed for the extensibility of the code as the lattice-based geometry representation limits the scope of application to certain types of problems.

Most of the MC codes employ the constructive solid geometry (CSG) representation to model the geometries, and presumably all the codes implement their own CSG module. However, the neutron tracing problem has an analogy to the ray tracing problem in graphics which is a very historic and extensively studied problem, and the algorithms and the libraries for the ray tracing are already well-established for GPU applications.

In this regard, to handle flexible geometries on GPUs, it is better to utilize a highly efficient ray tracing library rather than manually implementing the CSGs. A famous GPU vendor NVIDIA has been releasing a ray tracing engine OptiX [2] which works on their CUDA-enabled GPU cards. It provides a complete ray tracing pipeline and high level of flexibility so that several researches to exploit OptiX in physics simulations have been made. In neutronics, WARP [3] had first explored applying OptiX in the GPU-based MC neutron transport, and recently a novel port of OpenMC to GPUs [4] had exploited OptiX, though both works are not yet production grade.

This paper examines the use of OptiX for the general geometry neutron transport in PRAGMA and studies its feasibility. The algorithms, issues, and some preliminary results are presented.

2. Backgrounds and Algorithms

2.1. OptiX Ray Tracing Pipeline

The OptiX ray tracing engine is a programmable framework developed for NVIDIA GPUs. It provides a simple, recursive, and flexible pipeline for accelerating ray tracing algorithms for graphics, collision detection, visibility determination and etc.

Figure 1 shows the ray tracing pipeline in OptiX. The yellow boxes are the user-defined programs and the blue boxes are the OptiX built-in algorithms. Developers can implement all the programs in the yellow boxes in a way that is suitable for their own applications.

In MC neutron simulation, the ray generation program casts neutrons. Then, OptiX internally traverses neutrons throughout the geometry, and intersection and closest hit

programs provide intersections between the neutrons and the geometric primitives. Miss program is invoked when a neutron does not intersect any primitive and is used to handle void boundary conditions. Any hit program is not required in the MC simulations.

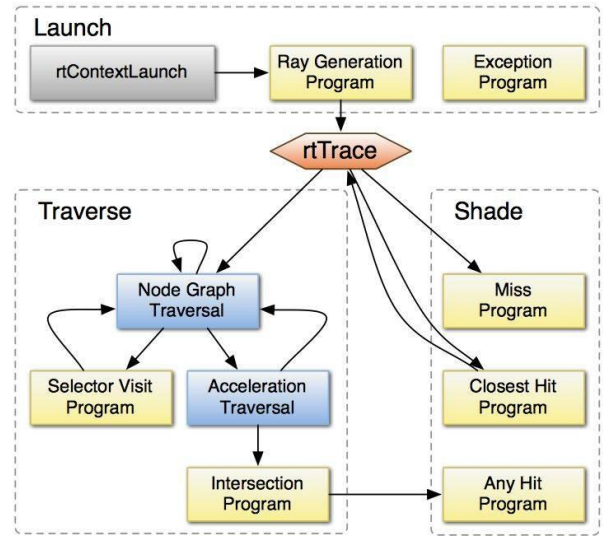


Figure 1. Call graph illustrating control flow through the ray tracing pipeline [5].

2.2. Ray – Primitive Intersection

Rays in OptiX are represented as parametric lines:

$$\begin{array}{c}
 \begin{array}{ccc}
 \varepsilon & & \\
 \leftarrow & \text{---} & \rightarrow \\
 O & & t_{\max} \\
 & & \uparrow \\
 & & t_{\min}
 \end{array} \\
 P(t) = O + t\mathbf{d} \quad (1)
 \end{array}$$

P : The end point of the ray

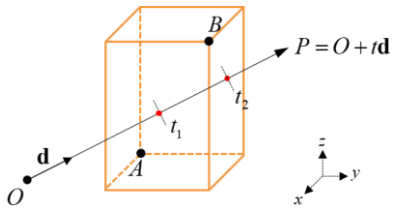
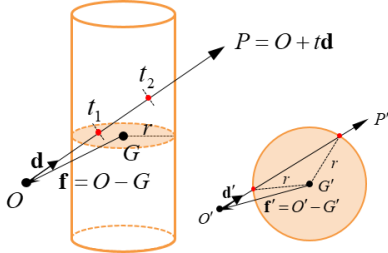
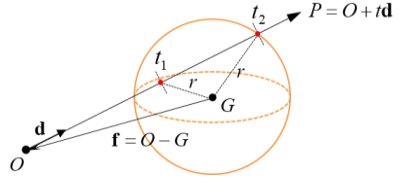
O : Ray origin, \mathbf{d} : Ray direction

t : Parameter that indicate distance between P and O

t_{\min} and t_{\max} are lower and upper threshold of the distance, respectively. The intersection is neglected if the distance to the intersection point is smaller than t_{\min} or larger than t_{\max} . Especially, t_{\min} is referred to as *scene epsilon* and it is one of the most important variable in preventing self-intersections, which will be explained later.

Each ray delivers data from the shading programs by a special variable called *payload*. It can contain any types of data specified by the developer. In our application, a payload contains distance-to-surface (DTS), hit primitive index, and such, which are determined by the closest hit program. When miss program is invoked, a flag that the ray had escaped the domain is returned by the payload.

Table 1. Intersection calculation algorithms of representative geometric primitives.

| Box Primitive | Cylinder Primitive | Sphere Primitive |
|--|--|--|
|  |  |  |
| $O + t\mathbf{d} = A, O + t\mathbf{d} = B$ $\Rightarrow t_A = (A - O) / \mathbf{d}, t_B = (B - O) / \mathbf{d}$ $t_1 = \max(q_x, q_y, q_z)$ $t_2 = \min(r_x, r_y, r_z)$ <p>where $q_u = (\min(t_{Au}, t_{Bu}))$ $r_u = (\max(t_{Au}, t_{Bu}))$</p> | $(P' - G') \cdot (P' - G') = r^2$ $\Rightarrow (\mathbf{d}' \cdot \mathbf{d}')t^2 + 2(\mathbf{f}' \cdot \mathbf{d}')t + \mathbf{f}' \cdot \mathbf{f}' - r^2 = 0$ $\therefore t_1 = (-b - \sqrt{b^2 - ac}) / a$ $t_2 = (-b + \sqrt{b^2 - ac}) / a$ $a = \mathbf{d}' \cdot \mathbf{d}' = \sin^2 \theta$ <p>where $b = \mathbf{f}' \cdot \mathbf{d}'$ $c = \mathbf{f}' \cdot \mathbf{f}' - r^2$</p> | $(P - G) \cdot (P - G) = r^2$ $\Rightarrow (\mathbf{d} \cdot \mathbf{d})t^2 + 2(\mathbf{f} \cdot \mathbf{d})t + \mathbf{f} \cdot \mathbf{f} - r^2 = 0$ $\therefore t_1 = -b - \sqrt{b^2 - c}$ $t_2 = -b + \sqrt{b^2 - c}$ <p>where $b = \mathbf{f} \cdot \mathbf{d}, c = \mathbf{f} \cdot \mathbf{f} - r^2$</p> |

In OptiX, each primitive is tied to a unique intersection program. All the intersection programs of the primitives other than the triangle should be implemented by the user. Box, cylinder, and sphere primitives were implemented in PRAGMA, whose intersection calculation algorithms are presented briefly in **Table 1**.

2.3. Cell Index Query Algorithm

Unfortunately, OptiX does not provide an easy way of finding the cell index of a neutron, as it can only find the index of the closest surface. **Figure 2** illustrates a typical problem situation that can be seen in the MC simulations. Consider the neutrons indicated as red rays, which flies across the water primitive (2) to reach the fuel primitives (1 or 3). For those neutrons, OptiX will return the indices of the fuel primitives which the rays hit, but they actually belong to the water primitive whose cross sections should be used for the flight kernels. Neutrons escaping closed primitives, which are represented as black rays, do not suffer from this problem.

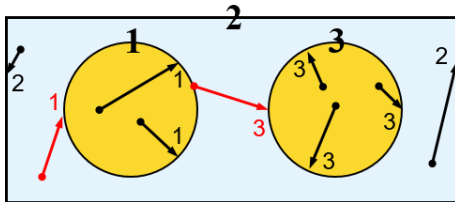


Figure 2. Problematic situations in cell index query.

To resolve this problem, a cell index query algorithm named as *Conditional Iterative Ray Tracing (CIRT)* was developed as shown in **Figure 3**, based on the crossing number algorithm in the point-in-polygon (PIP) problem [6]. The CIRT algorithm performs additional ray casting

after finding DTS to determine the cell index. At every intersection during ray casting, it is determined whether the ray is entering or escaping the intersecting primitive. If the ray is entering, *local_level* is set to -1; otherwise, it is set to 1. It is accumulated on the *global_level* variable and the ray casting continues until it becomes negative.

```

Set ray origin and direction
Launch rtTrace to get DTS and set global_level
while (global_level >= 0)
  Move ray origin
  Launch rtTrace and get local_level
  global_level += local_level
end while
Get primitive index
    
```

Figure 3. The CIRT algorithm.

The working principle of the CIRT algorithm is based on the fact that a ray that enters a closed primitive must escape the primitive. Entering and escaping give the local level of 1 and -1, respectively, which cancel out on the global level. As the result, the ray casting loop continues until it reaches the boundary of the containing primitive and returns its index, neglecting all the closed primitives in the trajectory. **Figure 4** demonstrates how the problem encountered in **Figure 3** can be resolved by the CIRT algorithm, in which the numbers indicate the global level.

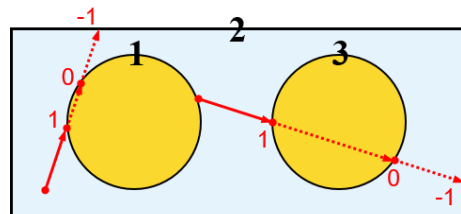


Figure 4. Example of CIRT.

As the result, the tracking algorithm using OptiX ray tracing (rtTrace) is constructed as **Figure 5**. rtTrace must be executed ahead of all the other calculation kernels as it determines the index of the cell where the neutron is located by the CIRT algorithm.

```

while (num_alive > 0)
  Launch rtTrace to get cell indices and DTSs
  Launch built-in simulation kernel
  parallel foreach num_alive neutrons
    Retrieve cell index and DTS from payload
    Calculate macroscopic cross section and DTC
    Move neutron to next position
    if (DTC < DTS) Process neutron reaction
  end parallel foreach
  Update num_alive
end while

```

Figure 5. Pseudocode of the tracking algorithm using OptiX.

2.4. Self-intersection Issues

Due to errors in the floating point arithmetic, a ray may intersect a given point repeatedly with an infinitesimally small distance, which is the case called *self-intersection*. Here, the scene epsilon parameter plays a role; it neglects intersections with distances smaller than a preset value, which is defined to be sufficiently small not to incorrectly ignore actual intersections.

However, the accuracy required in neutronics forces the scene epsilon values to be a very small value which is not sufficient to fully prevent the self-intersections. As self-intersection can make neutrons stuck on a surface or disturb CIRT by incorrectly changing the global level, it should be properly addressed.

Therefore, a conditional statement is augmented to the CIRT loop to skip the update of the global level if there is no change in the local level – primitive index pair of the two adjacent ray tracing steps. Additionally, when a neutron is stuck on a surface, the position of the neutron is perturbed conditionally based on the cumulative count of successive DTS events to make the neutron deviate from the surface.

3. Results

In this section, verification results of the OptiX-based geometry treatment module in PRAGMA are presented. Problems to be solved are 1.72 wt.% pin cell of APR1400, CANDU6 fuel bundle, and the Lady Godiva device. The new geometry module is verified against both the built-in module of PRAGMA and McCARD [7].

3.1. Verification of Cell Index Query

Figure 6 illustrates the result of cell index query for the CANDU6 fuel bundle. The cell indices of neutrons were retrieved during an MC run and plotted. Each cell is represented by a unique color. This verifies that the cell indices can be properly found by the CIRT algorithm.



Figure 6. Cell index query result for CANDU6 fuel bundle.

3.2. Verification with PRAGMA Built-in Solver

The 1.72 wt.% pin cell problem was solved with both the built-in module and the OptiX module to verify the consistency between the two solvers with the same cross section set. Note that the built-in solver of PRAGMA had been already verified against McCARD. Both modules employed 20/200 inactive/active cycles with 1,000,000 neutrons per cycle.

Table 2 compares the eigenvalue of the two solvers, and **Figure 7** compares their flux spectra. It is concluded that the OptiX module is fully consistent to the built-in solver, as both eigenvalues and the flux spectra coincide well with each other.

Table 2. Comparison of eigenvalues with built-in solver.

| Solver | Built-in | OptiX |
|-----------|-------------|-------------|
| k_{eff} | 1.20023 (4) | 1.20018 (4) |

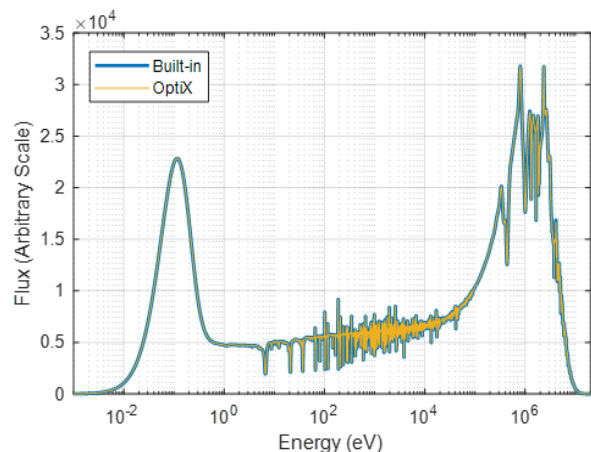


Figure 7. Comparison of flux spectra with built-in solver.

3.3. Verification with McCARD

Herein lie two problems that could not be handled with the built-in lattice geometry module of PRAGMA, which are the Lady Godiva device problem and the CANDU6 fuel bundle problem. Therefore, the verifications should be made with McCARD which is a general-purpose MC

code employing CSGs. Both codes used the cross section libraries based on ENDF-B-VII.1.

3.3.1. Lady Godiva Device

The Lady Godiva device is a highly enriched uranium sphere with a radius of 8.741cm. It contains 94.73 wt.% of ^{235}U and 5.27 wt.% of ^{238}U with the total mass density of 18.74g/cm^3 . It is close to the critical mass in the room temperature.

Table 3 compares the eigenvalues of the two codes. Both codes employed 1,000 cycles including 20 inactive cycles with 1,000,000 neutrons per cycle. It can be seen that both codes yield multiplication factors that are close to critical and that agrees with each other well within 1σ uncertainty, which indicates the soundness of the OptiX module.

Table 3. Comparison of eigenvalues with McCARD.

| Code | McCARD | PRAGMA |
|-----------|-------------|-------------|
| k_{eff} | 1.00139 (2) | 1.00137 (2) |

3.3.2. CANDU6 Fuel Bundle

A CANDU6 fuel bundle contains 37 natural uranium fuel rods capsuled by a pressure tube and a calandria tube. Both coolant and moderator use pressurized heavy water. The problem was set to cold zero power (CZP) condition, and 100 inactive cycles and 900 active cycles were used with 100,000 neutrons per cycle.

Table 4 compares the eigenvalues of the two codes, and **Figure 8** compares their flux spectra, which present a typical behavior of a heavy water system. As the flux spectra as well as the eigenvalues match perfectly, it is verified that the OptiX module can also handle irregular geometries correctly.

Table 4. Comparison of eigenvalues with McCARD.

| Code | McCARD | PRAGMA |
|-----------|-------------|-------------|
| k_{eff} | 1.13701 (4) | 1.13704 (4) |

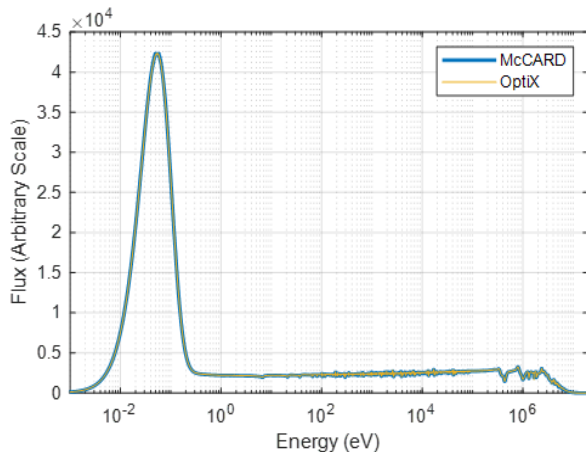


Figure 8. Comparison of flux spectra with McCARD.

4. Conclusions and Future Works

A flexible geometry module has been implemented in the GPU-based MC code PRAGMA exploiting a state-of-the-art ray tracing engine OptiX. MC neutron tracing problem has an analogy with the ray tracing problem in graphics, and in this regard, an interdisciplinary research that integrates the outcomes of computer graphics fields into the reactor physics had been performed.

A novel algorithm named Conditional Iterative Ray Tracing (CIRT) was developed to effectively query the cell indices under the ray tracing framework. Verification results for non-lattice problems such as the Lady Godiva device and the CANDU6 fuel bundle demonstrated that the ray tracing libraries developed for graphics purposes can be suitably utilized for physics simulations as well.

However, still substantial elaboration and optimization of the module are needed. The self-intersection issue has not been completely resolved yet, and this might lead to unexpected problems. Furthermore, the performance of the module has not been tuned. OptiX employs bounding volume hierarchy (BVH) to accelerate the scene traversal for high-performance ray tracing. To exploit the feature, an efficient node graph of the geometry objects should be constructed. Thus, developing an algorithm to effectively organize the primitives into group objects for the reactor geometries is crucial.

ACKNOWLEDGMENTS

This work was supported by KOREA HYDRO & NUCLEAR POWER CO., LTD (No. 2018-Tech-09).

REFERENCES

- [1] N. Choi, K. M. Kim, H. G. Joo, "Initial Development of PRAGMA – A GPU-Based Continuous Energy Monte Carlo Code for Practical Applications" Transaction of the Korean Nuclear Society Autumn Meeting, Goyang, Korea, Oct. 24-25 (2019).
- [2] S. Parker *et al.*, "OptiX: A General Purpose Ray Tracing Engine," ACM Transactions on Graphics Vol. 29, No. 4, Article No. 66 (2010).
- [3] R. Bergmann and J. Vujčić, "Algorithmic Choices in WARP – A Framework for Continuous Energy Monte Carlo Neutron Transport in General 3D Geometries on GPUs," Annals of Nuclear Energy 77, pp. 176-193 (2015).
- [4] J. Salmon and S. Smith, "Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC," Proceedings of the 10th International Workshop on Performance Modeling, Benchmarking, and Simulation of High Performance Computer Systems (PMBS), pp. 19-29 (2019).
- [5] NVIDIA, "NVIDIA OptiX 6.0 – Programming Guide," Revision 321453 (2019).
- [6] K. Hormann and A. Agathos, "The Point in Polygon Problem for Arbitrary Polygons," Computational Geometry, 20(3), pp. 131-144 (2001).
- [7] H. J. Shim *et al.*, "McCARD: Monte Carlo Code for Advanced Reactor Design and Analysis," Nuclear Engineering and Technology 44(2), pp. 161-176 (2012).