Chapter 17
A Robust Algorithm for Voxel-to-Polygon Mesh Phantom Conversion

Justin L. Brown, Takuya Furuta, and Wesley E. Bolch

17.1 Introduction

Since their early development in the late 1950s, general-purpose Monte Carlo (MC) radiation transport codes have utilized primitive geometric structures to define material interfaces in their transport geometry, e.g., planes, spheres, ellipsoids, and truncated cones. These structures were used from the 1960s to mid-1980s to geometrically represent the human body in both its outer body contour and internal organ structure. Their geometric simplicity was ideal for the limited computer technology at the time, and addressed the need for computational efficiency in particle tracking. These "stylized" phantoms, while at the time fit for purpose, did not provide an anatomically realistic representation of the human body, particularly in regard to organ shape and inter-organ tissue separation.

Beginning in the late 1980s, the need for improved anatomical accuracy, along with concurrent advances in computational memory and processor speed, led to the subsequent development and use of voxel-based human computational phantoms. Those phantoms were defined by a collection of rectangular parallelepipeds (voxels) of equal or non-equal size defining each tissue material. Voxel phantoms originate from the segmentation of CT or MR image data sets. Consequently, all tissue elements within a voxel phantom are generally of uniform size and shape (x,y,z dimen-

J. L. Brown
Medical Physics Graduate Program, University of Florida, Gainesville, FL, USA

T. Furuta
Nuclear Science and Engineering Centre, Japan Atomic Energy Agency, Tokai, Ibaraki, Japan

W. E. Bolch (✉)
Medical Physics Graduate Program, University of Florida, Gainesville, FL, USA

J. Crayton Pruitt Family Department of Biomedical Engineering, University of Florida, Gainesville, FL, USA
e-mail: wbolch@ufl.edu

S. Makarov et al. (eds.), *Brain and Human Body Modeling*,
https://doi.org/10.1007/978-3-030-21293-3_17

sions). The transition from stylized to voxel phantoms necessitated an increase in computational steps during radiation transport, as boundary crossing checks shifted from those associated with entering or leaving an organ or body region to those associated with entering or leaving each voxel defining that organ or body region.

Beginning in the early 2000s, a third generation of human computational phantom – mesh phantoms – was advanced, in which body regions and internal organ structures were once again represented, not by a collection of voxels, but by surfaces defined by 3D control points (non-uniform rational B-splines or NURBS) or arrays of polygons. These mesh-type phantoms allowed for the scalability and deformability provided by stylized phantoms, yet they retained the anatomical realism of voxel phantoms. In the coupling of mesh phantoms to radiation transport codes, however, a final step of voxelization had to be performed as the particle tracking algorithms employed at that time did not recognize NURBS or polygon mesh surfaces. Mesh phantom voxelization thus entailed filling these surfaces with an array of voxels of user-defined dimensions. A second advantage of mesh phantom voxelization was a resolution of potential surface overlaps and intersections introduced during phantom construction, rescaling, and/or deformation. The voxelization processes, by definition, eliminated these tissue incongruencies. Within the past few years, however, significant advances have been made in particle tracking algorithms so as to now enable the direct use of meshed geometries during MC radiation transport simulation. These developments were initially introduced into the MCNP code in 2009 [1], into the GEANT4 code in 2013 [2], and into the PHITS code in 2015 [3, 4]. Thus, there are a tremendous number of existing voxel-based computational phantoms that would now benefit from a conversion to mesh-type format.

This chapter reviews a computational algorithm developed to convert voxel phantoms to polygon mesh phantoms suitable for MC transport and importable into modern CAD software. The method eliminates geometric redundancies, allowing for a minimal and optimized geometric representation of the meshed structures. This feature is beneficial for computational human phantoms as voxel size is typically governed by the smallest anatomical structure to be represented, while a mesh phantom is not limited in this respect. The resulting algorithm allows users to continue to use the significant number of existing voxel phantoms that have been developed over the past 20 years without the need for labor-intensive manual modification. Additionally, the algorithm can be used with segmented image data to form mesh geometries free of intersections and incongruences so as to be used in simulation or CAD software.

17.2 Materials and Methods

17.2.1 Voxel to Mesh Conversion Procedure

The voxel-to-mesh conversion procedure is divided into six main steps: (1) data preparation, (2) gridded surface generation, (3) surface simplification, (4) line simplification, (5) polygon detection, and (6) polygon correction. The details of each

step are briefly described in this section, which also includes a discussion of the benchmarking procedures used to evaluate the conversion process.

Data Preparation

First, the phantom voxels are read into a three-dimensional array of specified size $<n_x, n_y, n_z>$. Next, two additional four-dimensional arrays are created of dimensions $<n_x, n_y, 3, 12>$ and $< n_x, n_y, 3, 8 >$ which represent sliding windows of temporary data used to ensure the uniqueness of every facet, vertex, and line that is generated in the newly created mesh phantom. The guarantee of element uniqueness is important to minimizing subsequent memory requirements during the handling of arbitrarily large arrays. It is important to note that the z-axis is chosen to be 3 units wide as this is typically the dimension along the phantom's cranial-caudal (and longest) direction this is chosen to minimize memory requirements. The array is 3 units wide as only adjacent z-slices of voxels can possibly contain information relevant to the current voxel. Several other arrays are also generated:

- The vertex array – an array of 3D points
- The line array – an array containing two integers representing two connected vertices within the vertex array
- The facet array – an array containing arbitrary numbers of integers representing connected lines within the line array
- The facet tag array – an array containing the ordered materials which separate the facets.

Gridded Surface Generation

The voxel data is parsed after the data is initialized. Each voxel is checked to determine if neighboring voxels are of a different material from the current voxel. If the neighboring voxels are of the same material, nothing is generated. If a neighboring voxel is found to be of a different material, the next step is to determine the facets to be produced. At this step, a facet is simply a rectangle between two voxels of different materials. As shown in Fig. 17.1, there is a possibility of 6 facets, 8 vertices, and 12 lines that could be produced for each voxel. Facets, vertices, and lines are produced depending on which adjacent voxels are of different materials. Given which neighbors are different materials, the required lines and vertices are determined. Once the required lines and vertices are determined, the sliding window of vertices and lines is checked to determine if this information already exists. If the data has already been generated, it is added to the current voxel position within the sliding window. If the data are not present, the data are generated and stored appropriately. The position of the vertices in 3D space is given by the required facets. At this point, a facet is composed of only four lines forming a rectangle. This process is repeated throughout the phantom array as the window is shifted along the longest axis through which it iterates. At this step, a surface mesh phantom has been

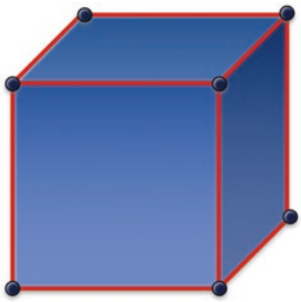

Fig. 17.1 Example of a single voxel and its potential 6 facets (blue), 8 vertices (black), and 12 lines (red)

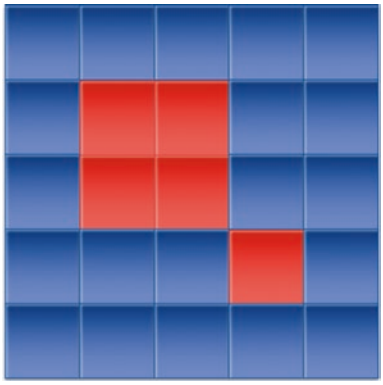

Fig. 17.2 A two-dimensional example of a voxelized surface after it has been converted to a gridded mesh. One material is depicted in blue and the other in orange

generated whose boundaries only differ between different materials (e.g., organs and tissue material of a given elemental composition and mass density). These boundaries are represented by a gridded surface which is further simplified and optimized as shown in Fig. 17.2.

Surface Simplification

The surface simplification process can begin once all necessary facets, vertices, and lines have been generated. Surfaces are first grouped by three values: (1) separated material, (2) whether or not x, y, or z is constant, and (3) the value of this constant. This grouping results in sets of surfaces which all separate the same material and are co-planar to one another (see Fig. 17.3). The purpose of this grouping is twofold. First, the grouping reduces the required computation time for the surface simplification step as comparisons only need to be made between grouped facets rather than across the entire list. Second, this grouping allows the surface simplification step to be performed in parallel.

The facets are then merged after grouping facets of the same material. A Boolean union operation is performed for every facet within each group. To determine if co-planar facets can be merged, the facets are checked to see if they share a common

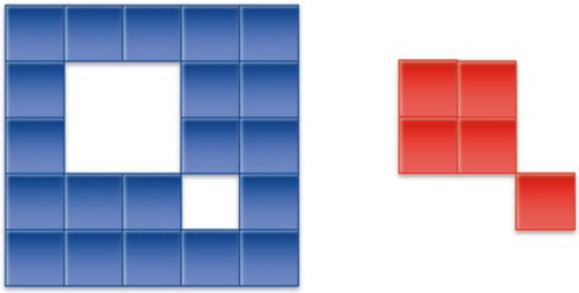

Fig. 17.3 Illustration of the facet grouping procedure depicting separation of facets into coplanar groups of the same separated material

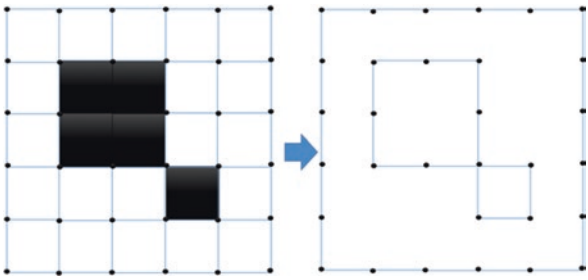

Fig. 17.4 Illustration of the surface simplification process for the blue surface group in Fig. 17.3. The black area on the left image depicts space no longer occupied by polygons

line. If the two facets indeed share one line, then they may be combined. The Boolean union process involves three sub-steps:

(i) The common line is determined and removed from both facet 1 and facet 2.
(ii) The remaining lines of facet 2 are added to facet 1.
(iii) Facet 2 is marked for removal.

This process is repeated until no more facets can be incorporated within each facet group. The facets marked for removal are then removed from the array. After this process is completed, unused lines and vertices are removed and facets and lines are updated to reflect new vertex and line positions within their respective arrays. At this point in the conversion process, the phantom surface mesh has been reduced to a minimal number of polygons, as shown in Fig. 17.4.

Line Simplification

After a minimum number of polygon surface representations have been generated, these polygons contain more lines than are necessary to enclose the required volume (e.g., organ or body region). Prior to simplifying the lines, they are grouped in a similar manner to the facets. First, lines are scanned iteratively to determine for every vertex how many lines use that vertex. Next, lines are subdivided into co-linear groups. This subdivision allows for the line simplification process to be performed in parallel and thus minimizes the required processing time needed since fewer comparisons need to be made.

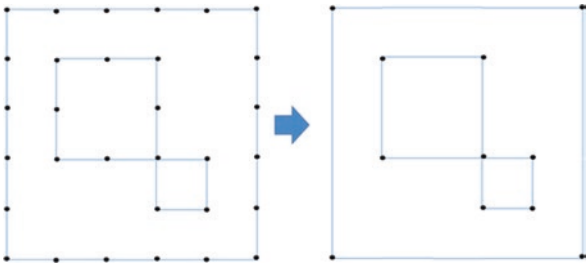

Fig. 17.5 Illustration of the line simplification process for the simplified surface group in Fig. 17.4

To simplify lines, each group of co-linear lines is scanned iteratively. Line pairs are flagged if both lines share a common vertex. If the lines share a vertex, a Boolean union operation can be performed if the vertex in common is only used by two lines globally within the phantom. If this is the case, the two lines and the vertex are not necessary to properly represent a given surface, and thus they can be removed without inducing a mesh overlap. The Boolean union process is performed for these two lines in a manner similar to that used for the facets:

(i) The common vertex shared by only line 1 and line 2 is determined.
(ii) The shared vertex in line 1 is replaced by the unshared vertex in line 2.
(iii) Line 2 is marked for removal.

This process is repeated until no additional lines can be incorporated within each group of lines. The lines marked for removal and all unused vertices are then removed. The line and facet arrays are then updated to reflect the new position of the vertices and lines in their respective arrays. At this stage, the mesh phantom is represented by the minimum possible number of surfaces and these surfaces are represented by the minimum possible number of lines as demonstrated in Fig. 17.5.

Polygon Detection

After these two simplification processes, the facets are now composed of an unordered set of lines and polygons. By construction, the lines contained within each facet must form at least one closed loop (i.e., a polygon). Within each facet, polygons are formed by simply end matching lines until all lines are used. If multiple polygons are formed by construction within one facet, one of these polygons must be interior to the other, thus forming a hole within the outer polygon. This is easily determined by a bounding box as demonstrated in Fig. 17.6. This process is repeated for each facet.

Polygon Correction and Hole Detection

Even though the technique described herein is computationally efficient, using an end-matching method to construct polygons can possibly create self-intersections within each polygon. These may occur because vertex repetition is not checked as each line is added to the polygon as it would result in a significant decrease in

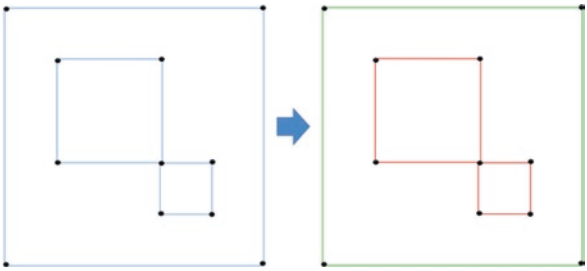

Fig. 17.6 Illustration of the polygon detection process for the simplified set of lines in Fig. 17.5

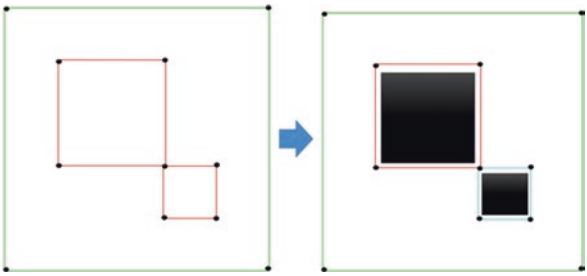

Fig. 17.7 Illustration of the polygon correction and hole detection process for the simplified set of lines in Fig. 17.6. Each color represents a polygon formed in the facet, with the black color illustrating the presence of a hole in the facet

computational efficiency. Instead, after polygons have been created, they are checked to see if any vertices other than the start/end vertex have been used multiple times. An "ear-clipping" method is employed in this situation. To ear-clip a polygon, one creates a new polygon from the lines between the vertex that is used multiple times. These lines are then removed from the larger polygon and a new polygon is added to the facet as demonstrated in Fig. 17.7. At this point, the mesh is now an intersection-free and redundancy-free (all vertices, lines, and facets are unique) mesh that is represented by the least number of surfaces. If the surface-mesh phantom is to be converted to a tetrahedral-mesh phantom, as required by the PHITS radiation transport code, the open-source conversion code TETGEN [5] may be utilized. The mesh can also be triangularized and exported in a file format acceptable to most modern CAD software codes.

17.2.2 Conversion Process Benchmarking

In testing the performance of the voxel-to-mesh conversion algorithm, two benchmarking tasks were performed. First, it was important to test that the algorithm is robust and can handle arbitrary datasets correctly. Thus, a series of random square binary voxel arrays were generated and then meshed to contain between 10^3 and 10^8 elements. One example is shown in Fig. 17.8. Second, it was important that the conversion algorithm performed efficiently in a practical setting. Thus, mesh conversions were applied to the UF/NCI reference adult male phantom [6] at voxel resolutions ranging from 1 cm^3 to 1 mm^3 as shown in Fig. 17.9. Finally, it was

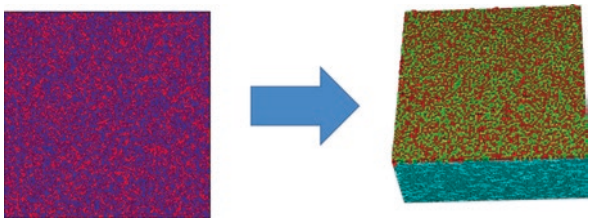

Fig. 17.8 Example of a 10^6 random binary voxel array (left) and its converted meshed format (right)

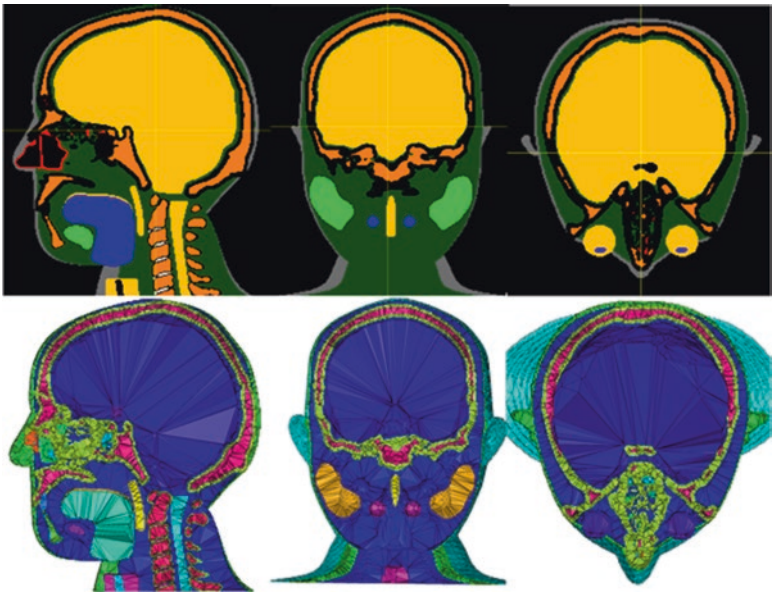

Fig. 17.9 Example of a voxel phantom (resolution of 1 mm^3) (top) converted into a mesh format (bottom)

important to assess how this conversion algorithm scales across multiple processors. Thus, the previous two benchmarking studies were performed using 1, 2, 4, 8, and 16 cores, respectively. All benchmarking tasks were run on the UF HiPerGator cluster using Intel E5-2698 v3 (2.3 GHz) processors. The code was compiled using Intel's C++ compiler with the –qopenmp and –O3 compiler flags.

17.3 Results

For the random array meshing benchmarks on a single core, the time to mesh for the highest resolution dataset ($250 \times 250 \times 250$) was 2.5×10^4 seconds, while the conversion time for the highest resolution head phantom was approximately 350 seconds.

Looking at the time breakdown for each step in the meshing algorithm, the majority of the compute time is devoted to the surface simplification step (see Sect. 2.1.3), which is expected as this step iteratively compares facets to one another causing this portion of the algorithm to have an order of n^2 performance (see Fig. 17.10). For the voxel-to-mesh phantom conversion, a more linear performance is seen, but this is

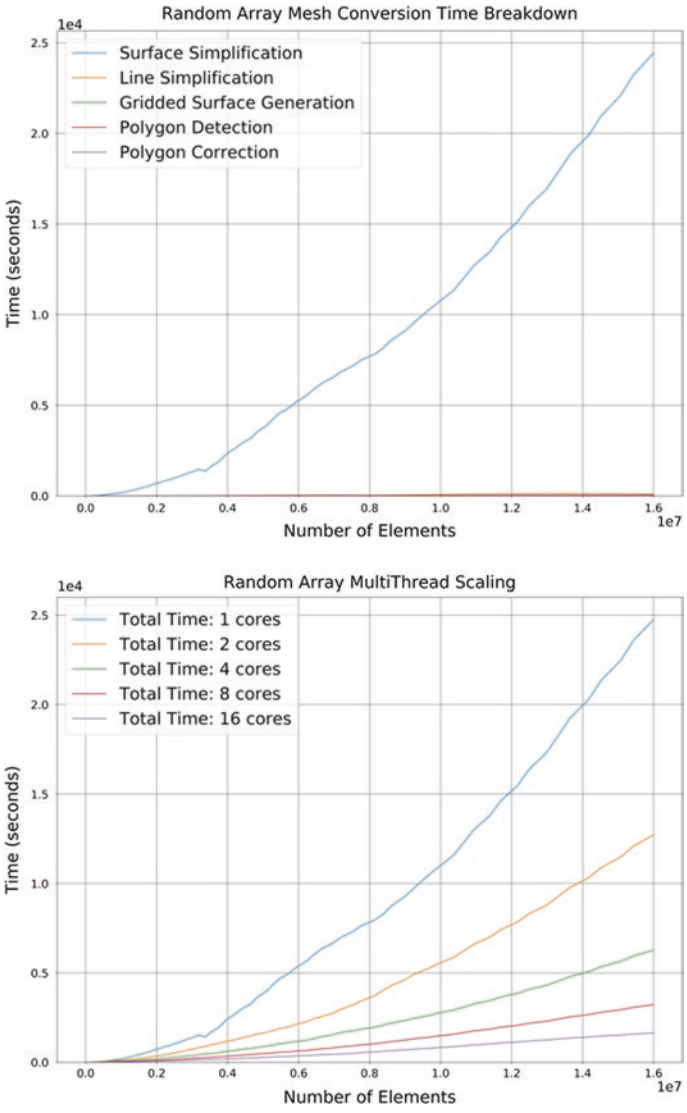

Fig. 17.10 Random array meshing time (units: 10^4 s) results per step (units: 10^7 steps) (top) and per multiprocessor scaling (bottom)

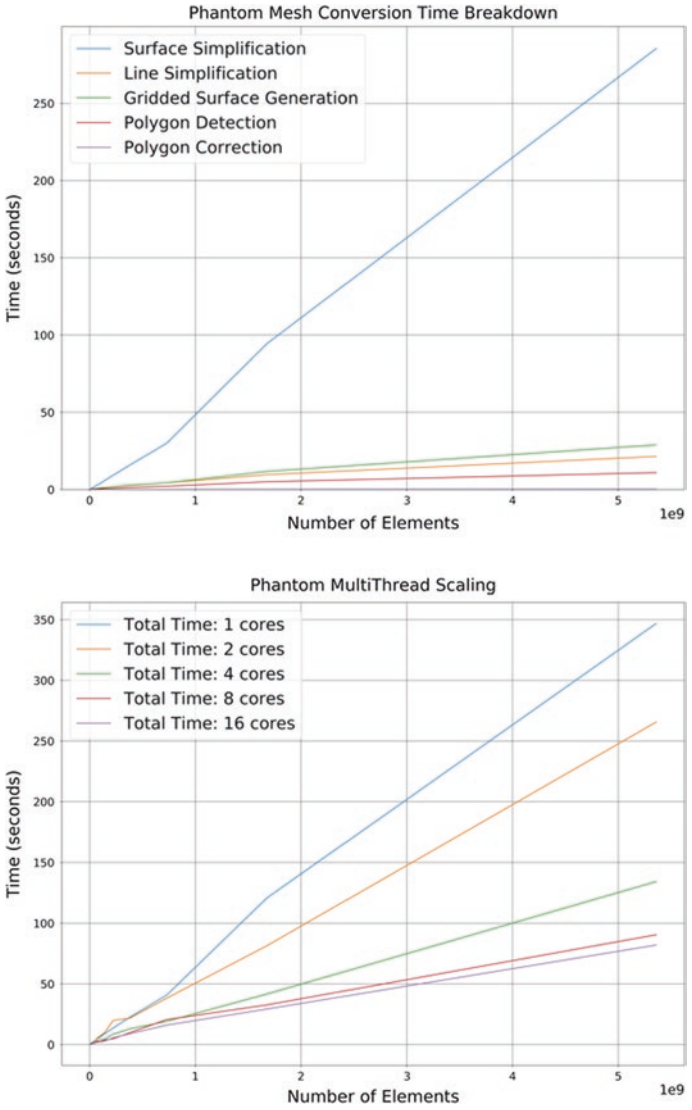

Fig. 17.11 Voxel phantom meshing time (units: s) results per step (units: 10^9 steps) (top) and per multiprocessor scaling (bottom)

likely due to the less randomized nature of the problem (see Fig. 17.11). Across multiple processors, both benchmarks saw performance gains although, as expected, they are not linear. The voxel-to-mesh phantom conversion speedup for 16 cores was approximately a factor of 4.1, whereas for 8 cores it was only a factor of 3.7. The diminishing returns are likely due to the implementation of OpenMP scheduling. The process can be better optimized in future development of this algorithm.

17.4 Conclusions

The presented methodology provides a fast and efficient method to convert voxel data to a polygon mesh format, containing no degenerate facets and no self-intersections, thus making it useful for input to Monte Carlo sampling codes and CAD programs. The algorithm can convert any segmented set of voxelized data to an optimized meshed surface suitable for a variety of applications such as Monte Carlo radiation transport or finite element simulations of the interactions between electromagnetic fields and the human body, e.g., during MRI.

Acknowledgments This work was supported in part by Contracts T72472 and T73057 with Wyle Laboratories and NASA Johnson Space Center, and grant R01 EB013558 with the National Cancer Institute.

References

1. Werner, C. J., et al. (2018). Title: MCNP Version 6.2 Release Notes. In *LA-UR-18-20808* (pp. 1–39).
2. Geant4 Collaboration 2017 Introduction to GEANT4, Release 10.4. https://geant4.web.cern.ch/support/user_documentation.
3. Sato, T., et al. (2018). Features of Particle and Heavy Ion Transport code System (PHITS) version 3.02. *Journal of Nuclear Science and Technology, 55*(6), 684–690.
4. Furuta, T., et al. (2017). Implementation of tetrahedral-mesh geometry in Monte Carlo radiation transport code PHITS. *Physics in Medicine and Biology, 62*(12), 4798–4810.
5. Si, H. (2015). TetGen, a quality tetrahedral mesh generator. *ACM Transactions on Mathematical Software, 41*(2), 11.
6. Lee, C., Lodwick, D., Hurtado, J., Pafundi, D., Williams, J. L., & Bolch, W. E. (2010). The UF family of reference hybrid phantoms for computational radiation dosimetry. *Physics in Medicine and Biology, 55*(2), 339–363.