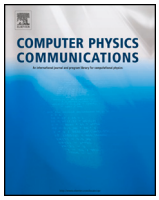
Computer Physics Communications 298 (2024) 109091

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc



Computer Programs in Physics

Enabling particle transport on CAD-based geometries for radiation simulations with penRed ☆,☆☆

S. Oliver [a], S. Rodriguez Bosca [b], V. Giménez-Alventosa [c,d,*]

[a] Instituto de Seguridad Industrial, Radiofísica y Medioambiental (ISIRYM), Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain
[b] European Laboratory for Particle Physics, CERN, 1211 Geneva 23, Switzerland
[c] Departament de Física Atòmica, Molecular y Nuclear, Universitat de València (UV), 46100 Burjassot, Spain
[d] Institut de Física Corpuscular, IFIC (UV-CSIC), 46100 Burjassot, Spain

ARTICLE INFO

Keywords:
Radiation transport
PENELOPE physics
Monte Carlo simulation
PenRed
CAD
Triangular surface mesh

ABSTRACT

Geometry construction is a fundamental aspect of any radiation transport simulation, regardless of the Monte Carlo code being used. Typically, this process is tedious, time-consuming, and error-prone. The conventional approach involves defining geometries using mathematical objects or surfaces. However, this method comes with several limitations, especially when dealing with complex models, particularly those with organic shapes. Furthermore, since each code employs its own format and methodology for defining geometries, sharing and reproducing simulations among researchers becomes a challenging task. Consequently, many codes have implemented support for simulating over geometries constructed via Computer-Aided Design (CAD) tools. Unfortunately, this feature is lacking in penRed and other PENELOPE physics-based codes. Therefore, the objective of this work is to implement such support within the penRed framework.

New version program summary

Program Title: Parallel Engine for Radiation Energy Deposition (penRed)
CPC Library link to program files: https://doi.org/10.17632/rkw6tvtngy.2
Developer's repository link: https://github.com/PenRed/PenRed
Code Ocean capsule: https://codeocean.com/capsule/1041417/tree
Licensing provisions: GNU Affero General Public License v3
Programming language: C++ standard 2011.
Journal reference of previous version: V. Giménez-Alventosa, V. Giménez Gómez, S. Oliver, PenRed: An extensible and parallel Monte-Carlo framework for radiation transport based on PENELOPE, Computer Physics Communications 267 (2021) 108065. doi:https://doi.org/10.1016/j.cpc.2021.108065.
Does the new version supersede the previous version?: Yes
Reasons for the new version: Implements the capability to simulate on CAD constructed geometries, among many other features and fixes.
Summary of revisions: All changes applied through the code versions are summarized in the file *CHANGELOG.md* in the repository package.
Nature of problem: While Monte Carlo codes have proven valuable in simulating complex radiation scenarios, they rely heavily on accurate geometrical representations. In the same way as many other Monte Carlo codes, penRed employs simple geometric quadric surfaces like planes, spheres and cylinders to define geometries. However, since these geometric models offer a certain level of flexibility, these representations have limitations when it comes to simulating highly intricate and irregular shapes. Anatomic structures, for example, require detailed representations of organs, tissues and bones, which are difficult to achieve using basic geometric objects. Similarly, complex devices or intricate mechanical systems may have designs that cannot be accurately represented within the constraints of such geometric models. Moreover, when the complexity of the model increases, geometry construction process becomes more difficult, tedious, time-consuming and error-prone [2].

☆ The review of this paper was arranged by Prof. W. Jong.
☆☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author at: Departament de Física Atòmica, Molecular y Nuclear, Universitat de València (UV), 46100 Burjassot, Spain.
E-mail address: gialvi@uv.es (V. Giménez-Alventosa).

https://doi.org/10.1016/j.cpc.2024.109091
Received 12 October 2023; Received in revised form 5 January 2024; Accepted 8 January 2024
Available online 22 January 2024

Also, as each Monte Carlo geometry library uses its own format and construction method, reproducing the same geometry among different codes is a challenging task.

Solution method: To face the problems stated above, the objective of this work is to implement the capability to simulate using irregular and adaptable meshed geometries in the penRed framework. This kind of meshes can be constructed using Computer-Aided Design (CAD) tools, the use of which is very widespread and streamline the design process. This feature has been implemented in a new geometry module named "MESH_BODY" specific for this kind of geometries. This one is freely available and usable within the official penRed package[1]. It can be used since penRed version 1.9.3*b* and above.

References

[1] V. Giménez-Alventosa, V. Giménez Gómez, S. Oliver, PenRed: an extensible and parallel Monte-Carlo framework for radiation transport based on PENELOPE, Comput. Phys. Commun. 267 (2021) 108065, https://doi.org/10.1016/j.cpc.2021.108065.

[2] Paul P.H. Wilson, Timothy J. Tautges, Jason A. Kraftcheck, Brandon M. Smith, Douglass L. Henderson, Acceleration techniques for the direct use of CAD-based geometry in fusion neutronics analysis, Fusion Eng. Des. 85 (2010), https://doi.org/10.1016/j.fusengdes.2010.05.030.

1. Introduction

Monte Carlo (MC) simulations of radiation transport are currently applied in several fields of research such as: radiation therapy and protection, electromicroscopy, x-ray fluence, accelerator technology and detector characterization, among other applications. Due to the huge amount of possible applications, and with the purpose of covering different configurations of the simulation system (particle types, physics models, energy ranges, geometry types, etc), several MC codes have been developed over the years. Some of them are EGSnrc [1], MCNP [2], PENELOPE [3], Geant4 [4], PHITS [5] or FLUKA [6]. In addition to these codes, there are many others that stem from them, specialized in specific applications or adding new capabilities.

One of the latest additions to this list of MC frameworks is the Parallel ENgine for Radiation Energy Deposition (penRed) code [7], a highly parallel, efficient, and adaptable framework for MC simulations. Written in C++, penRed incorporates the physics and functionalities of PENELOPE, making it a general purpose framework. In addition, penRed includes many other tallies and functionalities, especially for the field of medical physics [8].

While MC codes have proven valuable in simulating complex radiation scenarios, they rely heavily on accurate geometrical representations. In the same way as many others MC codes listed, penRed employs simple geometric quadric surfaces like planes, spheres and cylinders to define geometries. However, since these geometric models offer a certain level of flexibility, these representations have limitations when it comes to simulating highly intricate and irregular shapes. Anatomic structures, for example, require detailed representations of organs, tissues and bones, which are difficult to achieve using basic geometric objects. Similarly, complex devices or intricate mechanical systems may have designs that cannot be accurately represented within the constraints of such geometric models. Moreover, when the complexity of the model increases, geometry construction process becomes more difficult, tedious, time-consuming and error-prone [9].

In addition to this type of geometry construction, some MC codes, including penRed, allow the use of voxelized geometries. This can be useful when simulating, for example, phantoms based on scanner images. However, voxelized phantoms have a drawback in that they offer limited resolution due to the size of the voxels. Another complication is the lack of flexibility, making them challenging to translate or deform objects within the phantom in a simple way.

At present, due to the advancements in computer-aided design (CAD) technologies, the usage of software tools based on a graphical user interface (GUI) for three-dimensional geometry construction has significantly increased in recent years. Using CAD software, it is possible to create more complex and flexible 3D geometries with less effort and difficulty compared to create them using a set of primitive solids. Furthermore, CAD geometries help streamline the process of their constructions, especially when simulating elements of experimental setups that have been previously designed using this type of software. This approach avoids introducing user-induced errors in the geometry, which results in reducing errors in the MC simulation, saving time and effort in the geometry creation processes. This can be particularly usable when combining geometries with structural information from patients and applicators, for example.

To address these limitations of mathematical surfaces and voxelized geometries, some MC codes have implemented the possibility of defining geometries based on triangular meshed surfaces or irregular volumetric meshes of tetrahedral volumes, allowing the straightforward definition of complex structures using existing 3D design codes. Both approaches are flexible enough to accurately represent complex structures in three dimensions. For these reasons, such modeling capabilities have become increasingly prevalent in fields such as medical physics, radiation therapy and biomedical engineering, where accurate representation of complex anatomical structures and devices is crucial for precise simulations and treatment planning. Moreover, as this kind of geometry definition can be converted from standardized mesh formats, this approach streamlines the reproduction of simulations with different codes and research groups since the geometry can be shared easily.

Therefore, to achieve the benefits stated above, the objective of this work is to implement the capability to simulate on irregular and adaptable meshed geometries in the penRed framework. This has been done implementing a new geometry module named "MESH_BODY" specific for this kind of geometries, which is freely available and usable within the official penRed package.[2] It can be used since penRed version 1.9.3*b* and above.

2. State of the art

In this section the authors aim to describe the meshed geometries supported by different transport codes.

In 2004, Geant4 implements the `G4Tet` class to allow the capability of realize a tetrahedral meshes using a large number of individual tetrahedron objects: `G4Tets` [10]. Besides this feature, the `G4TessellatedSolid` class allows to implement directly in Geant4 a CAD-based geometry as polygon-mesh. However, when a very accurate modeling is need or a highly complex geometries are intended to be used, a large number of facets are required, which results in a long computation time. In 2013, with the aim to address this speed limitation, a new Geant4 solid class, `DagSolid` was developed [11].

[1] https://github.com/PenRed/PenRed.

[2] https://github.com/PenRed/PenRed.

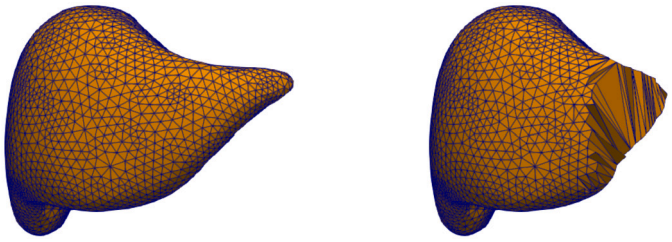


Fig. 1. Volumetric meshed liver model from the adult mesh-type reference computational phantom of the ICRP Publication 145 [16]. At left, the external surface of the liver is shown. On the right image, the same model was cut to show the internal structure with volumetric elements.

The `DagSolid` class is based on the direct accelerated geometry for the Monte Carlo (DAGMC) [9]. This software package consists on a ray-tracing acceleration using a hierarchical structure of oriented bounding box (OBB) tree with the aim of improving the calculation speed of interference detection. The authors of this work have been integrated DAGMC into many Monte Carlo radiation codes, including MCNP5 [12], MCNP6, Geant4, FLUKA, Tripoli4, OpenMC, and Shift codes. As example case, the integration with MCNP5 has been used to perform fusion neutronics analysis [9]. Later, in 2012, Los Alamos National Laboratory (LANL) develops an extension in MCNP version 6 to provide the capability for neutrons and photon tracking on an unstructured mesh [13] with no need of external tools.

Moreover, several other codes offer the ability to simulate intricate geometries, using methods like tetrahedral unstructured meshes or triangular surface meshes, to face the difficulties stated above. For instance, EGSnrc [1] includes the *EGS_Mesh* class for this specific purpose.

Speaking about PENELOPE-based codes, the tool penmesh [14] was designed to work with geometric structures based on triangular mesh surfaces and relies on the PENELOPE 2006 libraries. However, as far as the authors know, penmesh is no longer actively supported or available for use with the most recent PENELOPE versions. As a result, not only users of penRed but also those who use PENELOPE directly can find value in having this capability integrated.

3. Material and methods

The problem addressed in this work is not how to implement a ray-tracing algorithm for mesh-based geometries, as it is a well-known problem in computer graphics. Instead, the goal is on efficiently implementing it in the penRed environment to facilitate simulations in complex and organic geometries. To achieve it, this section describes the types of meshes that will be supported and outline a generic algorithm for incorporating new geometries into the penRed framework. Then, the validation tests and benchmarks are detailed.

3.1. Surface versus volumetric unstructured meshes

Firstly, we must decide whether to implement transport on volumetric or surface-based meshes in order to achieve the best balance for our simulations.

Volumetric unstructured meshes are commonly employed in finite element simulations, such as computational fluid dynamics problems [15]. In such scenarios, complex geometries are discretized into small volume elements (see Fig. 1), with each element storing the local mean state of the substance within its corresponding volume. These local states are then utilized to update the entire system based on a physical model, typically involving interactions with neighboring volume elements.

However, in radiation transport simulations, particles are individually simulated rather than being modeled as a substance filling the entire geometric system. Consequently, the discretization into smaller volume elements becomes unnecessary, and the geometry can be effectively represented using only the surface mesh with an empty interior volume. This approach is similar to the techniques employed in video games or animation rendering.

To determine the most suitable approach for penRed, we will analyze the advantages and disadvantages of both volumetric and surface-based meshes. It is important to note that regular volumetric meshes, such as voxelized geometries, are not under consideration, as they lack the capability to accommodate complex and organic surfaces and are already integrated into the penRed package.

On one hand, unstructured volumetric meshes use larger amount of memory to store the geometry compared to their surface-based counterparts. This is because the extra discretization of the geometry in smaller volume elements. The number of those elements depends on the desired resolution and the boundary shapes. Nevertheless, one advantage of volumetric meshes is the possibility to calculate magnitudes within the mesh element volumes themselves, especially useful for visualization and when the bodies are heterogeneous, as each element can be assigned different physical properties. However, in many simulations, we use homogeneous media within each body, and the region to be visualized is not the entire geometry. Consequently, volumetric meshes introduce memory wastage in regions where the simulation does not tally results. If necessary, it may be more efficient to create a detailed volumetric mesh to tally results only in the regions of interest.

On the other hand, performance strongly depends on the geometry, simulation parameters, and the optimization techniques applied in each case. However, regardless of the type of mesh used, the fundamental calculation for particle transport involves determining the intersection point of the particle track with either the volume elements for unstructured volumetric meshes or the surfaces of objects for surface-based geometries. This task is commonly referred to as the ray-tracing problem. In both cases, triangles are commonly employed to define both volume elements and object surfaces, simplifying the performance comparison to the number of triangles that need to be checked for particle transport.

In the case of volumetric meshes, the calculation of ray-tracing depends on the number of inner volume elements and can escalate rapidly, especially for high resolutions or organic objects with intricate surfaces. Alternatively, for surface-based meshes, employing techniques like octrees to subdivide surfaces into regions, minimizing the number of ray-tracing checks [14], enables us to restrict the testing of surface triangles to a small region close to the particle track. Moreover, even for complex geometries, the required memory is typically small enough to store the entire surface in GPU memory, allowing the simulation to leverage GPU acceleration for ray-tracing. However, implementing GPU acceleration for particle transport is beyond the scope of this work and may be considered in future versions.

For these reasons, in this work, we implement particle transport using ray-tracing on surface meshes. Specifically, we use triangular meshes, as this is a well-studied problem in computer graphics [17–19]. The algorithm used to calculate the ray-triangle intersection was presented by Möller [17], providing a good balance between memory usage and speed.

3.2. Algorithm

In this section, the algorithm implemented in penRed to model the particle tracking through geometries as a ray-tracing problem with surface-based elements is described.

In this algorithm, some restrictions will be imposed on the geometry definition in order to speed up the calculation without affecting the geometry construction flexibility. First, objects cannot intersect, understanding it as their surfaces crossing, but overlapping surfaces is allowed. This is not a significant limitation since the surface mesh provides enough flexibility to prevent object intersections. This approach

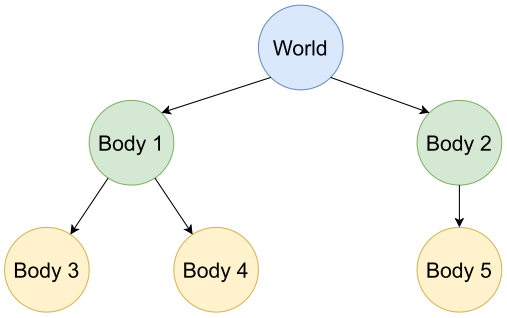


Fig. 2. Object genealogical tree.

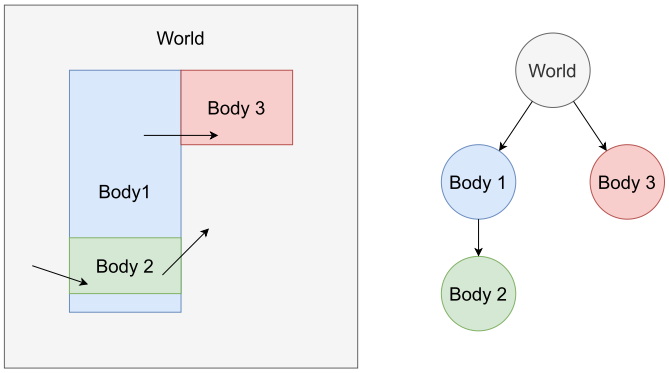


Fig. 3. Overlapping objects example. Geometry representation (left) and the corresponding genealogical tree (right). The geometry arrows represent particles crossing between objects.

allows us to structure the geometry objects with a genealogical tree as it is shown in Fig. 2, where each body completely contains its daughters.

Hence, there is no necessity to calculate the ray-tracing for all existing body surfaces to perform particle movement within the geometry, resulting in a reduction of the overall computational cost. In the initial stage, a particle situated within a specific body only needs to check its boundaries and the immediate offspring boundaries. Therefore, during this initial check, particles exclusively consider the bodies connected on the genealogical tree with the one containing them. However, upon crossing a boundary, it is essential to test if it overlaps with another surface. In such instances, the particle can skip intermediate bodies in the genealogical tree. The left image of Fig. 3 provides an example geometry which illustrates various overlapping cases, where each arrow denotes the movement of a particle within the geometry system. The starting point of the arrow indicates the initial body of the particle, while the endpoint indicates the final body after the particle movement. In addition, the right image of the Fig. 3 represents the corresponding genealogical tree. The three represented cases are described following,

- The top arrow is going from *Body 1* to *Body 3*. In this case, the particle skip the intermediate *World* body in the genealogical tree path, because bodies *1* and *3* share both, parent (*World*) and the crossed boundary by the particle.
- The bottom-left arrow is going directly from *World* to *Body 2*, although *Body 2* is not a direct *World* daughter, skipping *Body 1* in the genealogical tree path.
- Analogous to previous case, bottom-right arrow goes directly from *Body 2* to *World*, because the common boundary between *Body 1* and *Body 2* is crossed by the particle.

To determine if two or more bodies can share their boundaries, a collision box is constructed for each one, including all the body mesh triangles. Furthermore, this collision box is also used to check if a particle can cross any body triangle, avoiding calculations for distant bodies to speed up the simulation.

As illustrated in the previous example, the handling of the *World* body differs from others due to its lack of a parent. Consequently, if a particle crosses the *World* surface outward, it escapes from the geometry system. Furthermore, the *World* is treated as a surrounding object and is therefore not permitted to share boundaries with other bodies.

All these restrictions have been considered when defining the algorithm to move the particle within the geometry. Following the nomenclature used in PENELOPE and penRed, the geometry class method responsible for this process is the *step* function. The corresponding pseudocode is presented in Code 1, in the Appendix A, and described in this section.

In addition, generic assumptions required by any penRed geometry must be satisfied. These are outlined below:

- The geometry must consist of bodies filled with a single material. In this case, each body is confined by a mesh surface.
- **Material** index 0 is reserved for void regions.
- Optionally, bodies can be assigned a **detector** identifier.
- An **interface** is defined as a change in **material** or **detector** between bodies. Consequently, two bodies in contact with the same material do not constitute an interface unless they belong to different detectors.
- A particle escapes from the geometry system if it cannot reach a non-void region.
- When reaching an interface, the particle must be stopped unless the new material after crossing the interface is void. In such cases, the particle must be moved through the void until a non-void region is reached, or the particle escapes from the geometry.

For brevity, some functions of the code are not shown in this document. However, the complete code can be found in the penRed repository.[3] Note as well that this algorithm can be used to implement any geometry based on a single or collection of objects, as it is independent of their structure as long as it fulfills the penRed restrictions for geometry modules. These ones are detailed in the implementation documentation located in the penRed repository.[4]

The particle state includes information such as the position, direction, body, and material identifiers, among other quantities irrelevant to tracking. This information is necessary for the *step* routine, which needs to know the particle starting body and construct the ray based on the particle position and direction. Additionally, the *step* routine requires the maximum distance to travel in the current material (*DS*) and information about the geometry bodies (*bodies*), including the *world*.

As outputs, the *step* routine is expected to provide the updated particle state, including changes in position, material, and body index if a boundary has been crossed. It also calculates the distance traveled inside the original material (*DSEF*), the total distance traveled (*DSTOT*), considering the possibility of crossing void regions, and the number of bodies crossed (*NCROSS*).

Initially, the algorithm checks if the particle is inside the geometry system (Line 4). If not, the particle body index is expected to exceed the total number of bodies contained in the geometry. If it is outside, the algorithm then verifies if the particle is directed towards the *world* body. In that case (Lines 5 – 15), if the *world* has a non-void material, the particle movement is halted due to the material change (Lines 12 – 14). Otherwise, particle tracking continues, as the void region must be traversed. If the particle does not reach the geometry, it escapes, and the tracking terminates (Lines 16 – 23).

Next, we need to determine if the particle is within a void region or not, setting the *isInVoid* flag accordingly (Lines 26 – 29). Although the

[3] https://github.com/PenRed/PenRed.
[4] doc/PenRed_implementation_guide.pdf.

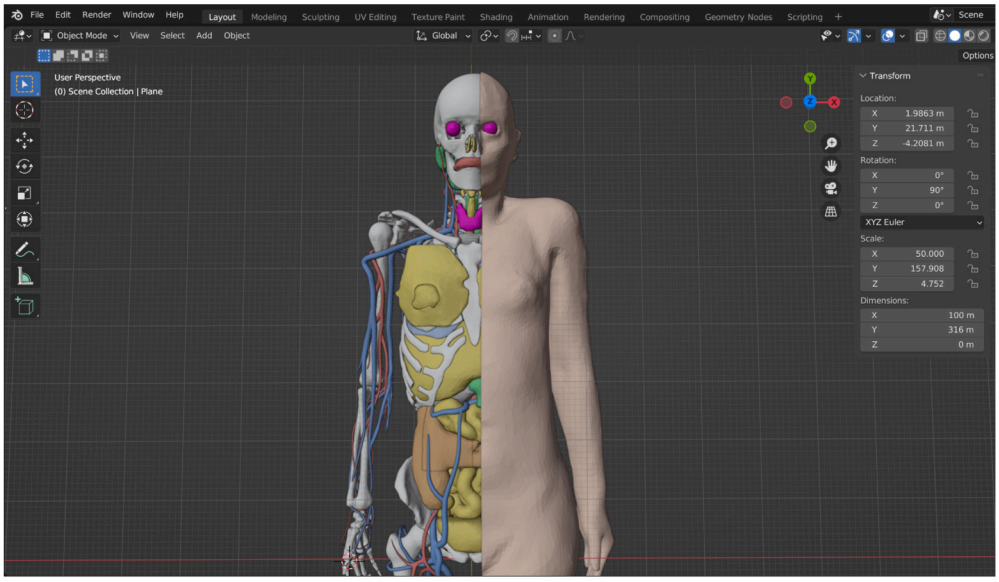


Fig. 4. ICRP145 female phantom mesh edited with Blender.

particle cannot be stopped in a void region once its simulation starts, it might be sampled in those regions before being moved to start the simulation. Subsequently, a loop initiates (Line 32), iterating over bodies as the particle moves through the geometry. The maximum distance to be traveled in each iteration depends on the body material. If it is a void region, no travel limit is applied; otherwise, the value of *DS* restricts the maximum distance to travel within the material (Lines 35 – 38). The maximum travel distance is constrained as it finds closer boundaries, avoiding computing distant bodies (Lines 40 – 50). Depending on the closest boundary, three possible cases must be considered, which determine the value of *travelType*:

- **0**: The particle cannot reach any boundary before the maximum travel distance. Therefore, the particle is moved, and the loop concludes (Lines 52 – 55).
- **1**: The particle crosses the current body boundary (Lines 56 – 70). If the crossed body is the *world*, the particle escapes from the geometry system, and the function terminates. Otherwise, overlapping boundaries with outer bodies are examined to determine the final body where the particle crosses.
- **2**: The particle crosses a daughter body boundary (Lines 71 – 73). Overlapping boundaries with inner bodies are examined to determine the final crossed body.

Once the closest distance and the next body to cross are determined, the particle is moved to the corresponding point (Line 84) and the variables *nextBody* and *nextMat* are assigned with the body index where the particle has moved and its corresponding material respectively (Lines 86 – 102). If *nextMat* is 0, it indicates that the particle is moving into a void region. Consequently, the count of bodies crossed (*NCROSS*) must be incremented if the particle is not already in a void region in order to prevent double counting (Lines 86 – 90). If *nextMat* is not 0 and the particle is situated in a void region, it must be halted, and the loop concludes (Lines 91 – 93). If it is not in a void region and the initial particle material differs from *nextMat*, the particle must also be halted due to the material interface (Lines 99 – 102). However, if both materials are identical, the particle will continue only if the detector number of the bodies match (Lines 95 – 98). Assigning bodies as "detectors" can be done during the simulation configuration to force an interface between bodies even if them share the same material. This feature is used in some tallies and, by default, a detector index 0 is assigned to all bodies. Once the loop is completed, the output values are updated, and the *step* function concludes.

3.3. Geometry construction

To construct geometries for simulation with penRed, the mesh geometry files must follow the format described in the documentation. However, users do not need to create these files manually nor convert them from other format themselves. Instead, we have developed a Python plugin for Blender[5] that facilitates the export of Blender-constructed geometries into the penRed format.

For instance, Fig. 4 depicts the Blender environment. In this case, the female phantom recently published by the International Commission on Radiological Protection (ICRP) in the ICRP145 [16] is imported into Blender, enabling the export of its geometry into the penRed format. This phantom has been chosen in this work because its high level of detail concerning the radiosensitive organs and tissues of the human body, and will be described in section 3.4.

In addition to harnessing Blender 3D environment and its versatile tools for constructing geometries, we can also leverage Blender ability to import a wide range of 3D mesh formats, including ABC, USD, OBJ, FBX, PLY, STL, and more. These imported meshes can then be exported to be used within penRed. The plugin itself, along with its documentation, is included in the penRed package for user convenience.

3.4. Validation tests and benchmark

To validate our implementation, we first compared the simulations available in the penRed examples. These examples are adaptations of the PENELOPE [3] package and have been carried out using both quadric and triangular mesh surface geometries. For the sake of brevity, we will discuss only three examples in this work:

- The *1-disc* example, with and without variance reduction techniques.
  The geometry of this example corresponds to a homogeneous disc phantom of Cu, with radius of 0.01 cm at plane XY and height of 0.005 cm at z axis with the base at $z = 0$ cm. The source is a point source of electrons, monoenergetic with energy of 40 keV located at $(x, y, z) = (0, 0, -0.0001)$ cm.

[5] https://www.blender.org/.

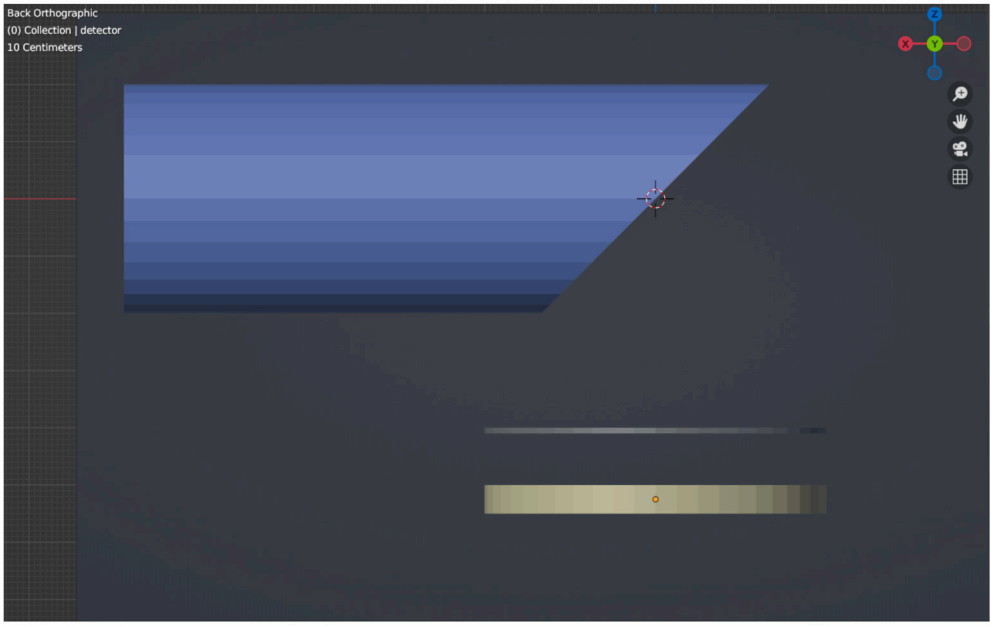


Fig. 5. X-Ray tube example geometry constructed with Blender. The components shown, arranged from top to bottom, are as follows: wolframium anode (blue), aluminium filter (gray) and silicon detector (yellow). (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

The quantities tallied in both versions of this example, are the cylindrical and spatial dose distribution, the energetic spectrum of emerging particles distribution, the energetic spectrum of particle fluence, the energy spectrum of particles impacting the disc, and the material energy deposition information. Notice that energy spectrums are normalized to the energy bin width.

The second version of this example, which incorporates variance reduction techniques, uses the same geometry as the first version and the same source configuration. In this scenario, interaction forcing, x-ray and bremsstrahlung splitting are activated. The configuration file specifies which body undergoes the variance reduction techniques (the disc phantom) and the type of particle and interaction to be forced-in, electrons in this case. In this version of the example, the number of histories of the simulation is reduced due to the applied splitting.

- The *3-detector-1* example features a geometry comprising a NaI cylindrical detector with a diameter and height of 5.08 cm, accompanied by a 1.27 cm Fe backing. Regarding the source configuration, this example is set up with a point-like Co-60 gamma-ray source emitting characteristic photons with equal probabilities of emission and energies of 1.17 and 1.33 MeV. The photons are directed to impinge on the NaI crystal from above. In this case, no variance reduction techniques are applied, and the tallied quantities include the total energy deposited in the detector material, the energy deposition spectrum registered by the NaI detector, as well as the energetic spectrum of emerging particles.
- The *4-x-ray-tube* example represents a simple x-ray generator. Its geometry includes a wolframium anode, an aluminium filter and a silicon detector, as it is shown in Fig. 5. In this example, the source is configured as a monoenergetic electron beam of 150 keV directed to the anode to produce bremsstrahlung photons. Since the efficiency of the photon production with this configuration is very low, variance reduction techniques are applied to increase the simulation efficiency forcing the photon production and splitting them. The simulation is configured to measure the energetic spectrum of particle fluence crossing the filter, the energy spectrums of particles impacting both, the filter and the detector, and the time of flight spectrum of impacting particles at the detector. Additionally, the energetic spectrum of emerging particles distribution and the energy deposition in all bodies and materials are tallied.

Table 1
Summarized simulation parameters of the presented examples.

	Histories	Source		Cutoffs (keV)		
	Number	Particle	E (keV)	γ	e^-	e^+
1-disc-noVR	$1.4 \cdot 10^8$	e^-	40	1	1	1
1-disc-VR	$1.1 \cdot 10^7$	e^-	40	1	1	1
3-detector	$1.2 \cdot 10^8$	γ	$(1.17, 1.33) \cdot 10^3$	5	50	50
4-x-ray-tube	$4.1 \cdot 10^6$	e^-	150	10	10	10

Some of the simulation parameters used for each example are summarized in the Table 1.

The process of adapting the examples, initially defined with the quadric geometry package, to be simulated using the surface mesh geometry package only affects the geometry definition. PenRed, being designed as a highly modular framework with automatic compatibility assurance among modules, allows seamless utilization of all tallies, variance reduction techniques, and other components for mesh simulations without any additional effort. Consequently, the only necessary modifications in the simulation configuration file are those related to the geometry configuration.

In addition, the geometry needs to be defined as a compilation of mesh surfaces following the penRed format. This process is streamlined through the use of the previously described Blender plugin. For instance, Fig. 5 illustrates the construction of the example *4-x-ray-tube* in the Blender environment. Furthermore, the plugin can be used to convert standard mesh file formats to the penRed one, providing the flexibility for the user to create the geometry with its chosen CAD software.

For more detailed information about each simulation, please refer to the penRed documentation. Furthermore, all the necessary files to run these simulations can be found in the penRed repository. Additionally, we adapted the ICRP145 phantom [16] with Blender to be simulated with penRed and compare the results with those provided using MCNP6. Specifically, we selected the simulation corresponding to the female phantom with the entire liver serving as a gamma source. The adult Mesh-type Reference Computational Phantoms (MRCPs) provided by the ICRP, are considered as the most advanced type of computational phantoms which can be implemented directly into MC codes. The

female MRCP height and weight is in accordance with the reference values, 163 cm and 60 kg. This one is composed of 2.6 million triangular facets including all the radio sensitive organs and tissues relevant to dose calculations due to radiation exposure.

In terms of benchmarks, we repeated the phantom simulation while applying different optimization parameters to measure the performance improvements. All simulations, tests and benchmarks, were conducted using a single node equipped with an Intel(R) Core(TM) i9-10900K CPU running at 3.70 GHz and with 64 GB of RAM. The used optimizations are focused on minimize the amount of ray-triangle intersections checks. These ones are listed below:

- **Body tree**: As discussed before, the bodies conforming the geometry follow a kinship hierarchy limiting the number of bodies that can be crossed by the particle.
- **Octrees**: Instead of testing all surface triangles belonging to a specific body individually, they are grouped into spatial regions defined by bounding boxes. Consequently, before initiating any ray-tracing check with a triangle in a given region, the algorithm ensures that the particle track intersects the corresponding boundary box. If not, it deduces that the track cannot cross any triangles within the tested region, leading to a significant reduction in the required computational time.
  Furthermore, the regions containing the surface triangles can be further grouped into higher-level regions, creating a hierarchy of regions. In penRed, we have established three levels of regions. The topmost level consists of a single bounding box encompassing the entire object surface. The second level regions comprises a set of sub-regions, which, in turn, contain the triangles to be tested.
  The subdivision of all geometry bodies into these three-level regions, or octrees, is automatically handled by penRed. However, users have the flexibility to adjust the sizes of different region levels, indicating the desired number of sub-elements contained in each one, to speed-up the simulation.
- **Triangle collision spheres**: Add a collision sphere to each triangle surrounding it. Before to apply the triangle-ray intersection algorithm, the distance between the track and the triangle center is calculated. If it is greater than the sphere radius the triangle is ignored.

As result, the whole geometry is subdivided in many levels, depending on the body tree structure, and three extra levels using the octrees in each body.

4. Results

4.1. Validation

The initial validation tests, which compared equivalent simulations using quadric and mesh-based geometries, demonstrate a high level of agreement between both geometry types. In Fig. 6, we can observe the results for the 1-disc example. The top graph displays the energy spectrum of the emerging electrons in the Z positive hemisphere, corresponding to the $ANGULAR\_DET$ tally, for the case with no variance reduction, whereas the bottom graph represents the energetic spectrum fluence, integrated across the entire disc volume, for the incoming particles, for the variance reduction case.

Concerning the *detector* and *x-ray* examples, Fig. 7 presents partial results. The top image illustrates the energy deposition spectrum recorded by the NaI detector, while the second image depicts the energy spectrum of the photons impacting the detector in the x-ray example.

The presented results clearly demonstrate a robust agreement between both geometry types. Furthermore, this behavior holds true for the results of all examples, including the omitted in this document for brevity.

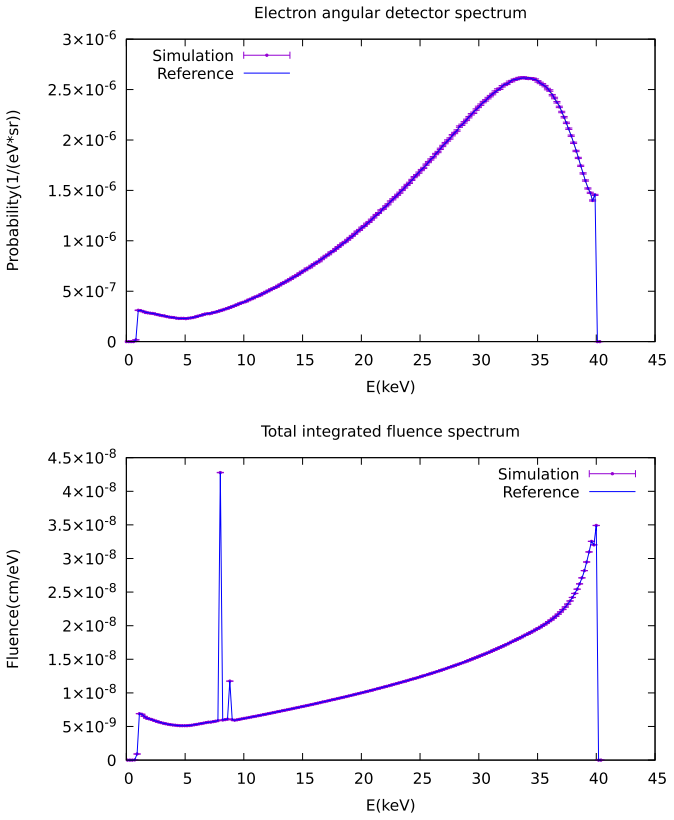


Fig. 6. Results corresponding to the example *1-disc*. Energy spectrum of electrons escaping the geometry system upwards in Z direction with no variance reduction (top) and energy spectrum of fluence registered integrated across the entire disc volume, for the variance reduction case (bottom). All results have been calculated with both, quadric (Reference) and triangular mesh (Simulation) geometry types and are normalized to energy bin width.

To thoroughly validate this agreement, we conducted a detailed examination of the deviations between simulations, affirming the absence of significant systematic differences between both types of geometries. To reach this conclusion, we compared all results, bin by bin, between both simulations types by calculating the absolute bin difference (d_i) as defined in Eq. (1),

$$d_i = value_i^{quad} - value_i^{mesh} \tag{1}$$

where i represents the bin number, $value_i^{quad}$ denotes the value obtained for the i-th bin from the simulation with the quadric geometry, and $value_i^{mesh}$ is the corresponding value from the mesh geometry simulation. Additionally, we calculated the standard deviation (σ_i) associated with d_i using the Eq. (2),

$$\sigma_i = \sqrt{(\sigma_i^{quad})^2 + (\sigma_i^{mesh})^2} \tag{2}$$

where σ_i^{quad} and σ_i^{mesh} are the standard deviations of bin i obtained from the quadric and mesh simulations, respectively. Finally, the deviation d_i is converted to standard deviation units (q_i) as,

$$q_i = \frac{d_i}{\sigma_i} \tag{3}$$

An illustrative example is provided in Fig. 8, where the value q_i has been represented for the energy spectrum of x-ray impacting photon (Fig. 7, bottom). As depicted, the discrepancies between both simulations consistently remain below the 3σ and are evenly distributed around zero. This indicates that the possible systematic uncertainties introduced by differences between geometry and tracking models are

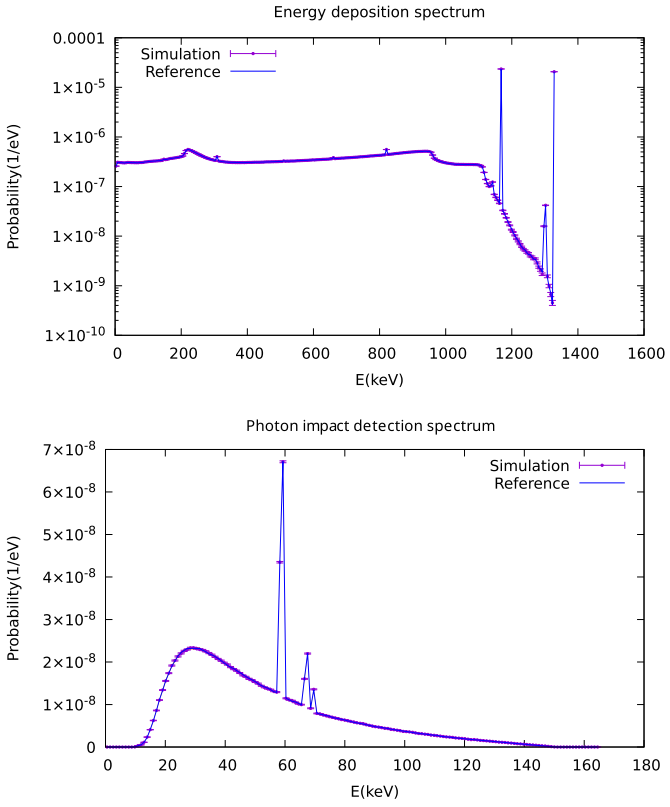


Fig. 7. Results for the NaI detector and the x-ray tube examples. The top image corresponds to the energy deposition spectrum registered by the NaI detector, while the bottom one shows the energy spectrum of the photons impacting the silicon detector of the x-ray tube. All results have been calculated with both, quadric (Reference) and triangular mesh (Simulation) geometry types and are normalized to energy bin width.

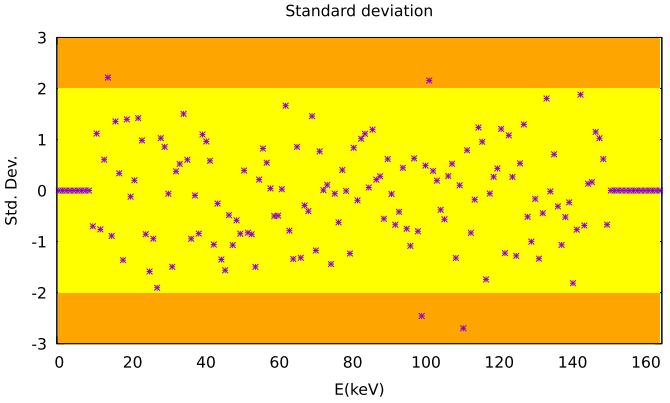


Fig. 8. Differences between quadric and mesh simulations for the energy spectrum of impacted particles, measurement at the silicon detector, in the x-ray tube example. Differences are shown bin by bin in standard deviation units.

not significant when compared to the statistical uncertainties inherent in the Monte Carlo simulation.

To conclude the validation process, we performed the phantom simulation employing identical simulation parameters to those disclosed by the ICRP, facilitating a direct comparison. The details of these parameters are outlined below. This specific example involves internal body irradiation, where the entire liver volume is designated as a monoenergetic gamma isotropic source with an emission energy of 1 MeV. The tracking cutoff for each particle type was established at 1 keV, meaning that any particle with energy below this value is considered to be locally absorbed. The simulation was executed with a total of $1 \cdot 10^7$ histories. The MCNP6 results provided by the ICRP, measure the absorbed dose in each material. Therefore, for the purpose of comparison, the equivalent magnitude was tallied through a penRed simulation.

The total penRed simulation time, including the initialization and geometry pre-processing step, in this conditions was less than 8 hours with a total memory usage of 1.75 GB using 20 parallel threads. Notice that this memory usage does not correspond only to the memory required to store the geometry, but includes also the materials databases, tallies, source information and all the other data used by the program.

To avoid values with substantial fluctuations resulting from low statistical counts, we have focused only on bodies with a energy deposition greater than the 0.5% of the energy deposited in the source material ($1.51 \cdot 10^5$ eV per history). Therefore, only materials with an energy deposition exceeding 754.68 eV per history are considered. The summarized results are presented in Table 2.

It is important to note that discrepancies may arise due many factors. First, both codes use different physics implementations and cross section databases, particularly concerning electron transport [20], especially for low-energy electrons [21] as well as the selection of multiscatter parameters [22]. In addition, it should be acknowledged that the results provided by the ICRP145 phantom package were based on a unstructured volumetric mesh with tetrahedrals as volume elements. Therefore, mesh preprocessing for penRed and MCNP6 simulations differed. This discrepancy could introduce also type B uncertainties into the simulation, specially on complex and thin objects. Nevertheless, as shown in the table, the highest disparity observed in the measured energy deposition is only 3.2%, which is a more than reasonable value taking into account these considerations.

4.2. Optimization

To test the impact of the previously discussed optimization techniques, several tests have been performed using the same hardware as the validation simulations. For the shake of brevity, only the tests performed for the MRCP female phantom are discussed following. Since we are not focusing on results accuracy, the particles cuttoffs have been relaxed to increase the number of simulated histories per second and achieve a stable speed value with fewer simulation time. These cuttoffs have been fixed to a range of 0.5 cm for all particle types. A comparison of simulation speed, in histories per second, for different number of octree split size is shown in Table 3. On one hand, for each cell in the table, the first column indicates the number of objective three level regions in each second level region in the octree hierarchy. On the other hand, the first row assigns the objective number of triangles in each three level region in the octree. Although the program tries to restrict the number of inner elements to the specified sizes, it prioritizes to join regions with a high overlaps allowing to extend the number of inner elements at most twice the specified size. Therefore, the number of elements in each region can differ significantly. Moreover, as the phantom has many bodies, the region size values have been set globally for all geometry bodies. As these parameters can be set individually for each body, it is possible to tune up the configuration to achieve a better performance.

As we can see in Table 3, the simulation speed strongly depends on the chosen values of number of elements in each region level. Therefore, it is worth to perform short simulations with different parameter values to maximize the performance, specially on long simulations. Finally, to evaluate the impact of employing collision spheres for individual triangles, we conducted the same simulation using the optimal octree parameters, specifically, 20 inner regions within each outer region and 5 triangles within each inner region, but with the triangle collision sphere check disabled. The resulting simulation speed was 30486 histories per second. Consequently, utilizing this technique is advantageous, as it results in a 43% increase in simulation speed in this case.

Table 2
Body energy deposition comparison between *MCNP6* and *penRed*.

Body	MCNP6 (eV/g)	penRed (eV/g)	Difference (%)
4300 Ribs cortical	2387.4 ± 9.0	2387.4 ± 9.0	0.00
4400 Ribs spongiosa	4044.2 ± 13.4	4052.4 ± 13.5	0.20
4900 Thoracic spine cortical	2848.3 ± 10.8	2887.7 ± 11.0	1.40
5000 Thoracic spine spongiosa	4440.7 ± 14.7	4411.7 ± 14.5	−0.65
5100 Lumbar spine cortical	1981.7 ± 9.0	2044.9 ± 9.0	3.20
5200 Lumbar spine spongiosa	3454.5 ± 12.8	3388.1 ± 12.5	−1.90
5600 Sternum spongiosa	774.6 ± 6.0	770.8 ± 6.0	−0.49
5700 Cartilage costal	1261.8 ± 7.6	1257.4 ± 7.5	−0.36
5800 Cartilage discs	1096.0 ± 7.0	1073.9 ± 7.0	−2.00
6200 Breast left adipose tissue	1000.4 ± 7.0	994.4 ± 7.0	−0.60
6400 Breast right adipose tissue	1787.6 ± 9.3	1786.5 ± 9.5	−0.06
6500 Breast right glandular tissue	1353.4 ± 8.3	1355.4 ± 8.5	0.15
7100 Gall bladder contents	2916.4 ± 12.2	2909.0 ± 12.5	−0.25
7200-7203 Stomach wall surface	3774.24 ± 13.5	3768.8 ± 13.5	−0.14
7300 Stomach contents	5040.9 ± 16.1	5032.9 ± 16.0	−0.16
7403 Small intestine wall surface	3980.0 ± 13.5	4002.3 ± 13.5	0.50
7602 Ascending colon wall surface	796.6 ± 6.0	785.9 ± 6.0	−1.40
8700 Heart wall	4617.6 ± 14.8	4601.9 ± 15.0	−0.34
8800 Blood in heart chamber	6291.5 ± 18.2	6257.2 ± 18.0	−0.54
8900 Kidney left cortex	1652.4 ± 8.8	1655.4 ± 9.0	0.18
9200 Kidney right cortex	3883.1 ± 14.0	3879.8 ± 14.0	−0.08
9300 Kidney right medulla	986.5 ± 6.9	979.8 ± 7.0	−0.68
9500 Liver	150940 ± 90	150984.0 ± 85	0.03
9700 Lung(AI) left	2696.0 ± 11.0	2695.9 ± 11.0	−0.014
9900 Lung(AI) right	9660 ± 22	9724.4 ± 22.0	0.66
11300 Pancreas	4535.7 ± 15.4	4530.3 ± 15.5	−0.12
12700 Spleen	1874.4 ± 9.6	1867.9 ± 9.5	−0.34

Table 3
Simulation speed in histories per second for different values of octree parameters. The top row indicate the objective number of triangles in each three level region. The left column specifies the objective number of three level regions in each second level one.

2nd\3rd	1000	200	100	50	20	10	5	2
10000	1177.0	8058.8	15375	15669	12036	10083	9254.8	7739.5
100	1165.2	8100.4	18535	25809	29876	29025	27969	26963
50	1174.9	8230.9	19401	28686	35401	35975	35214	34449
20	1188.4	8076.7	19739	31218	41002	43208	43580	42298
10	1168.9	8081.2	20054	31167	40459	41416	41408	39718

5. Conclusions and future work

In this work, we have expanded the abilities of penRed to simulate on triangular mesh-based shapes. This was a missing feature not only in penRed but also in other PENELOPE physics-based software. This enhancement now enables penRed users to simulate and create organic shapes and complex designs using specialized tools like Blender. Thanks to the Blender ability to import different mesh formats, users have the flexibility to choose their preferred design environment. Additionally, the complete description of the internal penRed mesh format is provided in the documentation, making it straightforward to create a custom format converter if needed.

Our results demonstrate excellent agreement between simulations based on quadric and mesh-based geometries. Furthermore, we conducted simulations using the ICRP145 phantom, showing good agreement between the results provided in the phantom package and the penRed simulations, confirming the accuracy of our code. Moreover, this simulation case only requires 1.75 GB of memory to store not only the phantom data, but also the material, tallies and other required information.

We have also provided an algorithm description that allows users to implement various types of geometries, empowering them to customize or create their own geometric modules.

Additionally, our benchmark tests indicate that better performance is achieved with relatively small octree sizes. However, the optimal size might vary depending on the specific geometry being simulated. Therefore, we recommend conducting an initial benchmark to fine-tune settings for long-duration simulations.

Looking ahead, our future plans involve test other ray-triangle intersection algorithms and implementing the transport algorithm on GPUs to boost simulation speed for these types of geometries.

CRediT authorship contribution statement

S. Oliver: Conceptualization, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **S. Rodriguez Bosca:** Investigation, Validation, Writing – original draft, Writing – review & editing. **V. Giménez-Alventosa:** Conceptualization, Investigation, Methodology, Resources, Software, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

All the data we utilized is accessible within the supplied "penRed" package, with the exception of the ICRP phantom. It was developed by an external group, and we do not have permission to distribute

Acknowledgements

VG would like to acknowledge the "Generalitat Valenciana" (GVA) grant PROMETEO/2021/064. The authors extend their profound appreciation to Professor Vicent Giménez Gómez for his invaluable contributions to the planning and development of the penRed project. It is through his dedicated efforts that we are able to progress with the project.

Appendix A. *Step* code

Algorithm 1 Step algorithm.

Input: State, DS, bodies, world
Output: State, DSEF, DSTOT, NCROSS

```
DSEF ← 0
DSTOT ← 0
NCROSS ← 0
if State.IBODY ≥ bodies.size then ▷ Means the particle is outside the geometry
    if State can cross world then
        ds2In ← State.distance(world)
        State.IBODY ← world.index
        State.MAT ← world.MAT
        DSTOT ← ds2In
        NCROSS ← 1
        State.move(ds2In)
        if world.MAT is not 0 then ▷ The world is not a void region
            DSEF ← ds2In
            end function
        end if
    else ▷ The particles does not reach the geometry
        State.IBODY ← bodies.size
        State.MAT ← 0
        DSEF ← ∞
        DSTOT ← ∞
        State.move(∞)
        end function
    end if
end if
                ▷ At this point, the particle is inside the geometry system
isInVoid ← false ▷ Flag if it is in a void region
if State.MAT is 0 then
    isInVoid ← true
end if
nextMat ← State.MAT
nextBody ← State.IBODY
while true do
    currentBody ← nextBody
    body ← bodies[currentBody]
    travel ← DS ▷ Set the maximum distance to travel in this iteration
    if nextMat is 0 then
        travel ← ∞
    end if
    travelType ← 0 ▷ Flag travel inside the actual body
    if State can cross body before travel then
        travelType ← 1 ▷ Flag travel crossing body boundary
        travel ← State.distance(body) ▷ Update maximum travel distance
        nextBody ← body.parent
    end if
    for each daughter ∈ body.daughters do
        if State can cross daughter before travel then
            travelType ← 2 ▷ Flag crossing daughter boundary
            travel ← State.distance(daughter)
            nextBody ← daughter.index
        end if
    end for
    if travelType is 0 then ▷ Particle remains in actual body
        DSEF ← DSEF + travel
        State.move(travel)
        break ▷ Stop loop
    else if travelType is 1 then ▷ Crosses body boundary
        if currentBody is world.index then ▷ Escapes from geometry
            State.IBODY ← bodies.size
            if State.MAT is 0 then
                DSEF ← ∞
            else if nextMAT is not 0 then
                DSEF ← DSEF + travel
            end if
            State.MAT ← 0
            DSTOT ← ∞
            NCROSS ← NCROSS + 1
            State.move(∞)
            end function
        end if
        solveOverlapsUp(travel,State,currentBody,nextBody)
    else
        solveOverlapsDown(travel,State,nextBody,nextBody)
    end if
    if nextMat is 0 then
        if State.MAT is 0 then
            DSEF ← DSEF + travel
        else
            DSTOT ← DSTOT + travel
        end if
    else
        DSEF ← DSEF + travel
        DS ← DS − travel
    end if
    State.move(travel)
    nextMat ← bodies[nextBody].MAT
    if nextMat is 0 then
        if not inVoid then ▷ Count void region cross only once
            NCROSS ← NCROSS + 1
            inVoid ← true
        end if
    else if inVoid then ▷ Void to material cross. Stop it.
        NCROSS ← NCROSS + 1
        break
    else if nextMat is State.MAT then ▷ No material change
        if bodies[nextBody].KDET is not body.KDET then
            NCROSS ← NCROSS + 1
            break ▷ Stop on detector change
        end if
    else ▷ Material changed, stop moving
        NCROSS ← NCROSS + 1
        break
    end if
end while
DSTOT ← DSTOT + DSEF
State.MAT ← nextMat
State.IBODY ← nextBody
```

References

[1] EGSnrc: software tool to model radiation transport, https://github.com/nrc-cnrc/EGSnrc. (Accessed 2 July 2017).

[2] Los Alamos Scientific Laboratory, Group X-6, MCNP: a General Monte Carlo Code for Neutron and Photon Transport, Dept. of Energy, Los Alamos Scientific Laboratory Los Alamos, N.M., 1979.

[3] F. Salvat, PENELOPE: A Code System for Monte Carlo Simulation of Electron and Photon Transport, OECD Nuclear Energy Agency, Issy-Les-Moulineaux, 2014.

[4] GEANT4 Collaboration, S. Agostinelli, et al., Geant4: a simulation toolkit, Nucl. Instrum. Methods A506 (2003) 250–303, SLAC-PUB-9350, FERMILAB-PUB-03-339.

[5] H. Iwase, K. Niita, T. Nakamura, Development of general-purpose particle and heavy ion transport Monte Carlo code, J. Nucl. Sci. Technol. 39 (11) (2002) 1142–1151, https://doi.org/10.1080/18811248.2002.9715305.

[6] A. Ferrari, P.R. Sala, A. Fassò, J. Ranft, FLUKA: a multi-particle transport code (program version 2005), CERN Yellow Reports: Monographs, CERN, Geneva, 2005.

[7] V. Giménez-Alventosa, V. Giménez Gómez, S. Oliver, PenRed: an extensible and parallel Monte-Carlo framework for radiation transport based on PENELOPE, Comput. Phys. Commun. 267 (2021) 108065, https://doi.org/10.1016/j.cpc.2021.108065.

[8] S. Oliver, V. Giménez-Alventosa, F. Berumen, V. Gimenez, L. Beaulieu, F. Ballester, J. Vijande, Benchmark of the penred Monte Carlo framework for hdr brachytherapy, Z. Med. Phys. (2022), https://doi.org/10.1016/j.zemedi.2022.11.002, https://www.sciencedirect.com/science/article/pii/S0939388922001040.

[9] P.P. Wilson, T.J. Tautges, J.A. Kraftcheck, B.M. Smith, D.L. Henderson, Acceleration techniques for the direct use of cad-based geometry in fusion neutronics analysis, in: Proceedings of the Ninth International Symposium on Fusion Nuclear Technology, Fusion Eng. Des. 85 (10) (2010) 1759–1765, https://doi.org/10.1016/j.fusengdes.2010.05.030, https://www.sciencedirect.com/science/article/pii/S0920379610002425.

[10] G4Tet.hh, Online https://gitlab.cern.ch/geant4/geant4/-/blob/master/source/geometry/solids/specific/include/G4Tet.hh. (Accessed 28 July 2023).

[11] M.C. Han, C.H. Kim, J.H. Jeong, Y.S. Yeom, S. Kim, P.P.H. Wilson, J. Apostolakis, Dagsolid: a new geant4 solid class for fast simulation in polygon-mesh geometry, Phys. Med. Biol. 58 (13) (2013) 4595, https://doi.org/10.1088/0031-9155/58/13/4595, https://dx.doi.org/10.1088/0031-9155/58/13/4595.

[12] F.B. Brown, R. Barrett, T. Booth, J. Bull, L. Cox, R. Forster, T. Goorley, R. Mosteller, S. Post, R. Prael, et al., Mcnp version 5, Trans. Am. Nucl. Soc. 87 (273) (2002) 02.

[13] R.L. Martz, Mcnp6 unstructured mesh initial validation and performance results, Nucl. Technol. 180 (3) (2012) 316–335, https://doi.org/10.13182/NT12-A15347.

[14] A. Badal, I. Kyprianou, D.P. Banh, A. Badano, J. Sempau, penmesh—Monte Carlo radiation transport simulation in a triangle mesh geometry, IEEE Trans. Med. Imaging 28 (12) (2009) 1894–1901, https://doi.org/10.1109/TMI.2009.2021615.

[15] H. Luo, J.D. Baum, R. Löhner, A fast, matrix-free implicit method for compressible flows on unstructured grids, J. Comput. Phys. 146 (2) (1998) 664–690, https://doi.org/10.1006/jcph.1998.6076, https://www.sciencedirect.com/science/article/pii/S0021999198960764.

[16] C.H. Kim, Y.S. Yeom, N. Petoussi-Henss, M. Zankl, W.E. Bolch, C. Lee, C. Choi, T.T. Nguyen, K. Eckerman, H.S. Kim, M.C. Han, R. Qiu, B.S. Chung, H. Han, B. Shin, Adult mesh-type reference computational phantoms. ICRP publication 145, Ann. ICRP 49 (3) (2020).

[17] T. Möller, B. Trumbore, Fast, minimum storage ray/triangle intersection, in: ACM SIGGRAPH 2005 Courses, SIGGRAPH '05, Association for Computing Machinery, New York, NY, USA, 2005, p. 7–es.

[18] J. Havel, A. Herout, Yet faster ray-triangle intersection (using sse4), IEEE Trans. Vis. Comput. Graph. 16 (3) (2010) 434–438, https://doi.org/10.1109/TVCG.2009.73.

[19] A. Kensler, P. Shirley, Optimizing ray-triangle intersection via automated search, in: 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 33–38.

[20] H.R. Sadoughi, S. Nasseri, M. Momennezhad, H.R. Sadeghi, M.H. Bahreyni-Toosi, A comparison between gate and mcnpx Monte Carlo codes in simulation of medical linear accelerator, J. Med. Signals Sens. 4 (1) (2014) 10–17.

[21] C. Valdes-Cortez, I. Mansour, M.J. Rivard, F. Ballester, E. Mainegra-Hing, R.M. Thomson, J. Vijande, A study of type b uncertainties associated with the photoelectric effect in low-energy Monte Carlo simulations, Phys. Med. Biol. 66 (10) (2021) 105014, https://doi.org/10.1088/1361-6560/abebfd, https://dx.doi.org/10.1088/1361-6560/abebfd.

[22] J.P. Archambault, E. Mainegra-Hing, Comparison between egsnrc, geant4, mcnp5 and penelope for mono-energetic electron beams, Phys. Med. Biol. 60 (13) (2015) 4951, https://doi.org/10.1088/0031-9155/60/13/4951, https://dx.doi.org/10.1088/0031-9155/60/13/4951.