EPJ Web of Conferences **302**, 11005 (2024) https://doi.org/10.1051/epjconf/202430211005
SNA + MC 2024

FLAIR3 – recasting simulation experiences with the Advanced Interface for FLUKA and other Monte Carlo codes

André Donadon[1], *Gabrielle* Hugo[1], *Christian* Theis[1], and *Vasilis* Vlachoudis[1*]

[1]Geneva 23 CERN CH-1211, Switzerland

Abstract. Particle tracking Monte Carlo codes are complicated programs to use, making them error prone for the end user. To address this problem, and simplify the use of (initially) FLUKA, Flair was born in late 2006. Over the course of time, Flair evolved so much that it established itself as the de-facto standard for working with FLUKA, even to the extent that users often confuse the two programs. Recently a major development effort was routed on separating the interface from the functionality and the tight links with FLUKA. This opened the door to support other Monte Carlo codes. Notably this effort was in line with the new generation of the FLUKA code v5, however also PHITS, MCNP and partially PENELOPE can benefit from this architectural change. The present paper describes the latest developments in the code and some of the most noteworthy Flair features.

1 Introduction

FLUKA's [1,2,**Error! Reference source not found.**] strict input format, as well as the auxiliary command line tools to manipulate and plot simulation results pose multiple problems to newcomers. These issues, witnessed especially in FLUKA beginner trainings, revealed early on the necessity of a simplified user environment. Thus, Flair[4,5] was born in late 2006 as a user-friendly interface to the FLUKA code, providing all necessary functionalities for the end user to build an input file from scratch via a graphical interface, to interactively edit and debug the simulation geometry, to perform the run, to monitor the run's progress, to post-process the simulation results, and to prepare publication-quality figures, relying solely on python [6,7] and the tkinter [8] graphical library as dependencies. Over the years, many capabilities have been added to simplify auxiliary tasks, such as intervention planning for radioprotection studies and radiation-treatment-planning simulation tools starting from DICOM [9] files, to name but a few. Eventually, Flair became a complete workbench, offering users an intuitive and user-friendly environment in which to perform FLUKA simulations while remaining practically agnostic to their underlying technical constraints and complexity.

* Corresponding author: Vasilis.Vlachouids@cern.ch

Flair quickly established itself as the de-facto standard way of working with FLUKA. The most notable addition in Flair was the inclusion of the geometry editor, built on a custom C++ library that allows users to visualize and graphically build error-free geometries. Flair v2 had a major facelift, offering a significant change in the user interface and make it more user friendly and intuitive (Figure 1). In 2019, in view of the phasing out of python v2, the Flair code base was completely restructured to be compatible with python v3, but most importantly, to decouple the graphical interface from the FLUKA dependence. This architectural change severed ties to a legacy python version that reached its official end of life 2020. Thus, the release of FLAIR v3 renders its predecessor version deprecated and obsolete. At the same time, this upgrade featured the integration of other Monte Carlo codes, constituting a major step forward. Currently, Flair support for FLUKA v5 is in a very advanced state, similarly for PHITS [10], while support for MCNP [11] and PENELOPE [12] is in progress. In addition, in Flair v3 the geometry editor module was further enhanced with a photorealistic 3D ray tracing engine called FARM (Flair Advanced Render Module), which opened a new chapter in the way to represent geometries and simulation results

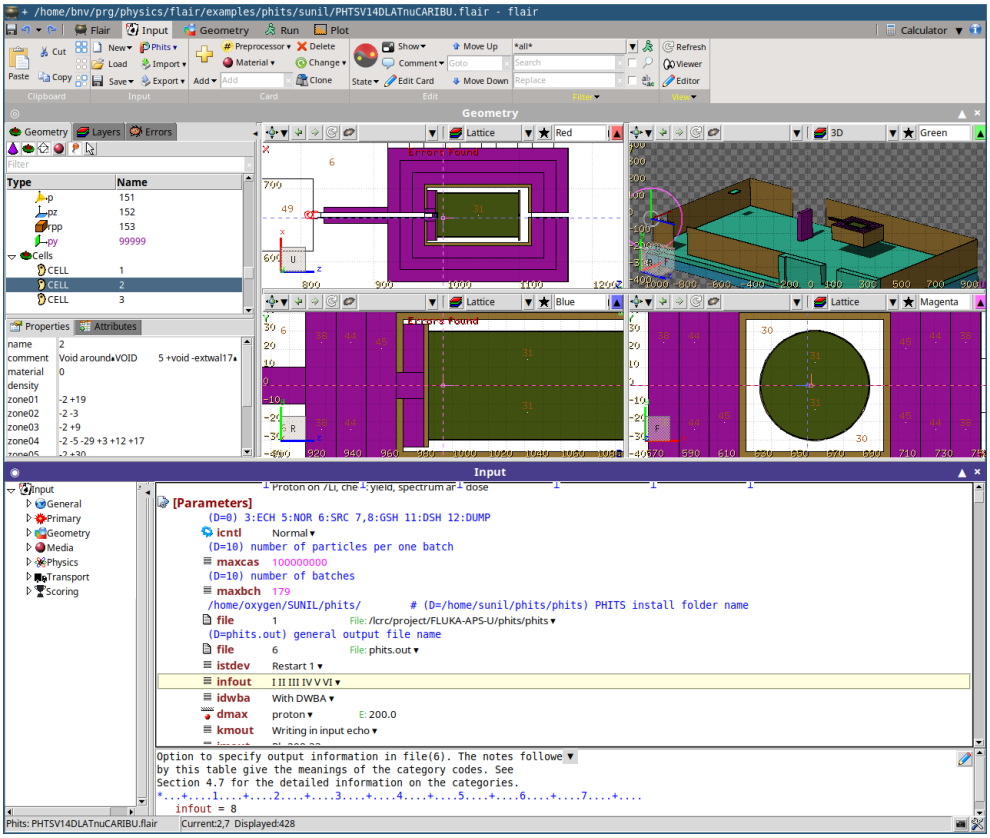

Fig. 1. Screen shot of flair interface showing the conversion of a FLUKA input to PHITS the rendering of the input and the geometry

2 FLAIR v3

Starting from Flair v3, an abstract Monte Carlo class was introduced to provide all the necessary methods and data storage in a generic way for the most commonly used Monte

Carlo codes in the radiation transport community. For every Monte Carlo code, Flair has a dedicated input class inheriting from the abstract input class, overriding some basic methods like I/O, preprocessing, geometry, and materials handling. The basic component in the input file is the Card class, a data structure which holds the following information: the card name, the list of card arguments (WHATs in the FLUKA jargon), the user comments, a free multiline text, and a dictionary for an arbitrary number of user variables with multiple values. In the past, a one-dimensional list of card instances was used to describe the FLUKA input file, while now it is converted into a tree structure of cards where each card an contain an arbitrary number of sub-cards. Moving from a one-dimensional list of cards to a multi-dimensional tree structure, enormously increased the complexity of the code, especially on how to address the cards with respect to editing operations and, most importantly, how to preserve the history of commands for Flair's undo/redo mechanism.

Each Monte Carlo operation mode in Flair is described in an external ASCII database file containing all information, units, particles, basic materials, and card description. For each card it contains the description, variable syntax, parameters, assertions for error checking, and rendering information. These databases offer an enormous flexibility to enhance the input description without the need of altering the Flair code. Additionally, they can be readily extended by non-programmers with new options as the different Monte Carlo codes continue to evolve.

On top of the Monte Carlo specific input cards, Flair has introduced a few extra cards and features. The Flair cards act as pre-processor cards to change the input cards that will be active when exported for the MC code. In combination with the expression mechanism in Flair, one can generate parametric inputs and submit multiple runs altering various options, cards, geometry, scoring, etc... with a simple mechanism on the Run page.

A translation module was introduced, permitting the conversion of inputs from one Monte Carlo code to another. There are clearly limitations mostly due to the differences in the Monte Carlo codes, however the conversion works well for the geometry and materials, which typically represent the biggest fraction of the work load.

Flair contains an internal database of materials, originating from various free sources. Despite the database helping the user enormously when building new inputs, caution should be exercised and the correctness of the information must be verified by the user, as is good practice for any simulation parameter.

2.1 Geometry kernel

The Flair geometry kernel is one of the most important components of Flair, offering tremendous capabilities in visualizing, graphical editing, debugging, and overlaying simulation results. It was initially built to support the FLUKA geometry description, which is however quite similar in logic with that of MCNP and PHITS, where bodies (equivalent to MCNP surfaces) are used to segment the 3D space, while portions of space are described as Boolean operations of these bodies/surfaces (named regions in FLUKA), equivalent to cells in MCNP. Since the concept of geometry description between FLUKA, MCNP and PHITS is very similar, accommodating the display of MCNP and PHITS geometries was almost straightforward. The main difference lied in the repetition capabilities in FLUKA with respect to those of MCNP and PHITS. FLUKA uses the concept of LATTICEs where user-defined regions/cells provide a transformation to another space location, which is quite

similar to the FILL mechanism, except that it operates in the same universe. Currently Flair v3 has been extended to support the fill/universe mechanism of MCNP/PHITS.

Geant4 GDML geometry support is achieved by internally converting the G4 solids into region/cell expressions with the introduction of facets described as bodies/surfaces. Currently it supports all G4 solids up to those relying on second degree equations while, for the moment, higher order solids like the twisted ones are implemented in an approximate way as a union of multiple 2nd degree surfaces. The special volumes, *replicavol*, *assembly,* and *paramvol* are supported, while support for *divisionvol* is still under development. The G4 geometry is hierarchical: each nesting level is described in Flair in a separate universe (name-based instead of integer-based), using the same fill/universe mechanism implemented for the MCNP/PHITS support.

In 2010, the geometry editor [13] module was added to Flair. It's a custom-built python extension written in C++, without external dependencies. All functionality was written from scratch, from the math libraries to the plotting functionality, fonts, 3D graphics, etc. The code is multithreaded and highly optimized, and is able to display even complicated 3D ray-traced images of the geometry practically in real time.

The geometry kernel is able to work either in 2D or in 3D mode. In 2D, every body/surface is converted to a logical operation of the bounding quadratic surfaces which are converted to conic sections on the 2D viewing plane. All conics are brought to intersect each other into multiple segments and every segment is analyzed and plotted accordingly. This is a very powerful approach, similar to what is used in MCNP. The geometry kernel can detect all geometry errors on the viewing plane, regardless of the extent of the geometry. Special attention was paid to performing numerically robust floating point calculations, featuring an internal mechanism for error correction.

The geometry kernel in Flair v3 was modularized using multiple classes for holding the data, general display and geometry operations, with dedicated thread-safe engines. The recent enhancements allow Flair v3 to be able to process PHITS, MCNP, PENELOPE, and Geant4 geometries, except for tetrahedral meshes and higher than 2nd-degree G4 surfaces. Thanks to its numerical robustness and speed, the Flair geometry kernel was used as external navigator in FLUKA v5, providing the possibility to FLUKA to run in seamless way combined geometries of Geant4, FLUKA, and/or PHITS/MCNP.

2.2 FARM

Effective visualization plays a pivotal role in comprehending and communicating simulation results, which often serve as the foundation for critical decision-making processes. Recent years have witnessed a substantial increase in expectations, driven by the ubiquity of high-quality imagery in everyday life and the widespread accessibility of powerful hardware and software solutions. Today's engineering 3D modelling packages seamlessly integrate with production-grade rendering tools, elevating visual quality to new heights that largely exceed what has been available in the domain of radiation transport codes so far.
In response to these expectations and to match today's standards, the FLAIR Advanced Rendering Module (FARM) has been developed. It is a physically based unidirectional path tracer that implements an array of features found in photorealistic production renderers. Among many others this encompasses ambient occlusion, image-based lighting, displacement mapping, global illumination, Quasi-Monte Carlo sampling and convergence based denoising. Seamlessly integrated with FLAIR's geometry kernel and following the

philosophy to be independent from 3rd party packages to maximize compatibility, it has been developed from scratch without utilizing auxiliary libraries. It offers fast, interactive 3D pre-visualization using a non-physical Phong shading model. For those seeking top-tier visual output, the "*high quality mode*" produces visually appealing photorealistic images for presentations by replicating the behavior of light in a physically accurate manner (see Figure 2). The latter mode is activated via a simple switch and requires minimal user interaction. Parametrization by the user, who naturally may not be a professional 3D artist, is streamlined to the choice of material type, reflectivity, roughness and sampling rate.

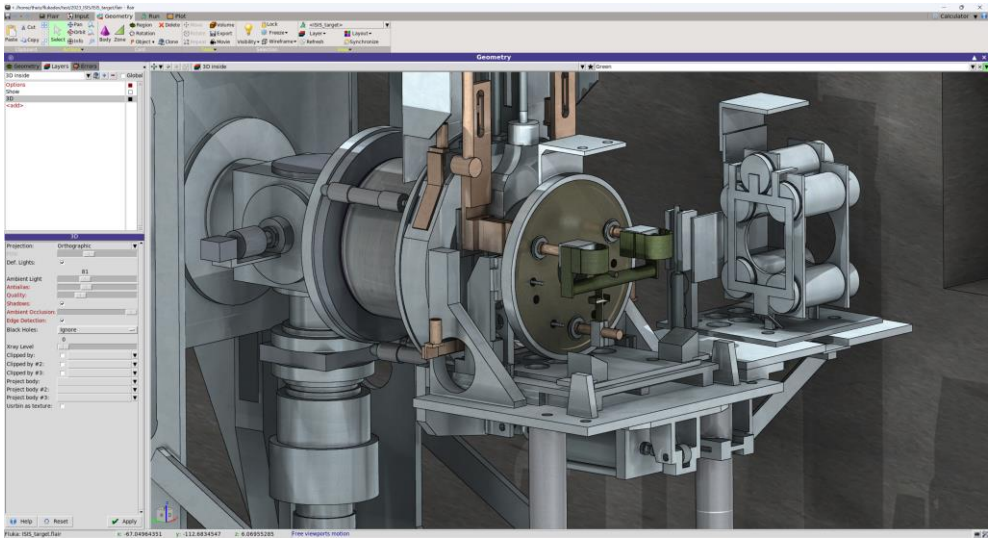

Fig. 2. Rendering of a FLUKA geometry using the high-quality mode of FARM.

At the same time FARM also includes an optimized renderer which targets medical applications, designed to facilitate the efficient volumetric visualization of voxel data at a quality level of commercial treatment planning systems. These two renderers seamlessly interact, offering a comprehensive solution for the three-dimensional display of voxel phantoms within their surrounding environment. In addition to portraying geometries FARM also offers the possibility to visualize data in 3D.

While photorealistic images excel in delivering stunning visuals, there is a recognition that they may sometimes divert the attention from the results that are intended for decision makers. To address this, the renderer incorporates special shader models popular in architecture. Geometries can also be rendered in subdued shades of white & gray with contours, emulating hand-drawn designs, while resulting data are shown in vibrant full color (see Figure 3).

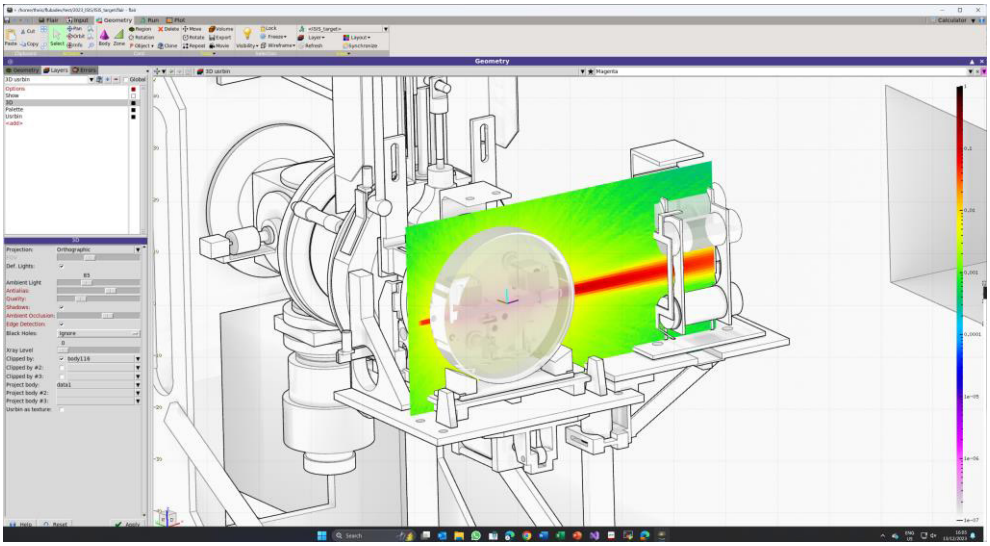

Fig. 3. Architectural rendering of a FLUKA geometry using the high-quality mode of FARM.

3 Medical applications

Unlike traditional treatment planning systems (TPS), general-purpose Monte Carlo codes such as FLUKA, Geant4, PHITS, and others offer significant advantages by accurately accounting for a broad spectrum of physical effects (such as complete secondary particle generation out of inelastic interactions), while their main drawback with respect to commercial treatment planning system was the computing efficiency. With the increasing availability of high-throughput computational power, Monte Carlo-based treatment planning systems are emerging as a practical alternative for clinical radiation therapy applications and are commonly employed for the Quality Assurance of TPS.

Flair v3 has undergone dedicated enhancements to create an integrated working environment, specifically tailored for the use of FLUKA in medical physics applications. Recent advancements include an improved handling of DICOM data, which facilitates the generation of voxel geometries. These geometries now feature built-in automatic conversion from Hounsfield units to material composition. Moreover, Boolean operations can be applied to predefined regions of interest (ROI) within the simulation, allowing for customization of material assignments. Additionally, Flair now incorporates a DICOM editor for common tasks such as anonymizing patient information and editing DICOM tags. The embedded RTViewer (Figure 4) assists users in comparing planned dose calculations from treatment planning systems with doses simulated by FLUKA. Furthermore, Flair includes an automatic dose-volume-histogram (DVH) generator. RTPLAN-modality DICOM files can be imported to generate corresponding primary-source cards for FLUKA. While RTDOSE-modality DICOM files can be converted to FLUKA's standard 3D mesh scoring format (USRBIN). This format not only allows for easy visualization in both 2D and 3D within Flair, but also serves as a volumetric primary source for subsequent simulations. The reverse conversion from USRBIN to RTDOSE is also supported.

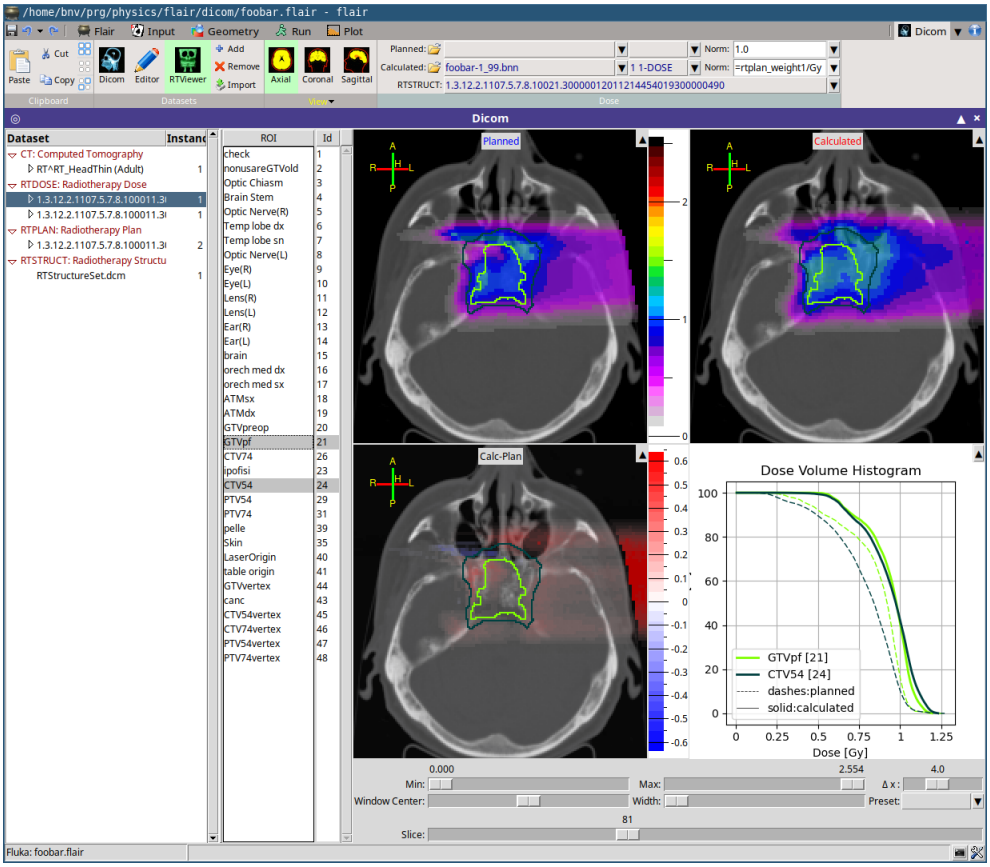

Fig. 4. RTViewer comparing the RTDOSE modality and the calculated DOSE from the simulation, and automatically creating the DVH (Dose Volume Histograms).

4 Run management

Flair v3 has the capability to submit Monte Carlo runs, selecting either single-node multi-threading and/or multi-processing on multiple nodes. It monitors the progress of the run, not based on the system process information, but rather mimicking human behavior by continuously surveilling the log and output files as they are produced. This approach makes it possible to provide real-time information on the run, regardless of the underlying platform in which the run is taking place, for example when using a Linux cluster.

The post processing programs that are used for merging and plotting are currently being transformed in way to accommodate the output formats of the various Monte Carlo programs.

5 Current status of Monte Carlo integration

The following sections describe the present state of Monte Carlo integration in Flair:

5.1 FLUKA v4

The FLUKA v4 is complete in every aspect since Flair was original designed for FLUKA

5.2 FLUKA v5

FLUKA v5 as it is built on top of Geant4, Flair support it includes most of the Geant4 functionality, with automatic generation of the databases to support the UI macro commands.

- *Input I/O*: complete
- *Input editing*: complete
- *Input conversion:* from v4 to v5 almost complete
- *Geometry visualization and editing*: very advanced with the exception of a few features as explained in the section 2.1
- *Running and monitor the progress*: fully operational
- *Data processing*: fully operational
- *Plotting*: fully operational
- *DICOM support*: not implemented yet

5.3 PHITS

- *Input I/O*: almost complete with the exception the binary lattice format.
- *Input editing*: very advanced about 880 cards/commands handled corresponding to about 80% of the total options in PHITS
- *Input conversion*: bidirectional from FLUKA v4 to PHITS and vice versa. It's quite advanced but not fully completed, operational for the common options of the two codes. Currently it can handle the primary source, materials, geometry, biasing and some scoring.
- *Geometry visualization and editing*: very advanced, it requires some improvements in the universes graphical editing and to implement the hex-lattices
- *Running and monitor the progress*: is operational needs some improvements to make it more interactive and to control the run
- *Data processing*: almost operational
- *Plotting*: Most of the 1D,2D and mesh scoring through VTK format are supported
- *DICOM support*: none

5.4 MCNP

- *Input I/O*: almost complete
- *Input editing*: only for the geometry and a limited number of cards
- *Input conversion*: bidirectional to/from FLUKA v4 only for the geometry, materials and importance biasing
- *Geometry visualization and editing:* as for PHITS is very advanced
- *Running and monitor the progress*: is able to run but no online statistics are available
- *Data processing:* not implemented yet
- *Plotting*: it can handle the 1D, 2D distribution and the mesh tallies format
- *DICOM support*: none

5.5 Penelope

- *Input I/O*: complete
- *Input editing*: complete
- *Input conversion*: none existing
- *Geometry visualization:* is complete while editing is still pending

- *Running and monitor the progress*: none
- *Data processing*: none
- *Plotting*: none
- *DICOM support*: none

6 Summary

Since its initial introduction to the public in late 2006, the Flair project has undergone continuous refinement and improvement, establishing itself as today's de-facto standard method for interacting with FLUKA. Its latest version (v3) features numerous important enhancements, particularly in the realms of integrating other MC codes, 3D visualization, and for medical applications. The FLUKA v5 integration is consistently kept up-to-date and is comprehensive, while the PHITS integration is quite advanced and steadily progressing. It can be expected to arrive soon at the same level as that of FLUKA v4 and v5.

References

1. https://fluka.cern
2. C.Ahdida et al., "*New Capabilities of the FLUKA Multi-Purpose Code*", Front. Phys., 27 January 2022 https://doi.org/10.3389/fphy.2021.788253
3. G. Battistoni, T. Boehlen, F. Cerutti, P.W. Chin, L.S. Esposito, A. Fassò, A. Ferrari, A. Lechner, A. Empl, A. Mairani, A. Mereghetti, P. Garcia Ortega, J. Ranft, S. Roesler, P.R. Sala, V. Vlachoudis, G. Smirnov, "*Overview of the FLUKA code*", Annals of Nuclear Energy 82, 10-18 (2015).
4. V. Vlachoudis, "FLAIR: A Powerful But User Friendly Graphical Interface For FLUKA", in Proc. Int. Conf. on Mathematics, Computational Methods & Reactor Physics (M&C 2009), Saratoga Springs, New York, 2009.
5. https://flair.cern
6. G. van Rossum, "*Python tutorial*", Technical Report CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam, May 1995.
7. https://www.python.org
8. https://docs.python.org/3/library/tkinter.html
9. M. Mustra, K. Delac and M. Grgic, "*Overview of the DICOM standard,"* 2008 50th International Symposium ELMAR, Borik Zadar, Croatia, 2008, pp. 39-44
10. T. Sato, Y. Iwamoto, S. Hashimoto, T. Ogawa, T. Furuta, S. Abe, T. Kai, Y. Matsuya, N. Matsuda, Y. Hirata, T. Sekikawa, L. Yao, P.E. Tsai, H.N. Hunter, H. Iwase, Y. Sakaki, K. Sugihara, N. Shigyo, L. Sihver and K. Niita, "*Recent improvements of the Particle and Heavy Ion Transport code System - PHITS version 3.33*", J. Nucl. Sci. Technol. 61, 127-135 (2024)
11. J. A. Kulesza, T. R. Adams, J. C. Armstrong, S. R. Bolding, F. B. Brown, J. S. Bull, T. P. Burke, A. R. Clark, R. A. Forster III, J. F. Giron, T. S. Grieve, C. J. Josey, R. L. Martz, G. W. McKinney, E. J. Pearson, M. E. Rising, C. J. Solomon Jr., S. Swaminarayan, T. J. Trahan, S. C. Wilson, A. J. Zukaitis. MCNP® Code Version 6.3.0 Theory & User Manual. Los Alamos National Laboratory Tech. Rep. LA-UR-22-30006, Rev. 1. Los Alamos, NM, USA. September 2022.

12. Baró, J., J. Sempau, J. M. Fernández-Varea, and F. Salvat (1995), "*PENELOPE: An algorithm for Monte Carlo simulation of the penetration and energy loss of electrons and positrons in matter*" Nucl. Instr. Meth. B 100, 31–46.
13. V.Vlachoudis, D.Sinuela-Pastor, "*Numerically robust geometry engine for compound solid geometries*", Proceedings of SNA+MC2013 (2013)