MadFLUKA BEAM LINE 3D BUILDER. SIMULATION OF BEAM LOSS PROPAGATION IN ACCELERATORS[*]

M. Santana-Leitner[†], Y. Nosochkov, T. Raubenheimer, SLAC, Stanford, USA

Abstract

Beam tracking programs provide information of orbits along the nominal trajectory for the design of beam-line optics. On the other hand, aspects like machine or radiation protection, which also inspect the transverse dimensions and volumes of components, are typically simulated with radiation transport Monte Carlo codes. Some other studies, like multistep collimator optimization or beam mis-steering phase space determination for failure shielding design, require combining features of both types of codes. A way around recurrently coupling such codes is to use radiation transport codes that include magnetic tracking capabilities, e.g. FLUKA, with full accelerator implementations. However, coding the entire 3D geometry of an accelerator along with the corresponding media and magnetic fields is often a daunting task, and so is its maintenance.

This paper presents MadFLUKA, a program that automatically produces FLUKA compatible geometries from MAD files. Objects selected from a user user-configurable database are auto-replicated with the rules of mad-deck files, and, exploiting the latest available FLUKA pattern repetition tools, a relatively light geometry is created. A FLUKA magnetic subroutine has been written to automatically read the lattice and reproduce the magnetic transport without further user coding. MadFLUKA is being used in the design of the Linac Coherent Light Source-II (LCLS-II), at SLAC.

MADFLUKA OVERVIEW

MadFLUKA not only converts MAD8 [1] lattice into 3D beam-line(s) compatible with FLUKA [2, 3], but also it automatically encodes the optics lattice.

The code will read and pre-process the `*.survey.tape` file(s), merging multi-line instructions into single objects, discarding unwanted components, transforming units and frames, identifying unique and first occurring objects as well as copies of those, adjusting dipole properties, allocating additional properties to objects, and storing into memory the result of all these transformations. Then files `beamline.geo`, `*.rot`, `*.mat`) will be generated (all these and the material database `material.dat` are linked to by the main FLUKA input file `beamline.inp`), as well as `beamline.opt` with additional information for the *magfld.f* magnetic subroutine. The user can perform a wide range of customizations, like implementing multiple beam-lines, setting discrimination rules, customizing the geometry or properties of objects or adding an additional layer of geometry, e.g. with information about the accelerator enclosure and shielding elements.

[*] Work supported by Department of Energy contract DE-AC02-76SF00515
[†] msantana@slac.stanford.edu

POST-PROCESSING MAD8 FILES

During initialization the MAD8 `*survey.tape` file is read and the following information is stored for each subcomponent: *keyword*, *name*, *Length*, *alpha* (α), *k1*, *k2*, *radius*, *engname*, *EGeV*, *TILT*, *E1*, *E2*, *H1*, *H2*, *X*, *Y*, *Z*, *SUML*, *THETA* (θ), *PHI* (ϕ), *PSI* (ψ). Additional beam parameters may be retrieved from the corresponding `*.twiss.tape` file, although in the current version of MadFLUKA, those are not used. Also, several files may be scanned, e.g. in LCLS-II the Hard-X-ray and Soft-X-ray beam-lines are both imported. Then, the following post-processing takes place:

Transforming Angles and Coordinates

MAD objects are oriented with three consecutive rotations centered on each component, i.e. first the azimuth $\theta_M(y)$ around absolute axis $\hat{\mathbf{y}}$, then a pitch , $\phi_M(x')$ around the resulting transverse axis $\hat{\mathbf{x}}'$, and finally a roll $\psi_M(z'')$ around the orbit longitudinal axis $\hat{\mathbf{z}}''$. FLUKA rotations are defined around the fixed reference frame, and are left-handed and inverted (from the rotated object to the straight prototype). Thus, after several matrix products and sign reversals and transformations, it is found that $\theta_M(y) = -\theta_F(y)$, $\phi_M(x') = -\phi_F(x)$ and $\psi_M(z'') = -\psi_F(z)$. These simple rules, along with a conversion from radian to degrees will be used to obtain FLUKA ROT-DEFI transformations. As for dimensions, they are converted from [m] (MAD8) to [cm] (FLUKA).

Filtering Components and Identifying Unique Components, Prototypes and Replica

The user can specify which components from the total MAD8 list will be considered for the generation of the FLUKA geometry/optics input. Discrimination can be performed as a function of the component type, its engineering name, position in the beam-line, etc. MadFLUKA will tag those components and, based on their names and/or physical properties (e.g. *SUML*), it will determine the repetition patterns for the remaining ones (i.e. which objects appear more than once and which are unique), it and will latch each component **j** with a flag (*latref*), defined as follows:

latref(j)=0 : j is discarded **or** is unique.

latref(j)=j : j has downstream replicas (= prototype).

latref(j)=i ($i < j$) : j is a replica of upstream component i.

A table is printed in file `lattice.dat` with the name of each component, its *latref* index and other properties like its position and orientation.

Re-orientating and Translating the Beam-line

MAD8 converted files may lead to beam-lines with unpractical orientation/position for FLUKA simulations. This can be the case if the main beam-line is not aligned with the absolute $\hat{\mathbf{z}}$-axis and/or its origin that does not match that of other objects (e.g. accelerator enclosures) in auxiliary geometry files. To do so, the user can use the coordinates ({x,y,z}) and angles (θ_u, ϕ_u, ψ_u) of a given component, printed in 'latref.dat' after a first run of MadFLUKA, and anti-rotate the entire beam-line with those mobile angles, specifying the {x,y,z}as the pivot point ($\overrightarrow{\mathbf{PIV}}$). A final translation ($\overrightarrow{\mathbf{DIS}}$) can be applied to shift the beam-line to a suitable location. The Cartesian coordinates ($\vec{\mathbf{r}}$) are transformed (to $\vec{\mathbf{r}}'$) as follows:

$$\vec{\mathbf{r}}' = \left\{ \overline{\overline{\mathbf{R_u}^{-1}}} \cdot \left(\vec{\mathbf{r}} - \overrightarrow{\mathbf{PIV}} \right) \right\} + \overrightarrow{\mathbf{PIV}} - \overrightarrow{\mathbf{DIS}} \qquad (1)$$

$\overline{\overline{\mathbf{R_u}}}$ is the rotation matrix for the user angles θ_u, ϕ_u, ψ_u.

As for the orientation angles, after the user-rotation, the angles read from the MAD file (θ_j, ϕ_j, ψ_j) do no longer represent the mobile angles of orientation of each component (θ_m, ϕ_m, ψ_m). To compute the angles j in the base u, the rotation matrix $\overline{\overline{\mathbf{R_{ju}}}} = \overline{\overline{\mathbf{R_j}}}^{-1} \cdot \overline{\overline{\mathbf{R_u}}}$ is evaluated numerically and compared to the analytical expression of a 3D-mobile rotation matrix $\overline{\overline{\mathbf{R_m}}}$. Then, an algorithm solves the resulting nine equations recurrently, discards unfeasible solutions and verifies which of the candidate results evaluates to the rotation matrix. The re-calculated positions and angles will be printed (again) to 'latref.dat' file and will be used thereafter as if they were the original 6-dimension coordinates.

Determining the Physical Length and Central Coordinates of Dipoles

In MAD8 the positions, angles, lengths, etc. actually refer to the central *trajectory* at a component rather than to the component itself. In most cases the two descriptions match, and components may be build in FLUKA geometry as if MAD8 values described the properties of those. However, for rectangular bends this does not entirely hold, e.g. the arc differs from the length., so corrections are necessary.

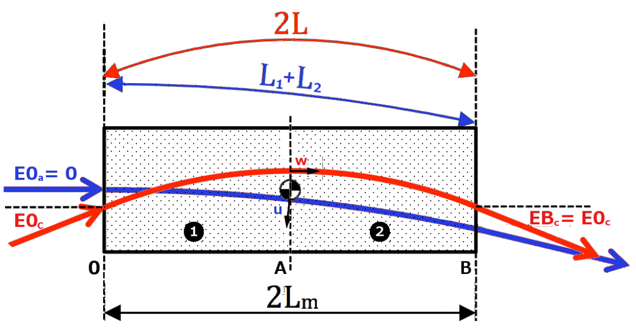

Figure 1: Typical rectangular dipole orientations, **c**entered in the trajectory arch or **a**lined with the incoming or outgoing beam.

The dipole semi-length (L_m) is computed from the MAD8 arches of the two halves (L_1, L_2) and the bend angle (α) as:

$$L_m = \frac{L_1 + L_2}{2} \cdot sinc\,(\alpha) \qquad (2)$$

MadFLUKA, compares the incoming and outgoing angles for the two halves and determines if the magnet is centered in the trajectory or if it is aligned to either the incoming or outgoing beam (Fig. 1). Based on that, the position of the magnet center $\vec{r_c}$ is computed as:

$$\vec{r_c} = \begin{cases} \vec{r_A} + \left(\hat{\mathbf{u}}_\mathbf{A} \cdot \frac{1}{2}\overrightarrow{AB}\right)\hat{\mathbf{u}}_\mathbf{A} + \left(\hat{\mathbf{v}}_\mathbf{A} \cdot \frac{1}{2}\overrightarrow{AB}\right)\hat{\mathbf{v}}_\mathbf{A} & : E_0 = E_B \\ \vec{r_0} + L_m \cdot \hat{\mathbf{w}}_\mathbf{A} & : E_0 = 0 \\ \vec{r_B} - L_m \cdot \hat{\mathbf{w}}_\mathbf{A} & : E_B = 0 \end{cases}$$

Where $\vec{r}_{0/A/B}$ are the orbit coordinates at the magnet front/mid/end planes, and $\hat{\mathbf{u}}/\hat{\mathbf{v}}/\hat{\mathbf{w}}_\mathbf{A}$ are the unitary transverse vertical/horizontal or longitudinal vectors at the magnet central plane. The later vectors are calculated as a function of the mobile axis angles, e.g.:

$$\hat{\mathbf{v}}_\mathbf{A} = \begin{Bmatrix} -cos\,\theta \cdot sin\,\psi - sin\,\theta \cdot sin\,\phi \cdot cos\,\psi \\ cos\,\phi \cdot cos\,\psi \\ sin\,\theta \cdot sin\,\psi - cos\,\theta \cdot sin\,\phi \cdot cos\,\psi \end{Bmatrix} \qquad (3)$$

BUILDING THE FLUKA BEAM-LINE

Next, MadFLUKA generates the bodies and regions of the geometry and assigns material properties to each. FLUKA uses CSG logic where *bodies* (cylinders, planes, boxes, etc.) define *regions* of space. Beam-line components will generally be made of several regions and each of those will be defined by several bodies. A material/compound and several properties can be assigned to each region.

Generating Bodies and Regions

In MadFLUKA, all regions of a component reside within its cylindrical or rectangular container body. Replica actually just consist of a container, as the inner information is imported from the corresponding (LATTICE card) prototype through a ROT-DEFI transformation. Body definitions are printed automatically for a set of pre-defined components, where dimensions are autoscaled from individual `*.tape` values (radius, length,...). The user may tune the appearance of components by adjusting several free parameters as a function of the engineering name, position, etc. The resulting equations of the bodies are oriented relative to $\hat{\mathbf{z}}$. Bodies of unique objects are all written around the origin, while prototypes are placed in the first available slot of a side gallery offset by 100 m. As for replica, the container body equation coincides with that of the corresponding prototype. The real position and alignment of components is achieved through transformation cards that affect their entire set of bodies.

Bodies are named with 8-character strings, including a 6-character prefix that inherits the `*.tape` component name.

Regions for predefined objects are automatically generated and identified with eight-character strings, the first six of which are the CAPITALIZED component name. For replica a single region is printed (equivalent to "= inside container").

Lattice and Transformation Files

LATTICE directives to replicate components are printed at the bottom of the `beamline.geo` file. As for the ROT-DEFI transformations that govern those replications and also the relocation of objects from their default position to the beam-line values, those are printed into `beamline.geo`. There, for any given component, if no rotation is involved, a single ROT-DEFI card will be printed with the pure translation from the MAD8 expected location to the position where bodies are centered around in `beamline.geo`. Otherwise, the transformation will chain up to five ROT-DEFI cards, i.e. the object is first brought from the beam-line to the origin, then three successive rotations are applied around the *fixed* axis, and finally the object is brought to the prototype coordinates in the virtual gallery.

ROT-DEFI transformations for auxiliary elements like buildings and walls should be included in `auxrot.rot` file. When writing those it is useful to remember that any rotation Φ is referred to the origin, and therefore the position $\vec{\mathbf{r}}$ is shifted by $\vec{\mathbf{dr}} = \{\overline{\overline{\mathbf{R}}}_\Phi - \overline{\overline{\mathbf{I}}}\} \cdot \vec{\mathbf{r}}$

Defining Beam-line Materials

MadFLUKA prints in `beamline.mat` the FLUKA ASSIGNMAT cards that associate regions with media and other properties (e.g. magnetic field ON/OFF)[1]. In doing so, the program can alter the default assigned material as a function of component-dependent variables, such as its *keyword*, *engname*, etc. Objects are broken up in cells so that, with different material assignments, several shapes can be achieved from a same prototype.

Beam Optics Implementation

The user does not need to write a single line of code to define the optics, and even re-compilation could be spared in most systems. This has been achieved by designing MadFLUKA logics to seemingly integrate with the FLUKA magnetic transport subroutine *magfld.f*, which was customized to interface with MadFLUKA output files. Indeed, all objects are aligned with $\hat{\mathbf{z}}$ (and then re-positioned through ROT-DEFI transformations), thus the magnetic field can be easily and generically coded as a function of the strength of the magnet and the magnet type. At initialization (i.e. the first time a particle enters a media where magnetic field is requested in ASSIGNMAT) *magfld.f* reads into an array `beamline.opt` file created by MadFLUKA, which links each component with its magnetic properties (ρ for dipoles, $k1$ for quadrupoles, $k2$ for sextuples) and with the ROT-REFI transformation that brings that component to the prototype position. An overview of the process that follows is:

1. For new particles or new magnetic zones (i.e. if FLUKA variables $Numpar(1)$, or $MREG$ or $MLATTC$ change) the rotation index (ind) is updated.

2. FLUKA function *DOTRSF* transforms the local coordinates to those of the corresponding prototype through the ind^{th} ROT-DEFI transformation in `beamline.rot`.

3. The magnetic inductances are then computed in terms of MAD8 (k_1[m^{-2}], $\rho = \frac{L}{\alpha}$[cm]) and FLUKA (P_{beam}[GeV], C_{light}[cm/s]) variables:

$$\frac{\vec{\mathbf{B}} \cdot c_{light}}{10^9 \cdot P_{beam}} = \begin{cases} \{\frac{1}{\rho(ind)\cdot 10^4}, 0, 0\} & : dipoles \\ k_1\{y, x, 0\} & : quadrupoles \end{cases}$$

4. $\vec{\mathbf{B}}$ is rotated (without translation) back to the reference frame of the current replica through instruction *UNRTO(1, BTX, BTY, BTZ, ind)*.

With few additional instructions in which failure rules of magnets are implemented (e.g. randomizing the strength and/or polarity of a bend within its power limits) it is possible to draw the phase-space of mis-steered beams and design the safety components (e.g. protection collimators) accordingly. When doing so, the randomization must be performed just once per magnet passage, e.g. when FLUKA Numpar(1) variable has been updated.

CONCLUSION

MadFLUKA is a new beam line and optics builder for FLUKA Monte Carlo code. An overview of its processes and options has been presented. MadFLUKA is being used to determine the phase-space of beam trajectories resulting from magnet failure in LCLS-II, so that design of protection collimators and local shielding can be optimized.

REFERENCES

[1] H. Grote, F. Ch. Iselin, "The MAD Program (Methodological Accelerator Design) Version 8.19, User's Reference Manual", European Organization for Nuclear Research, **CERN/SL/90-13(AP)**, Geneva, Switzerland, April 29, 1996.

[2] A. Fassò, A. Ferrari and P.R. Sala, "Electron-Photon Transport in FLUKA: Status", MonteCarlo 2000 Conference, Lisbon, Portugal, October 23–26 2000, A. Kling, F. Barao, M. Nakagawa, L. Tavora and P. Vaz eds., Springer-Verlag Berlin, p. 159–164 (2001).

[3] A. Fassò, A. Ferrari, J. Ranft and P.R. Sala, "FLUKA: Status and Prospective for Hadronic Applications", MonteCarlo 2000 Conference, Lisbon, Portugal, October 23–26 2000, A. Kling, F. Barao, M. Nakagawa, L. Tavora and P. Vaz eds., Springer-Verlag Berlin, p. 955–960 (2001).

[1] The actual definitions and properties of the materials are contained in the database `material.dat`, which is also linked by the main input file.