MARS15-MADX integration and its application for design of the Fermilab Booster collimation system

Igor S. Tropin[1*], Nikolai V. Mokhov[1]
[1]Fermi National Accelerator Laboratory, Batavia, Illinois 60510, USA
[*]tropin@fnal.gov

Abstract

Design of a high-efficiency collimation system with adequate shielding for personnel, environment and equipment in the collimation region is an essential part in development and modernization of accelerators. An approach for the solution of this class of tasks based on the integration of the MARS15 and MAD-X codes is presented. The system allows scalable runs on either a single host or a supercomputer in the MPI environment. A new collimation system for the Fermilab 8-GeV proton Booster has been designed using this new tool. The optimization studies have been performed on the ALCF THETA supercomputer. A noticeable improvement of the beam cleaning efficiency has been achieved in the simulations. The proposed shielding configuration and parameters guarantee the prompt dose levels on the berm, residual dose levels in the tunnel and radiation loads on the sump water will be below the administrative limits with required safety margins.

Introduction

The tools described in this paper were created and iteratively developed for solving problems where the primary beam plays the role of the source term and the effect of the beam halo particles lost on an aperture boundary is under investigation. A subject of a study can be, for example, collimation efficiency, radioactivation and radiation damage of equipment installed in the tunnel, ground water activation around the tunnel**,** dose on top of the berm**,** and skyshine.

A challenge with simulation of such effects in circular accelerators and storage rings is that beam halo particles can leave the aperture after passage of many turns in a ring. Precise multi-turn beam tracking usually relies on the accelerator physics methods and algorithms. It would be natural to use these specific methods inside the machine aperture, while simulating trajectories and particle-matter interactions outside the aperture with well-established multi-purpose Monte-Carlo codes. Based on the experience over the last decade, we have found that coupling of the MARS15 code [1-3] in the ROOT [4] mode for particle-matter stage with the MAD-X [5] modules for tracking in the machine aperture provides all one needs in such applications.

The STREG library from the MARS distribution includes the C++ tools which - among other things - allow creation of user-defined transport classes, called hereafter "steppers". This feature of the library is described in the first section. The stepper based on the MAD-X PTC module is created. This stepper is a part of the MARS MADX-Beamline library. The library implements interface between MAD-X – particle accelerator design and simulation tool - and the MARS15 code system. Features of the library and example of use in development of the MARS based application for the Fermilab Booster is presented in the second section. The Fermilab Booster, the proton synchrotron with circumference of 474 m, is a part of the Fermilab accelerator complex. In the current design, the Booster gets protons with kinetic energy of 0.4 GeV from the transfer line coming from the Linac and accelerates the beam to 8 GeV for the Main Injector accelerator.

There is a plan to substantially increase the beam power for the needs of the neutrino experiments. It turns out that the efficiency of the existing collimation system [6] is not enough to keep the radiation levels in and around the Booster at the current (or lower level) with the beam power being increased [7]. Based on the analysis of the Booster physical aperture, a proposal [8-10] called for a replacement of the current classical two-stage collimation system (with very thin primary collimators followed – at the appropriate phase advances - by a meter-long secondary collimators) with one or two "collimation units" made of a thick primary collimator followed practically immediately by a secondary collimator for the quasi-local absorption of the beam halo in the Booster realm. The last section describes the application of the newly-developed MARS15-MAD-X system for the optimization modelling of the collimation efficiency and radiation environment around the proposed "collimation units".

MARS STREG library

The library implements interface between the MARS15 general-purpose particle transport and interaction code [1-3] and the ROOT Geom library [4]. Corresponding subroutines from the Geom library are called from the basic MARS15 geometry module. The library contains two subroutines mandatory for any Monte Carlo particle transport code providing the answer to the question "where am I" and performing a "make-a-step" action. These subroutines are named *tgeo_find* and tgeo*_step1*, correspondingly. These and other service functions callable from the FORTRAN code are implemented in C++ and have "C" binding. FORTRAN interfaces for all such functions are combined in the

FORTRAN90 module named STREG, where the binding of the FORTRAN and C functions is described according to the ISO standard.

With this approach, the number and type of arguments passed from FORTRAN code to the C functions checked by the compiler, i.e. errors caused by mismatch in parameters of functions can be detected at an early stage of the geometry model building. The usage of the ISO standard method to call C functions from the FORTRAN code, instead of the method specific for the GCC compiler, improves portability of the MARS code. Besides that, it is no longer necessary to add underscore to the name of C functions to make it possible to call from FORTRAN. An overhead related to the use the ISO standard method is that the FORTRAN statement *USE STREG* should be added at the beginning of the declarative part of a FORTRAN code, where subroutines of the STREG library are called. Namely, in the current MARS15 version, the use-statement needs to be added to the following user's call-back subroutines, contained in m1519.f file:

1. *VFAN* – volumetric procedure, calls subroutine *tgeo_blvol* described in the STREG module.
2. *FIELD* – calculates the magnetic field components at the given point, dispatches call to function *tgeo_field*, if the ROOT geometry is active.
3. *REG1* – finds a region number for the given point in MARS non-standard, aka "user defined", geometry, to perform the task it calls function *tgeo_find* from STREG module. The subroutine is also used to import or create a ROOT geometry model.

The subroutines *VFAN* and *FIELD*, supplied with the MARS15 distribution in the m1519.f file, do not usually require any modification for use with the ROOT geometry model. However, content of the *REG1* procedure may require modification in the dependence on the way the geometry model is built – imported from the ROOT file or created "in-situ" inside the MARS15 application. Both use cases are supported by the subroutines available in the STREG FORTRAN module.

If a developer is fluent with C/C++ and ROOT, then the best way is to follow "in-situ" scenario, which implies that the geometry model is built by means of the ROOT tools inside the function named *tgeo_init*, called from the *REG1* subroutine. This use case is implemented in the *REG1* subroutine supplied with the MARS15 distribution in the m1519.f file. The following set of rules should be followed by a developer of the *tgeo_init* function:

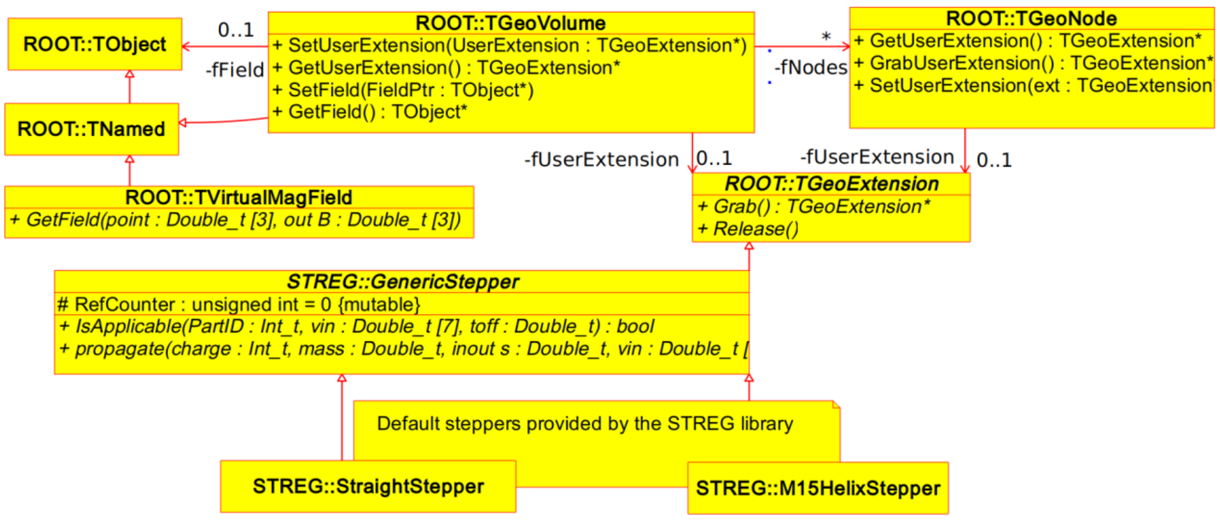
1. The function needs to be implemented in a separate file located in an application directory. Usually, it is called tgeo_init.cc.
2. The file tgeo_init.cc must be added to the list referenced by a variable SRCS of the GNUmakefile file.
3. The function must have the "C" linkage, i.e. a definition of the function needs to be enclosed in the *extern "C"* block.
4. *gGeoManager* object already exists when the function is called and can be immediately used inside the function.
5. *TGeoMedia* objects used to fill the volumes are not constructed inside the function, but retrieved from the *gGeoManager* object by means of the *TGeoManager::GetMedium* method either by the number or name specified in the MARS input file (usually MATER.INP).

For the rest, the rules of the geometry model creation are defined by the C++ and ROOT syntax and semantics. Based on the typical needs which came from application developments, the two complementary extensions were added to the library based on the hooks provided by the classes from the ROOT libGeom library.

One of the extensions is aimed to provide the possibility to change the particle tracking algorithm used in the MARS15 code, for example, to refine the transport in magnetic field or implement transport in the RF cavity.

For this purpose, the function *tgeo_step1* implements a polymorphic code, which is based on calls to the virtual functions *IsApplicable* and *propagate* for objects belonging to a user-defined class derived from the *GenericStepper* which in turn inherits from *the TGeoExtension* class defined in the ROOT libGeom library, as shown in Figure 1. The inheritance relationship allows association between objects of classes derived from the *GenericStepper* class with the ROOT geometry objects – volumes and nodes.

Figure 1: Class diagram for the STGEO base stepper classes



ROOT::TObject
0..1
-fField
ROOT::TGeoVolume
+ SetUserExtension(UserExtension : TGeoExtension*)
+ GetUserExtension() : TGeoExtension*
+ SetField(FieldPtr : TObject*)
+ GetField() : TObject*
*
-fNodes
ROOT::TGeoNode
+ GetUserExtension() : TGeoExtension*
+ GrabUserExtension() : TGeoExtension*
+ SetUserExtension(ext : TGeoExtension
ROOT::TNamed
-fUserExtension
0..1
-fUserExtension
0..1
ROOT::TVirtualMagField
+ GetField(point : Double_t [3], out B : Double_t [3])
ROOT::TGeoExtension
+ Grab() : TGeoExtension*
+ Release()
STREG::GenericStepper
# RefCounter : unsigned int = 0 {mutable}
+ IsApplicable(PartID : Int_t, vin : Double_t [7], toff : Double_t) : bool
+ propagate(charge : Int_t, mass : Double_t, inout s : Double_t, vin : Double_t [
Default steppers provided by the STREG library
STREG::StraightStepper
STREG::M15HelixStepper

The *tgeo_step1* ("make-a-step") function checks if the custom stepper is associated with the current node or volume. If it is, and call to a virtual member function *IsApplicable* for that stepper returns *true*, then the *propagate* method of the stepper attached to the node/volume is used to make a step in a node. If the custom stepper is not detected, then default steppers provided by the library are used. In the presence of the magnetic field the static object of the class *M15HelixStepper* is used to make the particle step, otherwise object of the class *StraightStepper* is used to propagate a particle.

For implementation of a new stepper class the following actions need to be performed:

1. Define a class derived from the abstract class *GenericStepper*
2. Implement two functions *IsApplicape* and *propagate*

The *IsApplicape* function should return *true*, if the algorithm implemented in the function *propagate* is applicable for a transport of the particle identified by *PartID* (see Figure 1) and having phase coordinates $vin=\{x, y, z, \Omega_x, \Omega_y, \Omega_z, p\}$ and time of flight *toff* (s); otherwise the function returns *false*. Components of the vector *vin* are *x,y,z* – the particle position in the global Cartesian application reference system (cm); $\Omega_x, \Omega_y, \Omega_z$ – components of the unit vector, indicating the motion direction of the particle; *p* – the particle momentum (GeV/c).

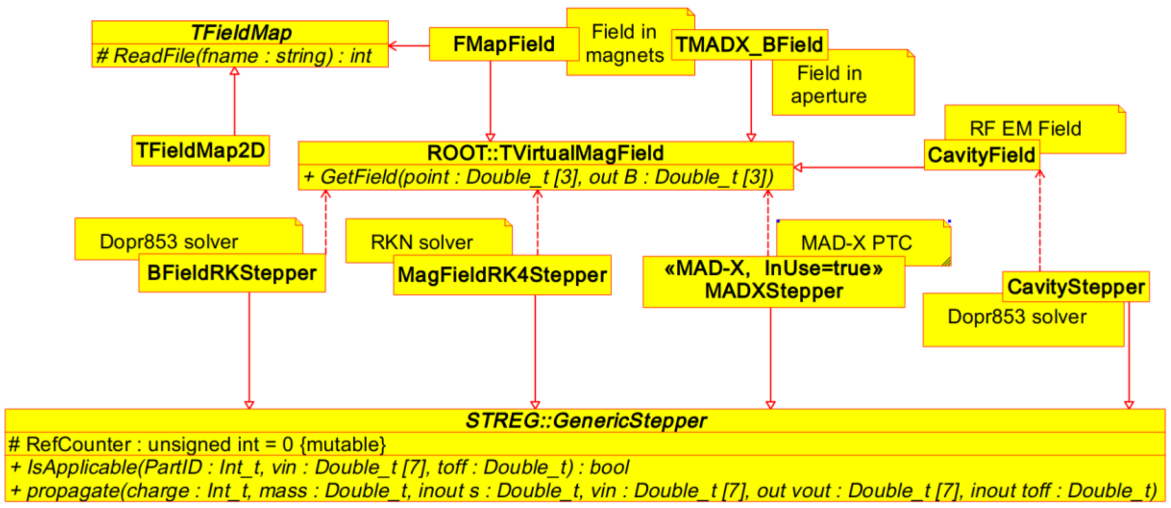
The function *propagate* should make a step *s* for the particle with the given *charge* and *mass* starting at the phase point *vin* and the time of flight *toff*. From the diagram shown in Figure 1 one can see that the parameters *s* and *toff* are used for input and output. At the input, parameter *s* contains the step size requested by the MARS particle tracking module, a pilot step. If boundary crossing is detected by the function, the value of *s* should be replaced by the pathlength passed by the particle till the volume boundary. Array *vout* should be filled with the particle phase coordinates at the end of step *s*, a value referenced by *toff* should be replaced by the time of flight till that point. The function should return zero upon successful completion, otherwise it should return -1. The special return code -9050898 tells MARS, that the trajectory is supposed to be continued by the *propagate* function of the custom stepper and should be interrupted in the MARS15 code. That is the case when the particle, moving in the beam aperture, is passed to the MAD-X PTC module.

Object of the developed class can be declared then in the *tgeo_init* function and associated with a volume or node of the ROOT geometry model by calling *SetUserExtension* member-function of the *TGeoVolume* or *TGeoNode* class. After that, at runtime, when the particle moves in a volume with the associated custom stepper, the *propagate* method of that stepper will be used to make a step in the volume, but only if *IsApplicable* method of the stepper returns *true* for the given input parameters. Otherwise, the built-in MARS15 steppes are used to make a step.

The technique described allows setting up special propagation rules in the dedicated regions without affecting underlining the MARS15 code. In general, it can be used to simulate arbitrary quasi-continuous physical interactions which can be experienced by the particle on the path between discrete strong, weak, and electromagnetic interactions modelled by MARS15. Examples include modelling coherent interactions in a bent crystal and impact of a gravitational force. The following stepper classes were developed:

1. *MagFieldRK4Stepper* – a stepper in magnetic field, 4-order Runge-Kutta-Nyström solver;
2. *BFieldRKStepper* – a stepper in magnetic field, 8-order Runge-Kutta solver;
3. *CavityStepper* – a stepper in time-dependent electromagnetic field, 8-order Runge-Kutta solver.

Figure 2: Class diagram for custom steppers and fields



TFieldMap
# ReadFile(fname : string) : int
FMapField
Field in magnets
TMADX_BField
Field in aperture
TFieldMap2D
ROOT::TVirtualMagField
+ GetField(point : Double_t [3], out B : Double_t [3])
RF EM Field
CavityField
Dopr853 solver
BFieldRKStepper
RKN solver
MagFieldRK4Stepper
MAD-X PTC
«MAD-X, InUse=true» MADXStepper
CavityStepper
Dopr853 solver
STREG::GenericStepper
# RefCounter : unsigned int = 0 {mutable}
+ IsApplicable(PartID : Int_t, vin : Double_t [7], toff : Double_t) : bool
+ propagate(charge : Int_t, mass : Double_t, inout s : Double_t, vin : Double_t [7], out vout : Double_t [7], inout toff : Double_t)

Note that the applicability rules for the steppers designed to transport particle in the magnetic field require the presence of a magnetic field in the given volume. In Figure 2 this fact is indicated by the UML “dependency” (dashed line) connections between the stepper classes and the ROOT

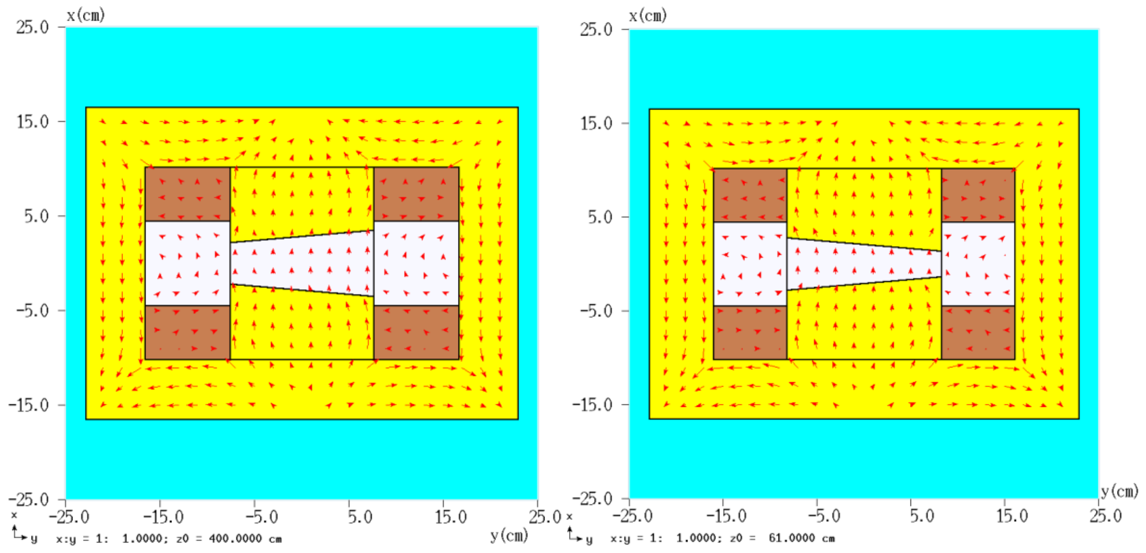
TVirtualMagField class. To satisfy the dependency, an object of a class derived from *TVirtualMagField* - *FMapField*, for example, must be associated with the volume, i.e. the object of the *TGeoVolume* class, using the *TGeoVolume*::*SetField* method. The developer needs to pay attention to the following facts: (1) ROOT allows the field association only with the *TGeoVolume* objects, there is no way to attach field to the geometry node (positioned volume); (2) the same *TGeoVolume* object can be referenced by multiple nodes; (3) all field classes defined in the STREG and other MARS15 libraries, contain the reference to the *TGeoMatrix* object, which is the argument of the field constructor function. The matrix defines transformation from the global reference frame to the local frame used for the field definition. If the same volume with the associated field is used in several nodes, the field inside all the nodes is calculated in the frame specified during the field object construction. In other circumstances, this behaviour can be considered either as a feature or a bug. The developer needs to be aware of this.

The current STGEO library, initially developed for integration of the ROOT TGeo package to MARS15, has the polymorphic implementation of the “make-a-step” task which makes it possible to use a custom propagation algorithm for the certain particle types of a certain phase volumes. Thereby, the library provides a straightforward way for coupling the MARS15 code with other simulation codes.

MARS–MAD-X coupling

The tools described here are provided in the MARS MADX-BEAMLINE library and include the stepper class based on the MAD-X PTC module and a geometry generator for the sequence of elements described in the MAD-X file, the MAD-X-MARS15-ROOT Beamline Builder (MMRBB). It is assumed that, ideally, the MAD-X input file is prepared by an accelerator physicist and - besides a description of the beam and the element sequence the beam is supposed to propagate through - contains the statements to create the survey and Twiss tables. It is expected that the elements which have the same design are declared in the MAD-X input file as belonging to the same MAD-X class. For example, two classes of the Booster combined-function magnets are declared with the cross sections shown in Figure 3.

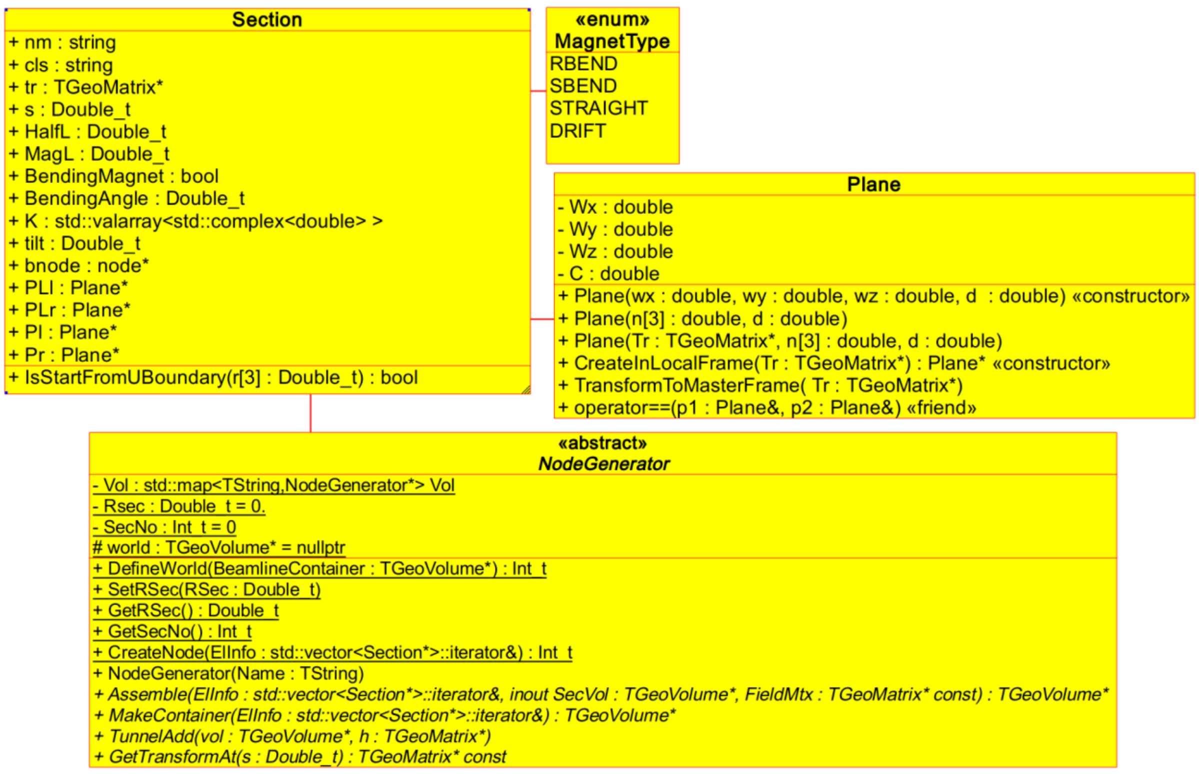
Figure 3: Geometry of the Booster combined-function magnets as implemented in the MARS15 model: DMAG class, defocusing dipole (left) and FMAG class, focusing dipole (right).


x(cm)
25.0
15.0
5.0
-5.0
-15.0
-25.0
y(cm)
x:y = 1: 1.0000; z0 = 400.0000 cm
x:y = 1: 1.0000; z0 = 61.0000 cm

There are 96 such dipoles in the Booster lattice which may have different magnetic fields so the implementation of the geometry model would take a considerable time even for a relatively small machine, like the Booster. The solution was found via MAD-MARS Beam Line Builder (MMBLB) which builds a longitudinal beam-aligned structure of the MARS geometry model using a MAD-generated optics file [11-13]. The MMBLB was successfully used by MARS users worldwide for 15 years. Recently, its capabilities have been substantially extended by switching to the MAD-X-MARS15 ROOT Beamline Builder (MMRBB). The general idea is to construct for each of the MAD-X class from the input file a generator of the three-dimensional representation of elements of that class, *aka* "factory" of the elements. In terms of the ROOT Geom library, the generator returns the unique instance of the *TGeoVolume* or *TGeoVolumeAssembly* objects filled with the geometry models of the MAD-X element. The generated object is placed then into the parent volume using a corresponding transformation matrix from the MAD-X survey module for the given element. It means that the creation of a survey table is mandatory in the MAD-X input file used in concert with the MARS15 application.

A development of a generator for the three-dimensional representation of elements of the particular MAD-X class element is the application specific task. For the geometry generator implementation, a class derived from the C++ abstract class *NodeGenerator* needs to be implemented by the application developer. The main operations of the class are presented in a class diagram shown in Figure *4.*

Figure 4 Base abstract class for implementation of geometry generator


Section
+ nm : string
+ cls : string
+ tr : TGeoMatrix*
+ s : Double_t
+ HalfL : Double_t
+ MagL : Double_t
+ BendingMagnet : bool
+ BendingAngle : Double_t
+ K : std::valarray<std::complex<double> >
+ tilt : Double_t
+ bnode : node*
+ PLl : Plane*
+ PLr : Plane*
+ Pl : Plane*
+ Pr : Plane*
+ IsStartFromUBoundary(r[3] : Double_t) : bool
«enum»
MagnetType
RBEND
SBEND
STRAIGHT
DRIFT
Plane
- Wx : double
- Wy : double
- Wz : double
- C : double
+ Plane(wx : double, wy : double, wz : double, d : double) «constructor»
+ Plane(n[3] : double, d : double)
+ Plane(Tr : TGeoMatrix*, n[3] : double, d : double)
+ CreateInLocalFrame(Tr : TGeoMatrix*) : Plane* «constructor»
+ TransformToMasterFrame(Tr : TGeoMatrix*)
+ operator==(p1 : Plane&, p2 : Plane&) «friend»
«abstract»
NodeGenerator
- Vol : std::map<TString,NodeGenerator*> Vol
- Rsec : Double_t = 0.
- SecNo : Int_t = 0
# world : TGeoVolume* = nullptr
+ DefineWorld(BeamlineContainer : TGeoVolume*) : Int_t
+ SetRSec(RSec : Double_t)
+ GetRSec() : Double_t
+ GetSecNo() : Int_t
+ CreateNode(ElInfo : std::vector<Section*>::iterator&) : Int_t
+ NodeGenerator(Name : TString)
+ Assemble(ElInfo : std::vector<Section*>::iterator&, inout SecVol : TGeoVolume*, FieldMtx : TGeoMatrix* const) : TGeoVolume*
+ MakeContainer(ElInfo : std::vector<Section*>::iterator&) : TGeoVolume*
+ TunnelAdd(vol : TGeoVolume*, h : TGeoMatrix*)
+ GetTransformAt(s : Double_t) : TGeoMatrix* const

The core function *Assemble* builds the element geometry model. This function is purely virtual in the base class, thus its definition in a derived class is mandatory. The meaning of the formal parameters is the following: *Elinfo* is a reference to the *Section* structure object containing complete information about the MAD-X element; *SecVol* is the beamline section volume. It is bounded by the shapes of the *TGeoTube* or *TGeoCtub* classes by default; *FieldMtx* – a transformation matrix which maps a global frame of the geometry model to a local frame used for definition of magnetic field in the element.

When implementing the *Assemble* method for the magnets, two objects of the *TMADX_BField* and *FMapField* classes (see Figure 2) are created by means of the C++ *new* operator and used then as input parameters of the *TGeoVolume::SetField* method to specify magnetic field in the magnet aperture and its body, respectively. The TMADX_BField class is based on an analytical representation of the field using the multipole coefficients defined in the magnet description in the MAD-X file. The parameter *Elinfo* from the argument list of the *Assemble* is passed to the constructor on creation of the *TMADX_BField* class object. The *FMapField* class is based on a tabular representation of the field. Building the class object, the developer needs to specify a reference to a table, often called the field map, which defines mapping between coordinates in a local reference system of a magnet to the components of the magnetic field vector. One more parameter passed to the constructor is a matrix which defines transformation of the local coordinates to the global frame of the model. A reverse matrix defines reverse transformation. The field map instance needs to belong to the class derived from the *FMapField,* and *ReadFile* and *GetField* functions, which are pure virtual in the ancestor. Generally, these functions could be implemented for three-dimensional magnetic field distributions, but in most cases, a 2D hard-edge approximation is quite adequate. The class *TFiledMap2D* provides the *ReadFile* function which reads an ASCII file generated by the OPERA code and implements the quadratic interpolation in the *GetField* function. Creation of the MAD-X/PTC stepper can also be done in the *Assemble* function. This is as easy in implementation as allocation of an object by means of the C++ *new* operator. A constructor of the *MADXStepper* class has two parameters. The first one is the same as that used for building the *TMADX_BField* class object. The second one is an array of 6 values, which has the same meaning as MAXAPER parameter in the PTC_TRACK command and is used for the same purpose. In the MAD-X, it is an "array defining upper limits for particle coordinates, essentially defining the aperture to trigger particle loss". It is also used in MARS15 as an acceptance window for the stepper. Association of the stepper object with the aperture volume is performed the same way as for the steppers described in Section "MARS STREG library". The following needs to be implement to the *tgeo_init* function to build a beamline model:

1. Call *NodeGenerator*::*DefineWorld* static method. The function sets up the volume specified in the argument as the container for entire beamline.
2. Call *NodeGenerator*::*SetRsec* static method. The beamline is built of the cylindrical sections – the volumes bounded by the objects of the *TGeoTube* and *TGeoCtub* classes. These sections supposed to be daughters of the volume used in the argument of the *NodeGenerator*::*DefineWorld* method. Each section contains the element geometry model created by the *Assemble* method. This is used to specify the outer radii for all the sections. The length of each the section corresponds to the length of the element. Reference to the section created by the beamline builder is passed to the *Assemble* method via the *SecVol* parameter. The number of sections created by the beamline builder is equal to the number of the non-zero length elements in the MAD-X file.
3. Create objects of the developed classes derived from *NodeGenerator* in memory. As it follows from the diagram shown in Figure 4, a string referenced as *Name* must be passed to the constructor of the base class. The *Name* is the name of the MAD-X class. Every time the object of the derived class is created, the base class constructor automatically saves its pointer to the static map *Vol* indexed by *Name*. The *Vol* is used then to find the appropriate geometry builder for the element of the MAD-X sequence. The memory allocation for the object of the class derived from *NodeGenerator* implicitly adds it to the map.

4. Call int *madx*(*char* fname*) function provided by the MARS-MADX-Beamline library. The function initiates the MAD-X session which is started from the execution of the MAD-X file given as the argument. The geometry construction starts after completion of the file processing by the MAD-X interpreter. For each element of the MAD-X sequence, the function tries to retrieve the geometry generator object from the map generated in step 3, using the MAD-X class name of elements as the key. On success, the virtual functions of the object are used to construct the section and geometry model of the element. If the geometry generator specific to the MAD-X class of the element is not found, the function tries to find a geometry generator mapped to the "default" keyword. As a last resort, the function creates just a cylindrical section with a length of the MAD-X element with the outer radius from step 2 and placement retrieved from the survey table. In the case that the geometry generator mapped to the keyword "Tunnel" is provided by the model developer, the tunnel becomes the daughter volume for the section volume, and the deepest node of the tunnel serves as a container for the element geometry model.

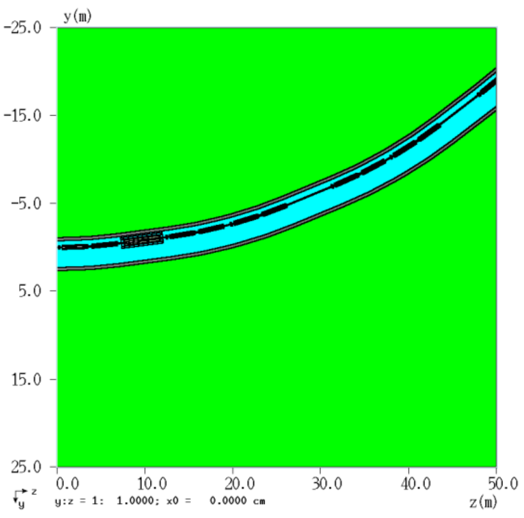
The MAD-X session initiated at step 4 continues until completion of the run. It is correct to think that MAD-X and MARS run concurrently with the synchronous execution of the MAD-X statements given as arguments of the *madx_stmt* function, called from the MARS15 code. In particular, if objects of the *MADXStepper* class were associated with the aperture volumes in the *Assemble* methods of the geometry builder objects created at step 3, the MAD-X commands PTC_CREATE_UNIVERSE and PTC_CREATE_LAYOUT are issued from *tgeo_init* routine followed by the call of the PTCTrackingActive() service function. The latter unlocks the steppers and provides the automatic execution of the MAD-X commands needed for the normal termination of the MAD-X PTC tracking and session itself upon completion of the MARS run. The library tools allow rapidly obtaining the working MARS15 model for the entire or any part of the beamline described in the MAD-X file and then gradually complicate the geometry and tracking models by means of the features provided by MARS15-MAD-X and ROOT. As a result, the same geometry model is used to transport particles in both MARS15 and MAD-X PTC parts. Beam particles assumed as lost in the MAD-X PTC module are taken care of by the MARS modules. If a particle is inside the acceptance of the MAD-X PTC tracking module, then the trajectory is passed for simulation to the MAD-X modules. An example of the application where cross-talk between two codes was used is given in the next section.

New Booster collimation unit optimization studies

As described in the introduction, a proposal [8-10] - driven by limitations of the Booster physical aperture - calls for a replacement of the current two-stage collimation system with one or two "collimation units" made of a thick primary collimator followed at a short distance by a secondary collimator for the quasi-local absorption of the beam halo [14]. The newly-developed MARS15-MAD-X system was used for the optimization modelling of the collimation efficiency and radiation environment around the proposed "collimation units" with preliminary results described in [15].

The Booster lattice consists of 24 sections, also referred as periods. The period is represented in the MAD-X file as the *line* [16]. To build the MARS model shown in Figure 5 (left), three *lines* (sequences) were included in the resulting sequence generated by the Booster MAD-X file: BCEL08, BCEL09, BCEL10. The total length of the resulting sequence is 59.3 m. The entire beamline in the sequence is wrapped in a uniformly shaped tunnel shown in Figure 5 (right). The origin of the coordinate system in the model is at the entrance to the BCEL08 *line*. Each of the periods consists of four dipole magnets. Corrector magnets shown in Figure 6 are located at the upstream ends of the first and third dipoles in each of the periods. The long drift spaces between the second dipole and corrector in each of the periods are supposed to be used for installation of the new collimation unit.

Figure 5: MARS15 model of the Booster periods 8-10 with the collimator unit in the long straight section 8 (left). Tunnel cross section (right).



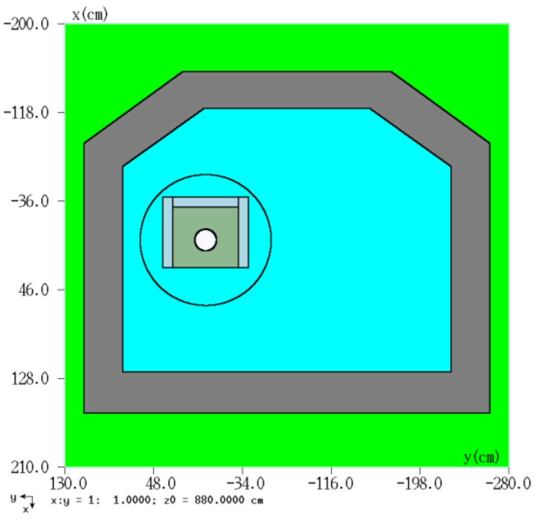
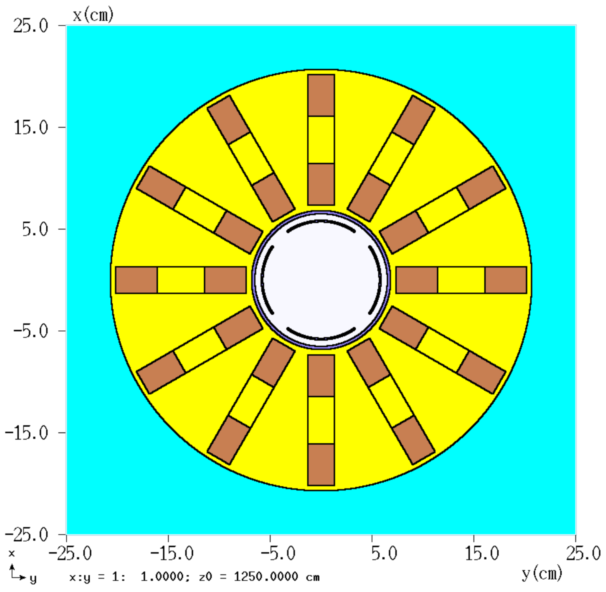


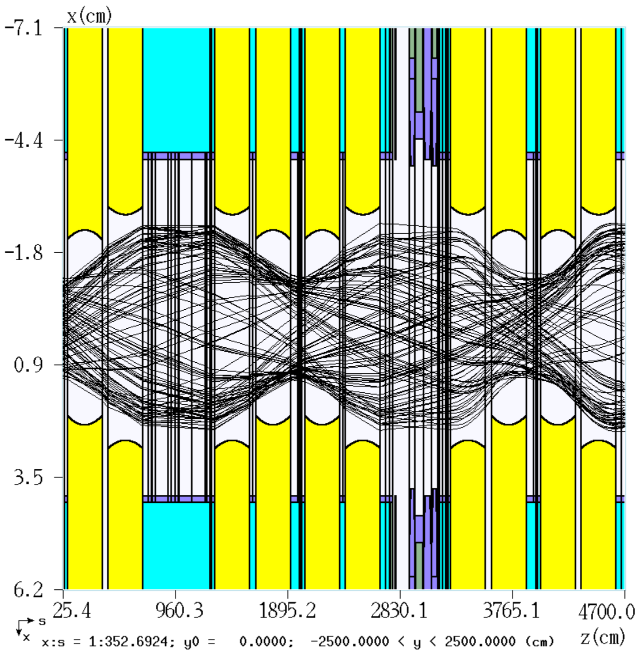
Figure 6: MARS15 model of the Booster corrector.



A new drift class was added to the MAD-X file and the element of this class was inserted in the BCEL08 *line* with a corresponding geometry generator created as a placeholder for the collimation unit installed in other period(s). The model can easily be extended to the full ring by adding the period *lines* in the resulting sequence of the MAD-X file.

The MAD-X PTC steppers are attached to the aperture volumes. Beam halo trajectories generated by means of the steppers are shown in Figure 7. The trajectories are shown in the coordinate system defined with respect to the design orbit for the Booster periods 7 and 8. The opportunity to rapidly change the range of elements included in the MARS model via MAD-X file configuration was used here. Vertical lines show boundaries of the elements from the MAD-X file. Yellow regions are the dipole magnet poles. Purple regions are metal parts representing the beam pipe and collimation unit installed in the Booster period 8. The copper primary collimators are followed by the massive secondary ones made of stainless steel and assembled in a single module (unit). In this study, the primary collimator thickness is varied in the range $T_0 \leq t \leq 16T_0$, where T_0 = 1.016 cm. The jaws of the secondary movable collimator have a total length of 60.96cm with the length of the flat part being 40.64cm. The aperture of stationary stainless-steel masks is 7.62×7.62 cm, with a total length of 45.72 cm and a flat part length of 25.4 cm. The unit is encapsulated in the steel shielding with overall size 60×60×400 cm (see details in Figure 10 below).

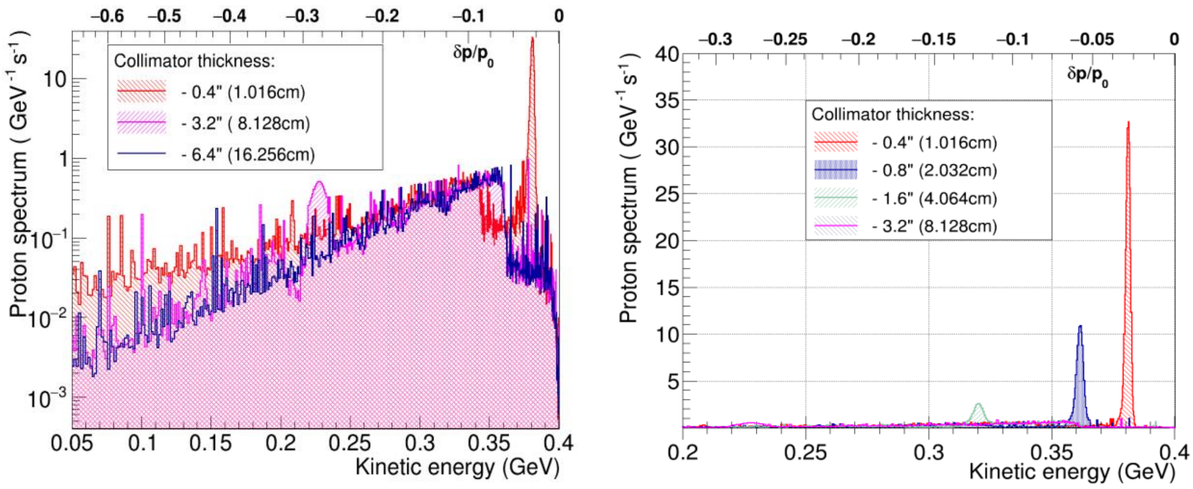
Figure 7. Proton trajectories of the 3.0 to 3.9 σ range through the two Booster periods with the collimation unit implemented (beam coordinate system, side view)



Contrary to the canonical two-stage approach used nowadays in multiturn collimation systems [17], the new Booster collimation unit is aimed at the single-pass beam halo shaving done at the proton injection energy of 400 MeV. The unit includes a relatively thick copper primary collimator, aimed to substantially increase a momentum spread of the scattered particles, and a massive secondary collimator(s), designed to intercept these particles in a local manner.

In this study, the proton spectra $dN_{pa}(E,t)/dE$ inside the aperture at the exit from the collimation unit (Figure 8) were simulated with the integrated MARS15–MAD-X system for the optimization of the primary collimator thickness t with respect to the highest collimation efficiency. The beam halo hits the aisle-side jaw of the horizontal primary collimator. The opposite jaw is in the garage position. The vertical coordinates $(y_{mad}, v = \alpha_y y_{mad} + \beta_y p_y$ are sampled from the restricted Gauss distribution with zero mean value and standard deviation $\sigma_y = 0.635\ cm$ in such a way that $abs(y_{mad}) < h$, where h is the half-height of the horizontal jaw. The horizontal coordinates are sampled from the uniform distribution $w(x_{mad}) = (b - a)^{-1}$, where $a = 3\sigma_x = 1.126\ cm$ is the jaw opening, $b = a + 1mm$. For this study the conservative scraping rate $N_a = 3.89 \times 10^{12}\ s^{-1}$ was used, which is 5% of the total beam intensity after the upgrade planned for the Booster. Jaws of secondary collimators in the considered collimator unit – both vertical and horizontal – were aligned with the $3\sigma_{x,y} + 2mm$ beam envelope. Figure 8 shows the calculated $dN_{pa}(E,t)/dE$ spectra. For the primary collimator reference thickness $t = T_0$, the spectrum has a pronounced maximum at 380 MeV. For larger t , the peak is shifted to lower energies *of* a decreased height. The peak disappears if the thickness exceeds a proton interaction length of 15 cm in copper.

Figure 8: Proton spectra $dN_{pa}(E,t)/dE$ at the collimaton unit exit for several thicknesses of the primary copper collimator for the entire energy range for *t/T₀* = 1, 8 and 16 (left) and its high-energy part *t/T₀* = 1, 2, 4 and 8 (right)



One can now calculate the number of protons above a certain kinetic energy threshold E_{th} in the aperture at the collimation unit exit relative to the scraping rate N_a , i.e. to the number of beam halo protons hitting the primary collimator jaw per second

$$N_{pa}(E_{th}, t) = \frac{1}{N_a} \int_{E_{th}}^{E_0=400\ MeV} dE\ dN_{pa}(E,t)/dE$$

This quantity can be referenced as a *collimation inefficiency*. Then the *collimation (absorption) efficiency* is $\varepsilon = 1 - N_{pa}$. The collimation inefficiency is shown in Figure 9 as a function of the energy threshold E_{th} for the primary copper collimator thickness in the range $1 \le t / T_0 \le 16$. For the relevant to the system performance threshold energies $0.32 \le E_{th} \le 0.38$ GeV, the collimation (absorption) efficiency could be as high as $\varepsilon = 0.9$ at $t = T_0$, $\varepsilon = 0.95$ at $t = 2T_0$, and $\varepsilon = 0.995$ at $t = 8T_0$.

Figure 9: **Collimation inefficiency as function of energy threshold for several thicknesses of the primary collimator jaws**

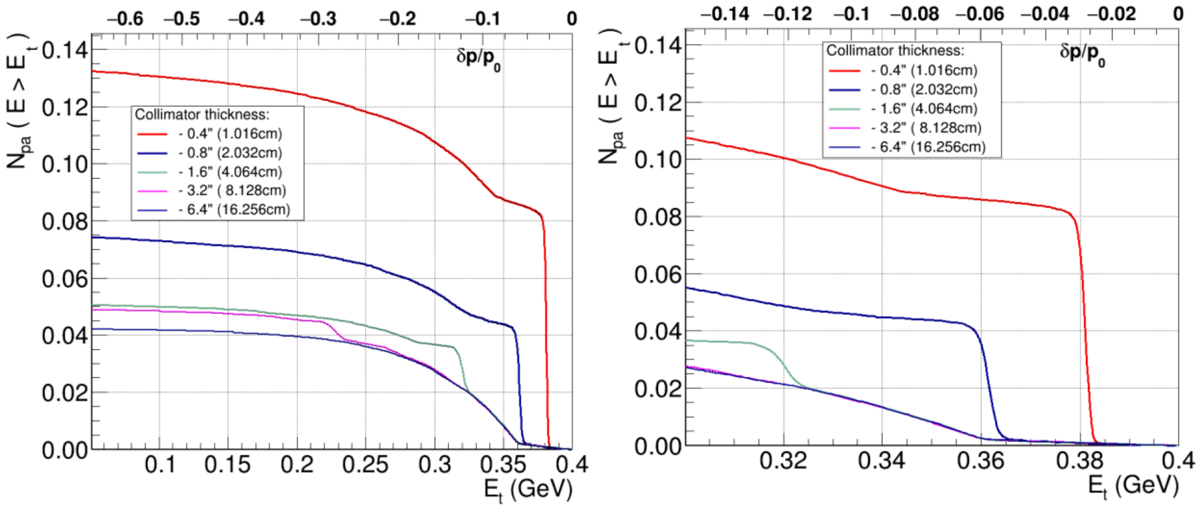
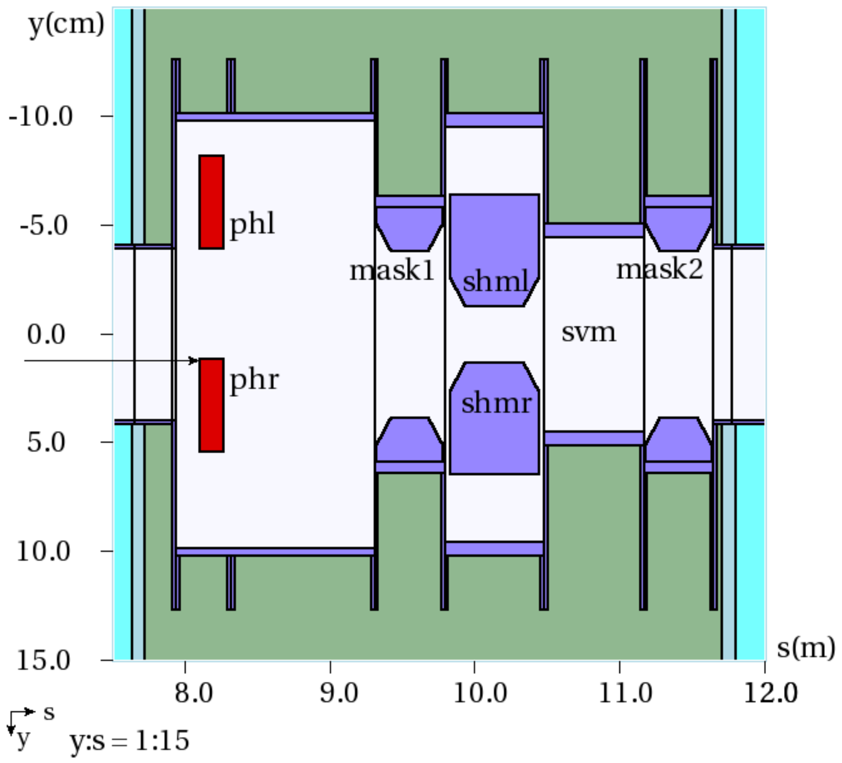


Figure 10: Plan view of the MARS15 model of the collimation unit in the beam coordinate system with primary collimators (red), as well as tapered masks and movable secondary collimators (purple). The unit is surrounded by steel shielding (grey) encapsulated in marble (light grey).

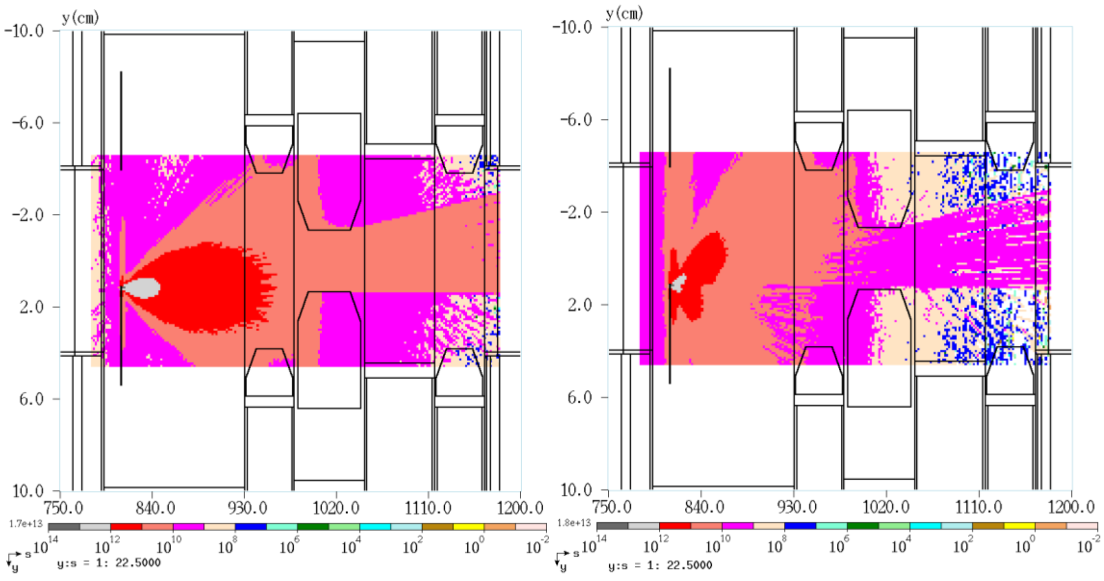


A MARS15 model of the collimation unit is shown in Figure 10. The unit includes two horizontal primary collimators right (phr) and left (phl) of the beam axis, two vertical primary collimators above (pva) and below (pvb) of the beam axis, two fixed aperture masks (mask1 and mask2) along with movable horizontal (shmr and smhl) and vertical (svm) secondary collimators. The fractions of 250-W beam halo power deposited in the collimation unit jaws are given in Table 1 for three thicknesses *t* of the primary collimators. The power deposition in the primary collimator active jaw (phr in this simulation) grows linearly with *t* in the range $1 \le t/T_0 \le 4$, saturating at larger thicknesses. That quantity at the exit of the collimation unit (mask2) decreases linearly with *t* in the entire range studied $1 \le t/T_0 \le 16$, consistent with the shown above increase of the collimation efficiency $\varepsilon = 1 - N_{pa}$. This effect is also clearly seen in Figure 11 for evolution of the hadron flux density above E_{th} = 1 MeV along the collimation unit and its substantial reduction at the downstream end of the unit for a thicker primary collimator. One can conclude here that the primary collimator with a thickness in the range $4 \le t/T_0 \le 8$, or roughly 4 to 8 cm of copper, provides the desirable increase of the collimation efficiency, with $t = 8T_0$ being a thickness of choice.

Table 1: Fractions (%) of beam halo power 250 W deposited in the collimation unit jaws.

t/T_0	phr	phl	mask1	shmr	shml	svm	mask2
1	5.53	0.014	20.04	13.2	20.1	8.72	0.3
4	21.6	0.21	20.8	4.48	5.95	1.03	0.07
16	54.0	2.0	9.27	0.15	3.71	0.65	0.02

Figure 11: Hadron flux density above E_{th} = 1 MeV in the collimation unit for two thicknesses of the primary collimator $t = T_0$ (left) and $t = 8\,T_0$ (right).



The hadron flux density above E_{th} = 30 MeV is often used for quick estimations of air activation in accelerator tunnels as well as ground and sump water activation of the soil outside the tunnel walls. The map of this quantity is shown in Figure 12 (left) in the collimation region and nearby downstream region. The total prompt dose map is shown in Figure 12 (right). The analysis has shown that with the optimized steel-marble shielding around the unit described above, the radiation field is adequately contained in the region with the prompt dose levels on the berm, residual dose rates on the shielding outside and radiation loads on the sump water being below the administrative limits with required safety margin. Residual dose rates on the upstream ends of the hottest machine components (Figure 13) are high – although quite typical for the Booster magnet faces – and may require implementation of steel masks in those regions

Figure 12: Hadron flux density distribution above E_{th} = 30 MeV (plan view, left) and total prompt dose distribution (side view, right) in the tunnel and outside the walls for the collimation unit region and the first 40 m downstream it. $t = 8\,T_0$ in both cases.

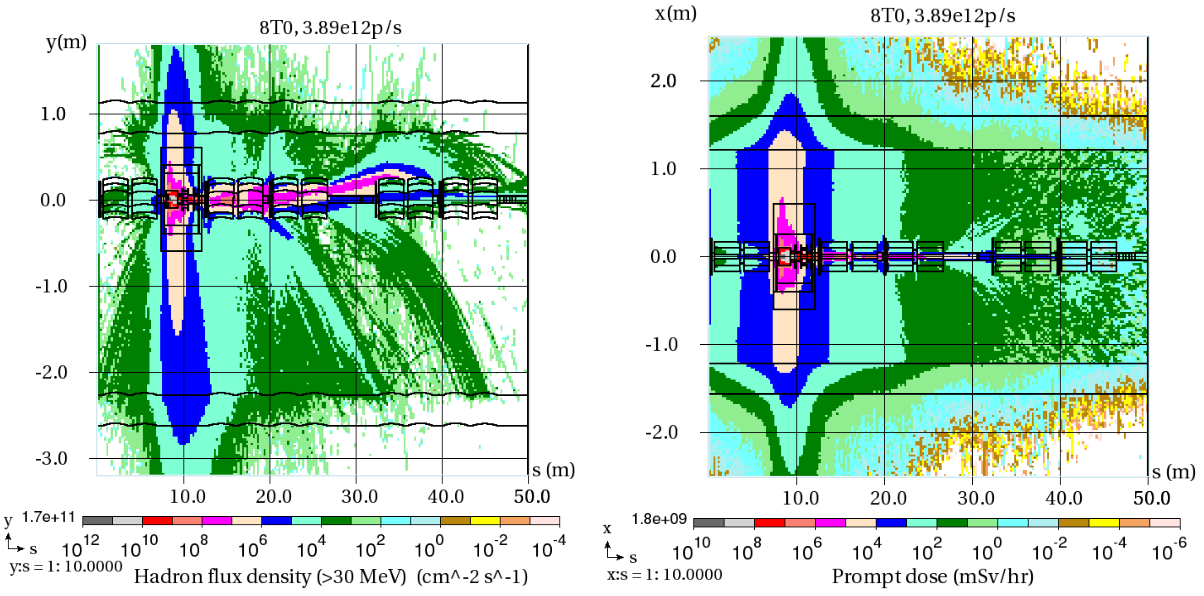
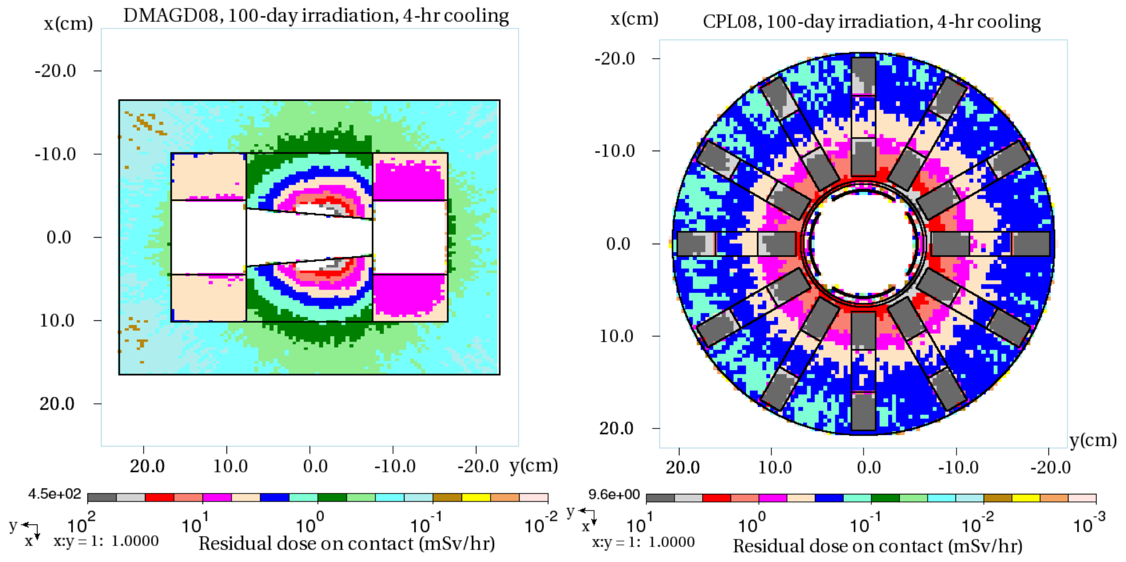


Figure 13: Residual dose isocontours on contact at the first dipole (left) and corrector magnet (right) downstream of the collimation unit after 100-day irradiation and 4-hours cooling.



Conclusions

The functionality of the STREG library of the MARS15 code which interfaced the ROOT geometry package with the MARS tracking engine has been extended through the polymorphic implementation of a "make-a-step" procedure. The polymorphism provides an opportunity to substitute the built-in MARS15 steppers by a user-defined algorithm without any changes in the main code. The approach was comprehensively tested and refined in implementation of the steppers for particle transport in magnetic and time-dependent electromagnetic fields. A stepper based on the MAD-X PTC module was implemented to the MARS15-MADX system making it even more powerful in accelerator and beamline applications.

The system described in this paper was developed, thoroughly tested and refined in the MARS15 code applications to the needs of the ESS, ILC, MAP and FCC projects. It has been successfully applied to the design optimization of the new Fermilab Booster collimation system. The parameters of the introduced collimation unit have been optimized with respect to the highest collimation efficiency. It was found that the primary collimator thickness should be $4 \leq t/T_0 \leq 8$, or roughly 4 to 8 cm of copper, with 8 cm being preferable. The final decision here is to be made based on thermomechanical analyses. The collimation unit shielding was optimized to provide the tolerable radiation fields in the unit itself, downstream magnets, in and around the tunnel.

Acknowledgements

The authors are thankful to C. Bhat, J. Johnstone, V. Kapin, W. Pellico and V. Sidorov for the valuable input and discussions. This document was prepared using the resources of the Fermi National Accelerator Laboratory (Fermilab), a U.S. Department of Energy, Office of Science, HEP User Facility. Fermilab is managed by Fermi Research Alliance, LLC (FRA), acting under Contract No. DE-AC02-07CH11359. It also used a Director's Discretionary allocation at the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] N. V. Mokhov (1995), "The MARS code system User's Guide", Fermilab-FN-628.
[2] MARS Code System (2019) https://mars.fnal.gov/
[3] N. V. Mokhov, P. Aarnio, Yu. Eidelman, K. Gudima, A. Konobeev, V. Pronskikh, I. Rakhno, S. Striganov and I. Tropin (2014), "MARS15 code developments driven by the Intensity Frontier needs", *Prog. Nucl. Sci. Tech.*, **4**, pp. 496-501.
[4] ROOT – Data Analysis Framework, Geometry (2019) https://root.cern.ch/geometry/
[5] MAD - Methodical Accelerator Design, CERN - BE/ABP Accelerator Beam Physics Group (2018) http://madx.web.cern.ch/madx/
[6] N. V. Mokhov, A. I. Drozhdin, P. H. Kasper, J. R. Lackey, E. J. Prebys and R. C. Webber (2003), "Fermilab Booster beam collimation and shielding", *PAC03 Cong. Proc.* C030512, pp. 1503-1505.
[7] V. V. Kapin, V. A. Lebedev, N. V. Mokhov, S. I. Striganov, and I. S. Tropin (2017), "Numerical simulations of collimation efficiency for beam collimation system in the Fermilab Booster", NAPAC-2016-WEPOA2, pp. 735-738.
[8] V. V. Kapin (2017), "A proposal for upgrade of Booster collimation system", Fermilab Beams-doc-5340, February 22, 2017.
[9] V. V. Kapin (2017), "Booster collimation upgrade plans", Fermilab Beams-doc-5930, November 20, 2017.

[10] V. I. Sidorov (2017), “New Booster collimation system conceptual design”, Fermilab Beams-doc-5927, November 15, 2017

[11] D. N. Mokhov, O. E. Krivosheev, E. McCrory, L. Michelotti, and J. F. Ostiguy (2000), “MAD parsing and conversion code”, Fermilab-TM-2115 (2000).

[12] O. E. Krivosheev, E. McCrory, L. Michelotti, D. N. Mokhov, N. V. Mokhov, and J. F. Ostiguy (2001), “A Lex-based MAD parser and its applications'', *Proc. 2001 Particle Accelerator Conference*, pp. 3036-3038, Chicago, IL, June 18-22, 2001; Fermilab-Conf-01/142-T (2001).

[13] M. A. Kostin, O. E. Krivosheev, N. V. Mokhov, and I. S. Tropin (2003), “An improved MAD-MARS beam line builder: user's guide," FERMILAB-FN-0738.

[14] V. V. Kapin (2018), “Collimator setting for the model 2”, Fermilab technical note, April 5, 2018.

[15] I. S. Tropin, N. V. Mokhov (2018), “Update on MARS studies of new Booster collimators”, Fermilab Beams-doc-6919, December 12, 2018.

[16] Booster MADX files (2016) https://cdcvs.fnal.gov/redmine/projects/booster_madx_lattice_files/repository

[17] N. V. Mokhov (2003), “Beam Collimation at Hadron Colliders”, *Beam Halo Dynamics, Diagnostics and Collimation (ICFA HALO’03), AIP Conf. Proc.* ***693***, pp. 14-19.