MARS15-Based System for Beam Loss and Collimation Studies

N.V. Mokhov and I.S. Tropin *
Fermilab, Batavia, IL, USA

Abstract
The system includes the newest version of the MARS15 code for particle production and transport in accelerator components integrated with a novel high-precision tracker/stepper for multi-turn tracking merged with MAD-X. The overall approach is described with all the essential features and programming issues highlighted. The application is illustrated for the Fermilab Recycler Ring and Integrable Optics Test Accelerator (IOTA) facilities as well as for the Higgs Factory muon collider and International e^+e^- Linear Collider (ILC) projects.

Keywords
MARS; MAD-X; ROOT; beam line builder; beam loss; collimation

1 Introduction

At hadron colliders, as at any other accelerator, the creation of beam halo is unavoidable. This happens because of beam-gas interactions, intra-beam scattering, beam particle collisions at the interaction points (IP), and particle diffusion due to RF noise, ground motion and resonances excited by the accelerator magnet nonlinearities and power supplies ripple Ref. [1]. As a result of halo interactions with limiting apertures, hadronic and electromagnetic showers are induced in accelerator and detector components causing numerous deleterious effects ranging from minor to severe. An accidental beam loss caused by an unsynchronized abort launched at abort system malfunction can cause catastrophic damage to the collider equipment. Only with a very efficient beam collimation system one can reduce uncontrolled beam losses in the machine to an allowable level Ref. [1, 2]. Modeling of beam losses in the machine, suppression of their rates at the critical locations via appropriately designed multi-stage collimation systems, and evaluation of the beam loss impact on the collimation system components, superconducting and conventional magnets are done by using – in concert – beam tracking, particle-matter interaction and material response simulation tools.

The contemporary approach consists of partitioning the spatial regions of interest into parts where the corresponding methods are applicable best. Inside the beam aperture, trajectories of the beam and beam halo particles are simulated by means of the deterministic methods of the accelerator physics implemented in the codes such as STRUCT Ref. [3] and SIXTRACK Ref. [4, 5]. In the regions filled with a substance, stohastic Monte-Carlo methods of particle interaction and transport are used, with the best fit to accelerator applications in the FLUKA Ref. [6] and MARS15 Ref. [7, 8] codes. Another code – BDSIM Ref. [9] – combines C++ in-vacuum accelerator style particle tracking and GEANT4 physics Ref. [10]. Successful applications of this approach at the SSC, LHC and Tevatron colliders as well as at their injector chains and fixed target experiments have proven its reliability and effectiveness.

When the approach is used, one needs to implement a way for the bi-directional exchange of the phase coordinates of particles crossing the boundaries between the regions. It seems obvious also that the geometry definition has to be shared between the two codes. A previous experience with the STRUCT-MARS bundle showed that the fulfilment of the requirements mentioned above is necessary to keep a consistency of the joint model and avoid doubling the functionality of one code inside another. For

*Corresponding author: tropin@fnal.gov

https://doi.org/10.23732/CYRCP-2018-002.57.

example, in the early version of the STRUCT-MARS bundle, the simplex (one-directional) data transfer was used: the phase coordinates of beam particles which were determined as lost on the aperture by STRUCT, were stored to a file to be read by MARS as a source term. Originally, there was no way to send back a particle from MARS to STRUCT in case of its backscattering to the aperture. Due to this limitation, a propagation of the beam particles through a thin scatterer (primary collimator) had later been re-implemented from MARS to STRUCT in a simplified form. Inasmuch, a description of the apertures in the two codes was implemented independently by means of different tools and even with respect to different coordinate systems. A special care was taken to keep the aperture definitions in the two codes consistent with each other. A quite important feature of the STRUCT-MARS coupled system in beam loss simulations was a modelling with a rather low threshold momentum $0.7p_0 \leq p \leq p_0$ for particles escaping primary collimators and other components and tracked through the system. Here p_0 is the beam momentum. Corresponding physics processes in this momentum region were described with a fast simplified MARS module included in the STRUCT code. This feature allowed getting representative beam loss distributions with a contribution from many other lattice components in addition to those from the primary collimators. The full shower simulations with MARS down to thermal neutrons were done then using such a source term to get detailed energy deposition and radiation maps.

The current MARS15-based system for beam loss and collimation modeling is described in this paper. It inherits the earlier approach of the MAD-MARS beam line builder Ref. [11–13] for creation of the beam line (accelerator) geometry now described by means of the MAD-X code Ref. [14] and usable directly in MARS15. Coupling to the MAD-X code gives us a possibility for precise and fast tracking inside the beamline vacuum with the MAD-X PTC module. The data between the MARS15 code and the PTC module are passed as parameters of corresponding functions. Such a way of the data transfer between the program modules is much more effective compared to other possible implementations. It seems obvious that copying data between actual and formal parameters of a function inside address space of a process is definitely much faster then passing data over network port as used, for example, in Ref. [15].

The PTC tracker and MARS15 use the same geometry definition to check the aperture boundaries. The ROOT TGEO package Ref. [16] used in MARS15 allows association of extensions defined by a user with the objects defined in the geometry model. This feature can be used by a user to attach to the node a stepper – the object which is used to make a step in the volume. The functionalities of this feature are described in Sec. 2. The tools require the ROOT version 6.12.04 and MAD-X v5.03.07 or newer ones. The tools used to prepare the beam line geometry shared between the MAD-X PTC tracking module and MARS15 are described in Sec. 3. Several use cases are given in Sec. 4.

2 ROOT-Based Geometry Library for MARS - libstgeo

The implementation of the MARS15 geometry is done using the features provided by the LIBGEOM library which is a constituent of the ROOT system Ref. [16] developed at CERN. It is often referred to as ROOT TGEO because most of the class names provided by the library have prefix TGEO. The ROOT geometry can be imported to the MARS15 application or created in a function called in the application. One can also combine an imported geometry module, e.g., the one of an entire collider detector in the GDML format, with the full 3D description of the collider geometry (Fig. 1) created by means of the tools described in Sec. 3. The TGEO package provides many remarkable features. One of them is a hook allowing to associate an electromagnetic field as well as any object of the user-defined class with the geometry objects. These features are actively used by the MARS15 developers. Recently it was decided to group the frequently used extensions into the library made available for all the MARS15 users.

In general, a geometry module of the considered Monte-Carlo codes performs the following two tasks: (1) finds an identifier of a region where a particle step starts, called a "where-am-i" task; (2) calculates the particle coordinates at the end of the step done in the given direction or at the point where the step crosses the nearest geometry surface – "make-a-step" task. For shortness, the class of objects

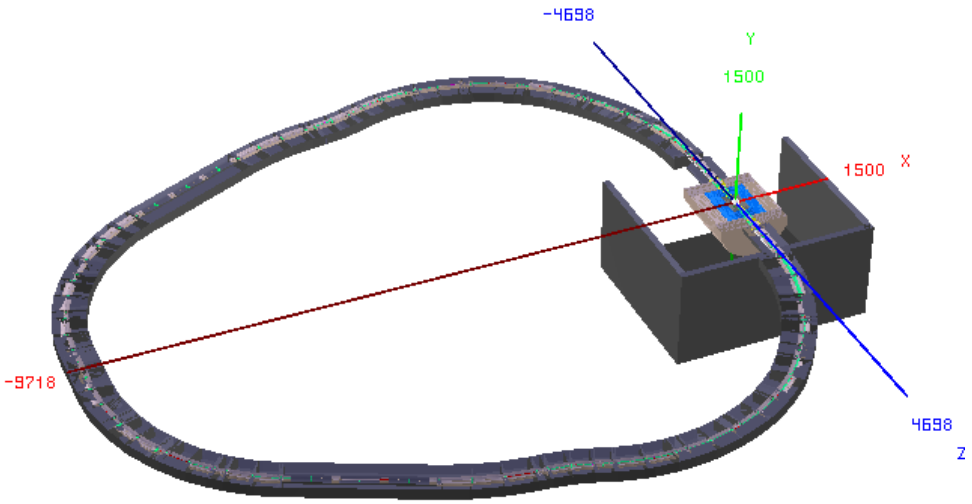

Fig. 1: MARS15 3D model of the Higgs Factory muon collider with the SiD-like detector where the silicon vertex detector and tracker are based on the upgraded version of the CMS detector

implementing the solution of the latter task referred to as "stepper".

In the standard implementation, a solution of the aforementioned tasks were built into the MARS native geometry module. For example, the stepper for charged particles in the magnetic field was implemented by means of a helix algorithm. A user had a control over the algorithm by means of parameters defined in an input file. It was challenging to replace the helix algorithm by something more sophisticated in multi-turn accelerator applications. After integration of the ROOT TGEO package to MARS15, it is now easy to do. By design, a stepper object can be associated either with the volume or node of the TGEO geometry. At the runtime, when the MARS15 tracking module requests a solution of the "make-a-step" task, the library function checks whether the user-defined stepper object is associated with the current node or the volume. If “Yes”, then it is tested if the stepper is able to make the provided step. If successful, the stepper makes the step. If there were no user-defined steppers associated with the volume, one of the two steppers provided by the library is used – the straight stepper or helix stepper. The latter is used when magnetic field is associated with the volume. Straight steppers are used in the regions without magnetic field. To keep a backward compatibility, both of the default steppers are controlled via parameters in `MARS.INP` file. This way the developer can create his/her own stepper used to transport particles in the dedicated regions. No changes in the existing MARS15 code are required.

For implementation of a stepper, the library provides the following abstract base class:

```
#ifndef TGEO_GENERIC_STEPPER_H
#define TGEO_GENERIC_STEPPER_H
#include <TGeoManager.h>
#include <TGeoExtension.h>
class GenericStepper: public TGeoExtension {
 private:
  mutable unsigned int RefCounter;

 protected:
 GenericStepper():RefCounter(0){};
 public:
  virtual TGeoExtension* Grab()
  {  RefCounter++;  return this;}
  virtual void   Release() const {
```

```
    assert(RefCounter > 0);
    RefCounter--;
    if (RefCounter==0) delete this;
  };
  // vin[0:2] - r; vin[3:5] - dir; vin[6] - momentum
  virtual Int_t propagate(const Double_t charge,
                          const Double_t mass,
                          Double_t* s,
                          const double vin[7],
                          double vout[7],Double_t* toff) = 0;
  virtual bool IsApplicable(const Int_t PartID,
                            const Double_t vin[7],
                            const Double_t toff)=0;
};
#endif
```

As one can see, the class `GenericStepper` is derived from `TGeoExtension` – the class defined in the ROOT TGEO package. A file named `GenericStepper.h` containing the declaration shown above can be found in the MARS15 distribution in the directory `$MARS15/linux/include/streg`. One can note also that `GenericStepper` is the abstract class, i.e., to implement a user-defined stepper, one needs to create a derived class containing two functions `IsApplicable` and `propagate`. The `IsApplicable` function returns `true` if the algorithm implemented in the `propagate` function, is applicable to a particle identified by `PartID` and having position, direction and momentum specified in a vector `vin`, and the time of flight `toff`. Components of the vector `vin` are $\texttt{vin}(0:2) = x, y, z$ – particle coordinates (cm) in the global frame of the application, and $\texttt{vin}(3:5) = \{\Omega_x, \Omega_y, \Omega_z\}$ – direction cosines of the unit vector, defining direction of the particle motion, $\texttt{vin}(6) = p$ – the particle momentum. The `const` specifier used in the declaration, indicates that all parameters are used only for input and are not changed by the function.

The function `propagate` makes a step for the particle with the given `charge` and `mass`. Parameters `s` and `toff` are used for input and output. At the input, `s` refers to the step size requested by the tracking module, a pilot step. If this step is shortened by a boundary crossing, it is expected that the function will replace the pilot step by the one till the boundary crossing. Similarly, the input `toff` is the reference to the time of flight at the step origin, with its value to be replaced by the time at the step end. The function returns zero upon a successful completion, otherwise it returns -1. There is a special return code -9050898 issued when there are no errors inside the function, but the current trajectory needs to be stopped. It is used when the stepper decides to pass the particle to another tracking module.

The following steppers are currently available in the library:

1. `MagFieldRK4Stepper` – a stepper in magnetic field, 4-order Runge Kutta 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.

All these steppers have been carefully tested. One of the tests was a comparison of the individual orbits generated with the MAD-X PTC and MARS15 `BFieldRKStepper` stepper in the Fermilab Recycler Ring. Figure 2 shows a perfect agreement between the two results.

Let's note that the applicability rules for the steppers described require the presence of the field in the region which the stepper is attached to. In terms of the ROOT TGEO package, it means that the field object has to be associated with the volume by means of `SetField` method defined in the `TGeoVolume` class. The latter should be derived from the abstract class `TVirtualMagField` defined in ROOT. The library provides an extendable set of such classes for electromagnetic fields defined in local coordinate systems and allowing tabular or analytical representaion. In most cases, the implemented steppers are used to solve the equation of motion in bounded regions. In general, this technique can be

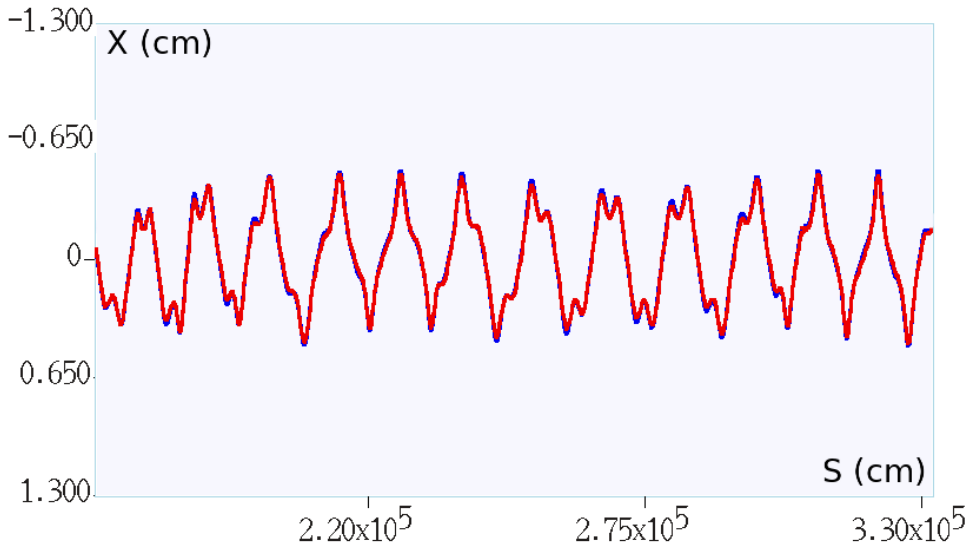

Fig. 2: Beam orbits in the Fermilab Recycler calculated with the MAD-X PTC module (blue) and the MARS15 `BFieldRKStepper` (red)

used to simulate arbitrary quasi-continuous physical interactions which can be experienced by particle on the path between the **discrete** strong, weak and electromagnetic interactions modelled by MARS15. Examples include modelling coherent interactions in a bent crystal and impact of a gravitational force.

3 MAD-X/ROOT MARS beam line builder (MRMBLB)

The tools described here are a successor of the code Ref. [13] with the following main distinctions:

- A geometry model is created by means of the ROOT TGEO library, while previously a MARS "non-standard" geometry option was used.
- A MAD-X input file is now used as input, the previous format [13] is also supported.
- The data provided in the MAD-X `BEAM` statement and tables can be accessed from the MARS15 code via the MAD-X `API` calls.
- The MAD-X survey table is used for the element positioning, while in the previous version a special function was used that duplicates the functionality of the MAD-X survey module.

As before, the user is expected to provide a so-called beam line element generator (henceforth called "factory") to allow descriptions of a single type/class, used in the beam line, to be searched then by a MAD-X class name. It is expected that the elements of the same type (having the same design) are declared as belonging to the same class in the MAD-X input file. This assumes that such a file is a database providing complete information on all the beam line elements.

Prior to creation of the complex builders for the magnets, collimators, etc., one would propose to figure out what beam pipe is most frequently used for the drift spaces in the beam line and implement a factory for that element first, then register the created factory as a builder of elements of the class having name "default". After that, if the beam line builder does not find the specific factory for the class, then an order will be placed to the factory to assemble default elements. After implementing such a default factory, one can visualize the ring or the beam line in the MARS15 GUI or the ROOT 3D viewer. The visualization is a powerful tool at the time of creation of the more complex elements. Here is an example of creating a factory for the beam pipes which can have magnetic field inside (f the field is defined for the elements in the MAD-X file). A declaration file for the factory reads:

```
#ifndef  MagRoundVacuumPipe_H
#define MagRoundVacuumPipe_H
#include "NodeGenerator.h" // Declarations provided by the beamline builder
class TVirtualMagField;
class MagRoundVacuumPipe: public NodeGenerator
{
  Double_t wall_thickness;
  Double_t R;
  TGeoMedium* STST;
  TGeoMedium* VAC;
  Int_t CopyNo;
 public:
  // Constructor
  MagRoundVacuumPipe(TString name,
                     Double_t ExternalRadius,
                     Double_t WallThickness,
                     TGeoMedium* WallMaterial);

  virtual TGeoVolume* Assemble(const std::vector<Section*>::iterator& ElInfo,
                     TGeoVolume* SecVol, TGeoMatrix* FildMtx);
};
#endif
```

The C++ header shown above declares a class MagRoundVacuumPipe as derived from a NodeGenerator. The latter is the abstract class for the factories provided by the MRMBLB library. When an object of this factory is created, the four parameters need to be provided to the constructor as indicated in the declaration. The first parameter is a name of the class for elements which factory will generate. If the name coincides with the class of element in the MAD-X file, then the factory will be used to generate a geometry of that element. The next three formal parameters following the name are used to specify parameters of the pipes to be generated in the factory: ExternalRadius,WallThickness and WallMaterial – the names are selfexplanatory. Having this declaration saved as MagRoundVacuumPipe.h, one can create the function which generates the beam line geometry in the application:

```
#include "blreg1.h"
#include "marbl.h"
#include "MagRoundVacuumPipe.h"
extern "C" {
  void tgeo_init()
  {
    TGeoMedium* AIR = gGeoManager->GetMedium("AIR");
    TGeoMedium* STST = gGeoManager->GetMedium("STST");
    Double_t Rmax=GetMarsREXT();
    Double_t Zmax=GetMarsZMAX();
    TGeoVolume* World = gGeoManager->MakeTube("World",AIR,0.,Rmax,Zmax);
    // Container for the beamline
    NodeGenerator::DefineWorld(World);
    // Radius for default cylindrical container for the beamline element
    NodeGenerator::SetRSec(Rmax);
    Double_t ExtR,WallThickness;
    // Create factory to be used inside madx function
    new MagRoundVacuumPipe("default",
                           ExtR=7.0,WallThickness=0.1,STST);
    // fname is name of the MAD-X input file
    std::string fname = "input.madx";
    madx(fname.c_str());
```

```
    gGeoManager->SetTopVolume(World);
    gGeoManager->CloseGeometry();
  }
}
```

In the given example, the ROOT geometry for the sequence of the elements defined in the MAD-X file `input.madx` – which is given as an argument to the MAD-X library function – is created. The factories of elements, declared before a call to MAD-X, are involved in the building process. In this example, the only generator for `MagRoundVacuumPipe` is used. At the time of the `tgeo_init` call, the MARS15 service functions have already converted a description of materials defined in the `MATER.INP` input file to the ROOT representation. Therefore, one can use the ROOT tools to retrieve media from the in-memory database either by name, as shown in the example above, or by index used in the `MATER.INP` file. The next action is creation of the ROOT geometry model. The top volume, `World`, in the presented example has a cylindrical shape and is filled with `AIR`. Dimensions of the volume – the radius, R_{max}, and the half length, Z_{max} – are retrieved from the MARS15 input data by means of two service functions which are components of the MARS15-ROOT interface.

The MARS15 application with the code provided above can be compiled, however, the linker will complain about the absence of the implementation of functions for the `MagRoundVacuumPipe` class. An example of its implementation is provided in Appendix A. When implementing the class, the user can assign particular steppers to the ROOT geometry objects. The stepper provided in the MARS15 library can be used here, or a stepper specific for the application can be created by the developer of the application.

For the tasks involving the long-term beam transport, a formal procedure and tools are proposed which define design of the applications with joint transport of the particles in MARS15 and in an arbitrary beam tracker. For prototyping the procedure, the MARS15 applications developed for simulations of energy deposition effects in the newly developed collimation system for the Fermilab Booster and the Recycler were used. In these applications a coupled transport with the MAD-X PTC tracker was implemented (Sec. 3.2).

3.1 MAD-X and MARS runtime interactions

The approach provided by the functions of the library consists of a concurrent execution of MAD-X and MARS15 codes where each of the participants keeps their own execution context. The developer can run the MAD-X session inside the MARS15 application, execute MAD-X commands from the MARS15 code, request a data kept in the MAD-X data structures by means of the MAD-X Application Program Interface (API).

To start a MAD-X session, the `C` function named `madx` should be called at the initialization stage. The function takes the string with name of the MAD-X input file as an argument and begins the session from execution of this file by means of the native MAD-X interpreter. It is supposed that the input file contains the following set of the MAD-X commands:

1. Beam description.
2. Definition of the sequence of elements which the beam is supposed to be propagated through.
3. Creation of the survey table for the sequence.
4. Creation of the twiss table.

The experience shows that the requirements 1 and 2 are always fulfilled in the MAD-X input file, however, often care should be taken to create tables mentioned in items 3 and 4.

By design, duration of the MAD-X session coincides with the runtime of the MARS15 executable. In particular, it means that the `STOP` statement, which resides at the very end of the MAD-X input file, needs to be removed or commented. Otherwise, the MAD-X session will be finished immediately after

execution of that statement and the tables built by the MAD-X script will be destroyed. An absence of the survey table makes it impossible the normal completion of function – the survey table taken from the in-memory data is used then for placement elements constructed by the MRMBLB. Upon successful completion of the MADX function, any MAD-X statement can be executed in an application code by means of the `madx_stmt` function. As an example, initialization of the PTC module reads as the following

```
madx_stmt("ptc_create_universe;");
madx_stmt("ptc_create_layout,model=3,method=6,nst=5,exact=true;");
PTCTrackingActive();
```

One can see that the `madx_stmt` function takes a single argument – a string filled by the MAD-X statement – which is passed for execution to MAD-X interpreter. The sequence commands like those shown above need to be executed in the source module at the initialization stage to make it possible to launch track in the PTC tracker by means of the MAD-X `ptc_start` statement. The last function – `PTCTrackingActive` – sets a global flag signaling to the MAD-X session manager that the external tracker is in use. This flag is also used for the flow control of the special stepper types which needs to be attached to the aperture volumes where one is supposed to use MAD-X. Setting the flag means also that a user shouldn't care about the execution of the closing sequence:

```
madx_stmt( "ptc_track_end; ");
madx_stmt( "ptc_end; ");
```

These functions and, therefore, the corresponding statements, will be automatically executed at the end of the MARS15 application followed by the MAD-X `STOP` command.

3.2 Interaction with external tracker

In the MRMBLB, the terms of the Open System Interconnection (OSI) reference model are used to define the tasks to be solved for establishing the interconnection between the processes, whether they are running on the same host or in a network environment. The functions are to be implemented for the following layers of the model:

Presentation layer – defines the rules (protocol) of the data conversion requested by the participants of the interaction. The functions of this layer implement the mapping of the phase space used in the MARS15 model to the one used in the PTC code and vice versa. A strict requirement is that the protocols and functions of the layers below are applied only if the mapping exists.

Session layer – defines the rules for service requests and service responses. In the current implementation, the session is always initiated by the MARS15 code. It requests the beam tracker to accept the particle with the given phase coordinates. The session duration coincides with the time required by the beam tracker to simulate the track. During the session, the beam tracker sends requests to MARS15 to check the aperture boundary crossing on the way between two phase points. The MARS15 responds with the code indicating whether the particle is lost or not on the way. In case of the boundary crossing, the phase coordinates of the lost particle are saved in the MARS15 stack for the further simulation. The session is finished when the particle leaves the geometry setup, crosses the aperture boundary or passes the predefined number of turns in a ring inside the beam aperture. In the current implementation, the MARS15 tracking code is suspended for all the session time. At the end of the session, the code kills the track and tries to find a particle in the stack to continue the simulation history.

Transport layer – defines the rules for data transfer. The data is transferred between the MAD-X PTC tracker and MARS15 as parameters of the functions. It means that the implementation of the transport level protocol is delegated to developers of the compiler used for compilation of the

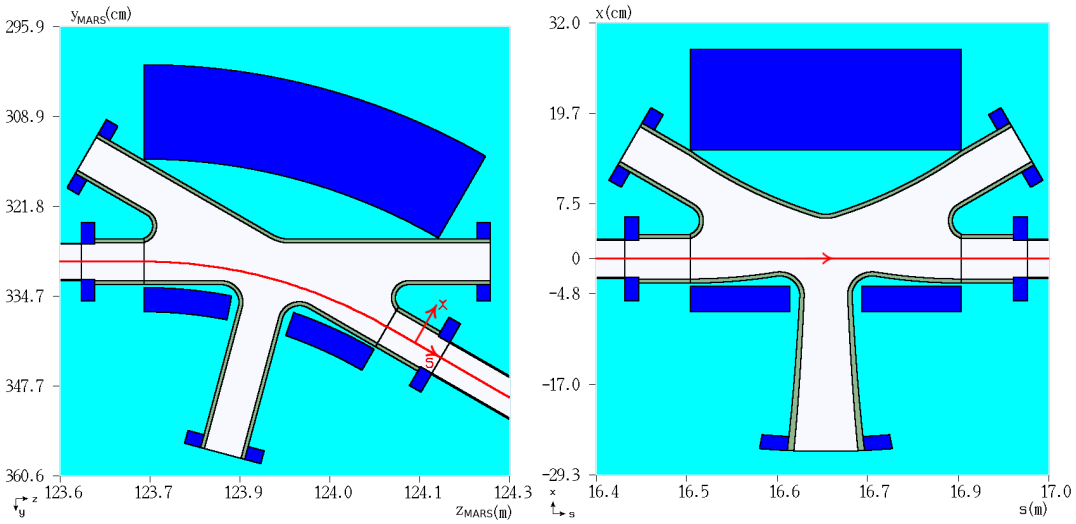

Fig. 3: Orbit-plane view of the B30 dipole region in the Fermilab Integrable Optics Test Accelerator (IOTA) ring as implemented in the MRMBLB model in the global frame (left) and in the MAD-X $\{x, y, s\}$ reference system (right). Axes of the MAD-X reference system are shown in red, arrow points a direction of the s-axis

codes. When the data needs to be passed over the network, a developer is forced to use a third party tool, like a `socket` library. Usually, this is to be avoided.

All these functionalities should be implemented for a set of dedicated volumes, i.e., those related to the beam line element apertures. The tools described in Sections 2 and Appendix A provide a straightforward way to realize this. In case of an arbitrary beam tracker, communications with the beam tracker developers might be needed. For example, the presentation layer functions can be specific for the chosen beam tracking program. It should also be taken into account that the applicability of the beam tracking code can be limited by the approximations used to create the model. As a result, not all the phase coordinates provided by the MARS15 session can be mapped to those of the beam tracker. The member-function `IsApplicable` of a stepper is the good place to address all the limitations. The mapping can be implemented in two stages. At the first stage, MARS15 global coordinates and direction of motion are transformed to the MAD-X $\{x, y, s\}$ reference system, as shown in Fig. 3. Such a kind of the direct and reverse transformations is provided by the MRMBLB library. Then, the particle coordinates and direction vector, expressed in this intermediate frame together with the particle momentum and time of flight, are converted to the phase coordinates of the particular beam tracking code.

The beam tracking session is to be initiated in the `propagate` function of the stepper class. It should be a synchronous call of a function implemented in `C`, `C++` or `FORTRAN`, which propagates the track in the beam tracker starting from the phase coordinates given as input parameters. In the prototype version with the MAD-X PTC tracker, the session is initiated by a consequent execution of the two MAD-X commands, `ptc_start` and `ptc_track` using the `madx_stmt` function described in Sec. 3.1. The PTC module is used in the "element-by-element" mode.

According to the session scenario, the next step would be to get a feedback from the beam tracker. To do that, a call to the function is to be added which takes the particle coordinates at two consequent points along the s-axis as input parameters. For the "thick" PTC tracker, it could be the points at the entrance and exit of the element. For the "thin" tracker, like the SIXTRACK or STRUCT, it could be the phase coordinates after the current and previous thin lenses. The purpose of the function is to detect the boundary crossing along the path between these points. In the implementation with the MAD-X PTC tracker, corresponding changes were made to the PTC module, `madx_ptc_track_run.f90` file of the

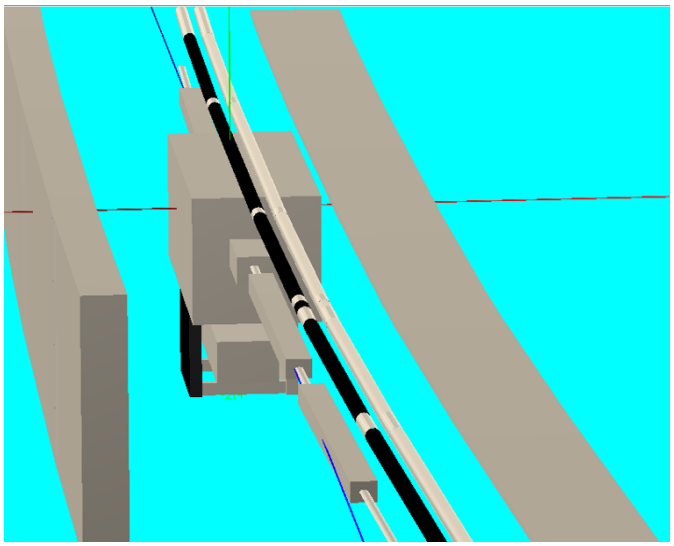

Fig. 4: 3D representation of the SCOLL613 secondary collimator

MAD-X distribution. To detect a boundary crossing, one of the trackers available in the stgeo library (See Sec. 2) is used. If the aperture boundary check is implemented in the beam tracker, then it can be used instead. Note that the aperture definition in the tracker should be used with care. As one can see in Fig. 3, the magnet aperture, which looks quite simple in the Lab system, can become non-uniform along the element, being expressed in the curvilinear design orbit reference frame. It is up to the developer of this function to decide whether to perform the boundary crossing for all elements, selected elements or elements positioned in the selected ranges. As prescribed by the session protocol, when a boundary crossing was detected, then the particle should be sent to the MARS15 stack. To do that, the functions named `stk_push` of the `libm15stack` library should be used. The return value should be agreed with developers of the beam tracking code. It needs to be defined to indicate whether the beam tracking code should kill the track or continue it. The implementation was done for the MAD-X PTC module. The user can implement an interface with the tracker specific for his/her application using the procedure described.

The scheme doesn't assume that the beam tracker provides any feature for parallel execution of the code. The MPI parallelism is implemented at the level of simulation of the MARS events. This was successfully tested on the ALCF THETA cluster for jobs of tens of thousand ranks.

4 Use cases

4.1 The Fermilab Recycler

The Fermilab Recycler is an 8-GeV storage ring used in the accelerator complex to provide an intense 120 GeV proton beam to the neutrino experiments. The Recycler resides in the same tunnel as the Fermilab Main Injector (MI) – accelerator which ramps proton beam energy from 8 GeV to 120 GeV. To obey the needs of the experiments, there is a plan to increase – in the nearby future – the 120-GeV beam power from 400-700 kW up to about 1 MW.

To keep beam losses and activation in the critical locations at the current or lower levels, the new multi-stage collimation system was designed and installed. The problem was that the collimator positions were limited longitudinally by the spaces available in the lattice and laterally by the ceiling and the tunnel wall. The place "613-616" was found in the tunnel at the junction with the NuMI tunnel. A detailed `MRMBLB` 3D model was built for the collimation region, some fragments are shown in Fig. 4 and Fig. 5.

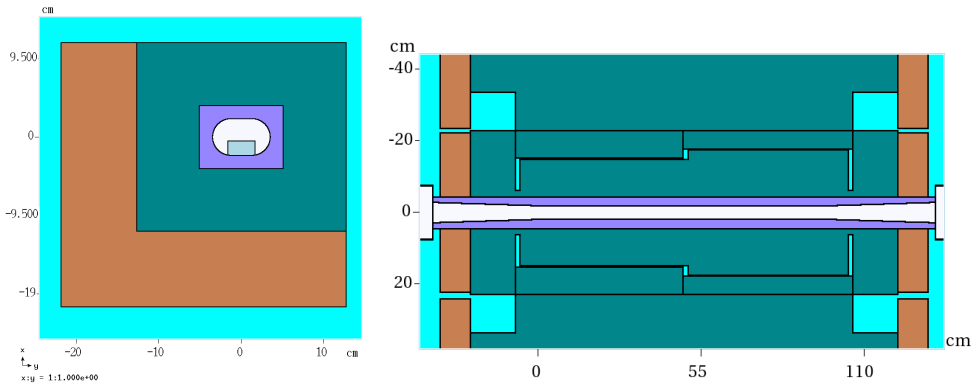

Fig. 5: Transverse cross sections of the primary collimator assembly (left) and the secondary 20-ton collimator (right)

Results of the multi-turn tracking and full MARS15 simulations were used for optimization of the collimation efficiency, the primary, secondary and tertiary (mask) collimator designs, with the heat load, cooling, activation and radiation shielding issues addressed on top of these. The fraction of the halo particles absorbed by secondary collimators after the passage through primary collimator are 24% (SCOLL613) and 63% (SCOLL616). These numbers are mostly due to a single pass of the beam halo protons through a primary collimator.

When passing through the primary collimator, beam protons undergo multiple scattering resulting in random energy loss and change their the direction of motion. Due to random energy-angular kicks caused by the primary collimator, proton path-lengths inside aperture are also random. Fig. 6 shows that most often protons are absorbed after four turns and it takes less than 20 turns for most protons to be lost in the lattice.

Calculated prompt dose isocontours in the collimation region are shown in Fig. 7. It was assumed here that 2 kW is delivered by particles of the beam halo onto the upstream boundary of the primary collimator made of TZM and having thickness $125\mu m$. The collimator is positioned at the boundary of $3\,\sigma$ beam envelope. Jaws of two secondary collimators are aligned along the $3.5\,\sigma$ boundary.

4.2 ILC

It has been recognized that the loss of dark current (DC) generated in the superconducting RF (SRF) cavities is the dominant source of radiation loads on the International e^+e^- Linear Collider (ILC) Main Linac components and related radiation environment in the tunnel. The DC-induced loads are several orders of magnitude higher than those induced by beam-gas Coulomb scattering and beam-gas bremsstrahlung. The DC production and handling algorithm in a detailed 3D MARS15 model, results of the beam loss, radiation load and dose calculations along with possible mitigation are described in Ref. [17]. Figure 8 (left) shows a fragment of the MARS15 3D model of a 9-cell SRF cavity.

A dark current (DC) formula reads

$$J = a\,\phi^{-1}\,F^2[-\frac{\nu(f)\,b\,\phi^{3/2}}{F}] \tag{1}$$

where J (A/m^2) is DC density, $a \approx 1.541434 \times 10^{-6} (A\,eV\,V^{-2})$, $F = E\,\beta$, $E\,(\frac{V}{nm})$ – time dependent electric field in the immediate vicinity of the cavity surface, $\beta = 100$ is the field enhancement factor, $\phi = 4.2(eV)$ is the local work function, $f = 1.439964\,(eV^2\,V^{-1}\,nm)\,F/\phi^2$, $\nu(f) \approx 1 - f + (1/6)\,f\,\ln f$, $b \approx 6.830890\ eV^{-3/2}\,V\,nm^{-1}$.

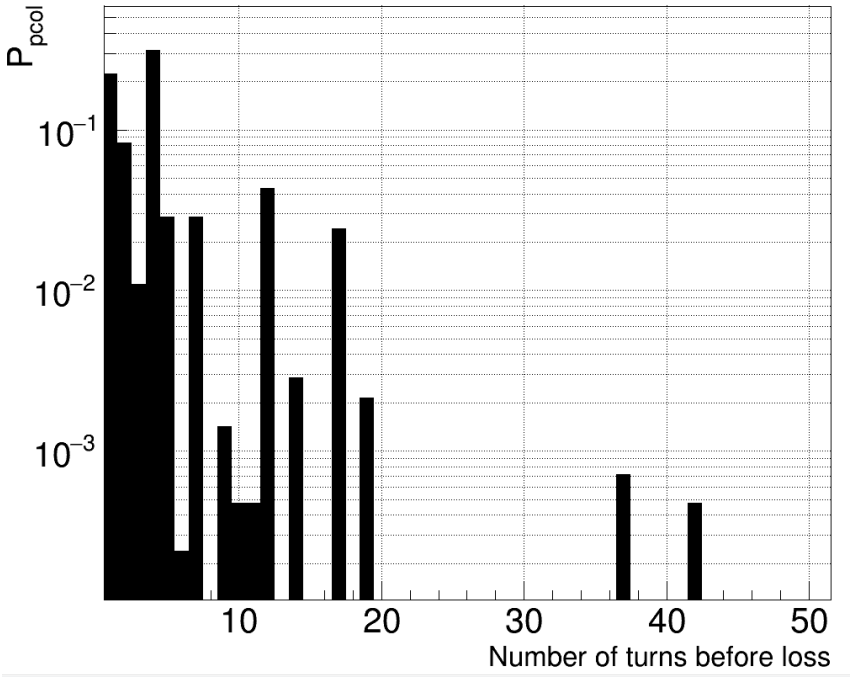

Fig. 6: Probability to be lost for the beam halo protons passed through the primary collimator as a function of the number of turns in the Recycler

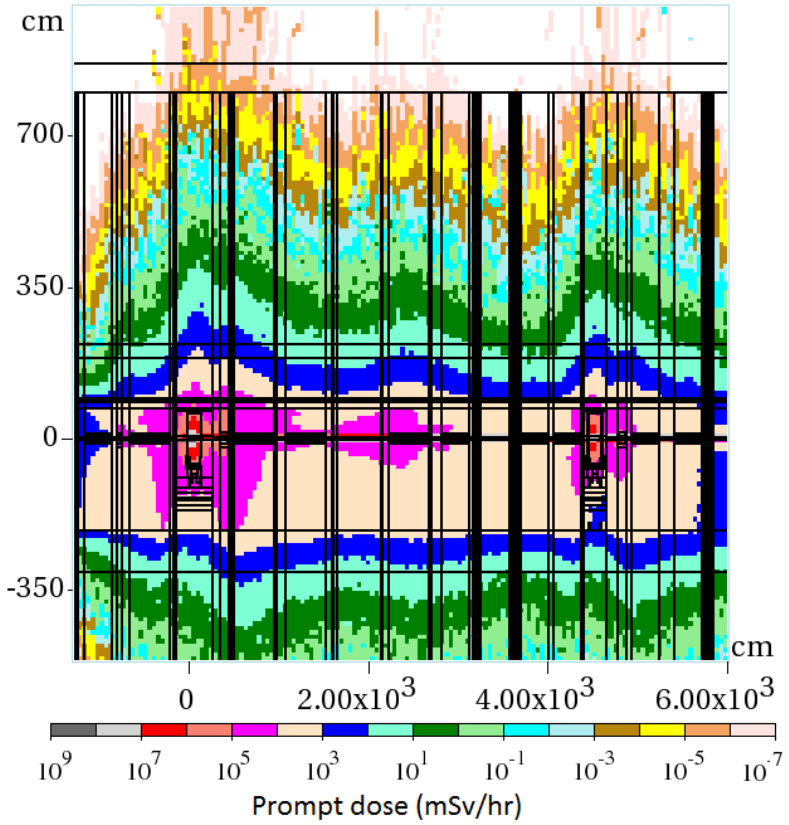

Fig. 7: Prompt dose distribution in the Recyler collimation region

Several DC electron tracks generated in the model are shown in Fig. 8 (right).

Analysis has shown that, in the case when field emission is described by Eq. (1) and emission points are distributed uniformly over the cavity surface, the forward DC from the cavity № 30 toward quadrupole 2 does not exceed 2.1 nA (see Fig. 9), with the average current over the 30 cavities being approximately 1 nA.

However, if a cavity with significant surface imperfections is placed at the very beginning of such a beam line segment, DC in the last cavity of the segment can exceed the assumed level of $50nA$ due to the significant amplification shown in Fig. 9.

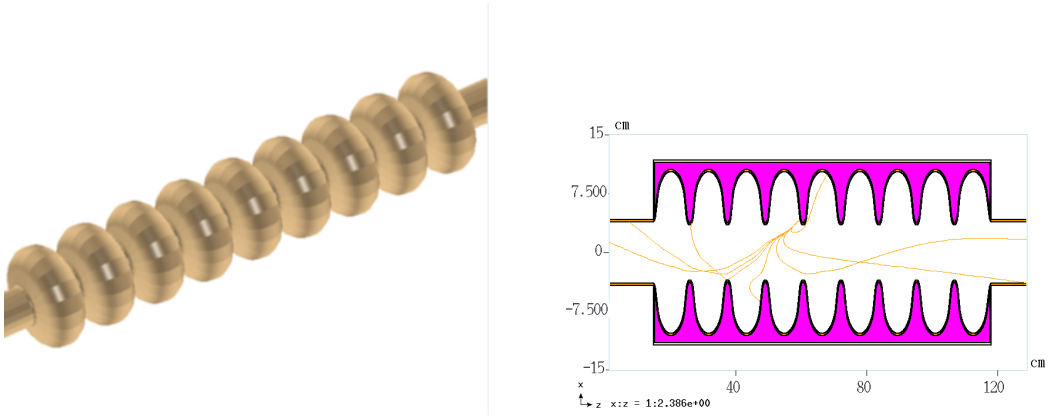

Fig. 8: MARS15 3D model of 9-cell SRF cavity (left) and sample DC electron trajectories in an SRF cavity (right)

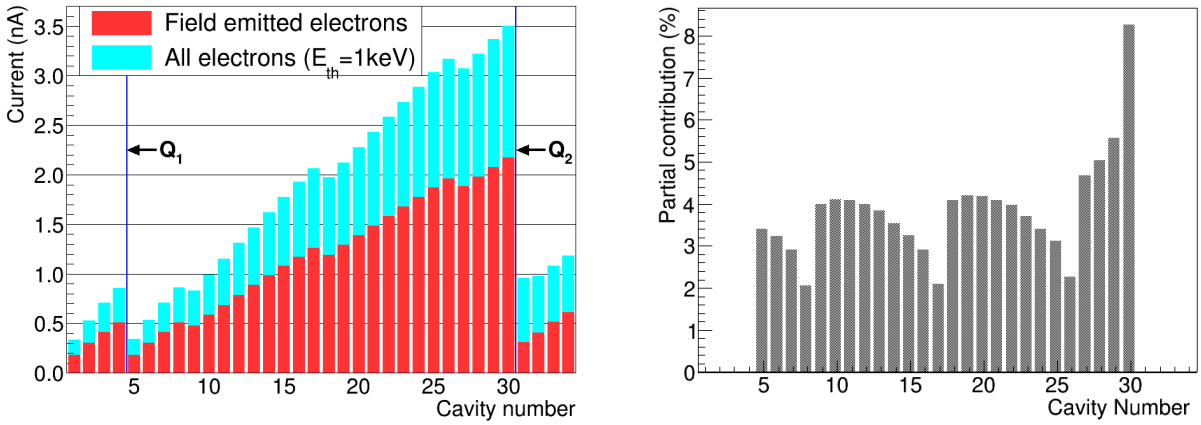

Fig. 9: Left: Current (nA) of accelerated DC electrons in the aperture at the downstream end of each cavity. The results include both field emitted electrons which did not experience any interaction with cavity walls and electrons that already experienced either elastic or inelastic scattering in cavity walls. The labels Q1 and Q2 indicate locations of quadrupole magnet 1 and 2, respectively. Right: Partial contribution of electrons emitted from the upstream cavities to the forward current at downstream end of cavity № 30

Appendices

A Node generator implementation

The implementation for the MagRoundVacuumPipe class according to declaration presented in Sec. 3 is given below:

```
#include <sstream>
#include <TGeoManager.h>
#include <TGeoTube.h>
#include "MagRoundVacuumPipe.h"
#include "TMADX_BField.h"
#include "MADXStepper.h"

MagRoundVacuumPipe::MagRoundVacuumPipe(TString name,
                                       Double_t ExternalRadius,
                                       Double_t thick,
                                       TGeoMedium* WallMedium):
    NodeGenerator(name),
    wall_thickness(thick),
    R(ExternalRadius),
    STST(WallMedium),
    CopyNo(0)
{
  VAC  = gGeoManager->GetMedium("BVAC");
}
// Creation of element
TGeoVolume* MagRoundVacuumPipe::Assemble(
           const std::vector<Section*>::iterator& ElInfo, // Information about current ele
           TGeoVolume* Sec, // input, container volume for the element
           TGeoMatrix* tr)  // input, matrix of element, useful in field definitions
{
 CopyNo++; // Copy number
 TGeoShape* SecShape=Sec->GetShape();
  Double_t nl[3]={0.,0.,-1.};
  Double_t nh[3]={0.,0.,1.};
  Double_t SecHalfL; // Half length of the element
  bool IsCtubSection=false;
  // Determine shape of a container section
  if(SecShape->IsA() == TGeoTube::Class()) {
    // Cylinder with parallel faces
     SecHalfL=((TGeoTube*) SecShape)->GetDz();
  } else if(SecShape->IsA() == TGeoCtub::Class()) {
    // Faces are not parallel to each other
    IsCtubSection=true;
    SecHalfL=((TGeoCtub*) SecShape)->GetDz();
    // Normal vecor for upstream and dowstream faces
    const Double_t* n1=((TGeoCtub*) SecShape)->GetNhigh();
    const Double_t* n2=((TGeoCtub*) SecShape)->GetNlow();
    memcpy(nl,n2,3*sizeof(Double_t));
    memcpy(nh,n1,3*sizeof(Double_t));
  }

  TMADX_BField* FieldInAperture=0;
  // Check precence of field
  if(((*ElInfo)->K).size()) {
    // Element has a magnetic field,
    // create mag.field object to be attached to the aperture:
```

```
    FieldInAperture=new TMADX_BField(ElInfo);
  }
  // Outer steel
  std::ostringstream ostr;
  ostr << (*ElInfo)->nm  << CopyNo;
  TGeoVolume* STube;
  if(IsCtubSection) {
    STube  = gGeoManager->MakeCtub(ostr.str().c_str(),STST,
                                   0.L, R,SecHalfL,
                                   0.L,360.L,
                                   nl[0],nl[1],nl[2],
                                   nh[0],nh[1],nh[2]);
  } else {
    STube  = gGeoManager->MakeTube(ostr.str().c_str(), STST,
                                   0.0L, R, SecHalfL);
  }
  // Inner vacuum
  ostr.str("");
  ostr << (*ElInfo)->nm << "RAP" << CopyNo;
  TGeoVolume* BeamAperture;
  if(IsCtubSection) {
    BeamAperture = gGeoManager->MakeCtub(ostr.str().c_str(),VAC,
                                         0.L, R-wall_thickness,SecHalfL,
                                         0.L,360.L,
                                         nl[0],nl[1],nl[2],
                                         nh[0],nh[1],nh[2]);
  } else {
    BeamAperture = gGeoManager->MakeTube(ostr.str().c_str(), VAC,
                                         0., R-wall_thickness,
                                                  SecHalfL);
  }
  // Associate field with volume
  BeamAperture->SetField(FieldInAperture);
  // Create stepper:
  MADXStepper* Stepper = new MADXStepper(ElInfo);
  // Attach stepper to the volume:
  BeamAperture->SetUserExtension(Stepper);
  STube->AddNode(BeamAperture,1);
  return STube;
}
```

Acknowledgements

The authors are grateful to I. Rakhno for collaboration and useful comments.

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. DEAC02-07CH11359.

This research used a computing award allocation from the ASCR Leadership Computing Challenge 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, “Beam Collimation at Hadron Colliders”, ICFA Workshop on Beam Halo Dynamics, Diagnostics and Collimation (HALO-03), Montauk, NY, May 2003, Ed. J. Wei, W. Fischer and

P. Manning; AIP Conf. Proc., Vol. 693, pp. 14-19.

[2] N.V. Mokhov *et al.*, "Tevatron Beam Halo Collimation System: Design, Operational Experience and New Methods", JINST, **6** (2011) T08005.

[3] I.S. Baishev, A.I. Drozhdin, and N.V. Mokhov, "STRUCT Program User's Reference Manual", Technical Report SSCL-MAN-0034, SSC Laboratory, 1994.

[4] http://sixtrack.web.cern.ch/SixTrack/

[5] G. Robert-Demolaize *et al.*, "A New Version of SixTrack with Collimation and Aperture Interface", Proc. of PAC2005, Knoxville, TN, USA (2005) pp. 4084-4086.

[6] A. Ferrari, P.R. Sala, A. Fasso and J. Ranft, "FLUKA: A Multi-Particle Transport Code", CERN-2005-010 (2005). http://www.fluka.org/fluka.php/

[7] N.V. Mokhov, "The MARS Code System User's Guide", Fermilab-FN-628 (1995), https://mars.fnal.gov

[8] N.V. Mokhov, P. Aarnio, Yu. Eidelman, K. Gudima, A. Konobeev, V. Pronskikh, I. Rakhno, S. Striganov, and I. Tropin, "MARS15 Code Developments Driven by the Intensity Frontier Needs", Prog. Nucl. Sci. Technol., **4**, pp. 496-501 (2014).

[9] L. Nevay *et al.*, Proc. IPAC 2014, Dresden, p. 182 (2014)

[10] J. Allison *et al.*, Nucl. Instr. Methods, **A835** (2016) pp. 186-225, https://geant4.cern.ch

[11] D.N. Mokhov, O.E. Krivosheev, E. McCrory, L. Michelotti and J.F. Ostiguy "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, "A Lex-based MAD Parser and its Applications", in *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, "An Improved MAD-MARS Beam Line Builder: User's Guide," FERMILAB-FN-0738.

[14] MAD - Methodical Accelerator Design, CERN - BE/ABP Accelerator Beam Physics Group. http://madx.web.cern.ch/madx/

[15] A. Mereghetti, F. Cerutti, R. De Maria *et al.* "SixTrack-Fluka Active Coupling for the Upgrade of the SPS Scrapers", Proc. 4th International Particle Accelerator Conference (IPAC 2013): Shanghai, China, May 12-17, 2013, http://JACoW.org/IPAC2013/papers/wepea064.pdf

[16] R. Brun and F. Rademakers, "ROOT - An Object Oriented Data Analysis Framework", Proceedings AIHENP`96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. http://root.cern.ch/

[17] N.V. Mokhov, I.L. Rakhno, I.S. Tropin, "Modeling Radiation Loads in the ILC Main Linac and a Novel Approach to Treat Dark Current", Fermilab-FN-1035-APC (2017).