Fermilab FERMILAB-Pub-03/133-E may 2003

Beam Modeling Tools for GEANT4 (and Neutrino Source Applications)

V. Daniel Elvira [a], P. Lebrun [a], P. Spentzouris [a]

[a] *Fermi National Accelerator Laboratory, P. O. Box 500, Batavia, IL 60510-500*

Abstract

Geant4 is a tool kit developed by a collaboration of physicists and computer professionals in the High Energy Physics field for modeling of the passage of particles through matter [1]. The motivation for the development of the Beam Tools is to extend the Geant4 applications to accelerator physics. Although there are computer programs for beam physics simulations [2–4], Geant4 is ideal to model a beam going through material or a system with a beam line integrated to a complex detector. There are many examples in the current international High Energy Physics programs, such as studies related to a future Neutrino Factory, a Linear Collider, and a very Large Hadron Collider.

Key words: C++, object oriented, geant4, accelerator, beam, simulation, neutrino, muon, cooling
PACS: 29.27.-a, 29.27.Eg, 13.15.+g

1 Introduction

Geant4 is a tool kit developed by a collaboration of physicists and computer professionals in the High Energy Physics (HEP) field for modeling of the passage of particles through matter [1]. The motivation for the development of the Beam Tools is to extend the Geant4 applications to accelerator physics. The Beam Tools are a set of C++ classes designed to facilitate the simulation of accelerator elements: r.f. cavities, magnets, absorbers. These elements are constructed from Geant4 geometry objects or *solid volumes*: boxes, tubes (cylinders), trapezoids, or spheres.

A variety of visualization packages are available within the Geant4 framework to produce an image of the simulated apparatus. The pictures shown in this

article were created with Open Inventor [5], which allows direct manipulation of the objects on the screen, plus perspective rendering via the use of light.

There are computer programs for beam physics simulations, but Geant4 is ideal to model a beam through a material or to integrate a beam line with a complex detector. There are applications where such modeling is essential. For instance, a critical part of the R&D associated with the Neutrino Source/Muon Collider accelerator is the ionization cooling channel, which is a section of the system aimed to reduce the size of the muon beam in phase space. The MuCool/MICE (muon cooling) experiments need accurate simulations of the beam transport through the cooling channel in addition to a detailed simulation of the detectors designed to measure the size of the beam. The accuracy of the models for physics processes associated with muon ionization and multiple scattering is critical in this type of applications. Another example is the simulation of the interaction region in future accelerators. The high luminosity and background environments expected in the Next Linear Collider (NLC) [6] and the Very Large Hadron Collider (VLHC) [7] pose great demand on the detectors, which may be optimized by means of an integrated simulation of the accelerator and detectors around the interaction region.

2 Geant4 Basics Relevant to the Beam Tools

Geant4 is an object oriented package developed in C++ by a collaboration of physicists and computer professionals for modeling of the passage of particles through matter. Geant4 applications include, but are no longer limited to, High Energy Physics. In addition to simulations of the new generation of HEP experiments, like the ATLAS and CMS detectors under construction at the CERN (European Organization for Nuclear Research) Large Hadron Collider (LHC), Geant4 is also used in other fields of research. One example is the study of radiation levels inside the international space station. In medical physics, Geant4 is used to help optimize the location and dose of radiation in the treatment of tumors.

As a tool kit, Geant4 provides a set of libraries, a `main` function, and a family of initialization and action classes to be implemented by the user. The names of the Geant4 library classes start with the prefix `G4`. The core of `main` starts with the construction of a `runManager`. This object of type `G4RunManager*` controls the flow of the program and manages the event loop within a run. The user initialization and action classes are singlets and their associated objects are constructed in `main`. The example described in this section is called `MuCool` and uses only some of the many available user classes. A section of the implementation of `main` in our example is shown below:

```
// set initialization classes

MuCoolConstruct *detector = new MuCoolConstruct();
runManager->SetUserInitialization(detector);
MuCoolPhysicsList *physList = new MuCoolPhysicsList();
runManager->SetUserInitialization(physList);

// set mandatory user action class

runManager->SetUserAction(new MuCoolPrimaryGeneratorAction);
MuCoolSteppingAction* stepAct = new MuCoolSteppingAction;
runManager->SetUserAction(stepAct);
runManager->SetUserAction(new MuCoolTrackingAction);
runManager->SetUserAction(new MuCoolEventAction);
```

These user class objects contain the information related to the geometry of the apparatus, the electromagnetic fields, the beam, and actions taken by the user at different times during the simulation.

2.1 Detector and Field Construction

The detector and electromagnetic field geometry, properties, and location are implemented in the constructor and methods of the `MuCoolConstruct` user class, which inherits from `G4VUserDetectorConstruction`. In the `Construct()` method, the user does the initialization of the electromagnetic field and the equation of motion. First, there is a variety of Runge-Kutta steppers to select from, to perform the integration to different levels of accuracy. Next comes the detector description, which involves the construction of solid, logical, and physical volume objects. These objects contain information about the detector geometry, properties, and position, respectively. Many solid types, or shapes, are available. For example, cubic (box) or cylindric shapes (tube), are constructed as:

```
G4Box(const G4String& pName, G4double pX, G4double pY,
      G4double pZ);
G4Tubs(const G4String& pName, G4double pRMin, G4double pRMax,
       G4double pDz, G4double pSPhi, G4double pDPhi);
```

where a name and half side lengths are provided for the box. Inner and outer radii, half length, and azimuthal coverage are the arguments of a cylinder (tube). A logical volume is constructed from a pointer to a solid, and a given material:

```
G4LogicalVolume(G4VSolid* pSolid, G4Material* pMaterial,
```

```
                const G4String& name)
```

The physical volume, or placed version of the detector is constructed as:

```
G4PVPlacement(G4RotationMatrix *pRot, const G4ThreeVector &tlate,
              const G4String& pName, G4LogicalVolume *pLogical,
              G4VPhysicalVolume *pMother, G4bool pMany,
              G4int pCopyNo);
```

where the rotation and translation are performed with respect to the center of its "mother" volume (container). Pointers to the associated logical volume, and the copy number complete the list of arguments.

2.2 Physics Processes

Geant4 allows the user to select among a variety of physics processes which may occur during the interaction of the incident particles with the material of the simulated apparatus. There are electromagnetic, hadronic, and other interactions available like: "electromagnetic", "hadronic", "transportation", "decay", "optical", "photolepton_hadron", "parameterisation". The information on physics processes is contained in the `MuCoolPhysicsList *physList` object. The different types of particles and processes are created in the constructor and methods of the `MuCoolPhysicsList` user class, which inherits from `G4VUserPhysicsList`.

2.3 Incident Particles

The user constructs incident particles, interaction vertices, or a beam by typing code in the constructor and methods of the `MuCoolPrimaryGeneratorAction` user class, which inherits from `G4VUserPrimaryGeneratorAction`.

2.4 Stepping Actions

The `MuCoolSteppingAction` user action class inherits from `G4UserSteppingAction`. It allows to perform actions at the end of each step during the integration of the equation of motion. Actions may include killing a particle under certain conditions, retrieving information for diagnostics, and others.

2.5 Tracking Actions

The `MuCoolTrackingAction` user action class inherits from `G4UserTrackingAction`. For example, particles may be killed here based on their dynamic or kinematic properties.

2.6 Event Actions

The `MuCoolEventAction` user action class inherits from `G4UserEventAction`. It includes actions performed at the beginning or the end of an event, that is immediately before or after a particle is processed through the simulated apparatus.

3 Description of the Beam Tools Classes

A particle accelerator consists of magnets and radio frequency cavities which are responsible for manipulating the particle beam. The magnets generate the forces necessary to focus and drive the beam through the accelerator complex. The radio frequency (r.f.) cavities provide the electric field for acceleration of the charged particles. In order to track the beam particles, we need an accurate description of the electromagnetic fields associated with the accelerator components. This Section is devoted to explain how to model these components using the Beam Tools. Although most of the classes available in the current version of the Beam Tools are associated with components of a Neutrino Source/Muon Collider machine, the object oriented design allows for easy extensibility to any accelerator system within and outside the field of high energy physics. Brief descriptions of each class and constructor are included below.

- The `BTSheet` class inherits from `G4MagneticField`. The class objects are field maps produced by an infinitesimally thin solenoidal current sheet. The class data members are all the parameters necessary to generate analytically a magnetic field in r-z space (there is φ symmetry). No geometric volumes or materials are associated with the `BTSheet` objects. `GetFieldValue` is a concrete method of `BTSheet` inherited from `G4Field`, through `G4MagneticField`. It returns the field value at a given point in space and time.
- The `BTSolenoid` class inherits from `G4MagneticField`. The class objects are field maps in the form of a grid in r-z space, which are generated by a set of `BTSheet`. The sheets and the `BTSpline1D` objects, containing the

spline fits of B_z and B_r versus z for each r in the field grid, are data members of `BTSolenoid`. No geometric volumes or materials are associated with `BTSolenoid`. The field at a point in space and time is accessed through a `GetFieldValue` method, which performs a linear interpolation in r of the spline fit objects.

- The `BTSolenoidLogicVol` class defines the material and physical size of the coil system which is represented by the set of current sheets. A `BTSolenoid` must first be constructed from a list of current `BTSheets`. The `BTSolenoid` object is a data member of `BTSolenoidLogicVol`. The `BTSolenoidLogicVol` class constructor creates `G4Tubs` solid volumes and associated logical volumes for the coil system, the shielding, and the empty cylindric regions inside them. Only the logical volumes are constructed here. No physical placement of a magnet object is done.
- The `BTSolenoidPhysVol` class is the placed version of the `BTSolenoidLogicVol`. It contains the associated `BTSolenoid` object as a data member, as well as the pointers to the physical volumes of its logical constituents.

The process of simulating a solenoid starts with the construction of a set of current sheets `BTSheet`, which will generate the field. The constructor of this class:

```
BTSheet(G4ThreeVector location, G4int id, G4int type,
        G4double thick, G4double rad, G4double len, G4double cur)
```

takes as input the sheet location, radius, length, and current. The other input parameters are not used for the moment. The `BTSheet` objects must be assigned to a vector of current sheets, `vector<BTSheet>`, which is latter provided to the `BTSolenoid` constructor:

```
BTSolenoid(G4double minrxy, G4double maxrxy, G4int numptrxy,
           G4double minz, G4double maxz, G4int numptz,
           vector<BTSheet> const &vsheets);
```

where `minrxy`, `maxrxy`, `minz`, `maxz` are the boundaries of the r-z grid, and `numptrxy`, `numptz` the number of r and z nodes respectively. The field is stored as a grid in data member arrays and spline fits. The more nodes in z, the better the accuracy of the spline fit; the more nodes in r, the better the accuracy of the interpolation. To ensure good accuracy, the field map should extend well beyond the physical limits of the magnet, since the field at a given point in space is the sum of contributions from all magnets in the accelerator lattice.

The next step is to construct the logical volume of a solenoid, that is a concrete coil system associated with the field. The `BTSolenoidLogicVol` constructor:

```
BTSolenoidLogicVol(BTSolenoid *theSol, G4Material *matCoils,
                   G4Material *shield, G4Material *vacuum,
                   G4String name, G4double LengthExtra,
                   G4double radExtra, G4double shieldThickness,
                   G4bool topIsCylinder, G4bool volumeSheet,
                   G4double solmaxstep);
```

takes the `BTSolenoid`, the material and dimensions of the coil system and shielding structure, and two `G4bool` arguments to either build the system as a ring or a solid cylinder (`topIsCylinder`), and make the sheets visible or not (`volumeSheet`). The `solmaxstep` argument is the maximum step size imposed by the user for integration of the equation of motion in the solenoid logical volume. Figure 1 shows a solenoidal copper coil system modeled with four infinitesimally thin sheets equally spaced in radius.

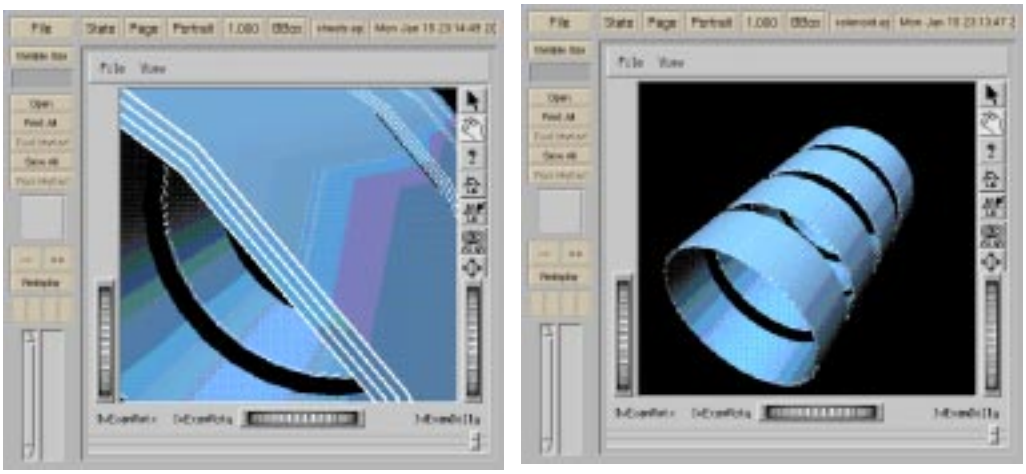

Fig. 1. Left: a solenoidal copper coil system modeled with four infinitesimally thin sheets equally spaced in radius. Right: array of four solenoids separated by gaps.

The last step in the construction process is to position the solenoid. The `BTSolenoidPhysVol` constructor takes the position of the object center with respect to its mother volume, a rotation around that center, and pointers to the associated logical volume, the mother's physical volume, and the global magnetic field object, `BTGlobalMagField *theField`:

```
BTSolenoidPhysVol(G4RotationMatrix *pRot,
                  const G4ThreeVector &position,
                  BTSolenoidLogicVol *theSolLV,
                  G4VPhysicalVolume *pMother,
                  G4bool pMany, G4int pCopyNo,
                  G4double reScaleFieldFactor,
                  BTGlobalMagField *theField);
```

A copy number, and a factor to scale the field value should also be provided. Figure 1 also shows a configuration of four solenoids separated by gaps.

3.1 Magnetic Field Maps

The Beam Tools also allow to simulate generic field maps using the `BTMagFieldMap` and `BTMagFieldMapPlacement` classes.

- The `BTMagFieldMap` class inherits from `G4MagneticField`. The constructor reads the map information from an `ASCII` file containing the value of the field at a set of nodes of a grid. No geometric objects are associated with the field. The field at a point in space and time is accessed through a `GetFieldValue` method, as in the case of the solenoid.
- The `BTMagFieldMapPlacement` class is a placed `BTMagFieldMap` object. Only the field is placed because there is no coil or support system associated with it.

The `BTMagFieldMap` object may be of two different `typeofmap`: "HardEdge" or "Interpolated". In the first case, the field is constant in between (r,z) nodes, taking the B_r and B_z value at the lower edge of the interval. In the second case, the field is evaluated in between nodes using a linear interpolation. The "HardEdge" option is useful, for example, to create square fields. The "Interpolated" option is aimed to reproduce accurately an arbitrary field. The `BTMagFieldMap` constructor requires arguments defining the map boundaries and grid resolution.

```
BTMagFieldMap(const char MBname[20], const std::string typeofmap,
              G4double zoff, G4double zlgth, G4double rlgth,
              G4int numMBnodesZ, G4int numMBnodesR);
```

The magnetic field map is placed by calling `BTMagFieldMapPlacement` constructor:

```
BTMagFieldMapPlacement(const G4RotationMatrix *rot,
                       const G4ThreeVector &position,
                       BTMagFieldMap *themap,
                       G4double rescaleFieldFactor,
                       BTGlobalMagField *theField);
```

which takes the global position of the field geometric center, a rotation, a pointer to the `BTMagFieldMap` which is being placed, the field scaling factor, and the global magnetic field object.

Radio frequency (r.f.) cavities are the elements providing the electric field for particle acceleration. This section explains how to simulate realistic r.f. systems using an r.f. device called Pill Box cavity. A description of a Pill Box cavity and other accelerator beam line elements can be found in the literature, for example Ref. [8]. But briefly, it may be described as an empty cylindric cavity with conductor walls. Ideally, it should be closed at the ends. Real cavities, however, deviate from the cylindric shape and have holes at the end to allow the passage of the beam. This "iris" may be closed with thin windows of a light material to improve the quality of the field without a significant degradation of the beam. The time dependent electric field inside the cavity is a sinusoidal wave of frequency ν generated by a voltage amplitud V_p.

The Beam Tools package provides the classes: `BTAccelDevice`, `BTPillBox`, `BTrfCavityLogicVol`, `BTrfWindowLogicVol`, and `BTLinacPhysVol`.

- `BTAccelDevice.hh` class is abstract. All accelerator device classes are derived from this class, which inherits from `G4ElectroMagneticField`.
- The `BTPillBox` class inherits from `BTAccelDevice` and represents single Pill Box field objects. An object of this class is a field with no associated solid. The time dependent electric field is computed using a simple Bessel function. An important feature is the reference particle mode, used to tune the phase of the r.f. wave to be synchronized with the passage of the beam (synchronous phase). In reference mode, the field is static (does not depend on time), and only an approximation to the ideal Pill Box field, as will be explained in Sec. 3.4. At a point in space and time, the field is accessed through a `GetFieldValue` method, and given by:

$$E_z = V_p \, J_0 \left(\frac{2\pi\nu}{c} r \right) \sin(\phi_s + 2\pi\nu t) \tag{1}$$

$$B_\varphi = \frac{V_p}{c} J_1 \left(\frac{2\pi\nu}{c} r \right) \cos(\phi_s + 2\pi\nu t) \tag{2}$$

  where V_p is the cavity peak voltage, ν the wave frequency, ϕ_s the synchronous phase, and $J_{0,1}$ the Bessel functions evaluated at $\left(\frac{2\pi\nu}{c} r\right)$. The radius of the cavity, R, is derived from:

$$\frac{V_p}{c} J_1 \left(\frac{2\pi\nu}{c} r \right) = 0 \quad \text{at} \quad r = R \tag{3}$$

$$\left(\frac{2\pi\nu}{c} R \right) = 2.405 \;\Rightarrow\; R = \frac{2.405\, c}{2\pi\nu} \tag{4}$$

- The `BTrfMap` class also inherits from `BTAccelDevice`. The class objects are electromagnetic field maps which represent an r.f. cavity. In this way, complex r.f. fields can be measured or generated and later included in the

simulation. The field map, in the form of a grid, is read in the `BTrfMap` constructor from an `ASCII` file. The `BTrfMap` object is a field, with no associated solid. A `GetFieldValue` method returns the field value at a point in space and time by means of a linear interpolation of the field grid.

- The `BTrfCavityLogicVol` class constructor creates solids and logical volumes associated with the r.f. field classes. In the case of a map, a vacuum cylinder ring represents its limits. In addition to geometric and material parameters of the cavity, the class contains field and accelerator device information.
- The `BTrfWindowLogicVol` class is used with `BTCavityLogicVol` to create the geometry and logical volume of r.f. cavity windows, including the support structure, which may be placed to close the cavity iris at the end caps.
- The `BTLinacPhysVol` class is a placed linac object. A linac, or linear accelerator, is a set of contiguous r.f. cavities, including the field, the support and conductor material, and windows. The `BTLinacPhysVol` constructor is overloaded. One version places a linac of Pill Box cavities and the other places field maps.

The inputs to the `BTPillBox` constructor are the cavity frequency, length, maximum gradient, and synchronous phase:

```
BTPillBox(G4double freqIn, G4double zLIn, G4double dzSkinIn,
          G4double eMaxGradIn, G4double phaseAccIn );
```

while the `BTrfMap` constructor needs, in addition, the boundaries of the map and the grid resolution.

```
BTrfMap(const char rfname[20], G4double freqIn, G4double zoff,
        G4double zPhaseIn, G4double zLIn, G4double rMax,
        G4double rEff,  G4double phaseAccIn, G4int numRFnodesZ,
        G4int numRFnodesR );
```

For a Pill Box cavity, the `BTrfCavityLogicVol` constructor needs a pointer to the associated `BTPillBox` object, the cavity walls material, the material filling the cavity, the logical volume name, the extra length necessary to accommodate the support structure, the wall thickness, and the maximum step size in the volume:

```
BTrfCavityLogicVol(BTPillBox *rfPillBox, G4Material *matCavity,
                   G4Material  *vacuum, G4String name,
                   G4double dLengthExtra, G4double wallThick,
                   G4double rfmaxstep);
```

For a field map, `BTrfCavityLogicVol` builds a cylindric tube bounded by a cylindric ring (wall) with the same geometric disposition as the Pill Box conductor ring, except that it is made of vacuum. Although this structure may

have nothing to do with the geometry of the real cavity which produced the field, it allows the user to visualize the boundaries of the r.f. field. In addition, it provides a dummy software structure to attach windows, if necessary. The `BTrfCavityLogicVol` constructor is, therefore:

```
BTrfCavityLogicVol(BTrfMap *rfmap, G4Material *matCavity,
                   G4Material  *vacuum, G4String name,
                   G4double dLengthExtra, G4double wallThick,
                   G4double rfmaxstep);
```

As a rule, the logical volume name must contain the "RF" string. This is a requirement for the automatic phase tuning to function in reference particle mode. The window solids and logical volumes are created by the `BTrfWindowLogicVol` constructor:

```
BTrfWindowLogicVol(double radius, double radiusOut,
                   double  thickness, double  rimThickness,
                   G4Material *matWindow, G4Material *vacuum,
                   G4String name);
```

The arguments are the window radius, the outer radius of the rim (window support structure), the window thickness, the rim thickness, the window material, the material filling the top volume which contains the window structure (vacuum), and the name of that volume. A step window with a center circle thinner than the outer ring, may be built by adding an outer ring to the flat window:

```
void AddOuterFoil(double rIn, double thickness, G4Material *mat);
```

The first argument is the ring window inner radius, the second its thickness, and the last its material.

The physical placement of the r.f. system is done by the `BTLinacPhysVol` constructor:

```
BTLinacPhysVol(G4int numCell, G4double *zLocCells,
               BTrfCavityLogicVol *rfCell,
               BTrfWindowLogicVol *rfWindow,
               G4VPhysicalVolume *pMother,
               BTGlobalEMField *theField);
```

where `zLocCells` is the array which contains the z positions of the `numCell` cavities with respect to the geometric center of the linac. The linac constructor also takes a pointer to its mother volume, as well as a pointer to the global electromagnetic field.

Figure 2 shows a single r.f. Pill Box cavity (in red), with an outer window ring (dark green) and an inner full window (light green). When a linac of more than two cavities is placed, contiguous cavities share a window. Figure 2 also shows a cooling channel where solenoids are embedded in large low frequency cavities. Since the beam circulates inside the solenoid, the cavity is represented by a field map (in red) restricted to a cylindric volume with radius slightly smaller than the inner radii of the magnets.

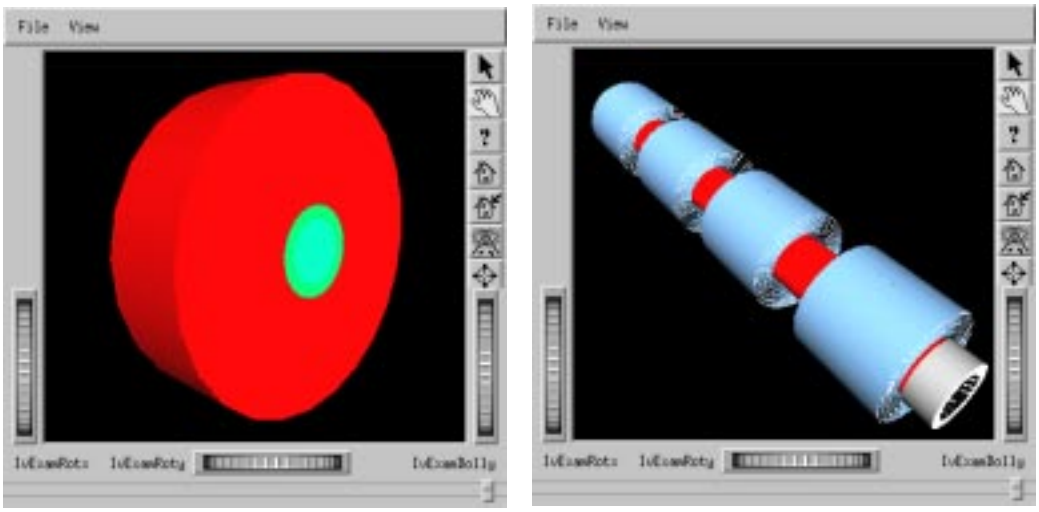

Fig. 2. Left: a Pill Box cavity (in red), with an outer window ring (dark green) and an inner full window (light green). Right: low frequency cooling channel. The red cylinders are the dummy software structure representing the limits of the electric field maps.

3.3 The Global Field Classes

A global field is the sum of all fields in the accelerator, available to the Geant4 tracking code as a particle propagates through the system. Two global field classes are used by the Beam Tools: one for the case that the field is purely magnetic, and another one in the case there is also an electric component.

- The `BTGlobalMagField` class inherits from `G4MagneticField`. A single `BTGlobalMagField` object is constructed from the sum of individual magnetic fields generated by all magnetic objects, like solenoids and magnetic field maps. The global magnetic field is accessed by a `GetFieldValue` method, where the sum of the fields is implemented. To perform these operations, the magnet objects and their positions in global coordinates need to be data members of the global field class. The `BTGlobalMagField` constructor only does a trivial initialization of some of the data members. The global magnetic field is actually filled or "included" by the `IncludeSolenoid` and `IncludeMaps` methods, which are invoked in the `BTSolenoidPhysVol` and

`BTMagFieldMapPlacement` constructors, respectively. In brief, the magnet objects, their location, and field scale factor are constructed or defined in the `Construct()` method of the detector construction user class. This information is then passed to `IncludeSolenoid` and `IncludeMaps` through `BTSolenoidPhysVol` and `BTMagFieldMapPlacement`. `IncludeSolenoid` and `IncludeMaps` assign this information to the data members of `BTGlobalMagField` for its use by `GetFieldValue`. Since this is the `GetFieldValue` associated with the field fed to the equation of motion, it will be called internally by Geant4.

- The `BTGlobalEMField` class inherits from `G4ElectroMagneticField`. A single `BTGlobalEMField` global e.m. object is constructed from the existing `BTGlobalMagField`, by adding the fields from the acceleration elements, such as r.f. cavities. The mechanism for feeding the global e.m. field information into Geant4 is the same as in the case of the `BTGlobalMagField`. The r.f. field is added to the global field by the `IncludeAnRFCell` method of `BTGlobalEMField`, which is called in `BTLinacPhysVol`. The global e.m. field has an additional feature related with the r.f. system phase tuning. The `SetPhaseDelayAtZ` method is invoked from the `UserSteppingAction` method only in reference particle mode. It uses the step object at the phase center of each cavity to calculate and set a phase delay for the cavity to operate at the required synchronous phase in a normal run. The `BTLinacCellPhaseInfo` class is convenient as it defines the right object to be manipulated by the global field. This information is passed to the `BTLinacPhysVol` constructor, then to the `BTLinacCellPhaseInfo` object, and finally to the global e.m. by `IncludeAnRFCell`.

3.4 Tuning the r.f. Cavity Phases

One of the critical elements of an accelerator simulation is the "r.f. tuning". In order to deliver the desired electric field during the passage of the beam through the cavities, a synchronous phase must be selected. This is how the r.f. wave is synchronized with the beam, more specifically, with the region of beam phase space that the user needs to manipulate. For this, there is the concept of a reference particle, defined as the particle with velocity equal to the phase velocity of the r.f. wave. If the kinematic and dynamic variables of the reference particle are set to values which are coincident with the mean values of the corresponding variables for the beam, the r.f. system should affect the mean beam properties in a similar way it affects the reference particle. Note that the r.f. wave does not necessarily have to be tuned to follow the mean velocity of the beam. Different applications may need a reference particle to represent the leading edge, the trailing edge, or any other sub-range of the total beam phase space.

The Beam Tools allow the use of a "reference particle" to tune the r.f. system before processing the beam. The time instants the particle goes through the phase center of each cavity are calculated and used to adjust each cavity phase to provide the proper kick, at the selected synchronous phase. As an example, Fig. 3 shows the reference particle trace through a cooling channel. The energy of the particle increases as it goes through a six cavity linac, and decreases through a liquid hydrogen absorber. The net energy gain after twenty unit cells is ∼50 GeV. This corresponds to a cavity synchronous phase of 25.5° for V_p=16 MeV/m at the time the reference particle goes through its phase center.

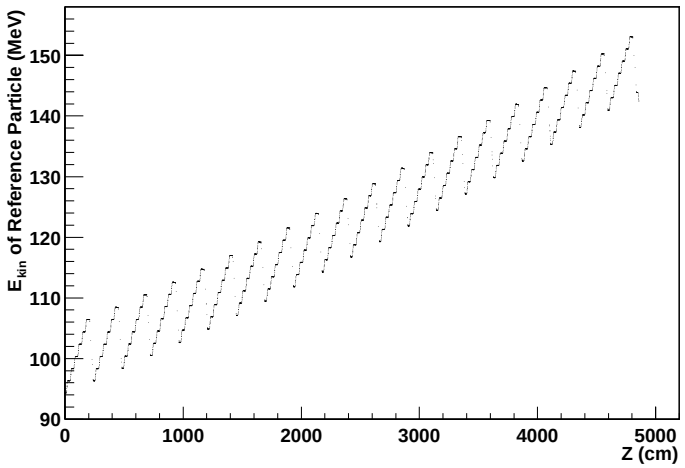

Fig. 3. Reference particle trace through a muon cooling channel. The r.f. system is tuned to provide an average net acceleration of 50 MeV to a beam with mean kinematic parameter values coincident with those of the reference particle.

Note that the r.f. model used to accelerate the reference particle is not the same the beam experiences in normal mode. While the latter could be a Pill Box or an arbitrary field map, the former is a Gaussian distributed field around the phase center of the cavity. The σ of the distribution is 20% of the length of the real simulated cavity or map, and the area under the curve is the total energy provided by that cavity to the particle.

3.5 Absorbers

The Beam Tools provide a set of classes to simulate blocks of material in the path of the beam. They are all derived from the abstract class of absorber objects `BTAbsObj`. The constructors create the solid, logical, and physical volumes in a single step. The absorbers are not a common element in accelerators because they typically degrade the beam. There are cases, however, where they can be useful [9]. For example:

- It may be necessary for some experiments to utilize beams of different sizes and qualities. Absorbers may be then used to degrade the beam emittance accordingly. Emittance is defined as the determinant of a six dimensional matrix built from x, p_x, y, p_y, $c \times t$, and E. It is a measure of the beam size in phase space.
- Transverse ionization cooling in muon beams may be achieved by reducing the beam total momentum by energy loss through an absorber material. After re-acceleration along the beam direction, the net result will be a reduction in the transverse emittance.
- Emittance exchange, a trade of transverse for longitudinal cooling, may be achieved with wedge or lense absorbers placed in a beam with transverse-longitudinal correlations. For example, if the p_z of a particle in a beam is a function of the distance r to the system center, a wedge or a lense can selectively reduce the speed of faster particles with respect to slower ones.

`BTCylindricVessel` is a system with a central cylindric rim, and two end cap rims with thin windows of radius equal to the inner radius of the vessel. The material is the same for the vessel walls and windows. The vessel is filled with an absorber material. The `BTCylindricVessel` constructor takes the absorber location in local coordinates of its mother volume, a pointer to the mother volume and the absorber material, the maximum step length in the absorber, the name of the object, the outer radius, the length, the absorber window material, the window radius, and its thickness.

```
BTCylindricVessel(G4ThreeVector location,
                  G4VPhysicalVolume* pMother,
                  G4Material *material,
                  G4double maxstep, G4String cylvesselname,
                  G4double cylvesselrad, G4double cylvessellen,
                  G4Material *cylvesselwmat,
                  G4double cylvesselwrad,
                  G4double cylvesselwthick);
```

The end cap inner radius adjusts itself automatically depending on the window radius. Realistic vessel windows are typically parabolic in shape to withstand pressure. The Beam Tools currently include only flat windows.

The grey cylinder in Fig. 4 is a schematic representation of a liquid hydrogen vessel with aluminum walls and windows. The Beam Tools also provide two constructors to simulate absorber lenses:
`BTParabolicLense` and `BTCylindricLense`. The first is a parabolic object with uniform density, and the second a cylinder object with density decreasing parabolically as a function of radius. From the point of view of the physics effect on the beam, both objects are almost equivalent. The `BTParabolicLense` constructor takes the object location with respect to the local coordinates of

its mother volume, a pointer to the mother volume, a pointer to the lense material, the maximum step length, a name, the length (maximum) at r=0, the radius, and the number of cylindric slices which make up the lense:

```
BTParabolicLense(G4ThreeVector location,
                 G4VPhysicalVolume* pMother,
                 G4Material *material, G4double stepmax,
                 G4String paraname, G4double parablength,
                 G4double parabradmax, G4int parabnumslice)
```

The lense is built as a set of short cylinders. The radius is maximum for the central cylinder and reduces symmetrically following a parabolic equation for the others in both sides.

The `BTCylindricLense` constructor takes essentially the same arguments as the parabolic lense, except that the number of slices is replaced by the number of rings. The object is built from concentric cylinder rings of the same length, different radius, and different densities to mimic a real lense.

```
BTCylindricLense(G4ThreeVector location, G4VPhysicalVolume* pMother,
                 G4Material *material, G4double stepmax,
                 G4String cyliname, G4double cylilength,
                 G4double cyliradmax, G4int cylinumrings)
```

Figure 4 shows a set of six parabolic lenses in the center of a complex magnetic system. The lenses are placed to mitigate the effect of the decrease in $\langle p_z \rangle$ at large radii in a magnetic field flip region, using an emittance exchange mechanism.

Wedge absorbers are also useful in some cases. They can be easily constructed using the Geant4 trapezoid shape `G4Trap`.

3.6 Sensitive Detectors

The sensitive detectors are empty software volumes which only provide boundaries, that is a volume transition, to force the stepper to make a pause and allow the user to execute some actions through `UserSteppingAction`. Sensitive detectors may also be used to force a change in the maximum step size to help the stepper not to miss an abrupt and short change in the electromagnetic field.

The Beam Tools provide a class of `BTSensDetGrid` objects, which are constructed from sensitive detectors previously defined in the detector construction user methods. Typically, these objects are accessed in `UserSteppingAction`

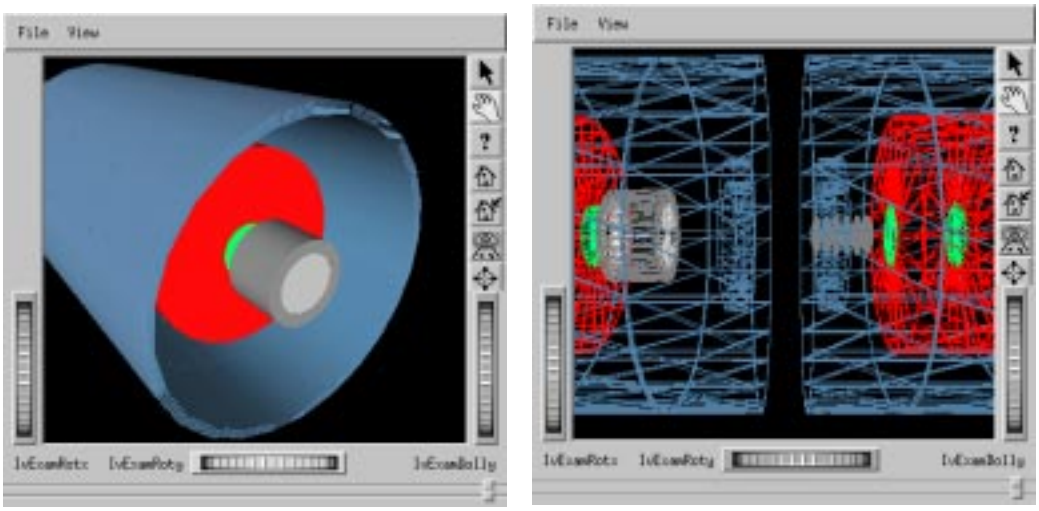

Fig. 4. Left: cooling unit cell composed of a solenoid (blue), surrounding the r.f. system (red) and the cylindric absorber vessel (grey). Right: six parabolic lenses inside a complex magnetic field.

to study the beam at different locations along the accelerator. A `BTSensDetGrid` object has information about the volume a particle traverses immediately before reaching its boundaries (`G4VPhysicalVolume *pFVol`), as well as a unique identification number `G4int iud`. The constructor also takes a pointer to the sensitive detector mother volume, a pointer to the sensitive detector physical volume, and a name.

```
BTSensDetGrid(const G4VPhysicalVolume *pTMother,
              const G4VPhysicalVolume *pFVol,
              const G4VPhysicalVolume *pDet, G4int iud,
              char* title);
```

3.7 Data Cards for Input Parameters

The Beam Tools use a native input parameter handler instead of the Geant4 messenger classes. This allows the user to pass parameter information to the simulation at run time. No compilation or linking is needed upon a modification of the parameter values in the input parameter file.

4 Applications to Neutrino Factory Feasibility Studies

The neutrino beam in a Neutrino Factory would be the product of the decay of a low emittance muon beam. Muons would be the result of pion decay,

and pions would be the product of the interaction of an intense proton beam with a carbon or mercury target. Thus the challenge in the design and construction of a Neutrino Source is the muon cooling section, aimed at reducing the transverse phase space by a factor of ten, to a transverse emittance of approximately $\varepsilon_x \sim$1 cm.

The ionization cooling technique, uses a combination of linacs and light absorbers to reduce the transverse momentum and size of the beam, while keeping the longitudinal momentum under control. There are two competing terms contributing to the change of transverse emittance ε_x along the channel:

$$\frac{d\varepsilon_x}{dz} = -\frac{\varepsilon_x}{L_{trans}} + f\left(\frac{\beta_\perp}{\beta^3 E m_\mu L_R}\right) . \tag{5}$$

The first is a cooling term, associated with the process of energy loss, and the second is a heating term related to multiple scattering. In Eq. 5, $\beta = v/c$, $L_{trans} = \beta^2 E \frac{dz}{dE}$, $\beta_\perp = \frac{2pc}{eB}$, and L_R is the absorber radiation length. $\beta_\perp = \frac{2pc}{eB}$, or "beta function", is a measure of the amplitude of the periodic transverse motion of the beam in the array of magnets.

Some of the ionization cooling systems recently studied with the Beam Tools are described briefly in the following sections. These examples are chosen to illustrate specific features of the available tools.

4.1 The Double Flip Cooling Channel (See Ref. [10])

The double flip cooling channel is a system consisting of three homogeneous solenoids with two field-flip sections. The first flip occurs at a relatively small magnetic field, B=3 T, to keep the longitudinal motion under control. The field is then increased adiabatically from -3 to -7 T, and a second field flip performed at B=7 T. Figure 6 shows a side view of a lattice unit cell, consisting of a six 201 MHz Pill Box cavities linac and one liquid hydrogen absorber, inside a solenoid. Details on the design and performance of this channel are available in Ref. [10,11]. Figure 5 illustrates the cooling effect, that is the reduction of the x (y) and p_y (p_x) distributions associated with the beam as it traverses the channel.

4.2 The Helical Channel (See Ref. [12])

The helical channel cools both in the transverse and longitudinal directions. The lattice is based on a long solenoid with the addition of a rotating trans-

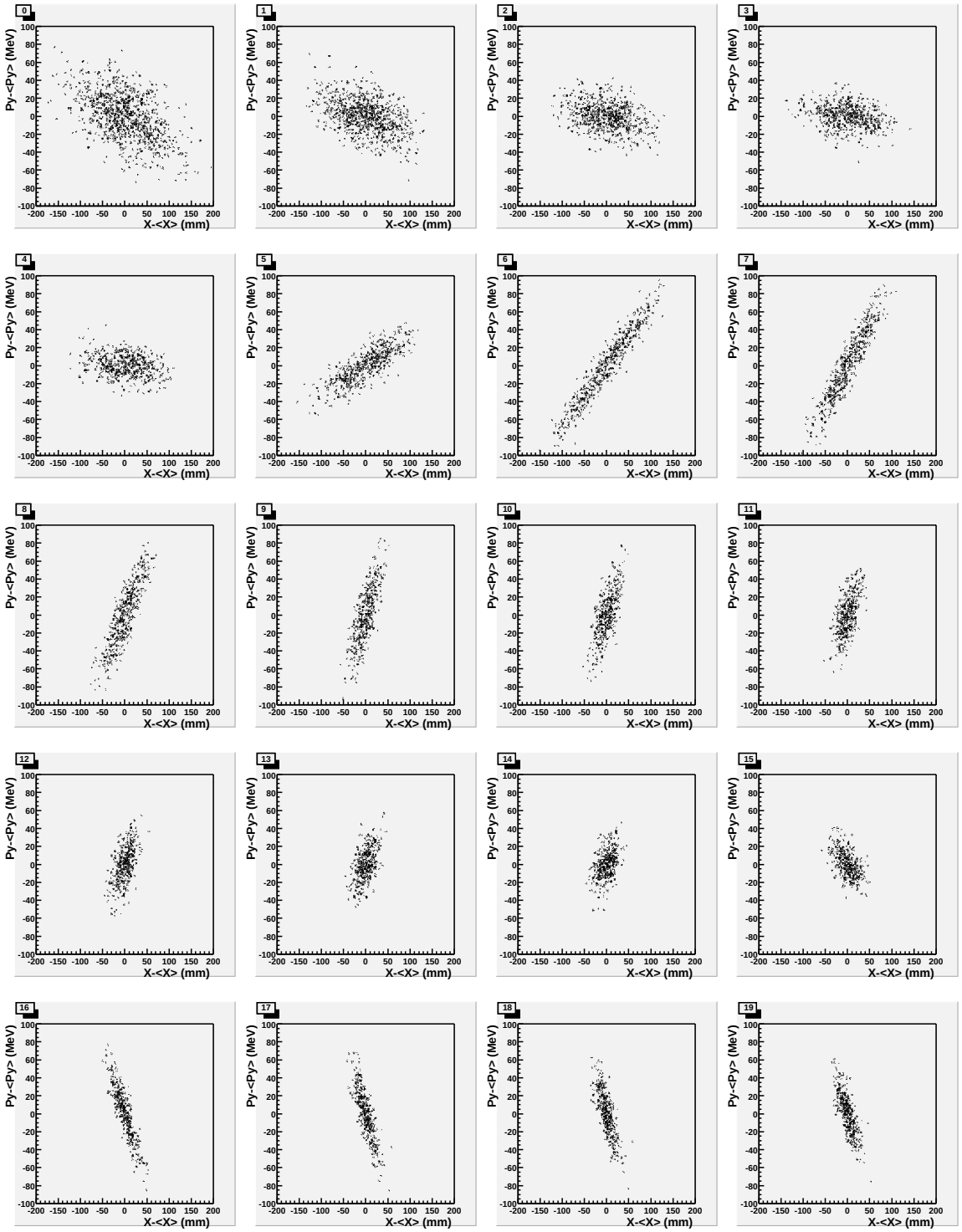

Fig. 5. The reduction of the x (y) and p_y (p_x) distributions associated with the beam is apparent as it traverses the double flip cooling channel.

verse dipole field, lithium hydride wedge absorbers, and 201 MHz r.f. cavities. Figure 4.4 shows a side view of the helical channel, including the wedge absorbers, ideal (thin) r.f. cavities, and the trajectory of the reference particle. The design details and performance of this channel are described in Ref. [12,13].

4.3 The Low Frequency Channel (See Ref. [14])

This is a design based on 44/88 MHz r.f. technology. A unit cell is composed of four solenoids embedded in four r.f. cavities, followed by a liquid hydrogen absorber. Figure 2 shows a unit cell of the low frequency channel, including the solenoids, the absorber, and the relevant section of the r.f. field map (inside the magnets). More information about this channel may be found in Ref. [14,15].

4.4 Other Systems

Among other simulations performed with the Beam Tools for Geant4 we may cite: the *Alternate Solenoid Channel (sFoFo)* [16], and a *High Frequency Buncher/Phase Rotator* scheme for the neutrino factory [17–19].

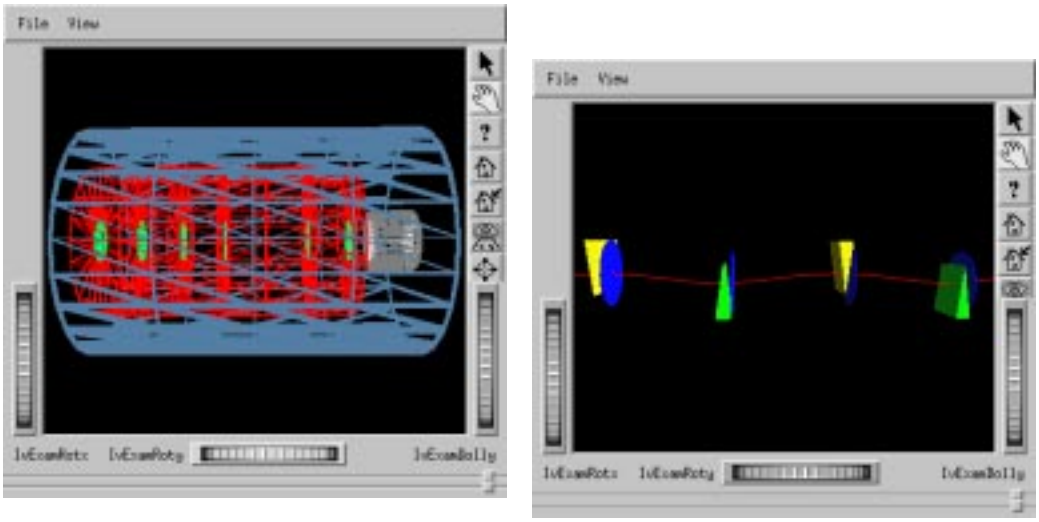

Fig. 6. Left: side view of the double flip channel unit cell, including the solenoid, the six Pill Box cavities, and the absorber. Right: image of the simulated helical channel, including the wedge absorbers (yellow and green), ideal (thin) r.f. cavities (blue), and the trajectory of the reference particle (red).

5 Summary

The Beam Physics Tools for Geant4 are used in numerous accelerator studies, reported in conference proceedings and proposals. Geant4 is especially suited to systems where accelerators, shielding, and detectors must be studied jointly with a simulation. The Beam Tool libraries, a software reference manual, and a user's guide, are available from the Fermilab Geant4 web page [20].

We thank Mark Fischler, for contributing the data cards and and spline fit classes, Jeff Kallenbach for helping with visualization issues, and Walter Brown for providing C++ consultancy. We are also grateful to the Geant4 Collaboration for useful discussions. In particular, we thank J. Apostolakis, M. Asai, G. Cosmo, V. Grichine, M. Maire, and L. Urban.

References

[1] S. Agostinelli *et al.*, CERN-IT-20020003, KEK Preprint 2002-85, SLAC-PUB-9350, submitted to Nuclear Instruments and Methods NIM A. Also See Geant4 web page at: wwwinfo.cern.ch/asd/geant4/geant4.html

[2] Cosy Infinity is an arbitrary order beam dynamics simulation and analysis code developed by M. Berz *et al.* See http://cosy.pa.msu.edu

[3] Methodical Accelerator Design is a computer program developed at CERN to design and calculate the properties of particle accelerators. http://mad.home.cern.ch/mad

[4] ICOOL is a software tool developed by R. Fernow to study ionization cooling of muon beams. See http://pubweb.bnl.gov/people/fernow/icool/readme.htm.

[5] Open Inventor. Registered trademark of Silicon Graphics Inc.

[6] See http://www-project.slac.stanford.edu/lc/nlc.html

[7] See http://www.vlhc.org

[8] "An introduction to the physics of high energy accelerators", D. A. Edwards, M. J. Syphers. A Wiley-Interscience publication.

[9] C. Ankenbrandt et al. (Muon Collider Collaboration) Phys. Rev. ST Accel. Beams 2, 081001 (1999) , Fermilab-Pub-98-179.

[10] "The Double Flip Cooling Channel", V. Balbekov, V. Daniel Elvira *et al.* Published in PAC2001 proceedings, Fermilab-Conf-01-181-T.

[11] "Double Field Flip Cooling Channel for Neutrino Factory (front-end simulation)", V. Balbekov, MuCool note #118, April 2000. http://www-mucool.fnal.gov/notes/noteSelMin.html

[12] "Simulation of a Helical Channel Using GEANT4", V. D. Elvira *et al.* Published in PAC2001 proceedings, Fermilab-Conf-01-182-T.

[13] "Conceptual Studies on Ionization Cooling of Muon Beam", Ya. Derbenev, MuCool note #185, 11/27/00.

[14] "Pseudo-Realistic GEANT4 Simulation of a 44/88 MHz Cooling Channel for the Neutrino Factory", V. D. Elvira, H. Li, P. Spentzouris. MuCool note #230, 12/10/01.
http://www-mucool.fnal.gov/notes/noteSelMin.html

[15] A. Lombardi, "A 40-80 MHz System for Phase Rotation and Cooling", CERN-NUFACT Note 34, August 2000.

[16] "Feasibility Study 2 of a Muon Based Neutrino Source", S. Ozaki *et al.* BNL-52623, Jun 2001.

[17] "High Frequency Adiabatic Buncher ", V. Daniel Elvira, MuCool note #253, 8/30/02.
http://www-mucool.fnal.gov/notes/noteSelMin.html

[18] "Fixed Frequency Phase Rotator for the Neutrino Factory", N. Keuss and V. D. Elvira,
MuCool note #254, 8/30/02.
http://www-mucool.fnal.gov/notes/noteSelMin.html

[19] "Exploration of the "High-Frequency" Buncher Concept", D. Neuffer
MuCool note #269, 2/10/03.
http://www-mucool.fnal.gov/notes/noteSelMin.html

[20] See Fermilab Geant4 web page at: http://www-cpd.fnal.gov/geant4.