BDSIM: An Accelerator Tracking Code with Particle-Matter Interactions

L. J. Nevay\textsuperscript{a,}*, J. Snuverink\textsuperscript{a}, A. Abramov\textsuperscript{a}, L. C. Deacon\textsuperscript{b}, H. Garcia-Morales\textsuperscript{a}, S. M. Gibson\textsuperscript{a}, R. Kwee-Hinzmann\textsuperscript{a}, H. Pikhartova\textsuperscript{a}, W. Shields\textsuperscript{a}, S. Walker\textsuperscript{a}, S. T. Boogert\textsuperscript{a}

\textsuperscript{a}Royal Holloway, University of London, Egham, TW20 0EX, United Kingdom
\textsuperscript{b}University College London, Gower Street, London, WC1E 6BT, United Kingdom

Abstract

Beam Delivery Simulation (BDSIM) is a program that simulates the passage of particles in a particle accelerator. It uses a suite of standard high energy physics codes (Geant4, ROOT and CLHEP) to create a Geant4 model of a particle accelerator that mixes accelerator tracking routines with all of the physics processes and particles of Geant4. This combination permits radiation and detector background simulations in accelerators where accurate tracking of all particles is required over long range or over many revolutions of a circular machine.

Keywords: Monte Carlo Simulation, Particle Accelerator, Geant4, Particle Physics, Particle Tracking

PROGRAM SUMMARY

Program Title: BDSIM
Licensing provisions: GNU General Public License 3 (GPL)
Programming language: C++, flex, bison
External routines/libraries: Geant4, CLHEP, ROOT, gzstream

Nature of problem: Simulate energy deposition and charged particle detector background in a particle accelerator originating from beam loss where particles may pass both through the vacuum pipe with magnetic and electromagnetic fields, as well as through the material of the magnets and accelerator itself. Simulate the

*Corresponding author
Email address: laurie.nevay@rhul.ac.uk (L. J. Nevay)

Preprint submitted to Elsevier
September 3, 2018
passage of particles both through and accelerator and material such as air.

Solution method: Automatic creation of a 3D Geant4 model from an optical description of an accelerator using a library of generic 3D models that are user extendable. Accelerator tracking routines, the associated fields and coordinates transforms are provided for accurate magnetic field tracking.

1. Introduction

Particle accelerators are the primary tool to study subatomic particles and to discover new particles. Their applications are widespread ranging from material treatment in manufacturing to radio-nuclide production for medical imaging [1]. They are increasingly being used for the electromagnetic radiation they produce in life-sciences to characterise biological samples [2–4]. Particle accelerators typically accelerate a beam of charged particles with radio frequency electromagnetic fields and guide them to a desired application using static magnetic fields through a series of connected evacuated ‘vacuum’ pipes.

An accelerator may lose some particles from source to delivery point due to the initial momentum distribution of the source particles and the finite extent of the accelerator components and fields. Non-linear or time varying fields may also lead to further losses of particles. The loss of charged particles (‘beam loss’) can lead to radioactivation of materials, transient radiation and energy deposition in materials and so accelerators are typically housed in a shielded environment to contain any radiation they may produce. Beam loss also has a strong influence on the thermal management of the accelerator.

In the case of high energy accelerators, even minute losses can lead to problematic radiation or heat loads. A further problem is that lost particles may reach the intended delivery point or target leading to unintended effects or signals that would be considered background. As the energy per particle increases, so does the length the particle can penetrate in material. In the case of the very highest energy accelerators such as the Large Hadron Collider (LHC) at CERN with 6.5 TeV protons, particles may penetrate tens of metres of rock or concrete. The LHC also uses cryogenic superconducting magnets to achieve the 8 T magnetic field necessary to guide the particle beams around the 27 km ring. These must be kept below 4 K to remain superconducting otherwise the magnet will quench depositing an extremely large stored energy.
into the coolant which would subsequently lead to catastrophic damage. Such cryogenic devices have an energy deposition limit of a few millijoules per cubic centimetre, which is minuscule in comparison to the total beam energy of several hundred megajoules (in the case of the LHC). This constraint requires that any losses must be both accurately and precisely quantified and controlled. Aside from the example of the LHC, any energy forefront accelerator will require the strongest possible electromagnetic fields that are provided solely by cryogenic superconducting magnets.

Cryogenic superconductors are also commonly used in accelerating superconducting radio frequency (SRF) cavities. These provided the highest accelerating gradient leading to a shorter accelerator for a desired particle energy. Similarly, the cryogenic heat loads must be accurately known and controlled to avoid damage or possible deceleration of the particles.

When a particle beam is stored for minutes to hours in a storage ring collider, various effects lead to the formation of a beam halo — particles that follow the main beam but with a large amplitude [5]. Halo must be continuously removed to avoid increased energy deposition and to protect both the accelerator and any detector close to the beam.

A further consideration is the interface between the accelerator and a detector referred to as the machine detector interface (MDI). The beam size is often strongly manipulated to create a small focus at the centre of the detector to increase the collision rate (luminosity) between the two crossing beams. This can lead to increased losses and background radiation that may penetrate the detector — ‘non-collision background’. Such background may give the appearance of potential new physics if not accurately accounted for. Often, the direction or timing of such signals can be used to discriminate against genuine collisions, but this should be minimised as much as possible to avoid degradation in the ability of the detector to correctly identify the collision events.

Simulating the effect of non-collision backgrounds requires both a simulation of the losses in the accelerator and their passage into the detector environment and through the detector itself. This often results in step by step disparate simulations.

Many low energy accelerators such as those for medical therapy may transport or deliver particle beams partly through air. In this case, an optical accelerator code cannot accurately predict the effect on the beam size and momentum distribution due to its interaction with air. Therefore, a 3D model with physics processes is required.
To predict the losses throughout a machine, a trivial estimate can be made by comparing the aperture to nominal beam size throughout the machine. However, the nominal beam size is typically derived from the linear lattice functions and does not account for the variation in particle momentum nor any nonlinear fields. Therefore, to accurately predict the losses, a particle tracking simulation is performed. A particle distribution is sampled and a Monte Carlo simulation performed by calculating the individual trajectories of particles until a termination condition is reached. Such a condition may be a single passage through a model or a certain number of revolutions of a circular accelerator. If the aperture is included in the simulation, particle tracking may be stopped when the particle position exceeds that of the aperture boundary.

A common tool for accelerator design is MAD-X [6]. This provides the ability to define a sequence of magnets, calculate the optical functions as well as an interface to the Polymorphic Tracking Code (PTC) [7] for individual particle tracking. MAD-X is commonly used to prepare an input model for the SixTrack tracking code [8] for long term symplectic tracking and dynamic aperture studies. In both the case of PTC and SixTrack, the particles are tracked throughout the complete model irrespective of aperture information. To estimate losses, the output trajectories can then be filtered by an independent program to apply the aperture model and terminate the trajectories at the appropriate point. The termination points of all the trajectories can then be collated to form a loss map [9]. However, this is a customised workflow rather than a publicly available tool.

Whilst this is a demonstrably successful technique [9], the simulation stops at the point where the particle touches the aperture. High energy particles will scatter and possibly disintegrate creating large amounts of radiation on a length scale that increases with the energy of the incident particle. For a given energy particle travelling in a material, a stopping distance can be calculated and it could be assumed that although the impact is not simulated, any subsequent radiation would occur on this length scale. However, as the particles are relativistic, they impact and scatter at very low angles and so it’s possible for a particle to re-enter the vacuum pipe. In this case the particle may travel quite some distance before again impacting the aperture. In the case of nuclei, fragmentation may occur producing nuclear fragments or protons with a momentum inside the acceptance of the accelerator and therefore travel a great distance. It is therefore crucial to simulate the interaction of losses with the accelerator as well as their subsequent propagation.
and secondary radiation to make an accurate prediction. For charged particle background in a detector it is also crucial to simulate the interaction with the accelerator as the background is predominantly composed of secondary particles produced by the accelerator.

Simulations that handle the interaction with matter are commonly made to predict particle detector response and precision. A 3D model with material specification is required as well as a library of particles and physics processes. If a magnetic or electric field is present, support for this is also required. Geant4 [10] and FLUKA [11] are two software packages that provide the capability to simulate the passage of particles in matter. Geant4 provides an open source C++ class library where the user must write their own program to construct the geometry and run the simulation. It does however provide a lot of utility that may be used to add more complicated features such as visualisation and an interactive interpreter. FLUKA is a closed-source Fortran code where the user describes their model through input text files. In both cases, significant effort is required to describe the geometry and materials particular to a given experiment or accelerator to be simulated. Furthermore, the user must supply a numerical field map for each volume they require to have a magnetic or electric field.

To calculate the particle motion in an arbitrary field, numerical integration is used. Numerical integration, while flexible can suffer from the accrual of small numerical errors that can eventually lead to gross inaccuracies if used repetitively. Limiting these effects by permitting only small steps in the field may make the simulation prohibitively computationally intensive as each high energy ‘primary’ particle may lead to thousands of ‘secondary’ particles that all must be tracked through the field. For the purpose of an accelerator, numerical integration is often not suitable as it is not sufficiently accurate after the many steps required through the large number of different magnetic fields, hence the use of dedicated accelerator tracking codes. However, accelerator tracking codes do not provide the physical processes or the 3D geometry required for such a simulation, so the simulation required is not possible.

Beam Delivery Simulation (BDSIM) [12] is a program that solves this problem by creating a 3D model using the Geant4 library with the addition of accelerator tracking routines. Geant4 was chosen as it is open source and so permits the extension of tracking routines as well as being in a more modern flexible language.

Accelerators are typically constructed with as few types of magnets as
possible and feature repetitive patterns of a set of magnets. Whilst the aperture of the vacuum pipe may vary in size, most designs fall into a small set of cross sections. BDSIM provides a library of scalable 3D components that provide the most commonly used magnets and apertures for an accelerator.

BDSIM constructs a 3D model using this library from an optical description of an accelerator, i.e. one that describes the length, type and strength of each magnet in a sequence. Along with each 3D model of the different types of magnets, appropriate fields are provided that are calculated from the rigidity-normalised strength parameters mostly commonly used to specify accelerator magnets and used in accelerator modelling tools such as MAD-X.

For tracking, Geant4 numerical integration routines such as a 4\textsuperscript{th} order Runge-Kutta integrator can be used, but a set of integrators more suited to accelerator tracking are provided. For dipole and quadrupole fields, an exact analytical solution is possible for the particle motion and numerical integration is not required. BDSIM provides the coordinate frame transformations and these routines that are automatically associated with each magnet. For higher order magnets such as sextupoles or octupoles, 2\textsuperscript{nd} order Euler integrators are provided. This is discussed in greater detail in Section 3.

With BDSIM, the user may progress from a generic model to a more specific one by adding externally provided geometry for particular elements, or by placing such geometry beside the accelerator in the model. Users can overlay their own field map on top of parts or all of components and choose between provided interpolators. A human-readable input syntax is used so the user may provided input text files to describe the model and need not write code nor compile it.

BDSIM provides a unique simulation capability that can also be accessed in a very short timescale from an optical accelerator description. The distinctive capabilities allow both energy deposition throughout an accelerator to be simulated as well as interfaces between accelerators and detectors. The implementation and a worked example are described in the following sections.

2. Implementation

Geant4 is a C++ class library that provides no program the developer or user can run. A developer must write their own C++ program to instantiate classes representing geometrical shapes, materials, placements of shapes in space as well as physics processes and the Geant4 kernel. As C++ is a compiled language, this would generally make any Geant4 model fixed in design.
However, to simulate any accelerator, a more dynamic setup is required. BDSIM uses human readable text input files with a syntax called GMAD. The GMAD syntax (Geant + MAD) is designed to be as similar as possible to that of MAD8 and MAD-X that are common tools for accelerator design and therefore it will be immediately familiar to a large number of users. This is significantly less labour intensive than writing and compiling C++ code.

BDSIM uses Flex [13] and GNU Bison [14] to interpret the input text files and prepare the necessary C++ structures for BDSIM to create a Geant4 model. The parser is easily extended by the developer allowing the possible introduction of new features in future. The most minimal input includes

1. at least one beam line element
2. a sequence (‘line’) of at least one element
3. declaration of which sequence to build
4. the particle species
5. the particle total energy

and would be written as

\[
\text{d1: drift, l=1*m;} \\
\text{l1: line=(d1);} \\
\text{use, l1;} \\
\text{beam, particle="e-",} \\
\text{energy=10*GeV;} \\
\]

Additional options and sets of physics processes may also be specified. After parsing the input text files, the Geant4 model is constructed by instantiating various construction classes that are registered with the Geant4 kernel class \text{G4RunManager}. The various aspects of the model construction are described in subsections 2.1 — 2.7.

2.1. Geometry Construction

The model is built from a sequence of unique elements that may appear multiple times in a varied order. As there are 26 different elements defined in BDSIM including 12 types of magnets with 8 different styles that can be combined with any one of 8 aperture cross sections, there is a large number of possible geometry combinations. It would be impractical to have one C++ class for each combination and it would not be trivial to extend the code to include new aperture cross sections or magnet styles. BDSIM is therefore
designed in such a way that independent pieces of geometry can be con-structed and then placed safely either alongside each other or in a hierarchy. This allows beam pipes and magnets to be constructed independently and assembled. Furthermore, it makes extension of different apertures or magnets trivial. Independent *factories* for magnet yokes and beam pipes allow any combination of aperture and magnet style to be created. Only one class is required for a magnet and that uses the factories to create the yoke and beam pipe it requires.

With the Geant4 geometry system, it is entirely possible to construct a nonphysical geometry that has overlaps between volumes beside each other or volumes that protrude outside their parent volume. Such errors are only highlighted to the user if they purposefully scan the geometry for errors or worse, during a simulation when the tracking routines fail to navigate the geometry hierarchy correctly or produce an undesired result. Once one of the Geant4 CSG primitive classes is instantiated, it is not possible to know its extent without querying a tracking point as to whether it lies inside or outside the volume. To circumvent this, BDSIM records the asymmetric extents in three dimensions along with every piece of geometry created. Whilst the cuboid denoted by these extents does not represent the surface of the volume, it is sufficient for ensuring that no overlaps will occur. Any piece of geometry in BDSIM is therefore represented by the base class `BDSGeometryComponent` that handles the extents.

To correctly navigate the geometry hierarchy, Geant4 must be able to numerically determine whether a point in 3D Cartesian coordinates lies inside or outside of a volume. Therefore, two volumes placed adjacent to each other at the same level in the geometry hierarchy must have a finite space between them. Geant4 defines a geometry tolerance that is the minimum resolvable distance in the geometry and therefore the tolerance when estimating the intersection with a surface of a trajectory. The tolerance is set explicitly in BDSIM to 1 nanometre and this is defined as a constant throughout the code called ‘length safety’ that is used to pad all geometry hierarchy.

The geometry is constructed by the `BDSDetectorConstruction` class that uses a component factory (`BDSComponentFactory`) to create the individual components required. A component registry is used to reuse previously constructed components saving a considerable amount of memory for large models. Components whose field is time dependent and therefore depends on the position in the beam line are created uniquely to ensure correct tracking. Each component is appended to an instance of
BDSBeamline, which interrogates each element and prepares the transform (rotation and translation) required to place that element on the end of the beam line in 3D Cartesian coordinates. When the construction of the beam line elements is complete, they are placed in a single container ‘world’ volume. Between construction and placement, the physical extent of the beam line is determined and these are used to dynamically construct the world volume of the appropriate size for the model.

Some elements may make use of geometry provided in external files. Such geometry is constructed again through a factory interface with a different loader for each format supported. The primary format is GDML [15], which is the geometry persistency format of Geant4. External geometry can either be placed in sequence in the beam line, wrapped around a beam pipe as part of a magnet or irrespective of the beam line in the world volume at an arbitrary location.

Due to the Geant4 interface, the fields for each element are not constructed at the same point as the geometry. Geant4 requires all fields to be constructed and attached to logical volumes at one point in the program. When the geometry for an element is constructed, a field recipe and logical volume to attach it to are registered to a field factory. The factory is then used by the Geant4 interface to construct and attach all fields at once.

2.2. Coordinate Systems & Parallel Worlds

The majority of accelerator magnetic fields as well as externally provided field maps are defined with respect to the coordinate frame of the element they are attached to. Accelerator specific numerical integration algorithms for calculating a particle trajectory through an element are typically defined in a curvilinear coordinate frame that follows the trajectory of a particle with no transverse position or momentum and with the design energy through that element — the Frenet-Serret coordinate system. Contrary to this, Geant4 uses the coordinate frame of the world volume that are 3D Cartesian coordinates. BDSIM provides coordinate transforms between these systems to permit the use of accelerator tracking routines.

For a given global Cartesian position, Geant4 can provide the transform from the volume that that point lies within to the world volume and vice-versa irrespective of the depth of the geometry hierarchy. A transform for N levels higher in the geometry hierarchy can also be requested. As the depth of the geometry hierarchy may vary from component to component,
this facility cannot be used. The local coordinate frame of any given volume is also not necessarily the required curvilinear frame.

To provide the correct transforms into the curvilinear frame, BDSIM constructs a separate 3D model (a ‘parallel world’ in Geant4 terminology) with different geometry than that of the beam line. In this parallel world, a simple cylinder of the same length as the accelerator component is placed at the same position as shown in Figure 1.

Any point in the world can then be queried in the parallel world and the transform used from the volume found at that location to the world volume. This will be a transform from the world volume to the local coordinate system of the cylinder whose axis is degenerate with the curvilinear system required. In the case of a component that bends the beam line, many small straight cylinders with angled faces are constructed, but the cylinder coordinate system is still different. Here we provide a dedicated transform.

As already discussed, all Geant4 geometry must have a numerically resolvable gap between adjacent solids and so each parallel world cylinder is placed with a small gap between it and the next one in the beam line. However, if a point is queried in this gap, an incorrect transform will be found. To overcome this, a third parallel world is built with bridging cylinders. If the world volume is found while searching the parallel curvilinear world, then the bridging world is subsequently used. This ensures a continuous coordinate system irrespective of the limits of the geometry system. A quadrupole, curvilinear cylinders and bridging cylinders are shown in Figure 2.
2.3. Physics Processes

Geant4 provides a large library of physics processes. A Geant4 application must instantiate the required particles and then instantiate the required physics process classes and attach them to the applicable particles. Geant4 includes a more general set of ‘physics lists’ that provide commonly used sets of physics processes applicable to many particles for a given application and energy range. These modular lists are selectable by the user. It is left to the user to select the most relevant physics processes for the simulation as using all of the physics processes by default would result in an extremely computationally expensive simulation. Additionally, there may be more than one physics model relevant that the user may wish to choose from. BDSIM provides simple names that map to the most common physics lists constructors in Geant4 as described in Table 1.

If no processes are selected, only tracking in magnetic fields will be present, i.e. particles will pass unimpeded through matter. Whilst this may appear inaccurate, it is computationally efficient and is useful to validate particle tracking and beam distributions.

2.4. Run and Event

A BDSIM simulation progresses at two levels; a Run and an Event. An event is the smallest unit of simulation where one initial particle or set of particles is tracked through the model. All information may be collected on an event level basis and each event is entirely independent of another. A run is a unit of simulation containing $N$ events where the geometry and physics processes do not change. A run may be summarised by surveying all events generated in that run.
Table 1: Examples of name mapping of Geant4 physics lists in BDSIM.

| BDSIM Name       | Geant4 Class                                      |
|------------------|--------------------------------------------------|
| em               | G4EmStandardPhysics                              |
| em_livermore     | G4EmLivermorePhysics                             |
| decay            | G4DecayPhysics                                   |
| ftfp_bert        | G4HadronPhysicsFTFP_BERT                         |
| ftfp_bert_hp     | G4HadronPhysicsFTFP_BERT_HP                     |
| hadronic_elasic  | G4HadronElasticPhysics                           |
| ion              | G4IonPhysics                                     |
| shielding        | G4ShieldingPhysics                               |
| stopping         | G4StoppingPhysics                                |
| synch_rad        | G4SynchrotronRadiation                           |

BDSIM can be run in two ways; interactively with a visualiser; or in batch mode without visualisation. The later is considerably faster and used for large scale simulation, whereas interactive visualisation is typically used in the preliminary stages to verify the model and typical outcome of an event. In the case of batch mode, BDSIM performs 1 run with the desired number of events. Interactively, the user may issue the following command on the interactive terminal:

```
/run/beamOn N
```

where \( N \) is the number of desired events. In this case, BDSIM creates 1 run with \( N \) events for each time the command is issued.

Geant4 provides several places in the framework where the developer can insert their own code and gain access to simulation information. BDSIM implements actions at both the beginning and end of each event and run where information is collected, histograms prepared and data written to the output.

2.5. Output

A Geant4 simulation produces no output information by default as the total possible information is unmanageable. Therefore, it is left to the developer to provide classes that process the information available during the simulation from the Geant4 kernel to create the desired reduced output information. BDSIM uses the Geant4 interfaces and records information from the simulation in two ways.
Firstly, several sensitive detector classes are provided and automatically attached to various volumes. These are registered with Geant4 and are provided with access to all particle tracks through the volumes they are attached to. BDSIM includes such a class to record the energy deposition in all volumes in both 3D Cartesian and curvilinear coordinates.

Secondly, BDSIM records trajectory information independently of volumes at an event level. As the full set of trajectories for all particles in an event could reach several gigabytes per event, there are several user options in BDSIM to select which trajectories are desired. Each trajectory consists of a series of trajectory points that each contain the spatial coordinates of that point, particle species, total energy and the physics process ID associated with the last step. Even though a user may filter to select only a certain type of particle, the trajectories are stored in a linked manner so that any individual trajectory point stored can be fully traced back to the primary particle in the event.

In a conventional tracking simulation the coordinates of each particle are updated after passing through each element in the accelerator. Therefore, the most common output is a list of all the particle coordinates after each element. To be able to compare BDSIM to tracking simulations, a similar functionality is provided. BDSIM has a sample command that inserts a “sampler” after either a single specified element or all elements in the beam line. A sampler is a 1 nm thick plane that is $5 \times 5$ m wide transversely. There is no ability to record particles on an arbitrary plane or surface of a volume in Geant4, so a volume must be created and a sensitive detector attached to it. The box is made as thin as possible to minimise artificially increasing the length of the beam line. The sampler plane records any particles passing through it.

Output information is stored in the ROOT data format [16, 17]. This is a well-documented, compressed binary format that is widely used in the high energy physics community. The ROOT data storage facilities are highly suited to storing information on an event by event basis and allow direct serialisation of the developer’s C++ classes. These can be loaded using the compiled software, but the file also contains a complete template of all classes used such that this is not required and the data can always be loaded even if the original software is lost.

Although written in compiled C++, the ROOT framework provides reflection for the classes stored that allows the compiled code to be easily loaded and used interactively in the ROOT interpreter as well as in Python with
the exact same functionality allowing users to explore the data interactively with ease.

Aside from raw information, histograms of energy deposition and primary particle impact and loss points are recorded with each event. These can be averaged in analysis to produce the mean energy deposition across all events, but with the correct statistical uncertainty. This treatment is only possible with event by event data storage.

2.6. Primary Particle Generation

Generally an event may start with several initial particles (“primaries”), however, in the case of an accelerator it is more typical to simulate a single particle sampled from a beam distribution.

To begin each event, BDSIM draws randomly a set of coordinates from a distribution chosen by the user. BDSIM provides 12 possible distributions. The most basic is the “reference” distribution where each particle is the same for each event with a fixed set of coordinates as chosen by the user. These are by default aligned to the axis of the accelerator with no transverse position or momentum. This distribution is used to validate the reference or design trajectory.

A 6D Gaussian distribution is provided where the user may specify the standard deviation \( \sigma \) in each dimension as well as the off-diagonal correlation terms in a 6x6 matrix. The Twiss parameterisation common to accelerator beam descriptions is provided, which in turn uses the 6D Gaussian generator.

As one of the purposes of BDSIM is to simulate lost particles interacting with the accelerator, which usually occurs with a low frequency, several distributions are provided that make the simulation more efficient. One such distribution is the “halo” distribution that provides a particle distribution according to the nominal Gaussian beam, but at a large number of \( \sigma \).

In all cases, the generator uses a single instance of the pseudo-random number generator from the CLHEP library. The generator used is the HEP-JamesRandom engine. The user may specify a starting seed value or one is automatically generated from the computer time. The seed and state of the generator are both recorded in the output file so a simulation may be entirely reproduced.

2.7. Simulation Control

In each event, particles are tracked until they reach zero kinetic energy or they leave the world (the outermost) volume. With high energy primary
particles, a very large number of secondaries can be produced and as the particles decrease in energy the number of secondaries can grow, which is known as infrared divergence. Such detail at low energy may not be necessary and may dominate computationally. To avoid this, it is necessary to have some control over the physics processes other than including only those relevant in the physics list.

Geant4 provides the ability to provide tracking cuts in a volume. Here, a minimum energy and maximum time can be specified. If a particle has an energy under or a time greater than these limits, the particle will be removed from tracking. BDSIM provides a method to set limits that will be attached to all volumes and also records the final energy of the particles to conserve energy per event.

In addition to the tracking cuts, Geant4’s primary mechanism is a range cut. A range cut is a distance assigned to a particle species that a secondary of that species would be required to travel. If the secondary would not travel at least that range, the energy deposition is recorded but the secondary particle is not produced. The range is internally converted in Geant4 to a material and particle specific energy cut. This strategy provides the greatest accuracy with the least computation [10]. BDSIM provides an interface to set the range cut for protons, electrons and positrons, photons and a default.

Such limits are often necessary as a simulation may not include physics processes that would lead to the natural termination of a particle. For example, synchrotron radiation can produce a very large number of photons making it computationally expensive to track all the secondary particles. The rate scales $\sim E^4$ so it can dominate when simulating high energy events. For this reason it is often omitted. The omission however, may lead to low energy particles spiralling in a magnetic field, such as that of a dipole, indefinitely. BDSIM numerical integrators have special treatment of spiralling particles, but the user limits provide a convenient method to avoid such scenarios that may lead to long running events with no gain in information.

One further consideration is a circular accelerator. With a circular accelerator and no synchrotron radiation, the particle energy will not decay as the magnetic field does no work. BDSIM therefore provides a circular option to limit the number of turns any primary particle can complete. A $5 \text{ m} \times 5 \text{ m} \times 1 \mu\text{m}$ box is inserted between the beginning and end of the lattice that is orientated to make a large transverse plane to the beam. A dynamic set of user limits is attached as well as a special Geant4 sensitive detector class. The sensitive detector class records the number of turns completed by
the primary particle and when this has reached the specified number of turns, the user limits are dynamically changed to reject any particle over 0 eV. This makes the box an infinite absorber, stopping any particle that hits it.

2.8. Variance Reduction

With the large number of physics processes included in each physics list as well as the large number of possible outcomes from each interaction, the outcome or process of interest may occur at a low frequency per event. An efficient simulation would simulate the outcome you wish to characterise for each event, i.e. it is impractical to simulate tens of millions of events to observe only a few occurrences of the desired outcome. When analysing a data set, this effect leads to a large variance of results when in certain areas of parameter space. Biasing is a form of variance reduction where a process or outcome is made artificially more frequent, but recorded with a corresponding statistical weight. Multiplying the simulated frequency by the weight gives the correct physical result, but due to the more frequent occurrence, with a reduced variance. Traditionally, the developer had to write their own C++ wrapper for a given process they wish to study, but recent developments of Geant4 have introduced a more generic biasing interface [18]. Geant4’s generic biasing interface provides both physics based (process cross-section) and non physics based (splitting and killing). An interface to process biasing is provided in BDSIM that allows the user to scale the process cross-section by a given factor for a given particle.

2.9. Visualisation

At the initial stages of a study, it is crucial to visualise a model to validate its preparation as well as the expected outcome of a typical event. Visualisation is accomplished through an interface to the Geant4 visualisation system that provides a variety of different visualisation programs depending on the software environment.

The default and recommended visualiser is the Geant4 QT visualiser. This provides a rich interface with an interactive 3D visualisation where the model can be viewed with solid surfaces or as a wire-frame, both with and without perspective. A built in command prompt allows extensive control of the simulation and the visualisation, such as geometry overlap checking and particle track colour schemes.

To visualise an event, all trajectories must be stored. Additionally, the model and all trajectories must be converted to polygon meshes and rendered
on screen. Whilst this is handled by the Geant4 visualisation system, this overhead leads to the interactive visualisation system running events approximately an order of magnitude slower than they would without visualisation. The visualisation is crucial in the first stages of a simulation for validation but beyond that it is more desirable to generate as many events as possible in a given time. BDSIM by default will run interactively, but if executed with the \texttt{--batch} option, it will not use the visualiser and complete the simulation with only text output to the terminal. Batch mode is much faster and suitable for simulations that may be run on a computing farm where no graphics systems are available.

2.10. Analysis

Once BDSIM has produced output, this may be analysed using an included analysis suite. This covers the most common basic analysis but also includes an interface for the user to include more complicated analyses.

The main analysis tool is \texttt{rebdsim} (‘ROOT event BDSIM’). This uses a simple text file as input that defines histograms to make from the output structures contained in output files. These can be 1 to 3 dimensional histograms made on an event by event basis or as a simple integration across all events. Any histograms stored in the raw output that were produced in BDSIM, such as energy deposition and primary impact location, are combined to produce a mean histogram across all events. The histogram definition in rebdsim specifies which variables in which ‘Tree’ in the output to make the histogram from as well as binning and a “selection”. The selection is an optional weighting that can be a numerical factor, another variable in the data, a Boolean expression based on data variables or a combination of all. This is an interface to that of \texttt{TTree:Draw} in ROOT.

The event level structure in the output is paramount to correctly calculating the variance and therefore the statistical uncertainties of any histograms. Conventional tracking programs only simulate one particle per ‘event’ so there is no need to structure data in this way. However, with the complex tree of particles and physics processes that can happen per event in a radiation simulation, it is crucial to structure the data in this way. The default histograms made by rebdsim are made event-by-event and so have the correct statistical uncertainties.

Using the “chain” feature of ROOT many files can be analysed together behaving as one. Furthermore, a tool \texttt{rebdsimCombine} is provided to combine the output from several instances of rebdsim. For large data sets it would be
prohibitive to analyse the whole data set serially, so it is better to analyse small chunks in parallel and then combine the resultant histograms. This strategy results in the exact same numerical answer but in a fraction of the time.

The ROOT output format and included analysis tools provide great flexibility in the storage and analysis of simulation data. They also ensure the process is scalable to the very largest data sets dealt with today on the multi-terabyte level.

3. Accelerator Tracking

The Geant4 model constructed by BDSIM provides all the required fields for an accelerator, however, the commonly used numerical integration algorithms provided with Geant4 such as a 4\textsuperscript{th} order Runge-Kutta integrator will not provide sufficient accuracy for tracking particles through an accelerator. Whilst these numerical integrators are suitable for arbitrary spatially and time varying magnetic and electric fields, the small errors from the numerical integration can accumulate with many successive uses leading to inaccurate results, hence these are rarely used for accelerator tracking. For the specific static magnetic fields of an accelerator, such as a pure dipole or quadrupole field, there are exact solutions that provide more accurate tracking. However, these typically use a coordinate system that follows the accelerator. Algorithms for these tracking routines as well as coordinate system transforms are included with BDSIM and used by default.

The majority of accelerator tracking algorithms use a coordinate system that follows the path of the reference particle (no transverse momentum and exactly the design energy) as opposed to 3D Cartesian coordinates [19]. The Frenet-Serret is such a curvilinear coordinate system as shown in Figure 3.

![Figure 3: The curvilinear Frenet-Serret coordinate system (x, y, s) shown following the trajectory of the reference particle. The global Cartesian system is shown (X, Y, Z).](image)
With the Frenet-Serret coordinate system each element’s numerical accuracy is independent from its location and there are no repeated transformations into and out of the system the algorithms are specified in.

Accelerator tracking algorithms can broadly be classed in two categories: thick, and thin. In the thick regime, elements are as long as they are in reality. In the thin regime, a series of instantaneous “kicks” is applied to the trajectory that affect only the transverse momentum and not the position of the particle. These thin kicks are used in conjunction with drift sections to achieve a similar result. As expected, a single kick is not a physically accurate representation of the passage of a particle through a magnetic field, however, the weighted combination of many small kicks in combination with drifts provides a physically accurate representation. The thin regime is often used as it is more computationally performant for a solution that conserves the volume in phase space. A solution that conserves phase space volume \textit{(symplectic)} is required as the tracking algorithms may be applied many times and any small errors will accumulate until the result is no longer physically correct. General symplectic thick solutions exist, but are often more computationally expensive. An accelerator ‘lattice’ is typically first described by a thick lattice that matches the physical accelerator and then potentially converted to a thin lattice.

Geant4 provides a series of numerical integrators that are supplied with spatial coordinates and the field values at that location and are required to predict the particle motion. These must be able to handle a variety of situations that are not encountered in an accelerator tracking program:

- an arbitrary step length
- particles with different masses
- particles with different charge or that are neutral
- backwards travelling particles
- particles travelling parallel to a magnetic field

To introduce accurate accelerator tracking to a Geant4 model, any new integrator must be able to handle these situations. Furthermore, the 3D nature of the model prevents the representation of the accelerator as thin lenses. As Geant4 assumes numerical integration, any new integrator must also estimate a numerical uncertainty associated with the calculation.
BDSIM includes a set of integrators that encapsulate thick tracking algorithms that use the Frenet-Serret coordinate system with special provision for the aforementioned tracking scenarios that may occur in a Geant4 model. The coordinate transforms between 3D Cartesian and the Frenet-Serret system make use of the parallel geometry constructed by BDSIM. For scenarios where these algorithms cannot be used, a 4th order Runge-Kutta integrator is used. It is foreseen that no one set of algorithms may be applicable to all situations, or they may be extended in future, sets of integrators are used. The user may select from different predefined sets depending on the application.

A full mathematical description of all BDSIM integrators is given in the provided documentation [20].

For the most accurate model, the user may choose to use a field map from simulation or measurement. Such field maps can be loaded by BDSIM and overlaid on to both BDSIM and user-provided geometry. The supplied field maps can be easily converted from other formats using the included pybdsim Python utility and can be compressed using gzip. The gzstream library [21] is used to dynamically decompress the text files as they are loaded. BDSIM includes a variety of interpolators to provide the continuous field values in 3D coordinates required by the simulation from the field specified at discrete points in the field map. The user may choose any of the Geant4 numerical integrators ranging from low order Euler to 8th order Runge-Kutta ones.

4. Example

To demonstrate the capabilities and unique features of BDSIM, an example model is provided and illustrated here. The model is of a fictional accelerator that demonstrates many features found in a variety of real accelerators. The model consists of a 4 km racetrack ring with two straight sections that include a low-β collision point and a high beta collimation section. Each insertion has a 3-cell half dipole strength dispersion suppressor on each side. The most relevant parameters are shown in Table 2. The model and all materials required to prepare it are included with BDSIM as a walk-through example called “model-model”. A schematic layout is shown in Figure 4.

The model was created in MAD-X [6] and optical functions as calculated by MAD-X are shown in Figure 5. The low-β straight section is designed to create a small symmetric focus of the beam suitable for a collision point
Figure 4: Layout of the example model with race track design including two arc sections, and two insertion regions: ‘IR-collimation’ and ‘IR-lowbeta’. Between each arc and an insertion region there is a dispersion suppressor ‘DS’. The interaction point ‘IP’ is where the beam is focussed to a minimum size.

Table 2: Defining parameters of example model.

| Parameter         | Value                        |
|-------------------|------------------------------|
| Particle          | proton                       |
| Energy            | 100 GeV                      |
| Length            | 4 km                         |
| Dipole bending radius | 221.5 m                   |
| $\epsilon_{N,x,y}$ | $1 \times 10^{-6}$ m rad    |
| Collimators       | 10                           |
| Dipoles           | 256                          |
| Quadrupoles       | 252                          |
| $\beta^*_{x,y}$   | 2.517 m                      |
Figure 5: Optical functions of example model. The two insertion regions can be seen with reduced horizontal dispersion (red dashed). The first insertion creates a small symmetric beam suitable for collision with another beam or gas target. The second insertion produces a large beam size suitable for collimation. The schematic at the top of the plot depicts each magnet type in the accelerator (blue: dipoles; red: quadrupoles, yellow: sextupoles, black: collimator).

with potentially another beam or gas target. The other insertion is designed to expand the beam suitable for collimation or scraping of the beam.

To prepare a BDSIM model a rendered version is first made from MAD-X using the TWISS command that produces a sequential one-to-one representation of the machine in an ASCII file. This is trivially converted to BDSIM input format GMAD using a provided Python converter pybdsim. Any additional information not specified in this file such as aperture or collimation settings can be included in this conversion with user-supplied Python dictionaries. There is no standard format of auxiliary information about an accelerator so loading this information is left to the user, however Python is a widely used language for which many data loading and processing libraries exist. The automatically converted model includes a Gaussian beam distribution as parameterised by the Twiss parameters from the first element in the sequence. This fully functional BDSIM model created in minutes acts as a starting point that can be customised. Figure 6 shows the 3D visualisation of part of the model in BDSIM.

The MAD-X model does not contain any information about the collimators in the lattice. For a realistic model, an example set of collimator settings is provided that are calculated from a specific number of $\sigma$ in the beam distribution at that point. Table 3 illustrates the general settings used in a 3-stage collimation system that has ‘primary’, ‘secondary’ and ‘tertiary’ collimators that are placed at successively greater distances from the beam.
Figure 6: Visualisation of the example model in BDSIM using the Geant4 Qt visualiser. Only part of the machine is visualised showing a single proton impact and secondaries (colour coded by charge). The optional tunnel is automatically built by BDSIM to follow the beam line.

Table 3: $\sigma$ values used for collimator sections. The ‘open’ setting is used for the aperture in the non-collimating plane.

| Collimator Type | $\sigma$ opening |
|-----------------|------------------|
| primary         | 6                |
| secondary       | 7                |
| tertiary        | 8                |
| open            | 40               |

Each collimator consists of a square aperture with one narrow aperture and one open aperture that is designed not to impede the beam.

These conceptual settings are used in combination with the MAD-X optical functions to calculate absolute collimator openings in millimetres using an included Python script. These are shown in Table 4.

The model is prepared using the included `pybdsim` converter. This allows the extra information for collimators to be included as a Python dictionary that is prepared from the text file. The converter also allows aperture information as loaded from a MAD-X TFS file as loaded by `pybdsim`. The mature utilities permit aperture filtering and simple inclusion of user-defined supplementary information to the conversion as shown below.

```python
import pybdsim
```
Table 4: Collimator half openings as calculated from a given beam $\sigma$ and the optical functions at each collimator location. Primary collimators are of low-Z carbon, secondary collimators copper and tertiary absorbers tungsten that are typical of high energy proton collimation systems.

| Name     | Material | $\Delta_x$   | $\Delta_y$   |
|----------|----------|--------------|--------------|
| COLPRIMY | C        | 3.174$\times$10$^{-2}$ | 2.660$\times$10$^{-2}$ |
| COLSECY  | Cu       | 5.891$\times$10$^{-2}$ | 3.681$\times$10$^{-2}$ |
| COLPRIMX | C        | 3.133$\times$10$^{-2}$ | 6.066$\times$10$^{-2}$ |
| COLSECX  | Cu       | 3.157$\times$10$^{-2}$ | 3.250$\times$10$^{-2}$ |
| COLTERT  | W        | 1.434$\times$10$^{-3}$ | 4.437$\times$10$^{-3}$ |
| COLTERT2 | W        | 8.006$\times$10$^{-3}$ | 6.134$\times$10$^{-3}$ |
| COLTERT3 | W        | 5.225$\times$10$^{-3}$ | 5.843$\times$10$^{-3}$ |

import pymadx
ap = pymadx.Data.Aperture('../madx/ring_aperture.tfs')
ap = ap.RemoveBelowValue(0.005)
cols = pybdsim.Data.Load('collimatorSettings.dat')
pybdsim.Convert.MadxTfs2Gmad('../madx/ring.tfs', 'bdsim-model',
        aperturedict=ap, collimatordict=cols)

The user may use the converted model immediately, edit it to their needs or include the automatically produced GMAD files in their own models. Including the files in others allows the model to be safely regenerated at any point without losing any user-defined input.

After preparation of the model, the first step is to validate the optical functions of the BDSIM model to ensure correct preparation. To generate optical functions, the automatically provided Gaussian beam distribution according to the Twiss parameters at the start of the machine are used. The beam distribution is sampled after each element in the BDSIM model by including the sampler command:

```
sample, all;
```

This places a 1 pm thin box between every element that records the passage of any particles going through it once. The model is run in BDSIM with 1 - 10 thousand particles simulated with the following command:

```
bdsim --file=bdsim-model.gmad --outfile=optics1 --batch --
        ngenerate=1000
```
The output ROOT format file “optics1.root” is analysed by an included tool rebdsimOptics that calculates the optical functions and their associated statistical uncertainty from the finite population at each sampler.

```
rebdsimOptics optics1.root opticalfunctions.root
```

These are calculated by accumulating 1 - 4\(^{th}\) order power sums and calculating various moments of the distribution. The statistical uncertainty reduces with the number of particles simulated and it is recommended to simulate at least 1000 for a meaningful comparison. rebdsimOptics produces another ROOT format file with the optical function data. The optical functions as calculated from BDSIM can be compared using pybdsim. In this case, we compare against the original MAD-X optical functions the model was prepared from using the following command:

```
pybdsim.Compare.MadxVsBDSIM('../madx/ring.tfs', 'opticalfunctions.root')
```

This produces a series of plots that compare \(\bar{x}, \bar{y}, \sigma_{x,y}, \sigma_{x',y'}, D_{x,x'}, \alpha\) and \(\beta\). An example is shown in Figure 7.

When simulating the optical functions, ideally no particles are lost as this would affect the shape of the beam distribution and the validity of comparing \(\sigma\) of the distribution. However, once validated and a physics study is desired, the lack of beam loss makes the simulation very inefficient. For example, to witness an event at 6\(\sigma\), approximately 500 million events would be required on average to be simulated. Therefore, it is logical to select a distribution of
Figure 8: Example phase space distribution for a beam loss simulation using the ‘halo’ distribution based on the Twiss parameters at the entrance to the primary horizontal collimator. The horizontal phase space (left) is designed to intercept a horizontally orientated 2 jaw collimator, whereas the vertical phase space (centre) is the unaltered distribution according to the Twiss parameters. The combination of the two spatially is shown (right). The distribution is exaggerated here with a greater spread in $\sigma$ than is required to make the shape clear. The horizontal phase space is clipped such that only particles with $x > 5.7\sigma$ are produced.

Primary particles that will collide with the accelerator or exhibit the desired interaction. BDSIM includes a variety of bunch distribution generators.

For this example, the intent is to evaluate the performance of the collimation system. Any beam that exists grossly outside the collimator aperture will be immediately lost within one revolution in a circular machine. Similarly, any particle largely inside the collimator aperture will not intercept the collimator. We therefore simulate a thin annulus in phase space that aligns with the edge of the collimator closest to the beam. Such a distribution is provided by the ‘halo’ distribution in BDSIM. This generates particles uniformly in a given phase space volume and accepts or rejects the particle depending on its single particle emittance. The process is repeated until a set of suitable coordinates are achieved. Furthermore, spatial limits are provided that allow the phase space to be reduced. An example input distribution is shown in Figure 8.

In this example, the collimation system is designed to work independently in the horizontal and vertical planes. Therefore, each is simulated independently with the appropriate beam distribution for each dimension. These can later be combined for total beam loss information. Figure 9 shows the combined losses from the simulation of $2 \times 10^5$ 100 GeV protons in both the horizontal and vertical planes.

Practically, the simulation was performed on the 500-core Royal Holloway
Figure 9: Combined beam loss from a horizontal and vertical halo simulation with $4 \times 10^5$ primary protons at 100 GeV. Blue triangles show the point of first contact of primary particles and red circles, the stopping location. The integrated energy deposition is also shown in black recorded by the accelerator geometry.

Faraday Cluster in 2000 jobs that produced a total 60 GB of ROOT format output. These were analysed individually using rebdsim with the same analysis configuration ASCII file. The resulting 2000 rebdsim histogram files were then combined using the provided rebdsimCombine tool.

Figure 9 shows the proton impact and loss locations. The impact location is determined as the first point where a physics process is invoked along the step of the particle and the loss point is the end of the trajectory of the primary proton. This may be either due to an inelastic collision and subsequent fragmentation or simply absorption. Additionally, the energy deposition in all material in the accelerator is shown. A clear maximum in losses and energy deposition can be seen in proximity to the collimation section between 3500 m and 4000 m. However, a clear decaying tail of energy deposition is seen throughout the subsequent arc (in the direction of the beam). For the first 1 km of the machine there are very few particle losses but considerable energy deposition. Such energy deposition would not be shown from only tracking primary particles and is a unique feature of BDSIM.

Both the ‘hits’ and ‘losses’ are normalised to the number of events simulated so the rate can be scaled to realistic beam particle populations and bunch patterns easily. The energy deposition histogram is also calculated on a ‘per-event’ basis and these are averaged giving the mean energy deposition rate per proton simulated. This is only possible because of the event-by-

27
event storage of the output format and this is crucial to calculate the correct statistical uncertainty associated with each bin in the histogram.

Looking at the collimation insertion region in more detail as shown in Figure 10, the pattern of proton impacts and losses can be seen more clearly. The ‘hits’ and ‘losses’ are quite different indicating that primary protons can interact with a collimator and escape. Such information could be used to improve the efficiency of the simplistic collimation section in this model to better absorb the scattered or leaked protons.

The information shown from the simulation is already a powerful guide to the operating radiation produced in the machine. However, further information is easily available that allows an even greater understanding of the machine behaviour. By using the ‘sample’ command on each collimator, the complete distribution of particles after each element can be recorded.

Figure 11 shows the energy spectrum of particles recorded in a sampler placed after the primary horizontal collimator for the horizontal halo simulation. For each particle species a separate histogram was prepared with rebdsim by defining a “selection” that acts as a filter. Here, the integer particle ID from the Monte Carlo Numbering scheme [22] was matched. The spectra show significant fluxes of various high energy particles leaving the primary collimator and in particular a much broader spectrum of primary protons than were originally in the beam. These are of particular interest as
Figure 11: Particle energy spectrum of all particles recorded after the primary horizontal collimator. The data is integrated for all revolutions of the primary beam in the simulation (up to 200).

they may travel some distance before being lost in a position that would not be immediately associated with the collimator in question.

Furthermore, for the purpose of damage and activation it is useful to look at the neutron distribution as an example. Shown in Figure 12 is the 2D distribution of neutrons weighted by their total energy after the secondary horizontal collimator from the horizontal halo simulation only. Here, a clear shadow of the collimator can be seen. The rebdsim input syntax is shown below to highlight the simplicity of making per-particle-species rate normalised histograms from simulation data without the need for a complicated analysis.

```
Histogram2D
Event.
COLPRIMXEWFluxProtSecZoom
{100,100}
{-0.5:0.5,-0.5:0.5}
COLPRIMX.y:COLPRIMX.x
COLPRIMX.energy*(COLPRIMX.partID==2212&&COLPRIMX.parentID>0)
```

For circular models, the losses as a function of turn number are important. BDSIM records the turn number associated with all data in the output for circular models permitting turn-by-turn analysis. Figure 13 shows the
Figure 12: The energy-weighted neutron flux after the secondary horizontal collimator per proton impact on the primary collimator. Although the collimators have an asymmetric aperture, their outer shape is a square. Here the shadow of the square 0.5 m wide collimators can clearly be seen.

surviving fraction of the simulated halo beam after each turn. The periodic steps are due to the tune of the circular machine and the figure illustrates the different performance of the collimation system in each plane. Should it be desired, the data presented in Figure 11 and Figure 12 can easily be filtered by turn number to show the radiation on a particular turn or range of turns that may correspond to a sharp increase of losses.

5. Validation

5.1. Tracking Validation

Tracking accuracy is crucial for physics studies to correctly describe the machines being simulated. Imprecise tracking in any single component would result in incorrect subsequent tracking in the machine. This is particularly relevant to circular machines that would suffer immensely from cumulative errors due to the high number of components and repeated passage of particles.

Verification of the tracking routines in BDSIM is best demonstrated by comparison to existing particle tracking software. PTC is such a commonly used tool for accelerator tracking simulations with trusted reliability and
accuracy. The syntactic similarities of input model in BDSIM and PTC via the MAD-X interface makes basic tracking comparison models easy to write and simple to execute.

To ensure tracking accuracy, each component as well as a variety of machines have been compared to PTC. In each case, both a BDSIM and PTC model are prepared from an input MAD-X model. BDSIM is used first and the initial coordinates generated by BDSIM are then used as input to PTC to ensure they are the same. Comparisons are made both on a particle-to-particle basis and by comparing optical functions and beam sizes calculated using the included ‘rebdsimOptics’ tool. rebdsimOptics calculates the 1st to 4th order moments of the coordinates and uses these to calculate the sizes and optical functions of the distribution recorded after each element along with the statistical uncertainty from the finite number of particles sampled.

Particle to particle comparisons show residuals at the level of $10^{-9}$ in both position (m) and transverse (normalised) momentum for particles in the paraxial approximation for the most common design of magnets (length and strength) found in a large variety of accelerators. The comparison was made by specially compiling BDSIM with double precision output and increasing the precision of the output from PTC.

Optical comparisons for a variety of accelerators including the LHC, the
Accelerator Test Facility 2 (ATF2) at KEK, Japan, the Diamond Light Source (DLS) at Oxford, UK, show excellent agreement with both PTC and the optical functions from MAD-X. The LHC demonstrates accuracy through a large number of components. The ATF2 demonstrates correct transport through a highly non-linear lattice where pole face rotations and dipole fringe fields are important. The DLS demonstrates a machine whose performance is highly dependent on chromatic and dispersive effects.

Although the tracking routines provided are demonstrably accurate, the 3D model must include a small numerically resolvable gap between each volume. For long term tracking in circular accelerators, such geometrical effects accrue and can result in inaccurate tracking. To overcome such a limitation, the tracking can be corrected at each turn with an optional user-supplied one turn transfer map. The map can be easily generated with an optical program such as PTC and imported into BDSIM. A single particle will therefore start in BDSIM and be tracked throughout the circular machine. Upon completing one turn, the coordinates are discarded and replaced with ones calculated from the initial coordinates transformed by the one turn map. If the particle has scattered it is allowed to perform one more turn in BDSIM. As the map is chosen to be symplectic (energy conserving), long term tracking stability is ensured while retaining the benefits of a 3D Cartesian model. It also ensures there is no loss of precision in the tracking due to the possible offset in Cartesian coordinates and the limited precision of a double length floating point number in C++. Therefore, the maximum error accrued in large models such as the LHC with many thousands of elements is the maximum error accrued in one turn. In the case of the LHC this is approximately 1 nm.

6. Conclusions

BDSIM is an open source C++ code that makes use of widely used and current high energy physics software to provide a mixed 3D and accelerator tracking simulation suitable for predicting the radiation in and around an accelerator.

It significantly simplifies the construction of a 3D Geant4 model by providing a library of generic but scalable accelerator components as well as the appropriate fields and numerical integrators for the most accurate particle tracking. Additional thin elements such as multipoles and magnet fringe elements ensure accurate magnetic tracking that matches the most commonly
used accelerator tracking codes. Furthermore, the ability to track all particles through the accelerator with arbitrary step sizes is unique and allows prediction of the radiation transmitted through and around an accelerator. The data format and provided analysis tools are built on ROOT software used commonly in particle physics and are highly scalable and suitable for long term data storage.

The included Python utilities allow preparation of models in minutes that would normally take months to prepare and are easily extendable with further user input and customisation. Being open source, the program is highly transparent to the user with complete and regularly updated documentation. Furthermore, the software can be easily extended with user’s C++ code.

BDSIM has been engineered to be applicable to both the lowest and highest energy accelerators in use today. It provides a unique ability to perform a mixed accelerator and radiation transport simulation as well as remove many obstacles for developing a beam line model. The software was written to provide an ability not just for one simulation but in the most general way possible.

7. Acknowledgements

The development of BDSIM has received funding from the John Adams Institute at Royal Holloway. This work was supported by Science and Technology Research council grant “The John Adams Institute for Accelerator Science” ST/P00203X/1 and Impact Acceleration Account. It was also supported in part by EU FP7 EuCARD-2, Grant Agreement 312453 (WP13, ANAC2) and STFC, United Kingdom. Thank you to J. Albrecht, D. Brewer, A. Cooper, W. Parker, J. Van Oers and S. Williams for their contributions and testing.

References

References

[1] O. Barbalat, Applications of particle accelerators; rev. version (CERN-AC-93-04-BLIT-REV) (1994) 23 p.
URL https://cds.cern.ch/record/260280

[2] J. Arthur, et al., Linac coherent light source (LCLS) conceptual design report.
[3] K. Balewski, W. Brefeld, W. Decking, H. Franz, R. Rohlsberger, E. Weckert, PETRA III: A low emittance synchrotron radiation source. Technical design report.

[4] Detailed design report on the MAX IV facility (2010).

[5] G. Valentino, R. Aßmann, R. Bruce, F. Burkart, V. Previtali, S. Redaelli, B. Salvachua, G. Stancari, A. Valishev, Beam diffusion measurements using collimator scans in the lhc, Phys. Rev. ST Accel. Beams 16 (2013) 021003. doi:10.1103/PhysRevSTAB.16.021003.

[6] Methodical Accelerator Design X (2018).

[7] F. Schmidt, E. Forest, E. McIntosh, Introduction to the polymorphic tracking code: Fibre bundles, polymorphic taylor types and exact tracking.

[8] F. Schmidt, SIXTRACK version 1.2: Single particle tracking code treating transverse motion with synchrotron oscillations in a symplectic manner: User’s reference manual.

[9] R. Bruce, R. W. Assmann, V. Boccone, C. Bracco, M. Brugger, M. Cauchi, F. Cerutti, D. Deboy, A. Ferrari, L. Lari, A. Marsili, A. Mereghetti, D. Mirarchi, E. Quaranta, S. Redaelli, G. Robert-Demolaize, A. Rossi, B. Salvachua, E. Skordis, C. Tambasco, G. Valentino, T. Weiler, V. Vlachoudis, D. Wollmann, Simulations and measurements of beam loss patterns at the cern large hadron collider, Phys. Rev. ST Accel. Beams 17 (2014) 081004. doi:10.1103/PhysRevSTAB.17.081004.

[10] S. Agostinelli, J. Allison, K. Amako, J. Apostolakis, Geant4—a simulation toolkit, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment (3) 250 – 303. doi:https://doi.org/10.1016/S0168-9002(03)01368-8.

[11] G. Battistoni, T. Boehlen, F. Cerutti, P. W. Chin, L. S. Esposito, A. Fassò, A. Ferrari, A. Lechner, A. Empl, A. Mairani, A. Mereghetti, P. G. Ortega, J. Ranft, S. Roesler, P. R. Sala, V. Vlachoudis, G. Smirnov, Overview of the {FLUKA} code, Annals of Nuclear Energy 82 (2015) 10 – 18. doi:https://doi.org/10.1016/j.anucene.2014.11.007.
[12] BDSIM - Beam Delivery Simulation.
URL http://www.pp.rhul.ac.uk/bdsim/

[13] Flex website.
URL https://github.com/westes/flex

[14] GNU bison.
URL https://www.gnu.org/software/bison/

[15] R. Chytracek, J. McCormick, W. Pokorski, G. Santin, Geometry description markup language for physics simulation and analysis applications., IEEE Trans. Nucl. Sci. 53 (2006) 2892. doi:10.1109/TNS.2006.881062.

[16] R. Brun, F. Rademakers, ROOT: An object oriented data analysis framework, Nucl. Instrum. Meth. A389 (1997) 81–86. doi:10.1016/S0168-9002(97)00048-X.

[17] I. Antcheva, M. Ballintijn, B. Bellenot, M. Biskup, R. Brun, N. Buncic, P. Canal, D. Casadei, O. Couet, V. Fine, L. Franco, G. Ganas, A. Gheata, D. G. Maline, M. Goto, J. Iwaszkiewicz, A. Kreshuk, D. M. Segura, R. Maunder, L. Moneta, A. Naumann, E. Offermann, V. Onuchin, S. Panacek, F. Rademakers, P. Russo, M. Tadel, Root: a C++ framework for petabyte data storage, statistical analysis and visualization, Computer Physics Communications 180 (12) (2009) 2499 – 2512. doi:https://doi.org/10.1016/j.cpc.2009.08.005.

[18] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, Recent developments in geant4, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 835 (Supplement C) (2016) 186 – 225. doi:https://doi.org/10.1016/j.nima.2016.06.125.

[19] W. H., Particle Accelerator Physics, 3rd Edition, Springer, 2007. doi: 10.1007/978-3-540-49045-6.

[20] BDSIM manual.
URL http://www.pp.rhul.ac.uk/bdsim/manual

[21] D. Bandyopadhyay, L. Kettner, gzstream library.
URL https://www.cs.unc.edu/Research/compgeom/gzstream/
[22] M. Tanabashi, et al., Particle data group, Phys. Rev. D 98 (2018) 030001.