G4BEAMLINE PROGRAM FOR RADIATION SIMULATIONS*

Kevin Beard, Thomas Roberts, Muons, Inc, Batavia, Illinois, USA 60510
Pavel Degtiarenko, Jefferson Lab, Newport News, Virginia, USA 23606

Abstract

Our G4beamline program is a useful and steadily improving tool to quickly and easily model experimental equipment and shielding without user programming.

INTRODUCTION

In current research programs at accelerator facilities there is often a need to quickly evaluate and sometimes mitigate potential radiation consequences of variations in physical setups. Monte Carlo programs (such as Geant4[1]) are capable of realistically modeling such problems, but the technical details of setting up, running, and interpreting the required simulations are beyond the ability of all but the most expert researchers (Figure 1).

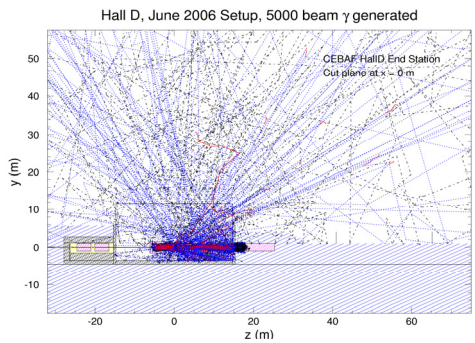

Figure 1: Typical radiation simulation.

We are planning to use G4beamline[2], a program that is an interface to the Geant4 toolkit that Muons, Inc. has developed to simulate accelerator beamlines, and extend it with a graphical user interface to quickly and efficiently model experimental equipment and its shielding in experimental halls. The program is flexible, user friendly, and requires no programming by users, so that even complex systems can be simulated quickly.

TECHNICAL APPROACH

Geant4 is an internationally supported particle tracking toolkit that was developed to simulate particle interactions in large detectors for high energy and nuclear physics experiments, and includes most of what is known about the interactions of particles and matter, including time-varying electromagnetic fields. G4beamline is a program that provides a highly flexible and user-friendly interface to the Geant4 capabilities relevant to simulating beamlines and requires no programming by users. We proposed to enhance G4beamline with a graphical user interface (GUI) for the specification of the system to be simulated. We also propose to use G4beamline and this new GUI to develop a universal source term calculation and benchmarking tool for radiation assessments.

G4beamline was conceived and designed as a flexible interface to the Geant4 toolkit specifically intended to permit the rapid prototyping and evaluation of different accelerator and beamline designs. It has become a "Swiss Army Knife" for Geant4 and is currently in use at Fermi, Jefferson, and Brookhaven labs, Universities of California Riverside and Mississippi, Illinois Institute of Technology, and by the MICE and PRISM collaborations. Its internal programming infrastructure is fully object-oriented, highly modular, and designed to be easy to extend.
G4beamline uses an ASCII file to specify all aspects of the simulation. It consists of a series of commands that control the simulation, define elements to be used in the simulation, place elements into the simulated world, and direct the generation of results. The complexity of the input file is comparable to the complexity of the system to be simulated (compared to system-specific simulation programs which are much more complex than the system itself). G4beamline has a rather large repertoire of common elements used in particle accelerators and detectors, such as bending magnets, quadrupole magnets, RF cavities, etc. In addition, G4beamline makes it easy to layout beamlines that involve bending magnets and the resulting rotations of elements – centerline coordinates are available that automatically handle the details so the user does not need to laboriously compute the locations and rotations of beamline elements.

An important feature of G4beamline's input file is that it is object oriented: one defines an object (with its internal structure) and then places that object into the simulated world (or into another object). This is simpler and more in line with the way people think of such problems than most other description-based simulation programs (e.g. MARS[3], MNCP[4]). Moreover, G4beamline's input file is formatted so that anybody familiar with the problem domain can simply read it, with minimal knowledge of the input syntax. For instance, it should be clear that this sequence puts a series of three 2x4x8" lead bricks into the world along the Z axis (units are mm):

```
box Brick length=203.2 width=101.6 \
  height=50.8 material=Pb
     place Brick z=0
     place Brick z=203.2
     place Brick z=406.4
```

The first line defines an object named "Brick", and each of the next 3 lines places an instance of it into the simulated world. Note that a basic Geant4 program fragment to do this would require about a page of C++ code; it would be readable only by someone expert in both C++ and Geant4, would be more error-prone, and would be more difficult to

* Work supported by DoE STTR grant DE-FG02-6ER86281

change if, say, one decided to use metric bricks rather than U.S. ones. This difference in complexity becomes much greater when complicated objects like magnets or RF cavities are used, or when rotations are involved.

It is easy to build up a library of frequently used objects, greatly reducing the effort and knowledge required to set up a simulation. A library may include many types of objects, such as common sources (radioactive, particle beam, cosmic ray), magnets (dipole, quadrupole, solenoid, …), shielding components (lead bricks, concrete shielding blocks, …), detectors (all types), moderated detectors (with different moderating layers), etc.

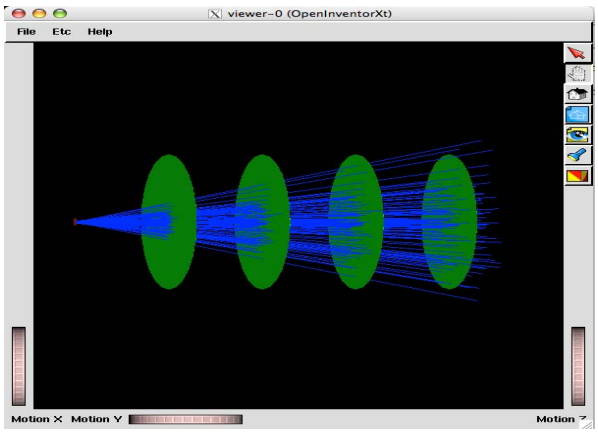

Figure 2: Muons passing through virtual detectors.

Once the system to be simulated is specified in the input file, particles are tracked using the full accuracy of the Geant4 toolkit. Extensive visualization capabilities are included in G4beamline, so the user can *see* what is being simulated, and where the simulated particles go (Figures 2 and 3). To permit trade-offs between accuracy and computation time, several different models for hadronic interactions are available, and over 100 megabytes of experimental data are available for use by the more detailed models; much of these data are specific to neutrons and photons.

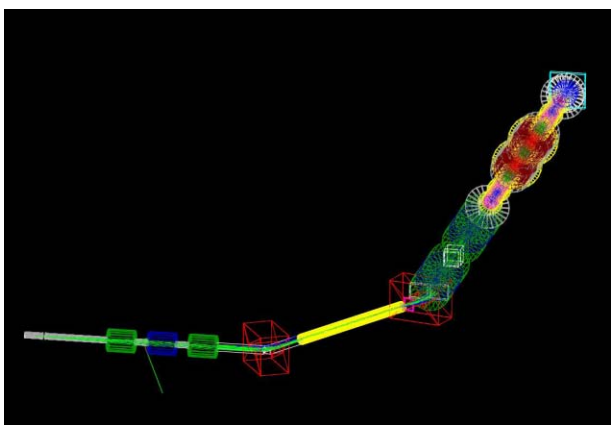

Figure 3: The Muon Ionization Cooling Experiment.

The existence of open-source toolkits from the physics and graphics communities has allowed a tremendous enhancement of software productivity. By building on the many staff-years invested by the Geant4 collaboration, and other open-source development communities, the development of G4beamline was quite modest compared to its capabilities. Table 1 shows that the total amount of code in G4beamline is more than 50 times what was actually written to develop it (excluding the C/C++ language libraries). While the number of lines of code is not a very good measure of code complexity, it is an indication of the fact that programs like G4beamline can be vastly more general and feature-rich than their bare size might imply. Moreover, the code size of the toolkits does not fully reflect the tremendous efforts and hundreds of person-years that have gone into them.

Table 1: Approximate Amount of Code Contained in G4beamline.

Software Component	Approximate Lines of Code
G4beamline	20,000
Geant4 Toolkit	750,000
Open Inventor Toolkit	300,000
OpenGL, Xwindows, etc.	>>100,000

At present G4beamline has a GUI that can only control the execution of the program and visualize the system , so we proposed to implement a graphical user interface (GUI) for G4beamline to construct and modify the description of the system to be simulated. At present, the GUI will be implemented in a stand-alone Java program separate from G4beamline itself, using an input file to control the program in the same manner as command-line users.

The current program's GUI (Figure 4) is implemented in Java, layered directly on the Java Swing toolkit. The new GUI to edit input files will be far more complex, and a more capable, higher-level toolkit is appropriate. A likely candidate is the Eclipse open development platform[5].

A key feature of our design for the GUI program is that the individual commands and elements it supports will be automatically generated from the G4beamline source code, so the two programs will automatically support exactly the same features as the list of features evolves. This is easy to do because of the modular structure of the G4beamline code; this basic approach is already used for the generation of the chapter on commands in the G4beamline User's Guide. All help text, argument lists, and argument descriptions are contained in the source code for each command, so the single source will be used 3 ways: in the program, in the User's Guide, and in the GUI.

Our software development methodology emphasizes rapid prototyping,, and as the saying goes, "we eat our own dog food" – so we have a strong incentive to develop the user-friendliest interface. We support Windows, Mac OS, and Linux standard user interfaces, and the underlying principle is: **A user interface is well-designed when the program behaves exactly how the user thought it would**. Another important principle is: **Simple things should be simple, but complex things should be possible**. We will implement a GUI that is flexible and 100% user configurable, but provide it with appropriate defaults so it works as users expect out-of-the-box.

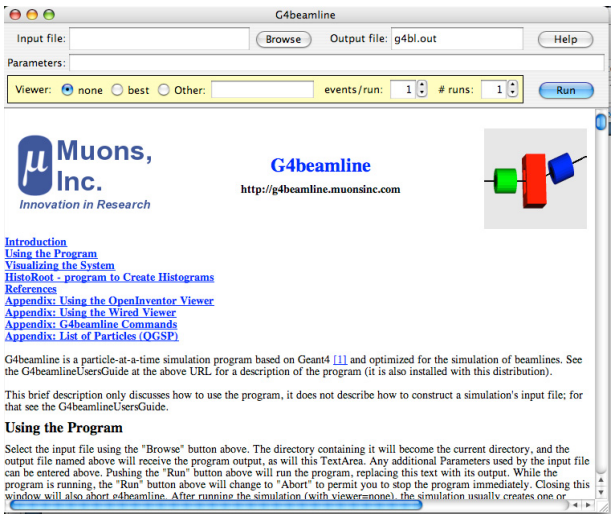

Figure 4: Current G4beamline GUI.

The result of this effort will be a graphical user interface to G4beamline that is as flexible and user-friendly as we can make it. It will be designed from the ground up with usability in mind and include extensive menus of elements and drag & drop placement in a scrollable and zoomable world.

Modern software development does not occur independently of applications. The effective implementation of these new capabilities for G4beamline requires that we use the program while developing them. This ensures that we do not spend time implementing useless features, and also helps us refine and optimize the user interface to the new features. In the software engineering literature this aspect of our methodology is known as "rapid prototyping", and we often implement one or more prototypes of a new feature before finalizing its implementation. For instance, specific requests from several current users have triggered this project.

For example, Jefferson Lab, has a need to quickly reevaluate radiation doses throughout the experimental halls in a rapidly changing environment. Each new experimental configuration requires a new assessment, as its beam requirements, physical layout, detector arrangement, shielding, etc. will be unique. To date, this has been done using Geant3 based models, requiring significant effort to modify for each experiment.

An additional feature that will be implemented using G4beamline and its new GUI, which will be of use at Jefferson Lab and in the wider Geant4 and accelerator communities, is the creation of a universal Geant4 source term calculation and benchmarking tool. Such a tool will be used to model particle yields from simple targets as a function of the target, beam particle, beam energy, secondary particle type, secondary particle energy, and secondary emission angle. This models the radiation fields around the target and can be used to more efficiently evaluate experimental backgrounds and required shielding for the experiments. The various experimental measurements of particle yields permit us to compare real data to the model, and to evaluate the accuracy of the physics processes and simulation parameters chosen for the model. We expect this to become a valuable tool for radiation assessments at many accelerator facilities. Such a tool could also be used by Geant4 developers in justifying the development and implementation of new physics models in Geant4.

FUTURE WORK

Muons, Inc. has the experience and knowledge to provide consulting services in many areas of accelerator and nuclear physics, and we are especially adept at applying and adapting G4beamline to specific customer needs. G4beamline itself is both an open source and a very open-ended tool, capable of simulating particle transport and radiation in just about any system for which a description can be written. As customer needs are rarely that broad, it can be advantageous to implement a tool specific to a particular situation for use by non-experts. An obvious possibility would be to implement a library of objects commonly used in such cases, and to package G4beamline and associated tools into scripts that would permit technicians who are expert in neither the physics nor these tools to use them for specific applications

REFERENCES

[1] Geant4 Toolkit - http://geant4.cern.ch
[2] G4beamline – http://g4beamline.muonsinc.com
[3] MARS – http://www-ap.fnal.gov/MARS/
[4] MNCP – http://mncpx.lanl.gov
[5] Eclipse open development platform - http://www.eclipse.org