Web Site

http://geant4.web.cern.ch/geant4/

Introduction

GEANT was developed by Rene Brun of CERN in the 1974 as well (it was a good time for programming. For fun, check out https://cds.cern.ch/record/118715/files/CM-P00059731.pdf. Also, for an amuzing note on the sociology of computing, see https://www.phenix.bnl.gov/WWW/lists/phenix-comp-l/msg00114.html). The original was written in the FORTRAN programming language. Just before the turn of the century, it was rewritten from scratch in C++ by the GEANT4 collaboration. Their web site is: http://geant4.web.cern.ch/geant4/

GEANT4 is a library of codes that can be used to simulate particle interactions with matter. It includes extensive geometry and visualization software. Using this software, you can specify the matter distribution, down to the nuts and bolts if you like, of a detector and see what will happen if you aim particles at it.

GEANT4 has an active users community that you can access at their listserve at:
http://hypernews.slac.stanford.edu/HyperNews/geant4/cindex

Our goal will be to learn enough GEANT to complete the exercise in the exercise link for this course, under GEANT.

Setting up your area and first run

You need a computer with CERN disks cvmfs mounted.

Let's get a taste of GEANT before we learn the details. We will use a simplified version of the B4a example that comes with GEANT. do the following:

```
mkdir junk
cd junk
git  clone https://github.com/saraheno/dualtoy.git
cd dualtoy
cmake -DGeant4_DIR=/cvmfs/geant4.cern.ch/geant4/10.5/x86_64-slc6-gcc63-opt/lib64/GEANT4-10.5.0 ../B4a
make
./dualtoy -c template.cfg -u Xm -o haha

(be a little patient… the x11 connection is slow)
Next to the “session” prompt in the window it pops up
  /run/beamOn 1
```

You should have seen a pretty picture pop up while it was running (if you did not, check that you had X11 running). Also, if you look now, you'll see a new file, haha.root. Open this file using root (root.exe haha.root) and use the TBrowser new command to look at the histograms contained inside.

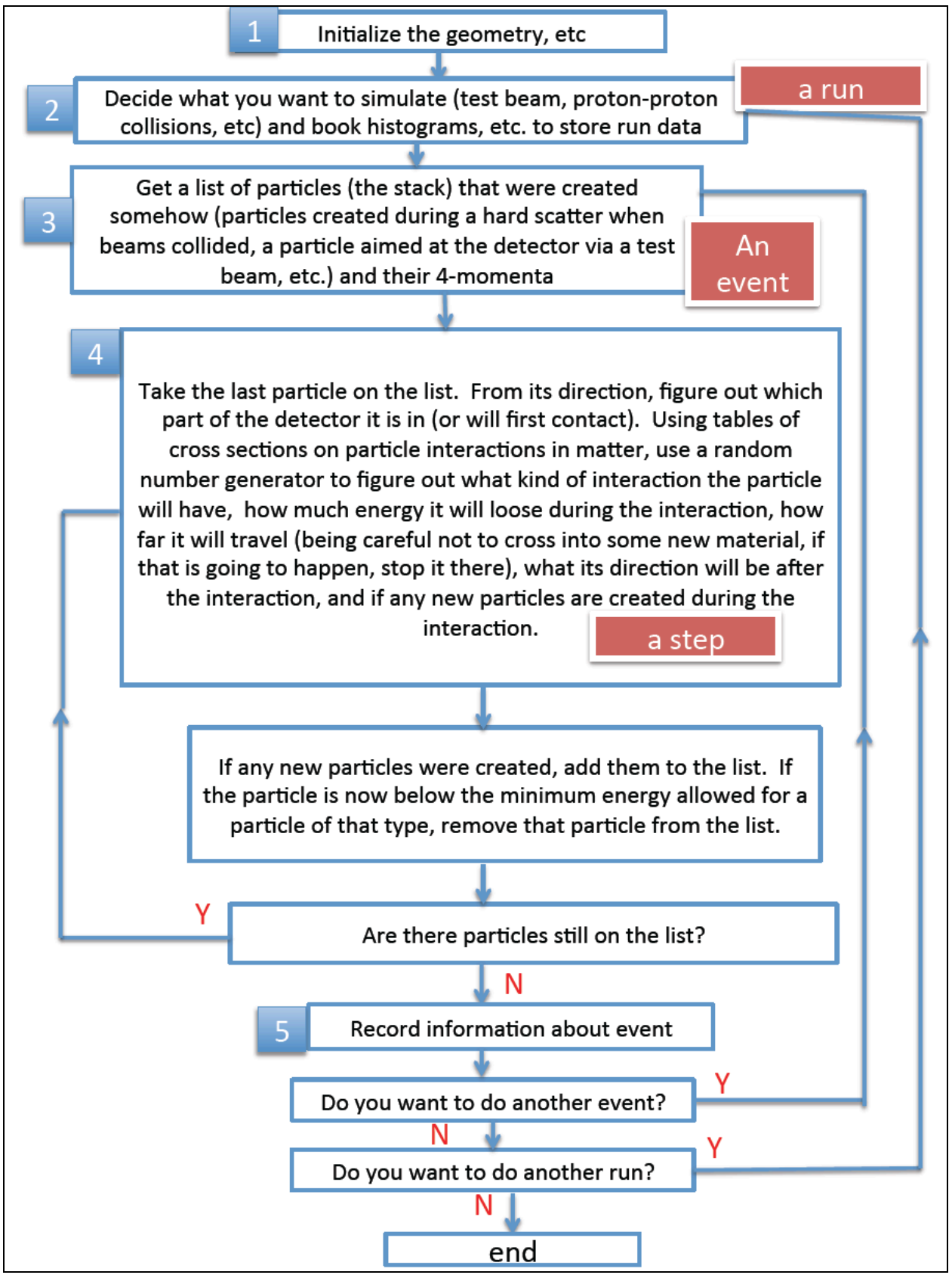
As you can see from the pretty picture that popped up, the code you cloned simulates a simple sampling calorimeter.

Understanding the structure of GEANT4

A GEANT executable is build from code provided by the GEANT4 collaboration and code that you yourself write. Unlike many of the programs you are used to, there is no default executable that can be run without any code being written by you.

The code provided by GEANT can be found in the $MYGEANT3 area (defined in g4setup.sh). Look in the "source" subdirector of this area, and in the subdirectories contained in that to get a feel for the size of the code provided by GEANT4. The example was adapted from is in $MYGEANT4/examples/basic/B4.

The code that is of the sort the user needs to provide is in the src subdirectory of the git repository you cloned. The code is fairly complex. To understand it, first we need to understand a bit about the structure of GEANT4 code and program flow. As an aid, look at the (simplified and thus not entirely correct) flow chart shown below.



Let's try to understand how the flow shown above matches to the code in $MYGEANT3, the dualtoy/src and dualtoy/include areas.

GEANT4 itself mostly handles the code for step 4 (code in $MYGEANT3). It also provides utilities for the user to use to build the geometry in step 1. It has some little code that can be used to produce an "event" (in step 3) consisting of a single particle. Usually, another code, such as PYTHIA, provides the list of initial particles needed by step 3.

The user code needs to be written in C++, and therefore makes use of classes. In general, classes are defined, and the classes may have methods that are called by the main GEANT4 code at various points in the flow. For the GEANT4 project you will be doing, you will need to understand well the following files:

- DetectorConstruction: step 1
- PrimaryGeneratorAction: step 2 and 3
- RunAction: step 2
- EventAction: step 5
- SteppingAction: step 4

Each of these contains code that is used at different points in the flow chart above, as indicated in the list. Let's look at them a bit in detail...

Creating the initial list of particles

Step 3 is handled by PrimaryGeneratorAction. Let's take a look at this in detail. First, look at:
dualtoy/include/PrimaryGeneratorAction.hh

You should see that in this file, a class called PrimaryGeneratorAction is defined. The class has a creator (PrimaryGeneratorAction()), a destructor (~PrimaryGeneratorAction()), as all classes must.

It also has a method

```
virtual void GeneratePrimaries(G4Event* event);
```

GEANT expects the user to create routines like this, and has code setup to call them at the right point in the flow described above. However, the user must write the code. To see the code that comes with this example, look at: dualtoy/src/PrimaryGeneratorAction.cc. Look at the code. Can you tell which code goes with the creator? Which code goes with "GeneratePrimaries"?

Exercise: Change the energy of the particle being created. How does the time required to do an event depend on the particle energy? Why do you think this is?

Reducing information

We do not store to memory or disk all the information about all the particles created in all the showers in all the events. Each shower can contain a very large number of particles, and when in dense media, each particle can take a very large number of steps. Instead, at step 4, we gather only that information we need for our purposes for each particle and each step. This information culling occurs in SteppingAction.hh and SteppingAction.cc. Let's look at them.

This code makes use of another code. If you look in dualtoy.cc, you'll see:

```
// Create output tree
//
G4cout << "before creating tree" << G4endl;
CreateTree* mytree = new CreateTree ("tree");
G4cout << "after creating tree" << G4endl;
```

This code uses include/CreateTree.hh and src/CreateTree.cc. In CreateTree.hh, we see lines like:

```
//integrated energy in each longitudinal layer
float depositedEnergyEscapeWorld;

float depositedEnergyTotal;
float depositedEnergyECAL_f[3];
float depositedEnergyECAL_r[3];
float depositedEnergySolenoid;
float depositedEnergyWorld;
```

These are variables where we will accumulate energies deposited in various sections of the calorimeter in SteppingAction.cc

Again we see that the .hh file is used to define a class. Again, the class has a creator, a destructor, and a method. Again, GEANT4 code will make sure the method is called for each particle and for each step of that particle.

gth to each of these numbers defined in the header, with B4aEventAction::AddAbs and B4aEventAction::AddGap

Let's go back toSteppingAction and look at the .cc file. You can see this has one main method, UserSteppingAction that takes as an argument a "step". In order to store energy deposited in the passive and active material separately, we first have to know which material the step was taken in (remember that a GEANT step can not cross boundaries: if the particle reaches a boundary, the step ends and a new step begins). This is done with the "GetVolume" method. We also care about the energy deposted in that volume and the path length of the particle in the volume. Note that the code checks the charge of the particle, and only includes the path length for charged particles.

Creating histograms

We want to record information about the locations and amounts of energy deposited in the various materials of the detector. We will use histograms and a tree to do this. The histograms are created ("booked") and the file they will be stored in is created in CreateTree. This work is done in steps 1 and step 2. However, the histograms are filled in SteppingAction.

Filling histograms

After an event is finished (no more particles in the stack), which is step 5, we need to fill the histograms with the information we have accumulated. This is done in CreateTree.

Handling of detector description (geometry)

This is done in DetectorConstruction.cc and DetectorConstruction.hh. The code here is executed during step 1 in the flow chart. It is also done in template.cfg. Look at this later file. What do you see? Also look in src/MyMaterials.cc. What do you see?

More practical running

While it is fun to see the visualization, what we really need is the histograms. We also want to run many events, not just one. To run without the visualization, do the following

- make sure you are in the right area
- make sure you have done "source g4setup.sh"
- ./dualtoy -c template.cfg -m run_50GeV_electron_N500.mac -o haha
- use root and TBrowser to look at the files in the root file it creates.

Even better, run the job on condor where

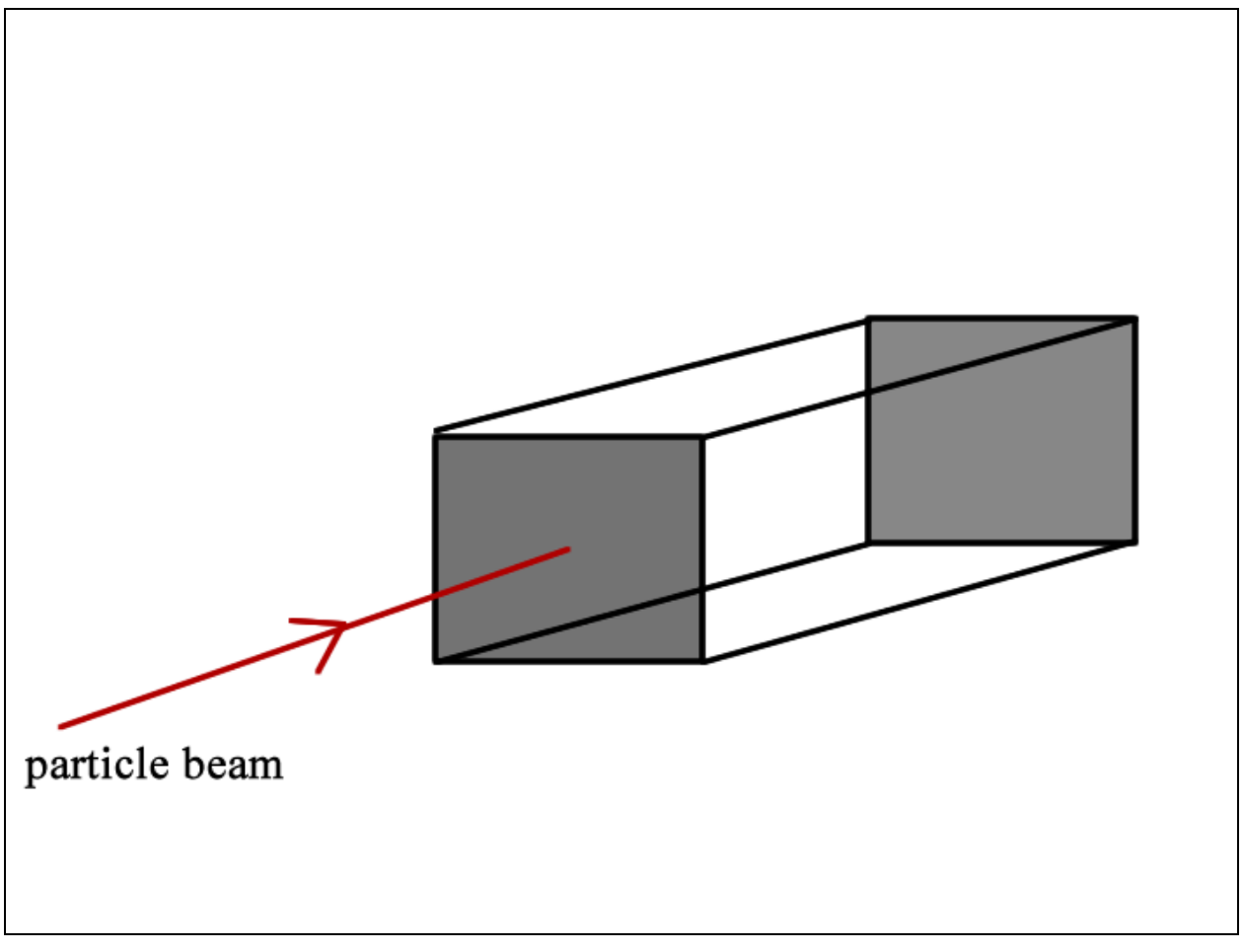
- cp /data/users/eno/dualReadout/build/condor* .
- edit the files so that they point to your areas, not mine
- submit with
    - condor_submit condor-jobs.jdl
- see if it is running with
    - condor_q -submitter your-user-name

for more on condor, see
https://sites.google.com/a/physics.umd.edu/umdt3/user-guide/submitting-analysis-jobs#TOC-Submit-jobs-to-the-cluster-using-Condor

1. The tutorial uses 50 GeV electrons. Make 1000 electrons.
2. Note that you cannot actually measure the energy deposited in the passive material, only in the active. So you will need a scale factor to apply to the energy deposited in the active area so that it gives the true energy on average after multiplying by the scale factor. How can you determine this scale factor?
3. What is the energy resolution for this calorimeter at 1 GeV? (which energy do you use to find this and why?)
4. vary the electron energy. Make a plot of the energy resolution versus electron energy.
5. Determine what fraction of the electron's energy escapes the calorimeter
6. change the particle to charged pions.
7. Determine what fraction of the pion's energy escapes the calorimeter
8. Adjust the thickness and width of the calorimeter so that it contains at least 90% of the pion's energy
9. Make a plot of the pion energy resolution versus energy, and overlay it with the one for electrons. (make sure the calorimeter is thick enough at all energies)
10. Change the passive material from lead to copper. How does this affect the resolution? Change it to Uranium. How does this affect the resolution?
11. Change the thickness of the liquid argon. How does this affect the resolutions?

Final Project

Your goal is to design the cheapest possible calorimeter that has a resolution for jets (sigma/E) for jets with energies between 20 and 2000 GeV better than 0.8/sqrt(E) and for electrons better than 0.13/sqrt(E) for electrons with energies between 10 and 200 GeV. The shape of the calorimeter should be rectangular. Assume the particles will be aimed at the center of the calorimeter, as shown below. Make the calorimeter as small as possible to save money.



Learn more

To learn more, it is best to go to the source. Look at:

- http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/index.html