Simulation on the Detector Elements of Proton Computed Tomography for Cancer Therapy

(Document)

Kavita Lalwani for pCT collaboration
Department of Physics and Astrophysics
University of Delhi, India
26 July 2012
## Contents

1 Detector Geometry and Parameters 3  
1.1 pCT-Geometry .................................................. 3  
1.2 Geometrical Parameters for pCT .............................. 4  

2 Class Structure 5  
2.1 Geant4 Class Categories ....................................... 5  
2.2 User Classes .................................................... 7  
2.2.1 Mandatory Classes ......................................... 7  
2.2.2 Optional user action classes ............................... 8  

3 Description of Macro files of pCT-Code 10  
3.1 How to define the main() program ? .......................... 10  
3.2 Description of Source and Header files ...................... 11  

4 Storing Data in Tree Format in ROOT File 13  
4.1 Variables used in Tree ........................................ 13  

5 Compilation and Running pCT-Code in Geant4 15  
5.1 Software ........................................................ 15  
5.2 CLHEP Build and Installation Procedure for Ubuntu ....... 15  
5.3 Geant4 installation procedure for Ubuntu ................... 16  
5.4 ROOT Installation ................................................ 17  
5.5 How to set Environment Variables ............................ 18  
5.6 Compile and Running pCT-code in Geant4 .................... 18  
5.7 control file “parameter.mac” .................................. 19  

6 Steps: How do we create MIRD Head in Geant4 ? 22  

7 Required Histograms ? 25  
7.1 Profile Histograms .............................................. 25  
7.1.1 Energy deposited in each tile as a function of depth(z)/Number of tile .............................. 25  
7.2 Entry and Exit Angle from Fibers ............................ 25  
7.3 Total Energy Deposited in Four Fibers ....................... 25
Chapter 1

Detector Geometry and Parameters

1.1 pCT-Geometry

Schematic of Proton Computed Tomography is shown in figure 1.

![Figure 1.1: Schematic of pCT.](image)

Our pCT Geometry consists of proton beam of kinetic energy of 200 MeV, Herman Head, is used as a target. Two fibers are used before and two after the Head phantom to measure the position of incoming and outgoing protons from the phantom. A calorimeter is used to measure the energy of outgoing proton from phantom.

In simulation, initially we started to work with pencil beam of proton of beam kinetic energy of 200MeV, because it is easier to understand the resolution for position and energy with pencil beam as compared to spread beam of protons. We have implemented Water Phantom as a target, as our body contains 97% of water and it is spherical and symmetric in shape, therefore it is easy to study the resolution for position and energy of each track. We have included four cylindrical fiber trackers made up of material polystyrene (two before phantom and two after the phantom), to measure the position of incoming and outgoing protons from the Water Phantom. Each tracker has 4 layers of fibers oriented in X and Y directions alternately, and are joined back to back through a Rohacell.

![Figure 1.2: Sketch for Fiber.](image)

To measure the energy of outgoing protons from Water Phantom, we have implemented a Calorimeter,
made up of 100 layers of scintillators of material polystyrene. An Alternate layer of Tyvek is used as reflector. To create 100 scintillator layers, We have used Replica Class in Geant4 simulation (see detail description in geantcode/src/ExN02DetectorConstruction.cc).

1.2 Geometrical Parameters for pCT

Figure 1.3: Sketch for pCT-Geometry in Geant4 simulation.

Figure 1.3 shows sketch for pCT-Geometry used in Geant4 simulation, where Black arrow shows the pencil beam of proton coming along z direction. Origin of beam is (0,0,-1500mm). We have Water phantom of diameter of 230mm (circle with blue colour) placed at z = 500mm. Than we have two upstream fibers of dimensions of 200×240 placed at 150mm and 300mm and two downstream fibers of dimensions of 240×300 at 700mm and 850mm along z. Each fiber has diameter of 0.5mm. At the end, We have a stack of scintillators called Calorimeter. The Number of scintillator layers in stack is 100 and each scintillator tile has thickness of 3mm and dimension of 270×360mm. Distance between center of phantom and the inner (upstream and downstream) trackers is 20 cm and distance between center of phantom and proton origin is 2m. Spacing between upstream tracking stations is 15 cm.

All geometrical parameters are listed in table 1.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beam Kinetic Energy</td>
<td>200 MeV</td>
</tr>
<tr>
<td>Diameter of Water Phantom</td>
<td>230mm</td>
</tr>
<tr>
<td>Diameter of Fiber</td>
<td>0.5mm</td>
</tr>
<tr>
<td>Dimensions of Fibers</td>
<td></td>
</tr>
<tr>
<td>Upstream</td>
<td>200×240mm</td>
</tr>
<tr>
<td>Downstream</td>
<td>240×300mm</td>
</tr>
<tr>
<td>Central Position of Phantom</td>
<td>500mm</td>
</tr>
<tr>
<td>Central Position of Four Fibers</td>
<td>150mm/300mm/700mm/850mm</td>
</tr>
<tr>
<td>Scintillator thickness</td>
<td>3mm</td>
</tr>
<tr>
<td>Tyvek thickness</td>
<td>0.15 mm (only 1 layer between any two tiles)</td>
</tr>
<tr>
<td>Dimensions of scintillator tile</td>
<td>270×360mm</td>
</tr>
</tbody>
</table>

Table 1.1: Geometrical Parameters for pCT.

After designing pCT-Geometry in Geant4, we have used OpenGL visualization driver to see the nice pCT-Geometry in Geant4 simulation (will see in chapter 7).
Chapter 2

Class Structure

2.1 Geant4 Class Categories

A class category contains classes which have a close relationship. The class categories and their relations are presented by a class category diagram as shown in the figure 1. Each box in the figure represents a class category, and a "uses" relation by a straight line. The circle at an end of a straight line means the class category which has this circle uses the other category.

![Figure 2.1: Geant4 Class Categories.](image)

The following is a brief summary of the role of class categories like Run, Event and track etc in Geant4 and others will be discuss later in further chapters.

1. **A Run in Geant4**: G4RunManager class manages the run.
   - A Run is represented by the G4Run class which provides summary results of each run
   - Within a Run the User cannot change:
     - The detector set-up
     - The settings of the physics processes
   - A Run can be defined as a collection of Events.
   - G4UserRunAction is an optional class to collect information before and after a Run.
   - A Run can be started with the command "BeamOn": G4RunManager* runmanager → BeamOn(numberOfEvent); The Run beam on command can be given using the UI command: /run/beamOn
• At the beginning of a run, the geometry is optimized for the navigation and cross section tables are calculated according to the materials in the simulation.

2. Event in Geant4: The Event is the basic unit of the simulation. G4EventManager class manages the flow of one Event (1 particle history).
   • At the beginning primary tracks are generated and pushed into a stack.
   • Each track is retrieved from the stack and tracked (The same happens for the secondary tracks)
   • When the stack is empty, the event is closed.
   • G4UserEventAction is an optional class to collect information before and after an Event.

The G4Event class represents an event. Its input are the list of primary vertex and particles. Its outputs are the hits and the trajectories collection.

3. Tracking a particle in Geant4: Track, Step, Step point and Trajectory

   • A track is a snapshot of a particle.
   • G4TrackingManager manages and process a Track.
   • A Track is represented by the G4Track class.
   • No track object persists at the end of the event: you must use the G4Trajectory object to record and visualize the tracks.
   • G4Track keeps information on the final status of the particle after the completion of one step.

   Step in Geant4

   • Step has two points and also delta information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
   • Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.
     • Because one step knows two volumes, boundary processes such as transition radiation or refraction could be simulated.
2.2 User Classes

![User Classes Diagram]

Figure 2.4: User Classes.

All must be instantiated in the main(). In our main(), we have to Construct G4RunManager (or derived class) and then set user mandatory classes to RunManager. We can define VisManager, (G)UI session, optional user action classes, and/or persistency manager in our main().

- G4RunManager controls the flow of the program and manages the event loop within a run. It also manages the initialization procedures.

2.2.1 Mandatory Classes

In this section, we will describe the Mandatory classes and their uses.

1. Describe Detector
   - We derive our own concrete class from G4VUserDetectorConstruction abstract base class.
   - In the virtual method Construct(),
     - Construct all necessary materials
     - Construct volumes of your detector geometry
     - Construct sensitive detector classes and set them to the detector volumes
   - Optionally we can define
     - Regions for any part of our detector.
     - Visualization attributes of detector element.

2. Select Physics Processes
   We derive our own concrete class from G4VUserPhysicsList abstract base class.
   - Define all necessary particles.
   - Define all necessary processes and assign them to proper particles.
   - Define cut-off ranges applied to the world and each region.
Physics Model
To include physical interactions of proton with matter, we have implemented the Quark-Gluon String Precompound (QGSP) model in our pCT-code. The QGSP Model is built from several component models which implements high energy inelastic scattering of hadrons by nuclei. It forms QCD strings by pairing a parton from the projectile hadron with a parton from a target nucleon. The strings are then excited by parton exchange and decayed to form final state hadrons. After the initial hadron-nucleon collision, the highly excited remnant nucleus is de-excited using the G4Precompound model. The QGSP model may be applied for incident nucleons, pions and kaons. The QGSP Model is used in following physics lists: QGSP\_BERT, QGSP\_BIC and QGSP\_HP.

We have compared the efficiency and accuracy of different models and found that QGSP-BERT gives best performance. It is Geant4 Bertini cascade model for primary protons, neutrons, pions and kaons below 10\,GeV and produced more secondary neutrons and protons yielding a better agreement to experimental data. Using QGSP\_BERT model in simulation, we have seen reduction in number of particles by about 20\% on the exit side of the water phantom as a result of nuclear inelastic events.

3. Generate primary event
   - Derive concrete class from G4VUserPrimaryGeneratorAction abstract base class.
   - Pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles.
   - Geant4 provides several generators in addition to the G4VPrimaryParticlegenerator base class:
     - G4ParticleGun
     - G4GeneralParticleSource
     - G4HEPEvtInterface, G4HEPMMCInterface

2.2.2 Optional user action classes
All user action classes, methods of which are invoked during BeamOn, must be constructed in our main() and must be set to the RunManager.

1. G4UserRunAction
   - BeginOfRunAction(const G4Run*)
     - Define histograms
   - EndOfRunAction(const G4Run*)
     - Store histograms

2. G4UserEventAction
   - BeginOfEventAction(const G4Event*)
     - Event selection
     - Define histograms
   - EndOfEventAction(const G4Event*)
     - Analyze the event

3. G4UserStackingAction
   - PrepareNewEvent()
     - Reset priority control
     - ClassifyNewTrack(const G4Track*)
       * Invoked every time a new track is pushed
       * Classify a new track – priority control
     - NewStage()
       * Invoked when the Urgent stack becomes empty
       * Change the classification criteria
* Event filtering (Event abortion)

4. **G4UserTrackingAction**
   - PreUserTrackingAction(const G4Track*)
     - Decide trajectory should be stored or not
     - Create user-defined trajectory
   - PostUserTrackingAction(const G4Track*)

5. **G4UserSteppingAction**
   - UserSteppingAction(const G4Step*)
     - Kill / suspend / postpone the track
     - Draw the step (for a track not to be stored by a trajectory)
Chapter 3

Description of Macro files of pCT-Code

3.1 How to define the main() program?

The Geant4 toolkit does not provide a main() method. The contents of main() vary according to the needs of a given simulation application and therefore must be supplied by the user. In this section we will discuss about how do we define the main() program for pCT-code. Below is the snapshot of main() program used in pCT-code (file in src/pCT.cc).

```cpp
#include "G4RunManager.hh"
#include "G4UserAction.hh"
#include "N03VisManager.hh"
#include "ExN03DetectorConstruction.hh"
#include "ExN03PhysicsList.hh"
#include "ExN03PrimaryGeneratorAction.hh"
#include "ExN03RunAction.hh"
#include "ExN03EventAction.hh"
#include "ExN03SteppingAction.hh"
#include "ExN03SteppingVerbos.hh"
#include "ExN03AnalysisManager.hh"
#include "QGSP_BERT.hh"

int main(int argc, char** argv)
{
  // Construct the default run manager
  G4RunManager * runManager = new G4RunManager;

  // Set mandatory initialization classes
  ExN03DetectorConstruction* detector = new ExN03DetectorConstruction;
  runManager->SetUserInitialization(detector);
  runManager->SetUserInitialization(new QGSP_BERT);
  or
  runManager->SetUserInitialization(new N03PhysicsList);

  // visualization manager
  G4VisManager* visManager = new G4VisManager;
  visManager->Initialize();

  // set user action classes
  runManager->SetUserAction(new N03PrimaryGeneratorAction(detector));
  runManager->SetUserAction(new N03RunAction);
  runManager->SetUserAction(new N03EventAction);
  runManager->SetUserAction(new N03SteppingAction);

  // get the pointer to the User Interface manager
  G4UImanager* UI = G4UImanager::GetUIpointer();
  if(argc==1)/ Define (G)UI terminal for interactive mode
  (G)UI->ApplyCommand("/control/execute prerun.g4mac");
  session->sessionStart();
  delete session;
  else // Batch mode
  G4String command = "/control/execute ";
  G4String fileName = argv[1];
  UI->ApplyCommand(command+fileName);
}

// jobs termination
delete visManager;
delete runManager;
return 0;
```

Figure 3.1: main() program.
In main(), we first create an instance of the G4RunManager class. It controls the flow of the program and manages the event loop(s) within a run. When G4RunManager is created, the other major manager classes are also created. They are deleted automatically when G4RunManager is deleted. The run manager is also responsible for managing initialization procedures, including methods in the user initialization classes. Through these, the run manager must be given all the information necessary to build and run the simulation, including

- How the detector should be constructed.
- All the particles and all the physics processes to be simulated.
- How the primary particle(s) in an event should be produced.
- Any additional requirements of the simulation.

In the main() the lines

runManager→SetUserInitialization(new ExN03DetectorConstruction);
runManager→SetUserInitialization(new ExN03PhysicsList);

create objects which specify the detector geometry and physics processes, respectively, and pass their pointers to the run manager. ExN03DetectorConstruction is an example of a user initialization class which is derived from G4VUserDetectorConstruction. This is where we describe the entire pCT-detector setup, including

- its geometry.
- the materials used in its construction.
- a definition of its sensitive regions.

Similarly ExN03PhysicsList is derived from G4VUserPhysicsList and requires to define

- the particles to be used in the simulation.
- the range cuts for these particles.
- all the physics processes to be simulated.

The next instruction in main()

runManager→SetUserAction(new ExN03PrimaryGeneratorAction);

creates an instance of a particle generator and passes its pointer to the run manager. ExN03PrimaryGeneratorAction is an example of a user action class which is derived from G4VUserPrimaryGeneratorAction. In this class, we describe the initial state of the primary event. This class has a public virtual method named generatePrimaries() which will be invoked at the beginning of each event.

After setting the PrimaryGeneratorAction Class, we now set the user action classes like ExN03RunAction, ExN03EventAction and ExN03SteppingAction etc via runManager.

To create interface manager we use following command in main()

G4UImanager* UI = G4UImanager::getUipointer(); This line issue the commands to the program.

We have also included an instance of analysis manager to book the analysis.

### 3.2 Description of Source and Header files

Now we will discuss source and header files used for pCT-Code

1. **ExN03DetectorConstruction** (header file) (source file)
   - derived from G4VUserDetectorConstruction
   - definitions of single materials and mixtures
   - Solids: Box, Tubs, cylindrical, spherical, ellipsoidal volumes etc.
   - G4PVPlacement without rotation and with Rotation.
   - Interactivity: change detector size, material, magnetic field. (→messenger class)
   - visualization
2. **ExN03PhysicsList** (header file) (source file)
   - derived from G4VUserPhysicsList
   - definition of proton, gamma, leptons and ions.
   - Processes: Transportation, Ionization, EM processes and nuclear inelastic scattering. Also included QGSP Model (see chapter 2).
   - Interactivity: SetCut, process on/off. (→ messenger class)

3. **ExN03PrimaryGeneratorAction** (header file) (source file)
   - derived from G4VPrimaryGeneratorAction
   - construction of G4ParticleGun
   - primary event generation via particle gun
   - Interactivity: shoot proton beam (pencil beam and spread beam) (→ messenger class)

4. **ExN03RunAction** (header file) (source file)
   - derived from G4VUserRunAction
   - draw detector and tracks
   - Interactivity: SetCut, process on/off.
   - Interactivity: change detector size, material, magnetic field.

5. **ExN03EventAction** (header file) (source file)
   - derived from G4VUserEventAction
   - store trajectories
   - print end of event information (energy deposited, etc.)

6. **ExN03SteppingAction** (header file) (source file)
   - derived from G4VUserSteppingAction
   - collect energy deposition, etc.

7. **ExN03VisManager** (header file) (source file)
   - derived from G4VisManager
   - Example Visualization Manager implementing virtual function

8. **ExN03SteppingVerbose** (header file) (source file)
   - derived from G4SteppingVerbose
   - give information regarding position and energy deposited at each step.

9. **ExN03AnalysisManager** (header file) (source file)
   - Create ROOT file, Tree and set their branches in order to store all variables in Tree Format.
   - Fill the Tree in ExN03EventAction.cc file to store each information on event by event basis.

10. **ExN03DetectorSD** (header file) (source file) **From Andrew**
Chapter 4

Storing Data in Tree Format in ROOT File

We store simulated data for example energy deposited, position coordinates (x, y and z) for each hit in all events in Tree (figure 5.1)

![Tree for storing simulated data.](image)

Figure 4.1: Tree for storing simulated data.

4.1 Variables used in Tree

- `event_id`: Number of events
- Flags for Particle. 1 for proton and 0 for secondary.
  1. FlagFibParticle: flags for particles in fibers.
  2. FlagWaterParticle: flags for particles in phantom (water/head).
  3. FlagCalParticle: flags for particles in calorimeter.
- Flags for Detector Elements:
  1. FlagFiblayer: 1 for fiber and 0 for Roha
  2. FlagCallayer: 1 for scintillator and 0 for tyvek
• Flags for Different Processes:
  1. FlagFibProcess: Flags for different interactions in fibers.
  2. FlagCalProcess: Flags for different interactions in calorimeter.

Note: Flags different interactions are:
  1. Flag 0 for Transportation
  2. Flag 1 for Ionization
  3. Flag 2 for Elastic
  4. Flag 3 for Proton Inelastic
  5. Flag 4 for Neutron Inelastic
  6. Flag 5 for Multi Coulomb Interaction (msc)
  7. Flag 6 for Ion Ionization (ionIon)
  8. Flag 7 for Compton Effect
  9. Flag 8 for Photo Electric Effect
  10. Flag 9 for Electron Ionization

• Deposited Energy of each hit:
  1. Edepfib: Deposited energy in fibers/roha.
  2. Edepcal: Deposited energy in calorimeter.
  3. Edepwater: Deposited energy in water/Head phantom.

• Position of each hit:
  1. PosXfib, PosYfib and PosZfib: Position X, position Y and position Z in fiber respectively.

• Counts for each hit:
  1. countfib: Counts the number of hits in fiber/roha.
  2. countwater: Counts the number of hits in phantom(water/head phantom).
  3. countcal: Counts the number of hits in calorimeter(scintillator/tyvek).

Variables for Fibers: From Andrew
  1. NFib:
  2. FibTrackID:
  3. LayerID:
  4. Event:
  5. CopyNo:
  6. XPosFib:
  7. YPosFib:
  8. ZPosFib:
  9. EDepFib:
  10. EDepFib:
  11. HitTimeFib:
  12. FibLayerFlag:
  13. FibProcessFlag
  14. FibProtonFlag: 
Chapter 5

Compilation and Running pCT-Code in Geant4

In this chapter, we will begin with the software version needs to be installed on our system and then we will discuss of setting some environmental variables. We will then discuss how do we download pCT-Code and compile it in Geant4. At the end we will show the results as Histograms.

5.1 Software

We should install following software packages:

- Geant4 package: geant4.9.4.p02
- CLHEP package: CLHEP2.1.0.1
- ROOT: 5.30.02a

In the following sections, we will describe the installation procedure of above softwares for different Operating system for example Ubuntu and RHEL.

5.2 CLHEP Build and Installation Procedure for Ubuntu

1. Install package from web(http://proj-clhep.web.cern.ch/proj-clhep/DISTRIBUTION/tarFiles/clhep-2.1.0.1.tgz)
2. Create a folder called G4install in your home directory, i.e., /home/username.
3. Unzip clhep-2.1.0.1.tgz into /home/username/G4install.
   Note: Installation directory is up to user where he wants to install the packages.
4. Cd /home/username/G4install/2.1.0.1/CLHEP
5. ./configure –prefix /home/username/G4install/2.1.0.1/
   Note: adapt prefix path according to our own installing directory. The configure script checks for required programs and libraries,and creates some files, E.g. makefiles and directories.
6. If no error occured in the configure process, one can start to build the CLHEP package using the “make” command.
   When make ends, you can see:
   ./build-header
   make[1]: Leaving directory ”/home/username/G4install/2.1.0.1/CLHEP”
7. Once the package was compiled successfully, CLHEP can be installed using the “make install” command.
8. We see following three directories inside 2.1.0.1:
(a) include: contains (in a defined directory tree structure) the C++ header files of CLHEP.
(b) lib: contains the (static and shared) CLHEP libraries.
(c) bin: contains configure scripts.

Finally, to save some disk space, we can remove the tar-ball.

5.3 Geant4 installation procedure for Ubuntu

Steps for building and installation of Geant4:

1. Install package from web: http://geant4.cern.ch/support/source/geant4.9.4.p02.tar.gz
2. Move it to /home/username/G4install.
3. Unzip tar-ball via command tar zxvf geant4.9.4.p02.tar.gz
4. To start the build process, execute ./Configure -build inside the geant4.9.4.p02 directory. Initially we get some general information (Screen 1).

![Figure 5.1: Screen 1, output for general information.]

5. As the next step the Configure script tries to determine our system and compiler (Screen 2).

![Figure 5.2: Screen 2 for system and compiler.]

6. We aim for a local installation and we do not care about portability, thus we accept the default ('n’) in the next step (Screen 3).

![Figure 5.3: Screen 3 for portability.]

7. Then we have to specify the source path and our install directory: "'/home/username/G4install/geant4.9.4.p02"."
8. We are then asked, if we want to put all header files in one directory: ....as you prefer....

9. We then have to specify the path to the data directories: “/home/username/G4install/data”

10. Then, we need to specify the path of the CLHEP installation (Screen 4). CLHEP_BASE_DIR = /home/username/G4install/2.1.0.1

![Screen 4 for CLHEP path.](image)

11. The next steps are to determine, if one wants static and/or shared libraries, and several questions concerning the visualization setup. For the visualization we can invoke the drivers we want. For the current version we have used OpenGL visualization driver.

12. Finally, we will see Screen 5.

![Screen 5 for final configuration.](image)

13. It may take a while until the libraries are built...

14. Once the build process is finished, install the package by executing: ./Configure -install. Finally Geant4 is installed in the directory: /home/username/G4install/geant4.9.4.p02.

NOTE: To set up environment, we can use the env.sh/env.csh script (depending which shell we use), which are located in the directory: /home/username/G4install/geant4.9.4.p02

### 5.4 ROOT Installation

1. Download Source Code (http://root.cern.ch/root/)

2. Installation via following commands:
   - ./configure linux
   - make
   - make install
5.5 How to set Environment Variables

To set up environment, we use file env.sh and set following environment variables:

- **G4INSTALL**: path where the Geant4 toolkit tree is installed.
  
  ```bash
  export G4INSTALL=/home/username/G4install/geant4.9.4.p02
  ```

- **CLHEP**: path to the CLHEP installation
  
  ```bash
  export CLHEP_BASE_DIR=/home/username/G4install/CLHEP
  ```

- **G4WORKDIR**: path of the user’s working directory
  
  ```bash
  export G4WORKDIR=/home/username/g4work
  ```

- **ROOTSYS**: path to root
  
  ```bash
  export ROOTSYS=/home/username/root
  ```

Installation directory is up to user where he wants to install the packages.

5.6 Compile and Running pCT-code in Geant4

We first describe how one can download pCT-code, change the files in code and upload pCT-code on nicadd CVS.

1. ssh username@t3nfs.nicadd.niu.edu
2. cd /xdata/username
3. mkdir pct
4. cd pct
5. cvs co g4dev, using this command we copy the most recent and changeable version of g4dev package (all files under g4dev folder on CVS server) into the local pct directory on /xdata/username/
6. To add new files into pCT-Code, use following commands: cp filename /xdata/username/pct/g4dev/pCT_QGSP_BERT (latest code) cvs add filename /xdata/username/pct/g4dev/pCT_QGSP_BERT (latest code)
7. Do update via command: cvs update
8. Commit the changes we have done
cvs ci, we get an e-mail confirmed all changes were done by this commit.
9. Give a new tag for example ”v00-00-01” to g4dev via command: cvs rtag v00-00-01 g4dev

Following are the steps used for the compilation and running pCT-code in Geant4.

1. We have already installed Geant4, CLHEP and ROOT on our local machine (discussed in section 5.1). Note: None of them (CLHEP/ROOT/Geant4) should be installed on CVS except pCT-code on which we are working on.
2. Set environment variables (CLHEP_BASE_DIR, ROOTSYS and Geant4) in env.sh
3. Do source env.sh in working directory
4. Give the name of executable (current file: pCT.cc) of our program in GNUmakefile which is placed inside the working directory.
5. Do “make clean” and then “make” inside the working directory to compile pCT-code
6. use command “/home/username/geant4/bin/Linux-g++/name of executable file (pCT) name of macro for external commands (parameter.mac)”
5.7 control file “parameter.mac”

To change the geometrical parameters for example scintillator tile size, dimensions, material and thickness of each detector element from outside pCT-Code, we use a messenger class called “DetectorMesenger”. After implementing geometrical parameters in Messenger class, add header file “DetectorMessenger.hh” in Detector Construction file and than use a macro file with extension “.mac” (current file: parameter.mac) inorder to change those parameters externally while running the code in Geant4 simulation. We create control file (latest one is “parameter.mac”) and set following commands to change the geometrical parameters outside the pCT-code.

- Set world sizes
  1. /N03/det/setWorldSizeX 800 mm
  2. /N03/det/setWorldSizeY 800 mm
  3. /N03/det/setWorldSizeZ 3050 mm

- Set number of layers /N03/det/setNbOfLayers 100

- Set calorimeter sizes (note: calor thickness = layer thickness*no of layers, calculated in program)
  1. /N03/det/setCalorSizeX 270 mm
  2. /N03/det/setCalorSizeY 360 mm

- Set layer sizes (note: layer thickness = tyvek1 thickness + scint thickness + tyvek2 thickness, calculated in program)
  1. /N03/det/setLayerSizeX 270 mm
  2. /N03/det/setLayerSizeY 360 mm

- Set material for world (Note: material for calor is same as for world i.e Air), scintillator, tyvek and water
  1. /N03/det/setWaterMat water
  2. /N03/det/setScintMat Scintillator
  3. /N03/det/setGapMat1 Tyvek1

- Set thickness of scintillator and tyvek
  1. /N03/det/setScintThick 3 mm
  2. /N03/det/setGapThick1 0.15 mm

- Set position z for fiber 1, fiber 2 fiber 3, fiber 4, water and calorimeter
  1. /N03/det/setPosZWater 500 mm
  2. /N03/det/setPosZFiber1 150 mm
  3. /N03/det/setPosZFiber2 300 mm
  4. /N03/det/setPosZFiber3 700 mm
  5. /N03/det/setPosZFiber4 850 mm
  6. /N03/det/setPosZCalor 1067.5 mm

- Set for Pencil Beam: off and Spread Beam: on
  1. /N03/gun/rndm off

Followings are the set of commands in external macro (parameter.mac) to see nice visualize pCT-Geometry in OpenGL driver (Graphics).

- To open a window of graphics in OpenGL use command:
  /vis/open OGLIX 800x600-0+0
• To draw detector volume
  /vis/drawVolume

• Specify view angle: /vis/viewer/set/viewpointThetaPhi 90. 0.

• Specify zoom value: /vis/viewer/zoom 2

• Draw smooth trajectories at end of event /vis/scene/add/trajectories smooth

• Draw hits at end of event: /vis/scene/add/hits

• if you prefer refreshing each event, comment out next line /vis/scene/endOfEventAction accumulate

• set verbose /tracking/verbose 1

• set number of events /run/beamOn 1

Figure 5.6 and 5.7 show the visualization of pCT-Geometry in Geant4 where Water phantom with pencil beam (1 event) and spread beam (100 events) can be seen. After completion of program in Geant4, we see the file with extension “.root” in our working directory. we open this root file via commands:

root -l "filename"

TBrowser b;

We see figure 5.8 as an output (from ROOT file) where we have stored all variables for example event id, deposited energy, position of each hit etc in the form of different branches in Tree “pctGeantTree”. All variables are explained in chapter 4. Double clicking on any branch, we see the histogram of corresponding variable. See for example event id which shows the number of events (1k) stored in Tree (figure 5.9).
Figure 5.8: ROOTtree for 1000 events.

Figure 5.9: Histogram for total number of events (1k).
Chapter 6

Steps: How do we create MIRD Head in Geant4?

Following are the steps used to understand the construction of MIRD Head in Geant4:

Figure 6.1: MIRD Head in geant4
Figure 6.2: MIRD Head in geant4

Step5 (Union of head1, head2)

Step6 (Union of head1, head2, head3)

Step6 (Union of head1, head2, head3, white colour)

Step7 (Insert skull in head, ellipsoid, no rotation)

Step8 (Insert brain in head, ellipsoid, no rotation)

Step8 (brain with white colour)

Figure 6.3: MIRD Head in geant4
Figure 6.4: MIRD Head in geant4

Figure 6.5: MIRD Head in geant4
Chapter 7

Required Histograms?

7.1 Profile Histograms
7.1.1 Energy deposited in each tile as a function of depth(z)/Number of tile
7.2 Entry and Exit Angle from Fibers
7.3 Total Energy Deposited in Four Fibers