Project in GEANT4 training course

Jonatan Adolfsson
Department of Physics, Lund University
E-mail: jonatan.adolfsson@hep.lu.se

9 November 2018

Contents

1 Introduction

The purpose of this exercise is to simulate a container scanner using the GEANT4 tool kit. The specifications are as follows: a container of dimensions $10 \times 3 \times 3$ m with 5 mm thick aluminium walls is placed in the scanner. This consists of one 10 cm thick x-ray/neutron detector at one of the long edges, and two 10 cm thick muon detectors, one above and one below the scanner. The distance between the container and each detector is 10 cm. The x-ray and muon detectors are made of CsI, whereas a plastic scintillator material is used for the neutron detector (the material should be toggled depending on beam type). For scanning, either 5 MeV x-rays or 15 MeV neutrons are used. These are fired at random positions normal to the container on the opposite side of the detector. Moreover, atmospheric muons are also used for the scan. These are assumed to all be negative, have an energy of 4 GeV, and hit the setup at random positions from straight above. The x-ray/neutron detector measures deposited energy, i.e. it acts as a calorimeter, whereas the muon detectors measure scattering angle between the two detectors.

Inside the container, it is required to place at least four objects of various sizes and materials, and simulate the performance of each detector for finding these objects. In my simulation, the container is operated by a dedicated smuggler, who is trying to use this container to smuggle one human, 16 boxes full of strong alcohol, one large box filled with firearms and one large piece of nuclear fuel (or is it a nuclear warhead?).

2 Implementation

2.1 Main programme

The run manager, scoring manager, detector construction, physics list, action initialisation, user interface, and visualisation are all initialised from the main programme, see source code in Appendix A. This programme accepts up to two command-line arguments. The first one is an integer, which is used to toggle the particle type, '0' (or no argument) for γ:s, '1' for neutrons, or '2' for muons. The second argument is optional, and should specify the name of an external macro, if one wants to execute the code without visualisation. This is particularly useful when simulating a large number of events.

If multi-threading is allowed for the current GEANT4 version (this is the case for the machine where I have run the code), a `G4MTRunManager` is used. Otherwise a `G4RunManager` is used. A scoring manager is set up to activate command based scoring. The geometry is set up by creating a `DetectorConstruction` object, which accepts a boolean defined by the beam type, in order to get the correct detector material for the calorimeter. The physics list is obtained from the environment variable `PHYSLIST`. In this simulation, this is set to `FTFP_BERT_HP` prior to execution, in order to get an accurate description of interactions between neutrons and materials. The action initialisation is set by creating an `ActionInitialization` object, where settings specific for this simulation are defined (see Sec. 2.3). Finally, an `UIManager` is created, which either executes the macro specified from the command line, or (if no argument is provided) starts a graphics user interface (GUI) for the detector, with visualisation of the setup.

2.2 Detector construction

The `DetectorConstruction` class overrides the abstract class `G4VUserDetectorConstruction`. In the constructor, the boolean variable used for specify an x-ray beam (as opposed to neutrons) is passed to a class attribute with the same purpose. This is in turned used where applicable in the implementation. The actual construction is implemented in the `Construct` method, where first all materials are defined, then all geometries, and finally visualisation attributes for the GUI. In the method `ConstructSDandField`, all sensitive detectors (SDs) are initialised and then connected to the logical volumes of the detector cells, which are initialised in `Construct` (no fields are used here). Here, three SDs are used: two muon detectors for measuring coincidences between muon hits, and one calorimeter for measuring energy deposited by γ:s or neutrons. The muon SDs are implemented in the `MuonHodoscopeSD` class, and store hit time and global position of the cell that has fired. Only the first hit (earliest time in the simulation) in each tile is recorded. The calorimeter is implemented in the `CalorimeterSD` class, and stores energy deposit and global position of the cell.

2.2.1 Container and detector geometries

The world volume is just a box, with dimensions $12 \times 6 \times 6$ metres, with its origin defined by the centre. The container is also centred at the origin, and is made of a solid aluminium box. The interior is implemented as another box, made of air and placed inside the aluminium box, with each half-side being 5 mm shorter than of the exterior box; see the following source code:

```
// Container geometry
G4VSolid* containerSolid = new G4Box("containerBox", 5.*m, 1.5*m, 1.5*m);
fContainerLogical
  = new G4LogicalVolume(containerSolid, aluminium, "containerLogical");
new G4PVPlacement(0, G4ThreeVector(0.,0.,0.), fContainerLogical,
  "containerPhysical", worldLogical, false, 0, checkOverlaps);
```

```
G4VSolid* interiorSolid = new G4Box("interiorBox", 4.995*m, 1.495*m, 1.495*m);
G4LogicalVolume* interiorLogical
  = new G4LogicalVolume(interiorSolid, air, "interiorLogical");
new G4PVPlacement(0, G4ThreeVector(0.,0.,0.), interiorLogical,
  "interiorPhysical", fContainerLogical, false, 0, checkOverlaps);
```

The detectors are made of boxes with dimensions $10 \times 3 \times 0.1$ metres, made of CsI (μ and γ detectors), or scintillator (neutron detector; this is toggled by the boolean variable used for specifying beam type). The same `G4Solid` is used for all detectors, but placed in different logical volumes. The muon detectors are rotated 90° during the placement phase, where they are attached to the world volume at the correct positions. Each detector is filled with detector cells of dimensions $5 \times 5 \times 10$ cm, in total 12 000 cells per detector. This is done by using parametrised volumes.

2.2.2 Objects inside container

The human is modelled by a volume made of water, consisting of four parts. A box of dimensions $150 \times 16 \times 30$ cm are used for the torso and legs, a solid sphere (`G4Orb`) of 20 cm diameter for the head, and two cylinders of length 70 cm and diameter 8 cm for the arms. There is a small gap between the head and the torso (the human has a neck), as well as between arms and body, and the arms are slightly rotated (about 17°) around the shoulders. The body parts are assembled into a box made of air, which is placed inside the container. This is rotated by 30° relative to the direction of the container.

The boxes with alcohol are created by using a parametrised volume. This consists of a box which for simplicity is made of air, of dimensions $2 \times 1 \times 1$ m, which is put behind the human, such that a beam hitting the human may also penetrate through the box. This is filled with 16 identical boxes of dimensions $0.5 \times 0.5 \times 0.5$ m, by using a parametrisation, also made of air. Each box is then divided into 128 cells (boxes made of air) of dimensions $6.25 \times 6.25 \times 25$ cm, using another parametrisation. The bottles of alcohol are put into these cells. Each bottle is generated as a `G4Polycone` with base diameter 6 cm, top diameter 2 cm, and height 24 cm, and made of glass. Another slightly smaller polycone, made of ethyl alcohol (used for imitating some really strong stuff, probably not even legal), is put inside each bottle, such that it everywhere is surrounded by 2 mm of glass. See source code below:

```
G4double ri[3] = {0., 0., 0.};
G4RotationMatrix* rot = new G4RotationMatrix();
rot->rotateX(-pi/2*rad);

// Boxes with alcohol (filled with bottles of REALLY strong stuff)
G4VSolid* alcoholBoxesSolid = new G4Box("alcoholBoxesBox", 1.*m, 0.5*m, 0.5*m);
G4LogicalVolume* alcoholBoxesLogical = new G4LogicalVolume(alcoholBoxesSolid,
  air, "alcoholBoxesLogical");
new G4PVPlacement(0, G4ThreeVector(3.5*m, -0.995*m, 0.5*m), alcoholBoxesLogical,
  "alcoholBoxesPhysical", interiorLogical, false, checkOverlaps);
// individual boxes, parameterised
G4VSolid* singleBoxSolid = new G4Box("singleBoxBox", 0.25*m, 0.25*m, 0.25*m);
G4LogicalVolume* singleBoxLogical = new G4LogicalVolume(singleBoxSolid, air,
  "singleBoxLogical");
G4VPVParameterisation* singleBoxParam = new BoxParameterisation();
new G4PVParameterised("singleBoxPhysical", singleBoxLogical,
  alcoholBoxesLogical, kXAxis, 16, singleBoxParam);
// cells with bottles of alcohol, parameterised
G4VSolid* bottleCellSolid = new G4Box("bottleCellSolid", 3.125*cm, 12.5*cm,
  3.125*cm);
G4LogicalVolume* bottleCellLogical = new G4LogicalVolume(bottleCellSolid, air,
  "bottleCellLogical");
G4VPVParameterisation* bottleCellParam = new BottleCellParameterisation();
new G4PVParameterised("bottleCellPhysical", bottleCellLogical, singleBoxLogical,
  kXAxis, 128, bottleCellParam);
G4double r1[3] = {1.*cm, 3.*cm, 3.*cm};
G4double z1[3] = {0., 6.*cm, 24.*cm};
G4VSolid* bottleSolid = new G4Polycone("bottlePolycone", 0, 2*pi*rad, 3,  z1, ri,
  r1);
G4LogicalVolume* bottleLogical = new G4LogicalVolume(bottleSolid, glass,
  "bottleLogical");
new G4PVPlacement(rot, G4ThreeVector(0., 12.*cm, 0.), bottleLogical,
  "bottlePhysical", bottleCellLogical, false, 0, checkOverlaps);
G4double r2[3] = {0.81*cm, 2.8*cm, 2.8*cm}; // 2 mm thick glass
G4double z2[3] = {0., 5.8*cm, 23.6*cm};
G4VSolid* alcoholSolid = new G4Polycone("alcoholPolycone", 0, 2*pi*rad, 3,  z2, ri,
  r2);
G4LogicalVolume* alcoholLogical = new G4LogicalVolume(alcoholSolid, alcohol,
  "alcoholLogical");
new G4PVPlacement(0, G4ThreeVector(0., 0., 2.*mm), alcoholLogical,
  "alcoholPhysical", bottleLogical, false, 0, checkOverlaps);
```

The box with firearms consists of a large box made of plastic (polystyrene; the firearms are assumed to be wrapped very tightly), of dimensions $2 \times 0.6 \times 0.4$ m. This is in turn divided into 160 equal cells of dimensions $10 \times 10 \times 30$ cm. Each cell contains a firearm, modelled as a steel tube of length 25 cm and diameter 3 cm. A hole is drilled through each tube by placing a cylinder of air inside it, with diameter 1 cm.

Finally, the nuclear piece consists of a cylinder of lead (used for shielding) with a conic top, with diameter 40 cm and height 70 cm. A cylinder of uranium with a diameter of 25 cm and 35 cm height is placed inside the shielding (originally, I tried to use plutonium, but this was not allowed by the physics list used). This object is placed quite far from the other objects in the container.

A screen shot of the setup is shown in Fig. 1.

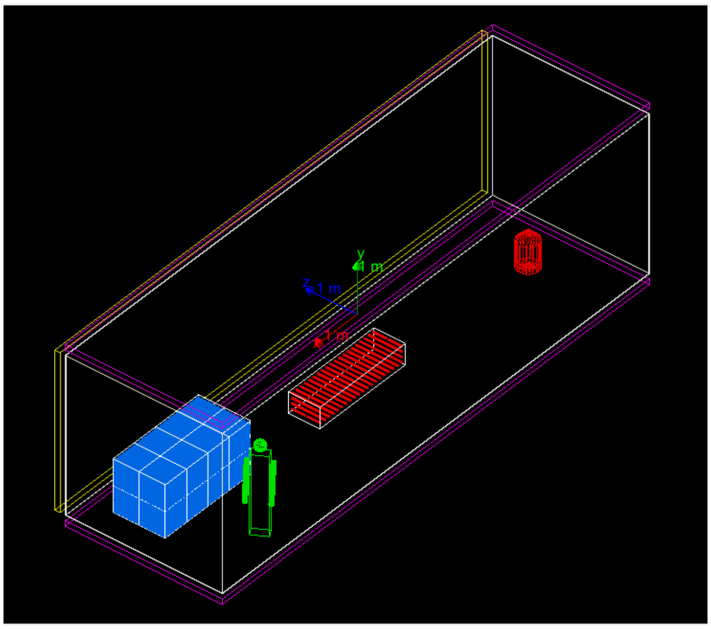


Figure 1 – Visualisation of the setup used in this project. The thicker white frames mark the edges of the container. The yellow wall by the side of the container is a calorimeter, used for detecting x-rays or neutrons. Similarly, the purple ones on top and bottom are muon detectors, used for tracking muons passing through the container. The green figure near the front models a human. The boxes near the human are filled with bottles of alcohol (blue). The box in the centre is filled with firearms (red). Finally, the conic red container at the far end is filled with nuclear fuel.

2.3 Action initialisation

The `ActionInitialization` class overrides the abstract class `G4VUserActionInitialzation`. Here, an integer is used as a constructor argument to set the beam type, which sets an attribute with the same purpose. All simulation settings are configured by the `Build` method. To configure the beam options, a custom `PrimaryGeneratorAction` object is created, see details below. Using this, the kinetic energy, direction, starting position, and particle type of the beam are set, following the specifications. The (initial) starting position is set to two metres from the origin; at the side of the container for the x-rays/neutrons, or just above the top detector for the muons. The x-ray/neutron beam is pointing in the positive z direction in Fig. 1, whereas the muon "beam" is pointing downwards (negative y direction). The generator action object is added to the configuration by calling `SetUserAction` from the mother class. See source code below:

```
void ActionInitialization::Build() const {
  PrimaryGeneratorAction* action = new PrimaryGeneratorAction();
  G4String particleType;
  G4double energy = 0; // kinetic energy
  G4ThreeVector beamAngle(0, 0, 1); // beam direction
  G4ThreeVector beamStartPosition(0, 0, -2.*m);
  switch (fBeamOption) {
  case 0: // x-ray beam
    particleType = "gamma";
    energy = 5.*MeV;
    break;
  case 1: // neutron beam
    particleType = "neutron";
    energy = 15.*MeV;
    break;
  case 2: // muon beam
    particleType = "mu-";
    energy = 4.*GeV;
    beamAngle = G4ThreeVector(0, -1, 0);
    beamStartPosition = G4ThreeVector(0, 2.*m, 0);
    break;
  }
  action->SetKineticEnergy(energy);
  action->SetParticleType(particleType);
  action->SetAngle(beamAngle);
  action->SetGunPosition(beamStartPosition);
  SetUserAction(action);
  SetUserAction(new RunAction);
  SetUserAction(new EventAction);
  G4StateManager::GetStateManager()->SetExceptionHandler(new
    MyExceptionHandler());
}
```

The `RunAction` and `EventAction` classes are custom classes for configuring the run and event action, respectively, see below. The reason for using a custom exception handler is that this is configured to suppress excessive warning messages from the neutron simulation, which would otherwise significantly slow down simulation speed and even cause a bad allocation exception during long simulations.

2.3.1 Primary generator action

The `PrimaryGeneratorAction` overrides the abstract class `G4VUserPrimaryGeneratorAction`. This class is used both for configuring the beam, as described above, and for generating primary particles in each event, which is done in the `GeneratePrimaries` method. Here, the position is first modified, so that the particle gun fires from a random position in the xy plane for the γ/neutron beam, or the xz plane for the muon beam, and then the primary vertex is generated.

2.3.2 Run and event action

The `RunAction` class overrides the abstract class `G4VUserRunAction`. This is used for initialising histograms and n-tuples for storing information from each event online, which later can be analysed offline. This is done in the constructor, where the analysis manager is called for creating the objects. Since command based scoring is used for analysing the results from the calorimeter, only data from the muon detectors are recorded here. Two histograms are used for recording hits in the two muon detectors as a function of x and z. This is done as a sanity check, and these histograms are not used in the actual analysis. For the analysis, an n-tuple object is created, which uses integer values for storing the number of hits from each event in each detector, and double values for storing the coordinates and time of the first hit in each detector from each event.

The `EventAction` action class overrides the abstract class `G4VUserEventAction`. This is used for filling the objects created by the run action. Two methods are overridden, `BeginOfEventAction` and `EndOfEventAction`. These are both called for each event. `BeginOfEventAction` is used for fetching the hits collections from the muon detectors. In `EndOfEventAction`, the histograms are filled and number of hits are recorded. Moreover, the method loops over all hits in each detector and records information of the first hit (earliest time in the simulation) in each detector. This is done since the n-tuple data structure only allows to store a single value for each variable from each event. The reason for using the first hit in the lower detector is that in case of several hits, it is most likely that the first one is from the hardest particle. Since most materials have quite low stopping power for muons, and these are quite energetic, it is very likely that the hardest particle indeed is the scattered muon.

2.4 Command based scoring

To obtain results from the calorimeter, a command based scorer is used. This is done by running the macro `scoring.mac`, which is executed from the GUI by using the command `/control/execute scoring.mac`. This macro creates a mesh over the detector and scores the energy deposited in each tile (the pixel size is also set in the macro). This is followed by disabling visualisation, running a simulation in the background, re-enabling visualisation, changing the view and finally drawing the scorer. See source code below:

```
# define scoring mesh
#
/score/create/boxMesh boxMesh_1
#
# Create a mesh large as the detector
/score/mesh/boxSize 5. 1.5 0.05 m
# Position it over the box
/score/mesh/translate/xyz 0 0 1.65 m
# mesh voxel size of 5 cm
/score/mesh/nBin 200 60 2
#
/score/quantity/energyDeposit eDep
/score/close
#
/score/list

/vis/disable
/run/verbose 1
/run/printProgress 5000
# Simulating 100000 events. Change this if running longer simulations
/run/beamOn 100000
/vis/enable

/vis/viewer/zoom 2
/vis/viewer/set/viewpointThetaPhi 0 0
/control/execute draw.mac
```

The last line executes another macro, `draw.mac`, which visualises the results from the scoring.

2.5 Offline analysis

For the muons, the scattering angle is obtained by analysing the output by using a `ROOT` macro. The output from the simulation is a number of `root` files, each from a different thread. These need to be merged in order to analyse all data efficiently. The analysis is run over all events. In order to obtain an average of the scattering angle in each detector tile, a `TProfile2D` histogram is filled. If there are no hits in either of the detector, the event is skipped (this means that the scattered muon has not hit the lower detector). Otherwise, the scattering angle θ is obtained as

$$\theta = \arctan\left(\frac{\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}}{\Delta y}\right),$$

where Δx and Δz are the distances in the xz-plane between the two hits, and Δy is the distance between the two detectors, which is 3.3 m. The angle θ is mapped to the position of the incoming muon, i.e. the hit position in the upper detector.

3 Results

3.1 Event displays

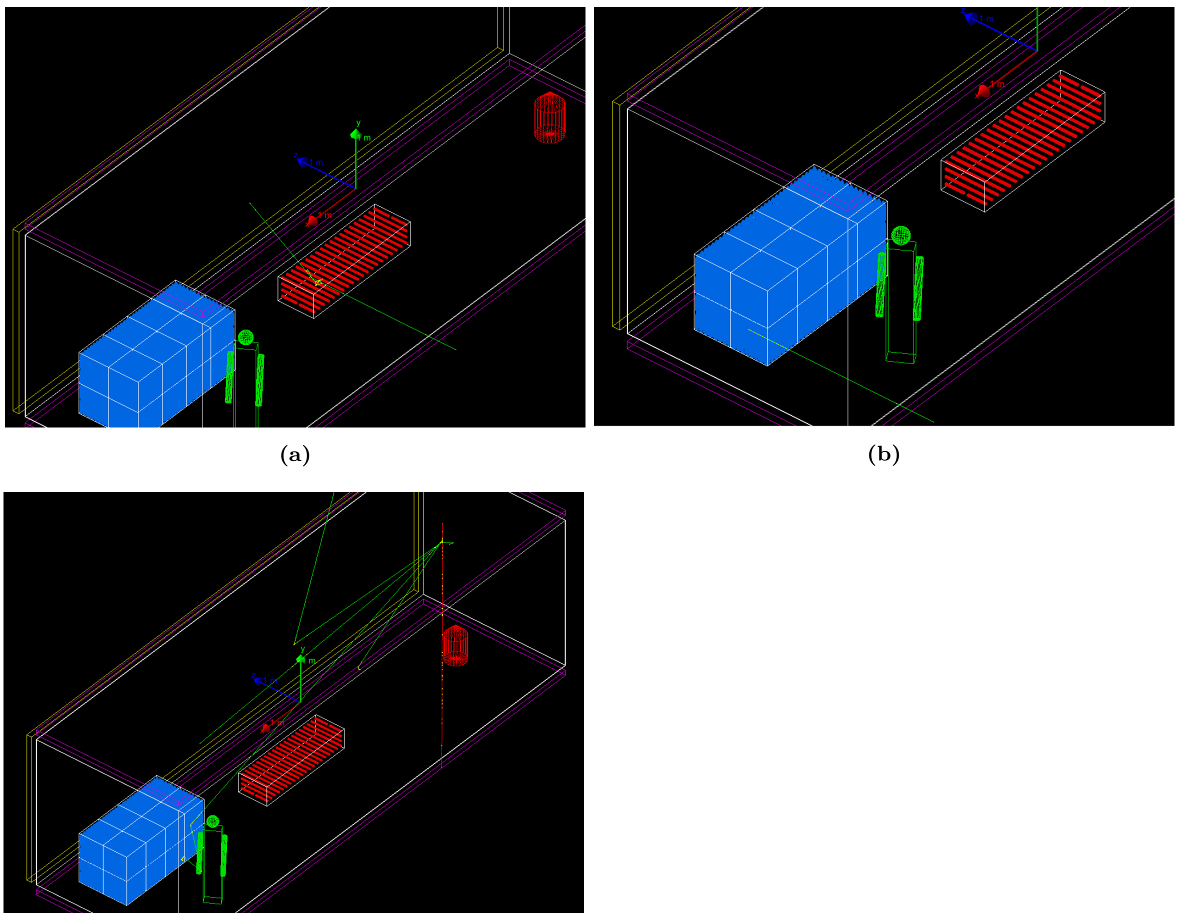
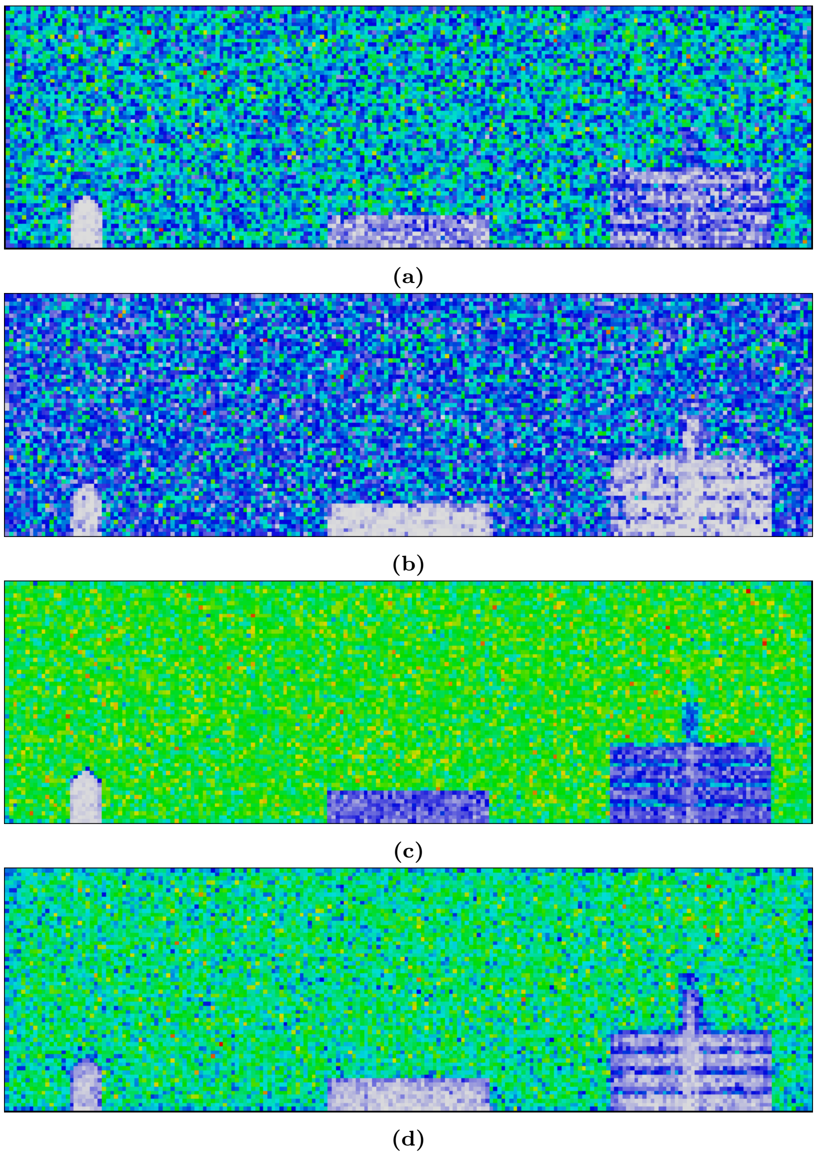

(a) (b) (c)

Figure 2 – Snapshots of **(a)** an x-ray event, **(b)** a neutron event, and **(c)** a muon event.

Snapshots of one event of each particle type are shown in Fig. 2. Apart from showing interactions involving three different beam types, they also show three different types of interactions. In Fig. 2a, the x-ray is scattered against the firearms box, which causes it to hit the detector at a position above the box. In Fig. 2b, the neutron is slowed down, and eventually absorbed, in the bottles of alcohol. Consequently, in both of these cases less energy is deposited just behind the object, which is the idea behind this scanner. In Fig. 2c, photons are produced when the muon interacts with the container roof. These are further scattered in the container, and may cause hits in the detectors. Since the muon has a very small scattering angle and a very high energy, it will reach the bottom detector before the photons. This shows why it is important to only consider the first hits in the detectors in each event, since otherwise secondary particles may give rise to false coincidences.

3.2 Detector measurements

3.2.1 X-rays and neutrons


(a)

(b)

(c)

(d)

Figure 3 – Energy deposit in the calorimeter using **(a)** 100 000 x-ray events, **(b)** 100 000 neutron events, **(c)** 500 000 x-ray events, and **(d)** 500 000 neutron events. Here, white corresponds to the lowest energy deposit, followed by blue, green, yellow, and red. Note that the absolute scales are not the same in all figures.

Energy deposits in the calorimeter are shown in Fig. 3, for both neutrons and x-rays, done for both 100 000 and 500 000 events. Already when using 100 000 events, all objects can be resolved by either beam, but the images are quite noisy. In particular when using x-rays, there is a large improvement in resolution when increasing the number of events to 500 000. Here, it is clear that the nuclear container, made of the high-Z materials lead and uranium, effectively blocks energetic photons, whereas these more easily traverse the other objects. One can also easily see the (1 cm wide) gaps between the bottles, but it is difficult to resolve the steel pipes (firearms). Although one can see the human in full length despite being partly hidden behind the box, it is difficult to see what it is.

The neutrons, on the other hand, are absorbed much more effectively in the hydrogen rich materials – water, alcohol, and plastics – than the γ:s, but not at all as effectively in lead and uranium. Therefore, it looks like all objects might be made from the same materials if only considering these images. Consequently, there is a larger contrast relative to the background for the hydrogen-rich materials. This makes it possible to resolve the human head, but otherwise the results are similar to using an x-ray beam.

3.2.2 Muons

Hit maps of the muon detectors are shown in Fig. 5, and a map over scattering angles in Fig. 4. All of these figures are generated using 2 million muon events. Note that since these plots are drawn in the xz plane, and not the xy plane, they are inverted (or look like if viewing the detector from below). The hit map over the top detector seems to follow a uniform distribution, except a slightly higher intensity near the edges, which is due to muons scattered against the container walls. This shows that the distribution of incoming muons is indeed uniform. For the bottom detector, the results are similar, but the intensity seems to be slightly lower just below the larger objects, although this is barely visible. This shows that the container objects indeed block the muons to a small extent.

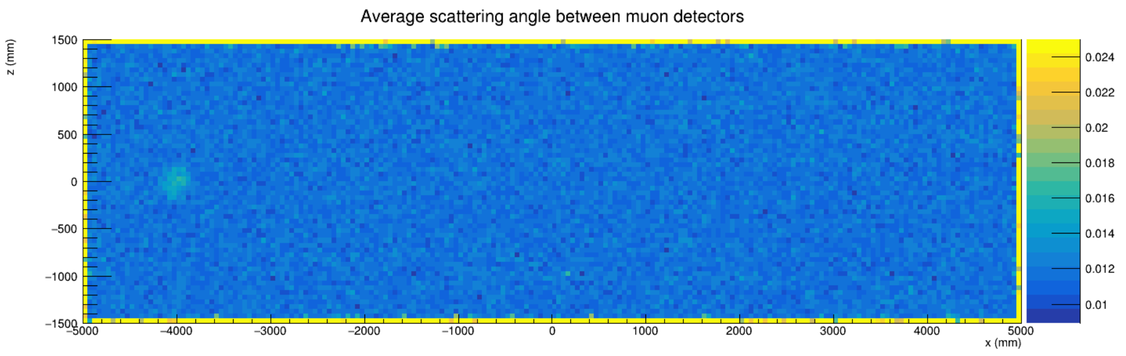


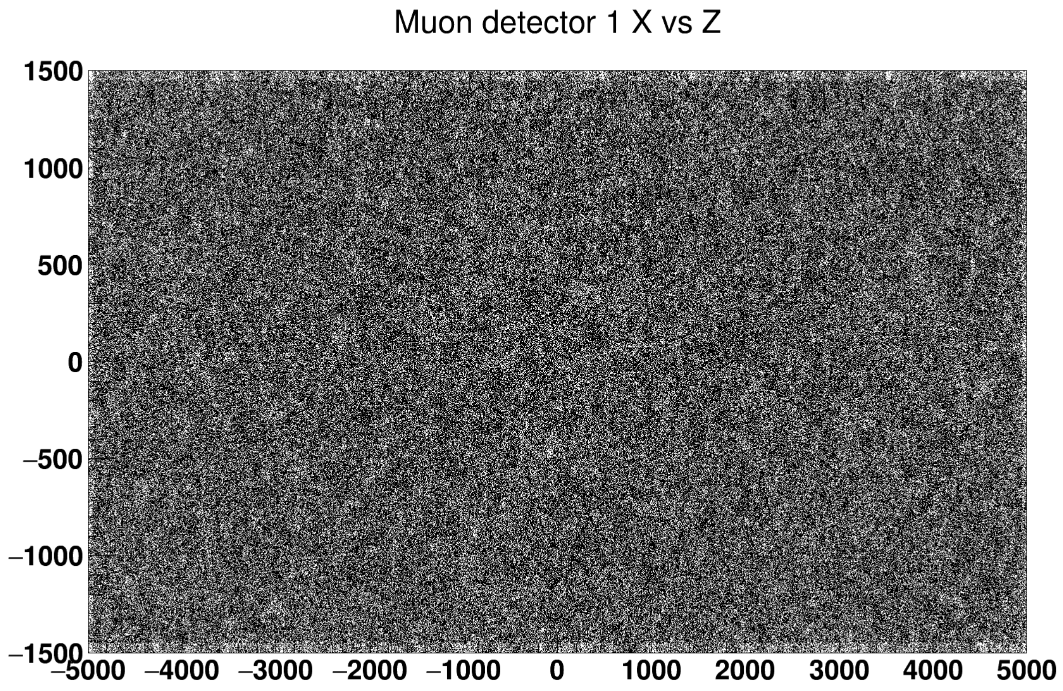
Figure 4 – Average scattering angle (in radians) in each detector cell for a simulation of 2 million muon events.

In the scattering map, the container walls give rise to relatively large scattering angles. The only object which is visible inside the container is the nuclear container, with scattering angles of about 1°. Apparently this is the only object giving rise to scatterings greater than fluctuations of the background (coming from scatterings with the roof and air molecules) at these statistics, showing that muons scatter more against materials with high Z.

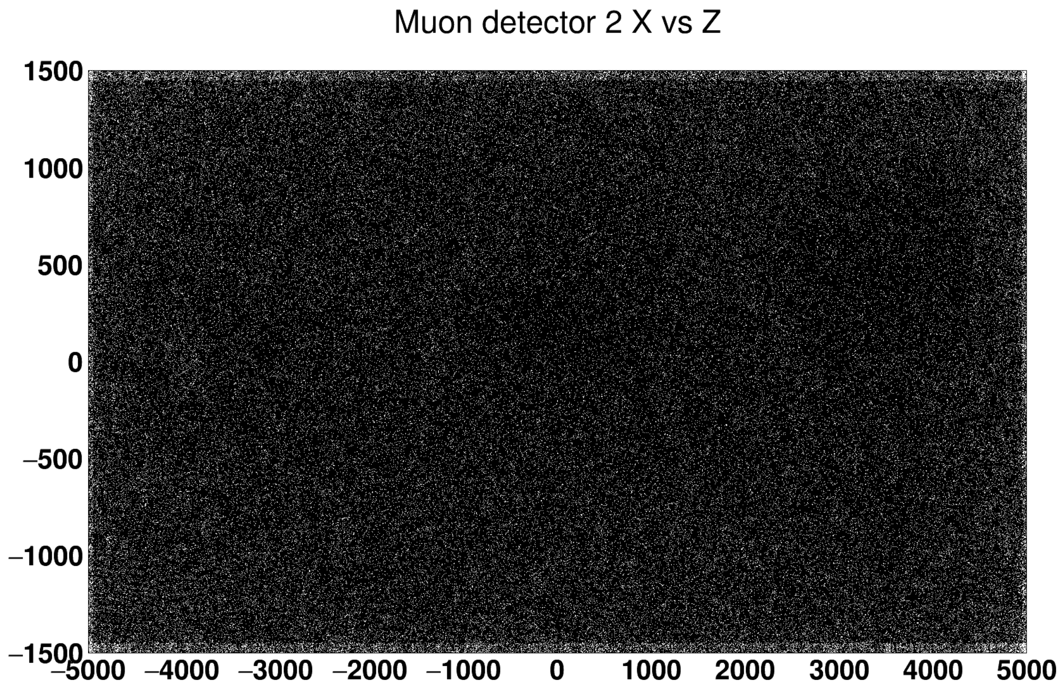
4 Discussion

4.1 Discussion of x-ray/neutron results

The results of these simulations show that the principle of measuring the energy deposit from an x-ray (or gamma ray) or neutron beam works for scanning a container with various objects. Combining the measurements from both beams makes it possible to distinguish between different materials. In this setup, one can see that the leftmost object in Fig. 3 effectively blocks photons but not equally well neutrons, suggesting that this object is made of a material with high Z, probably a heavy metal. The



(a)



(b)

Figure 5 – Hit maps (distribution of hits in the xz plane)) in **(a)** the top muon detector and **(b)** bottom muon detector, from a simulation of 2 million events.

other objects, on the other hand, are not so effective at blocking x-rays, but more effective against neutrons, suggesting that these materials are rich in hydrogen. It is difficult to determine exactly what these objects are, only that they probably are made of organic or water-rich materials, but when using

500 000 neutrons, the shape of a human starts becoming clear. From these results, I would therefore definitely look for people inside the container, and also investigate the heavy metal object, since this really looks suspicious (its shape resembles a bomb). The other objects do, on the other hand, not seem so suspicious, so the smuggler might get away with them. The firearms are so small that they are not resolved in the detector, in particular since they are aligned with the beam. Here one may ask if it really is realistic to pack them so tightly in polystyrene, but apparently this was a smart move of the smuggler. The boxes with alcohol could contain more or less anything based on the results of the scan, so they might also avoid a closer investigation.

Since using 100 000 events gives rise to quite blurry images, I would go for the larger number. In particular, the shape of the human is difficult to resolve when using the smaller number of events. Here, one should make a statement about the risks of scanning the container with ionising radiation, since humans or animals might be hidden inside. Here, the neutron beam results in a much larger dose equivalent in tissue than the x-ray beam, so I will only consider this one. Assuming the human blocks 0.5 m^2 of the beam area, he/she weighs 70 kg, the neutrons are fully absorbed in the tissue, and using a quality factor of 5 for the neutrons, the dose equivalent will be

$$\frac{0.5\,\mathrm{m}^2}{30\,\mathrm{m}^2} \cdot 5 \cdot 10^5 \text{ neutrons} \cdot 15\,\mathrm{MeV/neutron} \cdot 1.6 \cdot 10^{-13}\,\mathrm{J/MeV} \cdot 5/70\,\mathrm{kg} = 1.4\,\mathrm{nSv},$$

which is much less than what one gets from a medical scan. In fact, it is comparable to the natural background radiation one is exposed to during one minute (assuming an annual dose of 1 mSv/year), so I conclude this device is harmless.

In real applications, however, the background radiation may alter the measurements, requiring to use a beam of higher intensity. Since the container is so large, it may be difficult to fire 500 000 x-rays or neutrons uniformly over the detector in just a few seconds or so, which would be required to minimise the background. The simulation could however be made more realistic, however, and one could use a slightly smarter detector setup. First, the particle spectra are not monochromatic, neither if using an x-ray (continuous spectrum) or a gamma source (usually several spikes). The same goes for the neutrons. Second, the beam will probably be swiped over the area rather than appearing at random positions. Firing an equal number of particles towards each tile would decrease the fluctuations somewhat. Finally, I noticed during simulations that some particles are not fully absorbed in the detector, but escape through the other side, so it would be a good idea to use a thicker detector in order to ensure full absorption, which would also increase the scanning resolution.

4.2 Discussion of muon results

Using atmospheric muons for scanning a container is an interesting concept, but these simulation results show that it might be difficult to use for practical purposes. Of the objects placed inside the container, only the very dense one gives rise to scattering angles visible above the background, despite using a fairly large number of muons. In the hit maps, one can see hints of shadowing from the larger boxes as well, but these are very faint and barely visible, making it difficult to draw any conclusions. Both in the simulation and in reality it would be difficult to increase the statistics very much, since this already takes quite long time to simulate ($\sim$ 10 minutes). Two million muons correspond to about 67 000/m^2. As a rule of thumb, the rate of muons is about one per cm^2 per minute, meaning this simulation corresponds to about $400\,\mathrm{s} \approx 7\,\mathrm{min}$ exposure time.

In reality, this measurement would also be much more difficult. One needs to ensure both a good suppression of background sources, and take into account that not all muons come from straight above. Since the scattering angles are so small, one would therefore need to measure the directions of incoming as well as outgoing muons, which would require at least two detectors above and two below the container. It would be an interesting project on its own to simulate this in Geant4, but it would be far beyond the scope of this project.

5 Conclusions

The results of this simulation shows that one can obtain reasonable imaging results of objects placed inside a container, by scanning it with a moderate number of hard x-rays or neutrons. By combining measurements using both particle types, one can estimate which types of materials the objects are made of, and in particular whether they are hydrogen-rich or made of heavy metal. It is more difficult to determine exactly what objects are inside the container, but by using an intense enough neutron beam, the shape of a human may be resolved. For practical purposes, however, one should also take into account background sources and use a more realistic beam spectrum. It would also be a good idea to use thicker detectors, since particles may escape.

Using scattering of atmospheric muons to scan the detector is difficult to do in practice, and lighter materials may not be resolved at all, although one may use this technique for finding denser objects. In practical applications, a more advanced setup is required than the one used here, in particular since not all muons go in the same direction. Simulating this is a possible extension of this project.

A Source code of main programme

This is the source code for the main programme, excluding include statements:

```
/* The first argument should specify whether to use gamma beam (0), neutrons
   (1), or muons (2). If no arguments, a gamma beam is used.
   A second argument may be used to execute specific commands. */
int main(int argc,char** argv)
{
  // Detect interactive mode (if no argument) and define UI session
  G4UIExecutive* ui = 0;
  if (argc <= 2) { // No command line argument beyond particle type
    // Let G4UIExecutive guess what is the best available UI
    ui = new G4UIExecutive(1, argv);
  }

  // Construct the default run manager
  // Note that if we have built G4 with support for Multi-threading we set it here
#ifdef G4MULTITHREADED
  G4MTRunManager* runManager = new G4MTRunManager;
  // Set the default number of threads to be the number of available cores of the
  // machine
  // If not specified use 2 threads
  runManager->SetNumberOfThreads( G4Threading::G4GetNumberOfCores() );
#else
  G4RunManager* runManager = new G4RunManager;
#endif

  // Activate UI-command based scorer
  G4ScoringManager* scManager = G4ScoringManager::GetScoringManager();
  scManager->SetVerboseLevel(1);

  // Mandatory user initialization classes

  //====================
  // The Geometry
  G4int beamType = 0; // gamma
  if (argc > 1) // read beam type if specified
    beamType = atoi(argv[1]);
  runManager->SetUserInitialization(new DetectorConstruction(beamType < 1));

  //====================
  // The Physics
  G4PhysListFactory pls;
  G4VModularPhysicsList* physicsList = pls.ReferencePhysList();
  physicsList->RegisterPhysics(new G4StepLimiterPhysics());
  runManager->SetUserInitialization(physicsList);

  //====================
  // User action initialization
  runManager->SetUserInitialization(new ActionInitialization(beamType));

  // Visualization manager construction
  G4VisManager* visManager = new G4VisExecutive;
  visManager->Initialize();

  // Get the pointer to the User Interface manager
  G4UImanager* UImanager = G4UImanager::GetUIpointer();

  if (argc > 2) {
    // execute an argument macro file if exists
    G4String command = "/control/execute ";
    G4String fileName = argv[2];
    UImanager->ApplyCommand(command+fileName);
  }
  else {
    // We have visualization, initialize defaults: look in init_vis.mac macro
    UImanager->ApplyCommand("/control/execute init_vis.mac");
    if (ui->IsGUI() ) {
      UImanager->ApplyCommand("/control/execute gui.mac");
    }
    ui->SessionStart();
    delete ui;
  }
  // Job termination
  // Free the store: user actions, physics_list and detector_description are
  // owned and deleted by the run manager, so they should not be deleted
  // in the main() program !

  delete visManager;
  delete runManager;

  return 0;
}
```