technische universität
dortmund

Eine kleine Anleitung zur Installation und Verwendung von Geant4

Henning Manke
henning.manke@tu-dortmund.de

Lehrstuhl für Experimentelle Physik V
Fakultät Physik
Technische Universität Dortmund

Origin 2019, Update 2023

Inhaltsverzeichnis

Kapitel 1

Einleitung

Dieses Skript hilft hoffentlich bei der Installation und bei der ersten Simulation mit Geant4. Beachtet, dass sich diese Installation insbesondere auf MacOs bezieht. Andere Unix-Systeme benötigen gegebenenfalls leicht andere Befehle.
Den offiziellen *Geant4 Installation Guide* findet man hier. Dieser sollte inbesondere für die aktuellen Vorraussetzungen (siehe Abschnitt 2.1) und natürlich bei allen auftretenden Problemen zu Rate gezogen werden.
Wirklich hilfreich zur Installation ist auch dieses YouTube Tutorial. Auch die weiteren Videos dieser Tutorial Reihe können eine gute Hilfe für die ersten Schritte in Geant 4 sein. In dieser Anleitung werden Befehle und Code in 3 verschiedenen Eingabefenstern gezeigt.

```
$ cd ~ && mkdir Simulationen && cd Simulationen
$ # 1. Die normale Konsole. Daran zu erkennen, dass vor jedem Befehl ein \$-Zeichen
    steht, welcher nicht mit eingegeben wird. Bitte auf den Unterschied zwischen ~ und
     - achten.
```

```
/run/beamOn 10
# 2. Befehle für Geant4. Diese werden direkt in der visualisierten Umgebung
    eingegeben oder in das sogenannte RunFile geschrieben
```

```
1 // 3. C++ Code. Dieser wird in den Konfigurationsdateien der Simulation
      verwendet.
2 int main(){
3         cout << "Hello, World!";
4         return 0;
5 }
```

Kapitel 2

Installation

2.1 Vorraussetzungen

Für die Installation werden hauptsächlich Befehle in die Konsole eingegeben. Die Konsole findet ihr unter Programme/Dienstprogramme/Terminal.app. Das Dollarzeichen in der jeweiligen Zeile gehört nicht zum eigentlichen Befehl.

2.1.1 Homebrew

Für die Installation von Homebrew folgenden Befehl in der Konsole ausführen.

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/
    install.sh)"
```

2.1.2 XQuartz / X11

Zur Visualisierung eines interaktiven Fensters wird X11 benötigt. Als Ersatz für Unix-Betriebssysteme gibt es XQuartz. Dieses kann auf der offiziellen Homepage heruntergeladen werden. Alternativ die Installation über Homebrew, indem folgender Befehl in der Konsole ausgeführt wird

```
$ brew install --cask xquartz
```

2.1.3 cmake

Wieder kann Homebrew benutzt werden. Sollte cmake trotz Installation nicht genutzt werden können, da es nicht gefunden wird, kann der zweite Befehl helfen.

```
$ brew install cmake
$ export PATH="/usr/local/bin:$PATH"
```

Natürlich kann auch die binary-distribution auf der offiziellen Homepage heruntergeladen und ausgeführt werden, um cmake zu installieren.

2.1.4 Qt

Für die Visualisierung von Geometrien und Teilchenbahnen (zwingend nötig) muss auch Qt installiert werden.

```
$ brew install qt
```

Es kam vor, dass Geant4 auf MacOs Mojave nicht die aktuelle Version von Qt unterstützt hat. Sollte es nach der Installation Probleme mit der Visualisierung von Teilchen geben, kann Qt4 statt der aktuellen Version verwendet werden. Dafür zunächst Qt deinstallieren. Anschließend muss auch Geant4 neu installiert werden.

```
$ brew uninstall qt
$ brew tap cartr/qt4 && brew tap-pin cartr/qt4 && brew install qt@4
```

2.1.5 XCode

Die letzte Vorraussetzung ist ein C++ Compiler. Für MacOs wird Xcode verwendet und kann mit folgendem Befehl installiert werden.

```
$ xcode-select --install
```

Alternativ ist das gesamte XCode-Package inklusiver aller Editoren auf der Homepage verfügbar. Bei der ersten Version von MacOs Mojave gab es ein Problem mit der aktuellen Version von XCode. Daher musste neben der neuen Version von Xcode auch eine ältere Version, genauer 9.4.1, installiert werden. Die älteren Versionen sind hier zu finden. Nach der Installation von beiden Versionen kann mit Hilfe von

```
$ sudo xcode-select -switch /Applications/Xcode.app
$ sudo xcode-select -switch /Applications/Xcode_9.4.1.app
```

zwischen den beiden Versionen gewechselt werden (Der Pfad zu den XCode Programmen kann natürlich variieren). Nähere Informationen zu der Parallelinstallation verschiedener XCode-Versionen sind hier zu finden. Sollte die Parallelinstallation nötig gewesen sein, muss Geant4 ebenfalls neu installiert werden. Vor der Installation muss dann die ältere Version von XCode gewählt werden.

2.2 Installation von Geant4

Die letztendliche Installation von Geant4 wird einige Zeit in Anspruch nehmen (∼1h). Zunächst wird die aktuelle Version von der Homepage heruntergeladen (für Unix-Systeme das .tar Format). Für den Programmierkurs bzw. die Bachelorarbeiten im Jahr 2023 benötigen wir die Version Geant4-v11.0.3. Natürlich können bei Bedarf auch ältere Versionen installiert werden (`https://geant4.web.cern.ch/support/download_archive`). Die tar-Datei wird entpackt und der Ordner an einen beliebigen Ordner verschoben.

```
$ tar xzfv ~/Downloads/geant4-v11.0.3.tar
```

In diesem Beispiel heißt der Ordner geant4-v11.0.3 und wird in das Homeverzeichnis verschoben. Mit der Konsole wird der sogenannte „build“-Ordner erzeugt und in sein Verzeichnis gewechselt.

```
$ mkdir geant4-v11.0.3-build
$ cd geant4-v11.0.3-build
```

Anschließend wird ein cmake ausgeführt. Im Falle der Installation ist es einfacher eine gui zu haben, um die erforderlichen Argumente auf ”on”zu switchen.

```
$ ccmake ../geant4-v11.0.3
```

-> press ’c’ for configure
-> warnings are okay and can be ignored unless they are errors

```
CMAKE_INSTALL_PREFIX        ~/geant4-v11.0.3-install

GEANT4_BUILD_MULTITHREADED ON
GEANT4_INSTALL_DATA ON
GEANT4_INSTALLDATADIR ON
GEANT4_USE_G3TOG4 OFF
GEANT4_USE_GDML ON
GEANT4_USE_INVENTOR OFF
GEANT4_USE_INVENTOR_QT OFF
GEANT4_USE_OPENGL_X11 ON
GEANT4_USE_PYTHON OFF
GEANT4_USE_QT ON
GEANT4_USE_RAYTRACER_X11 OFF
GEANT4_USE_CLHEP OFF
GEANT4_USE_SYSTEM_EXPAT ON
```

-> press 'c' for configure
-> if it can't found some qt files, you can mostly ignore it, will complie anyway
-> press 'c' for configure
-> press 'g' for generate

Alternativ kann auch ohne gui der cmake Befehl ausgeführt werden (nicht zu empfehlen, besser, wenn man Übersicht über alle möglichen Argumente hat):

Es sollten zusätzlich einige optionale Argumente verwendet werden. Für die Visualisierung muss die Verwendung von XQuartz und Qt aktiviert sein (dies geschieht mit Hilfe von `-DGEANT4_USE_OPENGL_X11=ON -DGEANT4_USE_QT=ON`). Des Weiteren ist `-DGEANT4_INSTALL_DATA=ON` notwendig. Das Install-Prefix legt fest, in welchem Ordner letztendlich Geant4 installiert wird. In diesem Beispiel heißt der Ordner geant4-v11.0.3-install und befindet sich so wie der heruntergeladene Ordner und der build-Ordner im Homeverzeichnis. Zusätzlich sollte mithilfe von `-DGEANT4_BUILD_MULTITHREADED=O` Multithreading aktiviert werden. Dann kann eine Simulation auf mehrere Kerne gleichzeitig zugreifen. `-DGEANT4_USE_GDML=ON` benötigen wir (oder auch nicht?) für die Implementierung von stls. Insgesamt ergibt sich also für den cmake Befehl:

```
$ cmake -DCMAKE_INSTALL_PREFIX=~/geant4-v11.0.3-install -DGEANT4_INSTALL_DATA=ON -
    DGEANT4_USE_OPENGL_X11=ON -DGEANT4_USE_QT=ON -DGEANT4_BUILD_MULTITHREADED=ON -
    DGEANT4_USE_GDML=ON ~/geant4-v11.0.3
```

Für MacOs werden zudem `-DCMAKE_C_COMPILER=clang -DCMAKE_CXX_COMPILER=clang++` empfohlen, sind aber nach meiner Erfahrung nicht nötig. Weitere Informationen zu den optionalen Argumenten (beispielsweise um Multithreading zu aktivieren) sind der offiziellen Anleitung zu entnehmen.

Nun kommt der Befehl, für den je nach Rechenleistung des PCs etwa 1h eingeplant werden sollte. Durch das Argument `-jN`, wobei N der Anzahl der verfügbaren CPU-Kerne entspricht, wird die gesamte Rechenleistung eures PCs ausgenutzt.

```
$ make -jN && make install
```

Nun ist die Installation von Geant4 abgeschlossen. Wenn gewünscht, kann der build-Ordner, der entpackte Ordner und die heruntergeladene tar-Datei gelöscht werden. Es können ohne Probleme mehrere Versionen von Geant4 auf einem Computer installiert werden und auch parallel genutzt werden.

Kapitel 3

Die erste Simulation

Generell wurde in diesem Kapitel die Version Geant4.10.5 gewählt. Daher müssen die Pfade für neuere Geant4 Versionen angepasst werden. Die Basic Examples ändern sich meistens nicht, daher kann dieses Tutorial auch mit neueren Geant4 Versionen durchgeführt werden.

3.1 Starten der Simulation

Um die erste Simulation durchzuführen, wird hier ein offizielles Geant4-Beispiel verwendet. Das Beispiel findet man nach der Installation von Geant4 in:

```
~/geant4-v11.0.3-install/share/Geant4-11.0.3/examples/basic/B1
```

Dieser und alle folgenden Pfade können je nach Geant4-Version und nach gewünschtem Installationspfad variieren. Diesen Ordner (B1) würde ich zwecks des ersten Versuchs an einen beliebigen Ort kopieren (in diesem Beispiel wird ∼/Simulationen/ verwendet).

```
$ cp -r ~/geant4-v11.0.3-install/share/Geant4-11.0.3/examples/basic/B1 ~/Simulationen
    /
```

Anschließend wird, ähnlich wie bei der Installation von Geant4 ein build-Ordner zu der Simulation erstellt, in diesen gewechselt und cmake ausgeführt.

```
$ mkdir ~/Simulationen/B1-build
$ cd ~/Simulationen/B1-build
$ cmake ../B1
```

Wenn der Befehl fertig ausgeführt wurde sollte in der Konsole etwas wie

```
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/hmanke/Documents/Simulationen/CCB-build
```

zu sehen sein. Dann war cmake erfolgreich. Sollte eine Fehlermeldung erscheinen, dass cmake nicht weiß, wie es Geant4 bauen soll, muss der Pfad zur Geant4Config.cmake Datei, bzw. zum Ordner in dem sie liegt, angegeben werden. Dies geschieht, indem der Befehl um ein optionales Argument erweitert wird.

```
$ cmake ../B1 -DGeant4_DIR=~/geant4-v11.0.3-install/lib/Geant4-11.0.3/
```

Es kann sein, dass ihr in Zukunft bei jedem cmake (für Geant4) den Pfad angeben müsst. Es kann aber auch sein, dass es reicht, wenn cmake einmal die config gefunden hat. Probiert es einfach aus. War die Ausführung von cmake erfolgreich, wird

```
$ make -jN
```

eingegeben. N entspricht wieder der gewünschten Anzahl an verfügbaren CPU-Kernen. Wenn noch nichts im Beispiel von B1 verändert wurde, wird keine Fehlermeldung erscheinen. Sollte es Fehler oder zumindest Warnungen bezüglich der Simulation geben, werden diese während des Make-Befehls ausgegeben. Errors erscheinen immer wieder, da die Simulation nicht durchgeführt werden kann. Warnungen werden nach dem cmaken nur einmal ausgegeben.
Im nächsten Schritt wird folgender Befehl ausgeführt:

```
$ source ~/geant4-v11.0.3-install/bin/geant4.sh
```

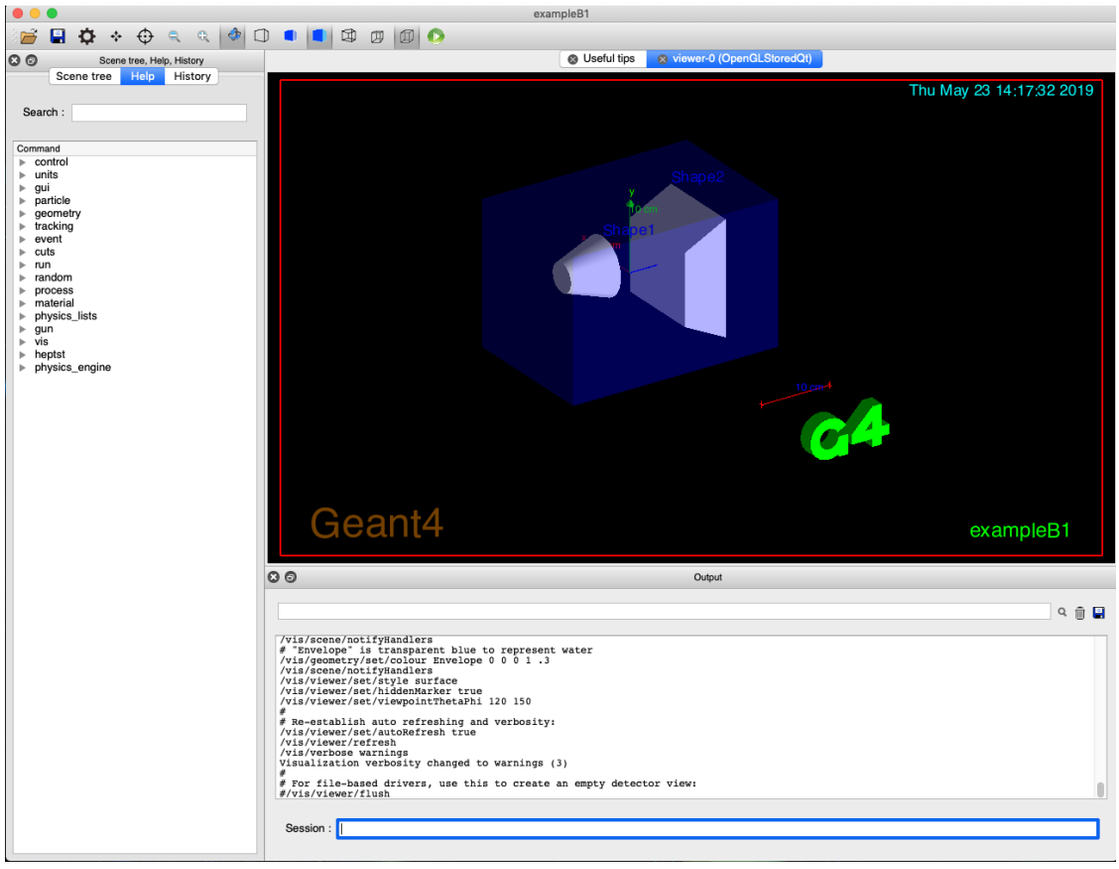
Ohne diesen Befehl kommt es beim Starten der Simulation zu einem „Segmentation Error 11“. In jeder Konsole, die eine Simulation starten soll, muss dieser Befehl ausgeführt werden. Daher ist es ratsam, sich ihn als alias in die `~/.zshrc` zu schreiben. Das kann dann zum Beispiel so aussehen:

```
alias geant="cd /Users/michelle/geant4-v11.0.3-install/bin && source geant4.sh && cd
    -"
```

Nun kann zum ersten Mal eine Simulation mit Visualisierung gestartet werden, in dem das Main-File der Simulation ausgeführt wird.

```
$ ./exampleB1
```

Es sollte folgendes Fenster erscheinen:



Es ist folgendes zu sehen: Innerhalb des schwarzen Fensters eine quaderförmige „world“ und darin zwei Volumen („Shape1“ und „Shape2“). Wird in das Eingabefenster (im Bild blau umrandet) der Befehl

```
/run/beamOn 100
```

eingegeben, werden 100 Partikel gemäß der Simulation losgeschickt. Hierbei handelt es sich zunächst um monoenergetische Photonen, die alle an einer Fläche der World erzeugt werden und in parallelen Strahlen

durch die Volumen geschickt werden (grüne Strahlen). Jegliche Wechselwirkungen werden automatisch von Geant4 erzeugt, sodass sich die Ausbreitungsrichtung natürlich auch ändern kann oder Elektronen (rote Strahlen) erzeugt werden können.

3.2 Verändern der Simulation - Detector Construction

Die Konsole befindet sich nach wie vor in dem build-Ordner der Simulation. In diesem Ordner wollt ihr die Dateien nicht aktiv verändern! Die gesamte Konfiguration der Simulation spielt sich in dem „normalen“ B1-Ordner ab. Öffnet mit einem Editor eurer Wahl (beispielsweise Sublime oder Atom) die Datei B1/src/B1DetectorConstruction.cc. In dieser Datei werden alle für die Simulation relevanten Volumina erzeugt. Beispielsweise befindet sich ab Zeile 70 der Abschnitt, in dem die vorhin angesprochene World definiert ist. Zunächst werden nur Variablen deklariert. Die Größe der Welt wird in Abhängigkeit der Größe des Envelops (folgender Abschnitt in der Detector Construction) definiert. Das Material „world_mat“ wird aus dem vorgefertigten Katalog von Geant4-Materialien abgerufen und ist Luft.

```
1  G4double world_sizeXY = 1.2*env_sizeXY; // env_sizeXY und env_sizeZ
       werden in Zeile 64 auf 20 und 30cm gesetzt
2  G4double world_sizeZ  = 1.2*env_sizeZ;
3  G4Material* world_mat = nist->FindOrBuildMaterial("G4_AIR"); // Abfrage
       des Materials aus der Bibliothek
4
5  G4Box* solidWorld =
6  new G4Box("World", // Name des Solids
7  0.5*world_sizeXY, 0.5*world_sizeXY, 0.5*world_sizeZ);
       // Größe (Achtung: Halbe Längen !)
8
9  G4LogicalVolume* logicWorld =
10 new G4LogicalVolume(solidWorld, // Verweis auf das Solid
11 world_mat, // Material festlegen
12 "World"); // Name
13
14 G4VPhysicalVolume* physWorld =
15 new G4PVPlacement(
16 0,                    // Rotation
17 G4ThreeVector(),      // Position, hier (0,0,0)
18 logicWorld,           // Verweis auf das LogicalVolume
19 "World",              // Name
20 0,                    // Mother Volume, "World" ist in der Hierarchie
       ganz oben
21 false,                // Keine Boolschen Operationen
22 0,                    // Anzahl der Kopien
23 checkOverlaps);       // Überprüfung auf Kollision mit anderen Volumina
```

Anschließend wird ein Volumen in 3 kleinen Abschnitten erzeugt. Zunächst das sogenannte Solid. In diesem Beispiel wird eine G4Box verwendet, sodass es sich um einen Quader handelt. Der Name des Quaders ist „World“ und die halben Längen in x, y und z Richtung werden angegeben.
Es folgt das Logical. Hier wird zunächst auf das Solid desselben Volumens verwiesen, anschließend das Material festgelegt und der Name wiederholt.
Zu guter Letzt kommt das Physical. In diesem werden zuerst die Rotation und die Position des Volumens definiert. Anschließend wird auf das Logical verwiesen und der Name ein letztes Mal wiederholt. Die Volumina in Geant4 sind hierarchisch angeordnet, das heißt es gibt ein oberstes Volume und alle anderen Volumen

befinden sich entweder darin oder wiederum in kleineren Volumina. Die World ist das oberste Volumen und erhält daher kein Mother Volume. Die World wird keiner boolschen Operation unterzogen und wird nur ein mal erzeugt. Der Befehl `checkOverlaps` sollte bei allen Volumina verwendet werden, damit bei der Simulation auf Kollision von Volumina hingewiesen wird.
Ebenso wird in der Detector Construction auch das Volumen Shape1 erzeugt. Ich empfehle an dieser Stelle nun das Volumen Shape1 zu verändern. Hierbei hilft einem der Guide *Geant4 Book For Application Developers, Kapitel: How to Define a Detector Geometry* (Link) weiter. Will man die Veränderung betrachten/testen, muss die Detector Construction gespeichert werden, und anschließend in der Konsole (die sich noch immer im build-Ordner befindet) erneut

```
$ make
```

ausgeführt werden. Nun werden alle Änderungen der Dateien im B1-Ordner umgesetzt. Mit Hilfe von

```
$ ./exampleB1
```

wird die Simulation erneut geöffnet und die Änderungen sollten nun zu sehen sein.
Es können beliebig viele neue Volumina erzeugt werden. Die Volumina sollten sich jedoch niemals überschneiden. Wenn liegen sie nur vollständig ineinander, und das Mother Volume wird entsprechend angepasst.
Des Weiteren können in der Detector Construction boolsche Operationen bei den Volumina durchgeführt werden, ein Volumen beliebig oft kopiert werden, Materialien neu definiert oder verändert werden.

3.2.1 Implementierung von CAD-konstruierten Bauteilen

Mittlerweile ist es möglich mittels diverser Programme konstruierte Bauteile in Geant4 zu implementieren. Dafür muss zunächst ein XML-Parser installiert werden:

```
$ brew install xerces-c
```

Zudem muss bereits bei der Installation von Geant4 eine Einstellung gewählt werden, sodass die `GDML` verwendet werden kann. Daher wird der bereits bestehende `cmake`-Befehl um ein zusätzliches Argument erweitert

```
... -DGEANT4_USE_GDML = ON ...
```

weitere informationen folgen von justine

3.3 Verändern der Simulation - Particle Generator Action

Eine weitere wichtige Datei ist die B1ParticleGeneratorAction.cc. In dieser Datei werden die erzeugten Partikel definiert. Der Partikel selbst, seine Ausbreitungsrichtung, seine Energie und wie viele Partikel dieser Sorte pro `\run\beamOn 1` erzeugt werden.

```
B1PrimaryGeneratorAction::B1PrimaryGeneratorAction()
: G4VUserPrimaryGeneratorAction(),
fParticleGun(0),
fEnvelopeBox(0){
        G4int n_particle = 1;
        fParticleGun  = new G4ParticleGun(n_particle);
        // default particle kinematic
        G4ParticleTable* particleTable = G4ParticleTable::
           GetParticleTable();
        G4String particleName;
        G4ParticleDefinition* particle = particleTable->FindParticle(
           particleName="gamma");
        fParticleGun->SetParticleDefinition(particle);
        fParticleGun->SetParticleMomentumDirection(G4ThreeVector
           (0.,0.,1.));
        fParticleGun->SetParticleEnergy(6.*MeV);
}
```

Natürlich lassen sich in solche Skripte beliebige Schleifen einfügen. Beispielsweise könnte die Energie gemäß der Wahrscheinlichkeitsverteilung eines Gamma-Energiespektrums randomisiert werden. Oder es werden je nach Zufalsszahl verschiedene Partikel simuliert. Ein Beispiel dafür findet sich in `/examples/basic/B5/src/B5PrimaryGeneratorAction.cc`. Andererseits können auch radioaktive Kerne erzeugt werden, sodass die gesamte Zerfallskette von Geant4 berücksichtigt wird. Näheres dazu findet man im Beispiel `/examples/advanced/radioactivedecay`.

3.4 Scoring

Das eigentliche Ziel der Simulation ist das Aufnehmen von Messgrößen. Dafür werden sogenannte „Scorer“ verwendet. Dabei gibt es zwei Arten: „Sensitive Scoring“ und „Command Based Scoring“. Sensitive Scoring ist deutlich schwieriger zu implementieren als Command Based Scoring, dafür sind mehr Möglichkeiten geboten. Tatsächlich sind Shape1 und 2 aus dem Beispiel B1 als sensitive Volumen implementiert. Sie geben bei Ende der Simulation die in ihnen deponierte Dosis aus.
Für die meisten Simulationen reichen Command Based Scorer aus. In diesem Beispiel wird in Beispiel B1 eine Dosisverteilung in x- und y-Richtung aufgenommen. Der Scorer hat seinen Mittelpunkt im Ursprung und ist in x-, y- und z-Richtung $20 \cdot 20 \cdot 1\,\mathrm{cm}^3$ groß. Um eine gewissen Auflösung in x- und y-Richtung zu erhalten, wird das Volumen in 400 sogenannte „Bins“ unterteilt.
Um Command Based Scoring in Geant4 zu integrieren, wird in das mainfile folgender Abschnitt hinzugefügt:

```
#include "G4ScoringManager.hh"    // oben im Header einfügen
G4ScoringManager::GetScoringManager(); // muss in die int main Schleife.
   Am Besten hinter die Einführung des run Managers
```

Zunächst wird die Simulation erzeugt und gestartet. Es wird davon ausgegangen, dass sich der Ordner der Simulation in `~/Simulationen/B1` befindet.

```
$ mkdir ~/Simulationen/B1-build
$ cd ~/Simulationen/B1-build
$ cmake ../B1
$ make -j4
$ ./exampleB1
```

In das Eingabefenster der Geant4-Visualisierung werden nun folgende Befehle eingegeben:

```
/score/create/boxMesh box_dosisverteilung # Name des Scorers
/score/mesh/boxSize 10 10 0.5 cm
/score/mesh/nBin 20 20 1  # Unterteilung des Volumens in Bins
/score/mesh/translate/xyz 0 0 0 mm # gegebenenfalls verschieben
/score/quantity/doseDeposit dDep
/score/close
```

Damit wurde ein Quader gemäß den oben geannten Daten erstellt. Mit Hilfe von

```
/score/drawProjection box_dosisverteilung dDep
```

kann der Quader auch sichtbar gemacht werden. Die effektiven Messpunkte liegen jeweils in der Mitte jedes einzelnen Bins. Nun werden 1000 Teilchen simuliert und anschließend die deponierte Dosis in eine Textdatei ausgegeben.

```
/run/beamOn 1000
/score/dumpAllQuantitiesToFile box_dosisverteilung dosiverteilung.txt
```

In jeder Zeile dieser Textdatei ist das Ergebnis eines Bins aufgelistet. Neben seiner Nummer (nicht seine genaue Position!) in x-,y- und z-Richtung ist die in ihm deponierte Dosis in der vierten Spalte und die Anzahl der Teilchen, die die Dosis erzeugt haben, in der sechsten Spalte notiert. Aus den Nummern der Bins sollten später zur besseren Darstellung der Dosisverteilung natürlich jeweils die Position des Mittelpunkts des Bins ausgerechnet werden. Wird ein Zylinder als Scorer verwendet, bezieht sich die Nummerierung natürlich auf Zylinderkoordinaten.

3.5 Simulation ohne graphische Oberfläche mit Hilfe eines Runfiles

Da Geant4 auf der Monte-Carlo-Methode basiert, müssen deutlich mehr Teilchen simuliert werden, um ein Ergebnis zu erhalten, das nicht mehr stark fluktuiert. Je mehr Teilchen simuliert werden, desto rechenaufwändiger wird die Simulation natürlich. Zudem stellt die graphische Darstellung für den Computer eine zusätzliche, unnötige Belastung dar. Funktioniert die Siimulation nun wie gewollt, kann man sich daher eines „Runfiles“ bedienen.
In der Simulation B1 existiert bereits eine solche Datei: `~/Simulationen/B1/run1.mac`. In dieser Datei stehen nun diverse Befehle, die sonst im Eingabefenster bei einer Simulation mit Visualisierung eingegeben werden könnten. Die Befehle

```
/control/verbose 2
/run/verbose 2
/event/verbose 0
/tracking/verbose 1
```

legen fest, wie viele Informationen zu dem jeweiligen Bereich ausgegeben werden. Dabei entspricht eine 0 gar keiner Ausgabe und eine 2 sehr vielen Informationen.
Anschließend werden die Partikel erneut definiert. Es werden zunächst fünf Photonen mit einer Energie von 6 MeV und anschließend ein Proton mit einer Energie von 210 MeV losgeschickt. Diese Befehle überschreiben die Parameter der `B1PrimaryGeneratorAction.cc`.
Führt man nun in der Konsole statt des normalen Starts der Simulation

```
$ ./example run1.mac
```

aus, wird die Simulation ohne Visualisierung gestartet und die Befehle des Runfiles werden nacheinander durchlaufen. Dieser Simulation wollen wir nun unsere Messung der Dosisverteilung hinzufügen. Die Reihenfolge der Befehle ist nicht unerheblich. Der Scorer muss selbstverständlich vor dem Aussenden der Teilchen implementiert sein. Die Ausgabe der aufgenommenen Dosis muss erst nach der Simulation der Teilchen

erfolgen. Folglich werden die Befehle, die den Scorer erzeugen

```
/score/create/boxMesh box_dosisverteilung # Name des Scorers
/score/mesh/boxSize 10 10 0.5 cm
/score/mesh/nBin 20 20 1  # Unterteilung des Volumens in Bins
/score/mesh/translate/xyz 0 0 0 mm # gegebenenfalls verschieben
/score/quantity/doseDeposit dDep
/score/close
```

vor der Erzeugung der Photonen kopiert. Die Ausgabe der Dosis mit Hilfe von

```
/score/dumpAllQuantitiesToFile box_dosisverteilung dosiverteilung.txt
```

wird am Ende der Datei eingefügt. Nun kann die Simulation auch mit sehr vielen Partikeln durchgeführt werden. Pro Simulation wird nur ein Kern des CPUs verwendet, da beim Installieren Multithreading nicht aktiviert wurde. Dementsprechend können mehrere Simulationen parallel in verschiedenen Konsolen oder als Hintergrundprozesse gestartet werden. Die Anzahl der Simulationen sollte die Anzahl der Kerne jedoch nicht überschreiten. Auch muss darauf geachtet werden, dass genügend Arbeitsspeicher zur Verfügung steht.

3.6 Erweiterung der Simulation zur statistischen Datennahme

Die einzelnen ermittelten Werte sind auf Grund der Monte-Carlo-Methode immer einer gewissen Fluktuation unterworfen. Um die Genauigkeit seiner Ergebnisse zu ermitteln, kann man sich eines einfachen Tricks bedienen. Es werden mehrere Runfiles erstellt, die exakt die gleichen Befehle beinhalten. Die Runfiles speichern ihr jeweiliges Ergebnis in einer jeweils anderen Textdatei. Die ermittelten Werte der einzelnen Dateien können mit Hilfe eines Skripts eingelesen werden, und so der Mittelwert und dessen Fehler berechnet werden.
Bei dieser Methode ist folgendes zu beachten: Zum Einen beruht die Simulation auf dem Randomisieren von verschiedenen Parametern. Um zu gewährleisten, dass die Variablen in den einzelnen Simulationen verschieden erzeugt werden, können bzw. müssen die „Seeds“ für den Zufallsalgorithmus explizit verschieden gesetzt werden. Dies kann auch innerhalb der Runfiles geschehen. Dafür wird an den Anfang der Datei folgender Befehl eingesetzt:

```
/random/setSeeds Zahl1 Zahl2
```

Dabei müssen Zahl1 und Zahl2 durch entsprechende Integer ersetzt werden. Die zweite Zahl muss größer als die Erste sein und darf 10^8 nicht überschreiten.
Zum Anderen werden diese Runfiles am Besten in dem normalen Simulationsordner erstellt. Damit Sie beim cmaken übernommen werden, müssen sie explizit in der Datei

```
~/Simulationen/B1/cmakeLists.txt
```

erwähnt werden. Ab Zeile 46 findet man in dieser Datei den folgenden Abschnitt:

```
set(EXAMPLEB1_SCRIPTS
        exampleB1.in
        exampleB1.out
        init_vis.mac
        run1.mac
        run2.mac
        vis.mac
)
```

Diese Liste innerhalb der Klammern kann mit den Dateinamen der neuen Runfiles erweitert werden. Um die Dateien über cmake zu kopieren, muss der Cmake befehl erneut ausgeführt werden.

```
$ cd ~/Simulationen/B1-build
$ cmake ../B1
$ make
```

Kapitel 4

Auslagerung der Simulationen auf Großrechner

Für eine genügend große Statistik ist es nötig die Simulation sehr oft und mit sehr vielen Teilchen durchzuführen. Dies übersteigt die Kapazitäten von gewöhnlichen Rechnern, weswegen man sich dann eines Rechenclusters bedient. Auf dem E5-Cluster ist ein Submittier-System namens HT-Condor installiert, dessen Dokumentation hier zu finden ist.

4.1 Vorraussetzungen

Um auf die Großrechner zugreifen zu können, müsst ihr dort als User registriert sein und mit einem ssh-key identifiziert werden. Für die Konfiguration könnt ihr folgendes Skript befolgen: https://git.e5.physik.tu-dortmund.de/e5/ssh-config.
In die Konsole wird folgender Befehl eingegeben:

```
$ curl -sSL git.e5.physik.tu-dortmund.de/e5/ssh-config/raw/master/setup.py >
    /tmp/install.py && python /tmp/install.py
```

Dabei bitte die Anweisungen in der Konsole beachten (insbesondere muss der Public-Key an die zuständige Person weiter geleitet werden).

4.2 Simulation auf dem Großrechner

Zunächst wird der Simulationsordner (in diesem Beispiel heißt dieser B1 und liegt in /Simulationen/) kopiert.

```
$ scp -i ~/.ssh/e5.key -r ~/Simulationen/B1
    username@interactive.e5.physik.tu-dortmund.de:/net/nfshome/home/hmanke/
```

Dabei ist es wichtig, dass der Pfad */.ssh/e5.key* zu eurem Key führt (sollte richtig sein, wenn ihr obiges Skript benutzt habt, um die ssh-keys zu erstellen). *-r /Simulationen/B1* legt fest, dass der gesamte Ordner *B1* kopiert wird. Der Letzte Teil des Befehls ist der Pfad zum Speicherort des Ordners. Dies sollte um eine gewissen Ordnung zu Erhalten in eurem eigenen Ordner auf den geteilten Festplatten sein. *username* ist entsprechend zu ersetzten.
Anschließend loggt man sich auf der Maschine, die das System verwaltet, ein.

```
$ ssh interactive.e5.physik.tu-dortmund.de        # beziehungsweise
$ ssh interactive         # wenn der server in der config gespeichert ist
```

Nach Bestätigung und eventuell folgender Passworteingabe befindet ihr euch nun mit der Konsole auf der interaktiven Maschine und solltet den Simulationsordner finden. Dies kann leicht mit

```
$ pwd
$ ls
```

überprüft werden. Anschließend wird wie bei der Simulation auf dem eigenen Rechner mit Hilfe von

```
$ mkdir B1-build && cd B1-build
```

ein build-Ordner erstellt. Um zu cmaken, wird Geant4 benötigt. Dies wird mit Hilfe eines Singularity Containers ermöglicht. Mit dem folgenden Befehl öffnet ihr in eurer Konsole einen Container, in dem Geant4 installiert ist.

```
$ singularity shell /ceph/Singularity/Geant41051-Singularity
```

Falls der Container nicht gefunden werden kann, schaut mit Hilfe von

```
$ cd /ceph/Singularity
$ ls
```

nach, welcher Geant4 Container vorhanden ist. Zurück im Build-Ordner wird mit

```
$ cmake ../B1 -DGeant4_DIR=/opt/geant4/lib/Geant4-10.5.1 && make
```

die Simulation gebaut und die interaktive Konsole innerhalb des Singularity Containers kann mit

```
$ exit
```

verlassen werden. Die Simulation ist fertig vorbereitet und kann verwendet werden. Für das Starten von Simulationen gibt es verschiedene Möglichkeiten. Möchte man die Simulation live verfolgen, kann wieder die Singularity Shell geöffnet werden und die Simulation gestartet werden. Allerdings nie mit Visualisierung sondern immer mit Hilfe eines Runfiles!

```
$ singularity shell /ceph/groups/e5a/Singularity/Geant41051-Singularity
$ ./exampleB1 run1.mac
```

So wird allerdings ein CPU von der interaktiven Maschine tarek blockiert. Dies kann für eine kurze Kontrolle, ob die Simulation korrekt durchläuft, geschehen. Für viele Simulationen sollte allerdings jeweils ein Job abgeschickt werden.

4.2.1 Abschicken eines Jobs im HT Condor System

Um einen Job in das Submitiersystem zu übertragen, müssen 2 Datein angelegt werden. zunächst die sogenannte sh-Datei. In dieser steht der Befehle bzw. die Befehle, die nacheinander durchgeführt werden sollen. Legt ein Dateiname.sh File an, in dem Beispielsweise folgende Zeilen stehen:

```
#!/bin/bash
#file name: Dateiname.sh
singularity exec /ceph/Singularity/Geant41051-Singularity ./exampleB1 run1.mac
```

Die Datei muss auf der interactiven Maschine noch als ausführbar gekennzeichnet werden, weswegen zusätzlich der Befehl

```
$ chmod u+x Dateiname.sh
```

auf die Datei angewendet werden muss. Des Weiteren wird ein submit-File benötigt. Dazu erneut im Texteditor eurer Wahl ein Dokument erstellen.

```
executable = Dateiname.sh
log = Dateiname.log
output = Dateiname-output.txt
error = Dateiname-error.txt
queue
```

Der Dateiname bei executable muss natürlich auch mit der entsprechenden .sh-Datei übereinstimmen. Zu Guter Letzt wird der Job in das System eingebunden:

```
$ condor_submit Dateiname.sub
```

Der Job wird gestartet, sobald die benötigten Kapazitäten im Rechencluster zur Verfügung stehen. Mit

```
$ condor_q
```

wird eine Liste mit allen submittierten Jobs angezeigt.

Es wird eine Option `maxhours` geben, mit der man die maximale Laufzeit seines Jobs einstellen kann. Default liegt bei einer Woche. Folglich werden alle Jobs nach einer Woche Laufzeit abgebrochen. Für vorraussichtlich längere Jobs muss die maximale Laufzeit entsprechend abgeschätzt und angepasst werden. Jobs mit einer Laufzeit von maximal einer Stunde werden in eine gesonderte Queue gesteckt, die mit höherer Priorität gestartet wird.

4.3 Weitere Befehle im Condor-System

1. `condor_status -submitters` Gibt eine Auflistung über alle User und der Anzahl der jeweils submittierten Jobs.
2. `condor_q` Zeigt die submittierten Jobs und deren jeweiligen Status. Running = Job wird ausgeführt. Idle = Der Job läuft noch nicht, weil die nötigen CPUs belegt sind. Hold: Wird nicht ausgeführt, bis der Job mit `condor_release` wieder freigegeben wird. Oft landen Jobs im Hold-Modus, deren `.sh`-Datei nicht mithilfe von `chmod u+x Dateiname.sh` als ausführbar gekennzeichnet wurden.
3. `condor_relase JobID` Löst den Job mit der entsprechenden JobID aus dem Hold-Modus.
4. `condor_rm JobID` Löscht den entsprechenden Job.
5. `condor_hold JobID` Setzt den Job in den Hold-Modus
6. `condor_history` Gibt eine Auflistung der bereits abgeschlossenen Jobs.