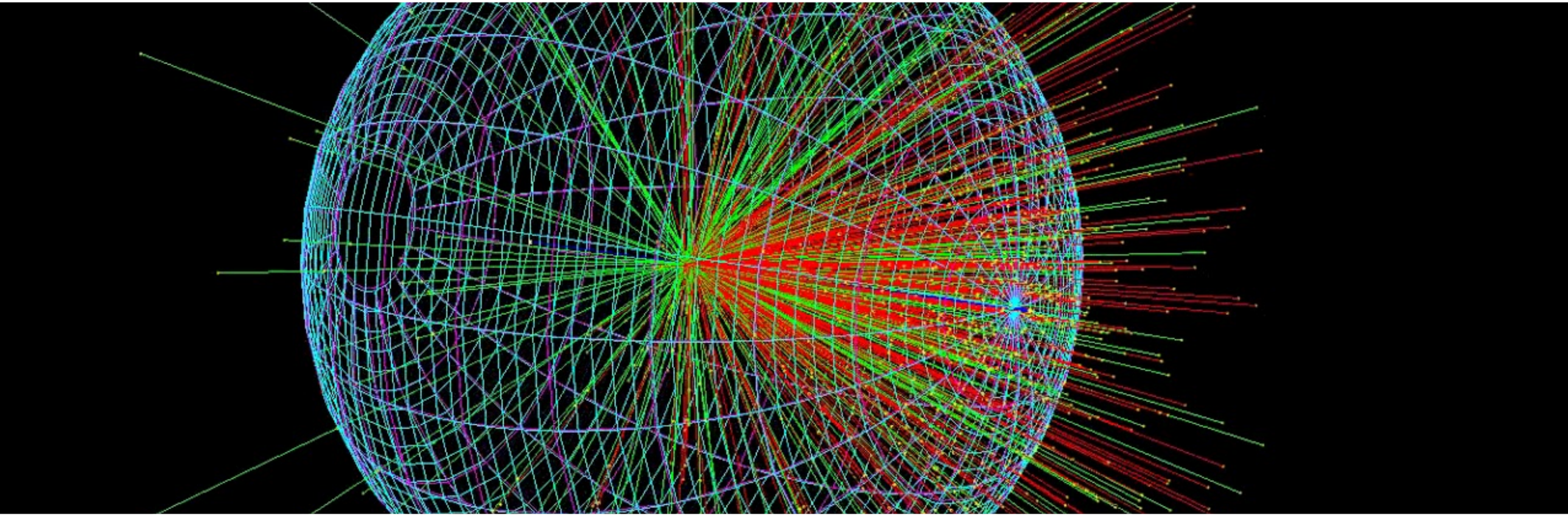
technische universität
dortmund

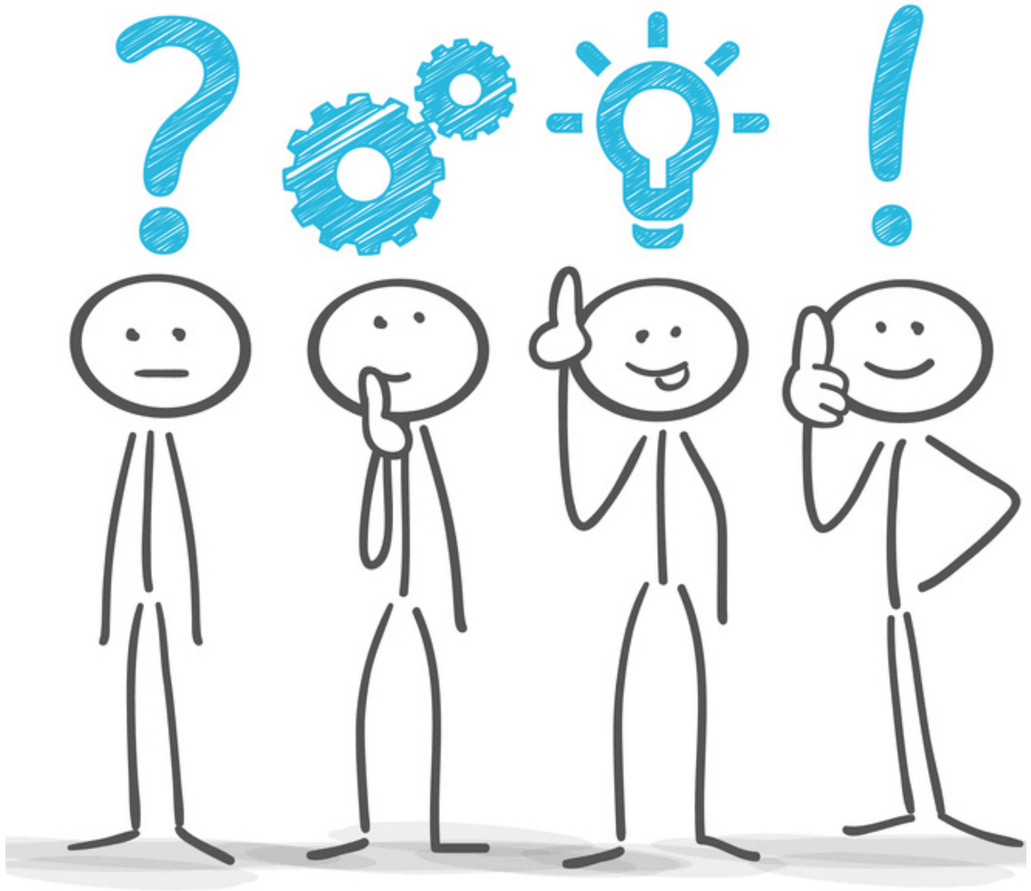


Einführung in Geant4 (und C++)

Henning Manke | henning.manke@tu-dortmund.de
Programmierkurs 2020 | Datum

Ziel des Kurses

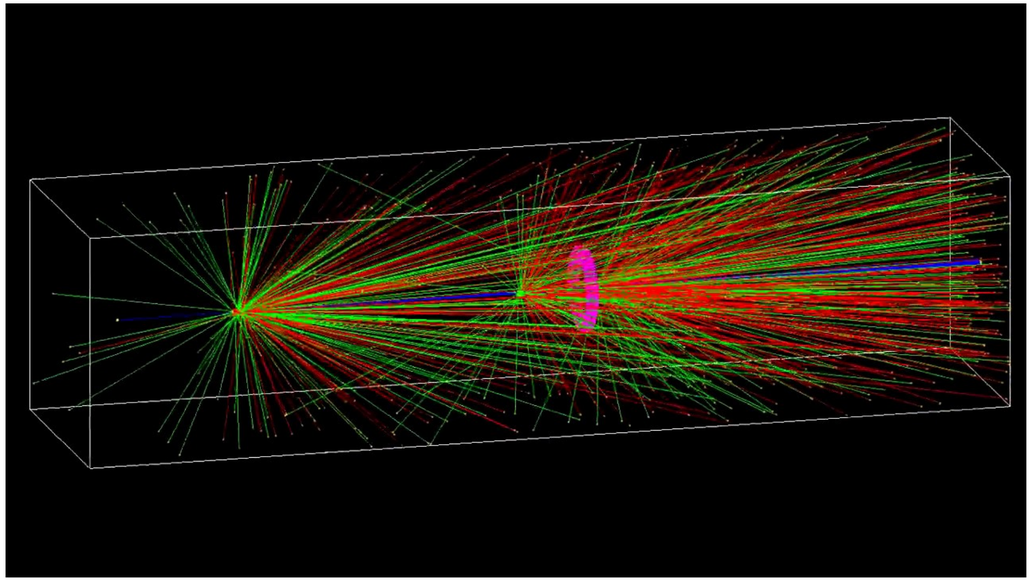
- Kein klassischer Vortrag, Mitmachen und Ausprobieren (auch währenddessen) ist explizit erwünscht!
- Dieser Kurs soll die Grundlagen zur Simulation mit GEANT4 vermitteln
- Stellt Fragen! Ich verurteile niemanden und „dumme Fragen“ gibt es nicht



© Matthias Enter -

Grundlagen

- Plattform zur Simulation von Teilchen
- Monte-Carlo-Methode —> große Statistik notwendig
- Für Hochenergiephysik entwickelt
  - Kann für niederenergetische Teilchen angepasst werden
- In C++ geschrieben
- Virtuelle Realität
  - Teilchen, Geometrie und Materialien definieren
- Datennahme
  - Beispielsweise Aufnahme von Dosisverteilungen



Installation

- Anleitung folgen
- Installation geschieht übers Terminal mit Cmake & Make
- Voraussetzungen: C++ Compiler (XCode), Xquartz (statt X11), CMake, QT
- Options setzen (vorher bedenken)
  - Install_Data, X11, QT, ... (Clang, Compiler)
- Zeit einplanen!

Installation (Beispiel für Version 10.6 in ~/Geant4/...)

```
brew install cmake
brew cask install xquartz
brew install qt
brew install xerces-c
xcode-select --install
```

- Download der neuesten Version: http://geant4.web.cern.ch/support/download
- Entpacken und den Ordner in ~/Geant4/ kopieren

```
cd ~/Geant4/
mkdir geant4.10.6-build && cd geant4.10.6-build
cmake ~/Geant4/geant4.10.6 -DCMAKE_INSTALL_PREFIX=~/geant4-.10.6-
install -DGEANT4_INSTALL_DATA=ON -DGEANT4_USE_OPENGL_X11=ON
-DGEANT4_USE_QT=ON -DGEANT4_USE_GDML=ON
make -j4 && make install
```


void generation(int A[max][
{
int i=0, j, res=0;
for(;i<(*vertex);i++)
{
j=0;
for(;j<=i;j++)
{
res= rand()%2;
A[i][j]= res;
A[j][i]= res

C++

C++

- Höhere Programmiersprache
- Sowohl objektorientierte als auch maschinennahe Programmiersprache
- Pros:
  - verbreitet
  - gut dokumentiert
  - potentiell sehr effizient
- Cons:
  - komplex (da abwärtskompatibel bis C —> sehr viele Inkonsistenzen)
  - Absolute Kontrolle und Performance <—> Einfachheit der Sprache
  - viele Wege führen nach Rom (aber welcher ist der schnellste?)

C++ - Programmaufbau

- Gliederung:
  - Compileranweisungen (z.B. #include = Einbindung von Bibliotheken)
  - Hauptprogramm
    - Main wird automatisch ausgeführt
- Kommentare
  - // einzeiliger Kommentar
  - /* mehrzeiliger Kommentar */

```
#include <iostream>

int main()
{
  x = y+2;
  x++;
  return x
}
```

C++ - Headerdateien

- Enthalten Funktionsdeklarationen und Ähnliches
- Beispiele:
  - #include <iostream> // Textausgabe aus der Standard Template Library
  - #include „some_header.h” // Eigener Header im Quellverzeichnis
- Der Code der Datei wird vollständig eingebunden und kompiliert
- Ist in der Standardbibliothek vorhanden, kann in jedem C++ Programm implementiert werden
- Dateien außerhalb der Standardbibliothek einfach ins gleiche Verzeichnis kopieren und dann einbinden

C++ - Variablendeklaration

- float = einfache Fließkommazahl
- unit8_t = Zahl mit 8Bit
- bool = True oder False
- int = Ganze Zahlen
- char = Einzelnes Zeichen
- static const = Konstante
- string = Zeichenkette

```
float Helligkeit = 0.5;
uint8_t color[3] = {0,0,150};
bool WW = false;
int modus = 5;
char delimeter[] = ",";
char mystring[11] = "Jetzt klappts";
static const uint16_t Es[2] = {0,1};
// string funktioniert erst nach
Einbinden der Klasse "string"
```

C++ - Schleifensyntax

```cpp
#include <iostream>

int main()
{
  for (int i = 0, i<10, ++i)
  {
    std::cout << i << std::endl;
  }
}
```

- {} rahmen Code ein. Variablen werden anschließend wieder gelöscht bzw. auf ihren vorigen Wert gesetzt

C++ - Schleifensyntax

- Vorsicht mit break und continue
  - Break beendet die schleife sofort
  - Continue springt zur nächsten Iteration der Schleife
- Erfüllen zwar ihren Zweck, sind aber für andere Nutzer schwieriger nachzuvollziehen

- lieber saubere Bedingungen formulieren
- (oder pragmatisch und vorsichtig verwenden)

C++ - Fallen bei Operatoren

```
int main()
{
  double a = 3.666666; // Gleitkommazahl
  a = 11/3; // = 3, weil zwei Integer dividiert wurden
  a = 11/3.0; // Gleitkommazahl
  a = 11.0/3; // Gleitkommazahl
}
```

Am Besten alles deklarieren!

Zeiger / Pointer

- Zeigt auf die Adresse einer Variable
- Ist abhängig vom Datentyp aber ist selber nur die Adresse
- Wird mit * deklariert und mit & nimmt es den entsprechenden Wert an

```
#include <iostream>

int main() {
    int   Wert;       // eine int-Variable
    int *pWert;       // eine Zeigervariable, zeigt auf einen int
    int *pZahl;       // ein weiterer "Zeiger auf int"

    Wert = 10;        // Zuweisung eines Wertes an eine int-Variable

    pWert = &Wert;    // Adressoperator '&' liefert die Adresse einer Variable
    pZahl = pWert;    // pZahl und pWert zeigen jetzt auf dieselbe Variable
```

Referenzen

- Zeigen intern auf eine Variable, sprich auf den Speicher

```
int  a = 10; // eine Variable
int  b = 20; // noch eine Variable
int &r = a;  // Referenz auf die Variable a

std::cout << "a: " << a << " b: " << b << " r: " << r << std::endl;
++a;
std::cout << "a: " << a << " b: " << b << " r: " << r << std::endl;
r = b;       // r zeigt weiterhin auf a, r (und somit a) wird 20 zugewiesen
std::cout << "a: " << a << " b: " << b << " r: " << r << std::endl;
```

Ausgabe:

```
a: 10 b: 20 r: 10
a: 11 b: 20 r: 11
a: 20 b: 20 r: 20
```

call-by-value und call-by-reference

- call-by-value:
  - Der Wert der Variable wird an die Funktion übergeben und explizit woanders gespeichert
  - Innerhalb der Funktion wird der Wert verändert, außerhalb nicht

- call-by-reference
  - Die Adresse der Variable wird an die Funktion übergeben
  - Änderungen in der Funktion wirken sich global auf die Variable aus

Klassen

- Klare Trennung von verschiedenen Zugriffen auf Daten
- Hierarchie innerhalb von Klassen möglich
- Aufbau:
  - Klassendefinition in der .h-Datei:
    - Festlegung von Eigenschaften und Methoden
  - Definition der Methoden in der .cpp-Datei

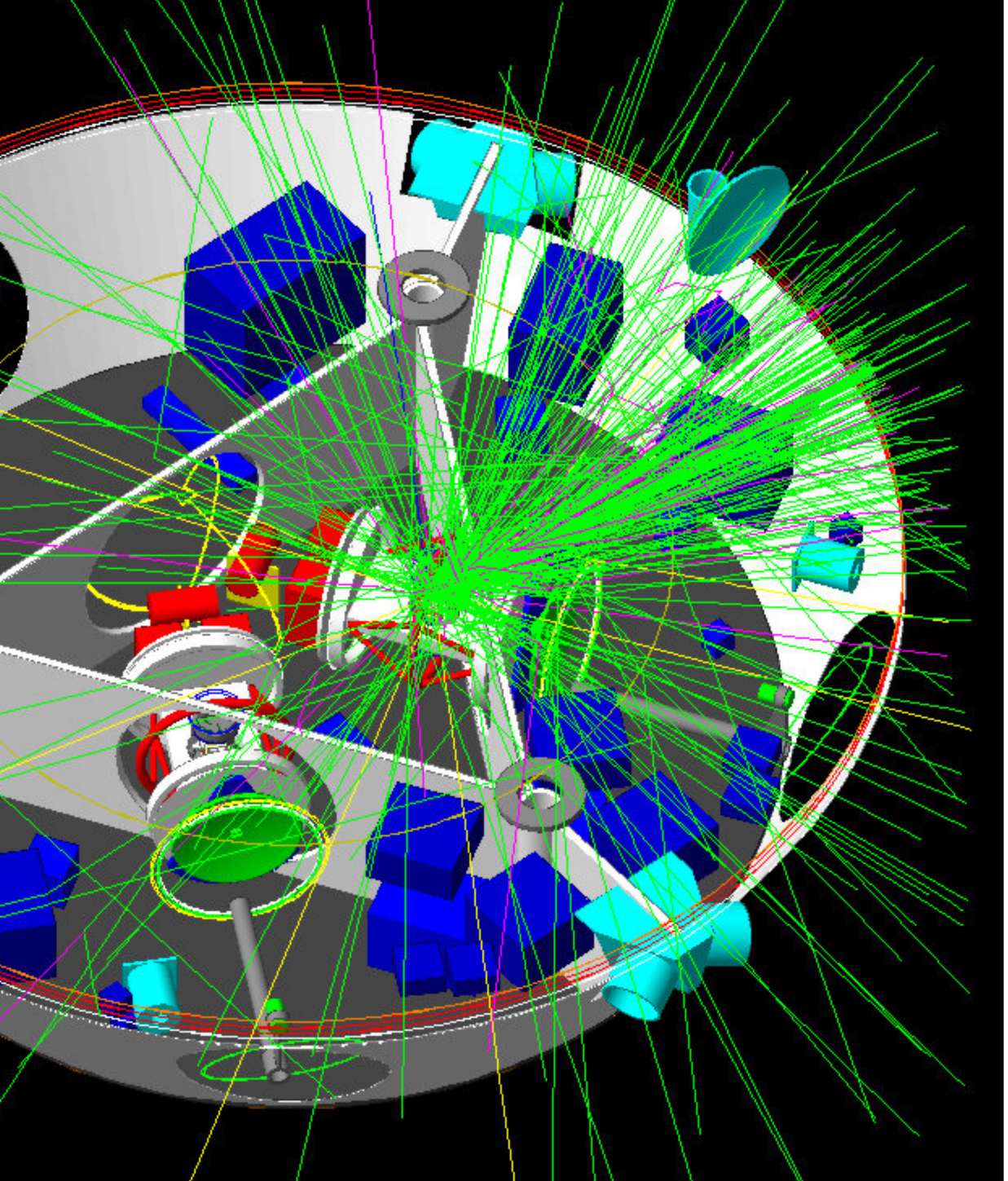
Dynamische Speicherzuweisung

- Variable wird innerhalb einer Funktion initialisiert
  - Wert geht anschließend verloren
- Speicher „reservieren” mit new,
  - Abhängig vom Datentyp und von der Länge des Arrays
  - Werte gehen nicht verloren, bis das Programm vollständig durchlaufen ist, oder explizit gelöscht werden

Dot- und Arrow-Operator

- Mit dem Dot-Operator kann auf einen Member einer Klasse zugegriffen werden
- Wird ein Pointer auf verwendet, vereinfacht der Arrow-Operator die Lesbarkeit

```
class Car
{
public:
   int number;
   void Create()
   {
      cout << "Car created, number is:  " << number << "\n" ;
   }
};


int main() {
   Car x;
   x.number = 123;
   x.Create();

   Car *y; // declare y as a pointer which points to a Car object
   y = &x; // assign x's address to the pointer y
   y->number = 456; // this is equal to (*y).number = 456;
   y->Create();
}
```



Zurück zu Geant4

Geant4 - Verlauf

- Initialization
  - Material/Geometrie
  - Teilchen/Prozesse/WW-Querschnitte
  - Welche Informationen will ich „scoren“?
- BeamOn
  - Teilchen werden losgeschickt —> Events
  - Mit wenigen Teilchen anfangen, Zeit stoppen, mit Sicherheitssaum hochrechnen
- Datenauslesen

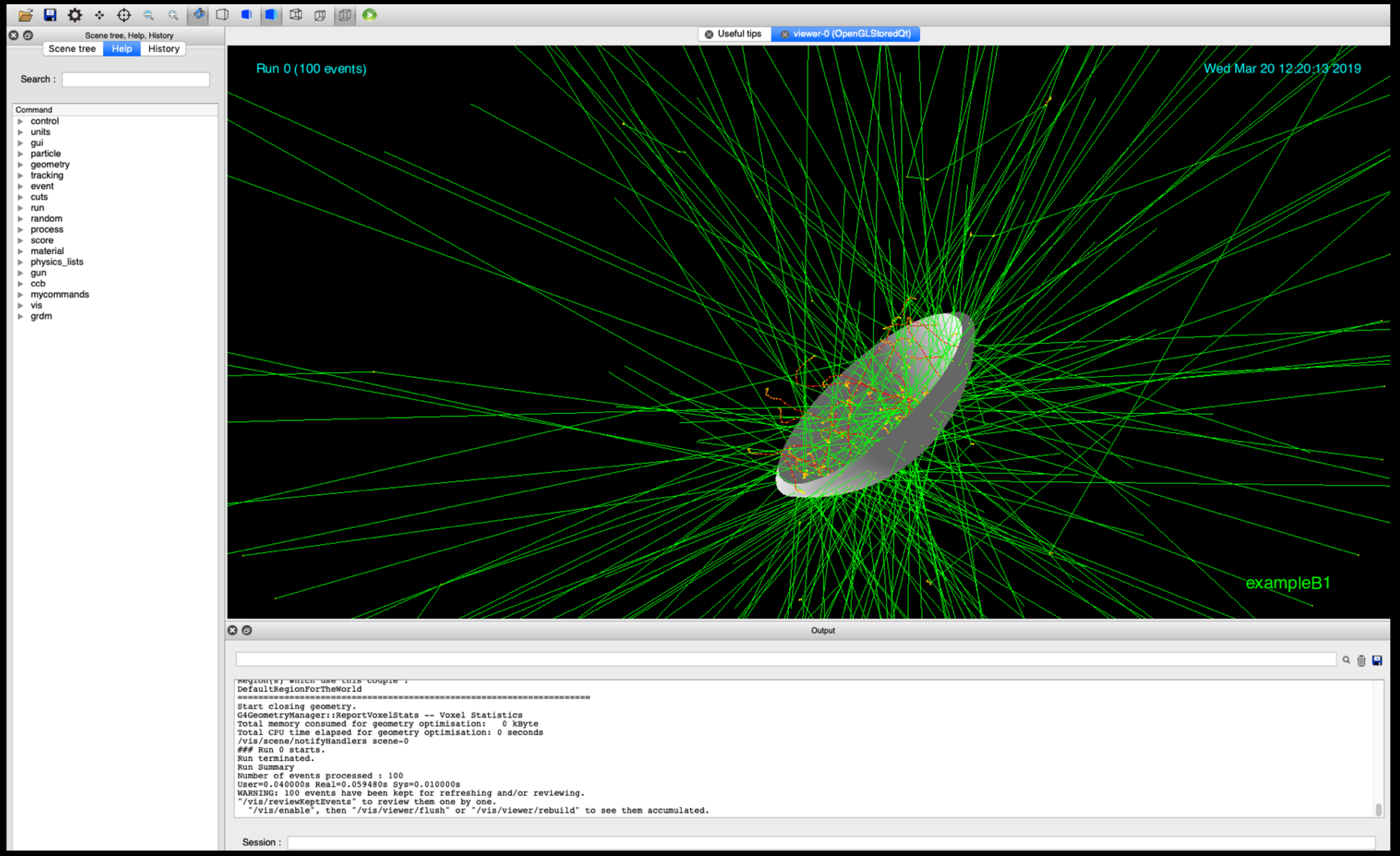
- Links:
  - Offizielle Dokumentation zu Geant4
  - Schönes Tutorial mit vielen Beispielaufgaben
  - Vortrag der Geant4 Programming School
  - Veröffentlichte CCB-Simulation
  - Präsentation über Materialdefinitionen

Aufbau und Durchführung einer Simulation

- Geant4 einbinden
- In den build-Ordner navigieren
- CMake
- Make
- Main-Datei ausführen

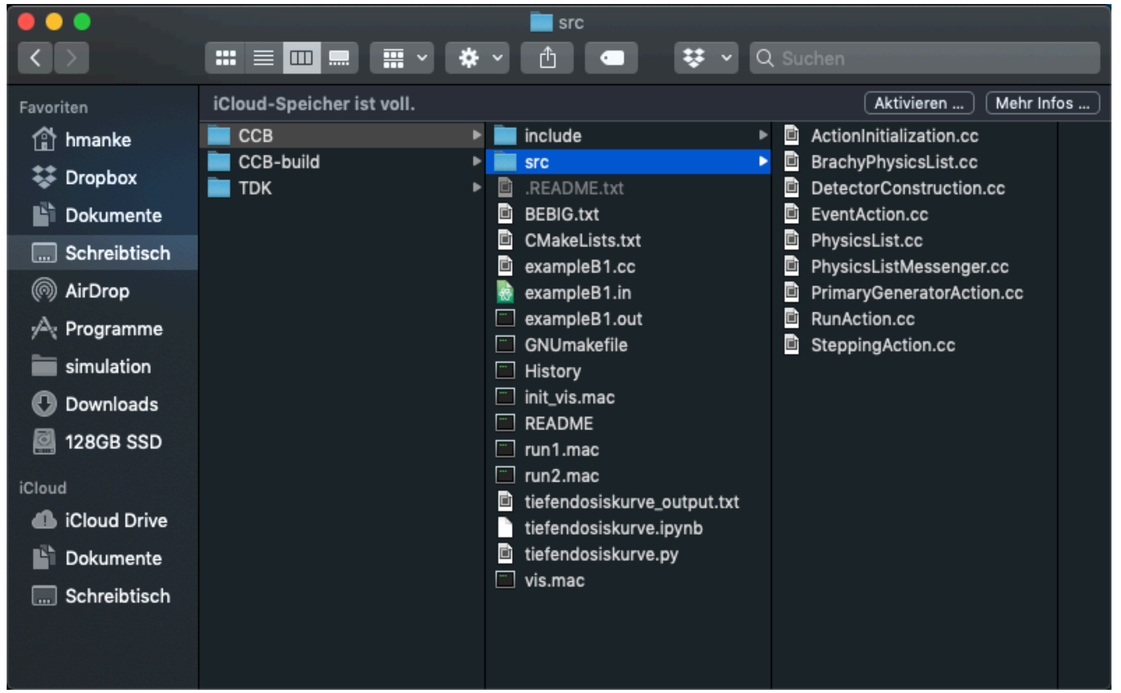
- Warnings kontrollieren und gegebenenfalls beheben, Errors beheben

```
$ source /PathTo/Geant4/geant4-install/bin/geant4.sh
### lädt die Einstellungen und Pakete
### kann auch in die bash_profile

$ mkdir /PathTo/Simulation-build
$ cd /PathTo/Simulation-build
### wechseln in einen Build-Ordner

$ cmake -DGeant4_DIR=/PathTo/geant4-install/lib/Geant4
/PathTo/Simulation
### alias in bash_profile erspart die flag

$ make -jN
### N = Anzahl verwendeter CPU-Kerne, meist 4

$ ./mainDatei
### Ausführen der Simulation mit Visualisierung

$ ./mainDatei runfile.mac
### Ausführen der Simulation ohne Visualisierung mit
### vorgefertigtem Skript
```


Scene tree, Help, History
Scene tree
Help
History
Search :
Command
control
units
gui
particle
geometry
tracking
event
cuts
run
random
process
score
material
physics_lists
gun
ccb
mycommands
vis
grdm
Useful tips
viewer-0 (OpenGLStoredQt)
Run 0 (100 events)
Wed Mar 20 12:20:13 2019
exampleB1
Output
DefaultRegionForTheWorld
Start closing geometry.
G4GeometryManager::ReportVoxelStats -- Voxel Statistics
Total memory consumed for geometry optimisation: 0 kByte
Total CPU time elapsed for geometry optimisation: 0 seconds
/vis/scene/notifyHandlers scene-0
Run 0 starts.
Run terminated.
Run Summary
Number of events processed : 100
User=0.040000s Real=0.059480s Sys=0.010000s
WARNING: 100 events have been kept for refreshing and/or reviewing.
"/vis/reviewKeptEvents" to review them one by one.
"/vis/enable", then "/vis/viewer/flush" or "/vis/viewer/rebuild" to see them accumulated.
Session :

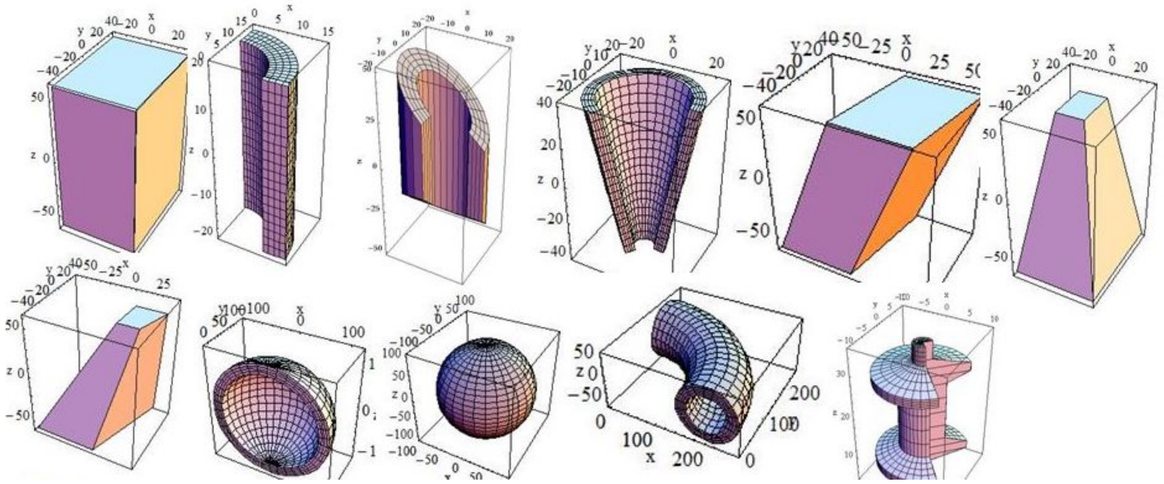
User classes

- Initialization classes
  - G4VUserDetectorConstruction
  - G4VUserPhysicsList
- Action classes
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction
  - G4UserEventAction
  - G4UserStackingAction
  - G4UserTrackingAction
  - G4UserSteppingAction
- Jeweils die .cc Dateien im src-Ordner



DetectorConstruction

- Construct()
  - Materialien definieren
  - Volumen definieren
    - Solids (Form, Maße)
    - Logical (Eigenschaften wie Material, Magnetische Felder, Sensitivität)
    - Physical (Position und Rotation)
    - Hierarchischer Aufbau (Welt à Volumen à Tochtervolumen)
- Visualisierung (Farben festlegen)
- Boolsche Operationen möglich



DetectorConstruction - Beispiel

```
G4ThreeVector positionCalorBox = G4ThreeVector(0,0,0);

solidCalorBox =
   new G4Box("CalorBox",CalorBoxX/2.,CalorBoxY/2.,CalorBoxWidthZ/2.);
logicCalorBox =
   new G4LogicalVolume(solidCalorBox , VacMaterial, "CalorBox",0,0,0);
physiCalorBox =
   new G4PVPlacement(0,                    // no rotation
                     positionCalorBox, // at (x,y,z)
                     logicCalorBox,    // its logical volume
                     "CalorBox",       // its name
                     logicWorld,       // its mother  volume
                     false,            // no boolean operations
                     0);               // copy number
```

Materialien

- Datenbanken
- Aus Elementen oder bekannten Materialien zusammensetzen und neu definieren
- Dichte neu zuweisen

```
G4double fractionmass;

G4Material* Air =
    manager->FindOrBuildMaterial("G4_AIR");
    manager->ConstructNewGasMaterial(„Air20”, „G4_AIR",
    293.*kelvin, 1.*atmosphere);

G4Material* lAr =
    manager->FindOrBuildMaterial("G4_lAr");
G4Material* lArEm3 =
    new G4Material("liquidArgon", density= 1.390*g/cm3,
    ncomponents=1);
    lArEm3->AddMaterial(lAr, fractionmass=1.0);
```

PhysicsList

- Physikalische WW, Querschnitte, … definieren
- Alles was ihr braucht ist quasi schon implementiert, bis auf niederenergetische Prozesse (—> Einbinden der BrachyPhysicsList)
- Verschiedene Simulationsalgorithmen (Single oder Multiple-Scattering)
  - Kompromiss aus geforderter Rechenpower/Zeit und Genauigkeit
  - Auch für alle Sekundärteilchen müssen Dateien mit Daten eingebunden werden

- Insbesondere: Bei der später verwendeten CCB-Simulation könnt ihr den Simulationsalgorithmus wählen!

PrimaryGeneratorAction

- ParticleDefinition
  - Teilchenart, Position, Richtung, Energie, …
  - G4ParticleTable à vordefinierte Teilchen
- ParticleCreation

```
B1PrimaryGeneratorAction::B1PrimaryGeneratorAction()
: G4VUserPrimaryGeneratorAction(),
  fParticleGun(0),
  fEnvelopeBox(0)
{
  G4int n_particle = 1;
  fParticleGun  = new G4ParticleGun(n_particle);

  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);
}

void B1PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{

  G4double envSizeXY = 0;
  G4double envSizeZ = 0;

  G4double size = 0.8;
  G4double x0 = size * envSizeXY * (G4UniformRand()-0.5);
  G4double y0 = size * envSizeXY * (G4UniformRand()-0.5);
  G4double z0 = -0.5 * envSizeZ;

  fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));

  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

Hauptprogramm

- G4RunManager
- Andere Klassen einbinden
  - BrachyPhysicsList.hh (!!!)
- setSeed (müssen bei statistischen Simulationen voneinander abweichen!)
- PhysicsList und damit den Simulationsalgorithmus bewusst wählen
- Visualisierung oder keine Visualisierung
- Und vieles mehr ...

Befehle geben

- Interaktiv in der GUI
  - Einzeln eingeben und abschicken
    - Tab-Autocomplete
  - Direkte Visualisierung
  - Super zum Kontrollieren
- Batch
  - Macro-Datei erstellen und durchlaufen lassen
  - Gut zum Ergebnisse produzieren
  - SetSeeds Zahl1 Zahl2
  - Scoring Meshes
  - beamOn
  - Score

```
### Kommentierung im Runfile mit #

/run/beamOn 100
### erzeugt 100 Teilchen gemäß Primary Generator
### Action

/control/execute/ file.txt
### führt ein komplettes File mit allen Befehlen
### aus

### Viele weitere Befehle findet ihr in unseren
### Files
```

Aufgaben

1. Kopiert euch das Beispiel
   `/PathTo/geant4.10.05-install/share/Geant4-10.6.0/examples/basic/B1`
   an einen beliebigen Ort und führt es aus
2. Öffnet die Datei /B1/src/B1DetectorConstruction.cc
   I. Verändert nach Belieben die Volumen shape1 und shape2
   II. Fügt eine neue Geometrie ein (hohle Halbkugel, Material: Silber)
   III. Anmerkung: Cmake nach jedem Hinzufügen von Dateien, dabei CmakeLists.txt im Auge behalten. Make nach jeder Änderung innerhalb von Dateien
3. Öffnet die Datei /B1/src/B1PrimaryGeneratorAction.cc
   I. Ändert Energie, Richtung und Startposition der Teilchen.

Aufgaben

5. Öffnet die Datei /B1/run1.mac und macht euch mit den Befehlen vertraut
   I. Führt in der Konsole ./exampleB1 run1.mac aus
   II. Fügt der Datei einen sogenannten Command based Scorer hinzu:

/score/create/boxMesh boxMesh_1
/score/mesh/boxSize 5. 5. 5. Cm
/score/mesh/nBin 1 1 1
/score/mesh/translate/xyz 0 0 0 cm
/score/quantity/doseDeposit dDep
/score/close

#/score/drawProjection boxMesh dDep
#/score/colorMap/setMinMax ! 0.01 0.99

/score/dumpAllQuantitiesToFile boxMesh Scorer_run1.txt

- Links:
  - Offizielle Dokumentation zu Geant4
  - Schönes Tutorial mit vielen Beispielaufgaben
  - Vortrag der Geant4 Programming School
  - Veröffentlichte CCB-Simulation

Aufgaben

6. Ladet euch die Simulation eines CCB Applikators herunter
   I. Link
   II. Führt die Simulation aus und macht euch mit den Geometrien vertraut
   III. Schaut euch die erzeugten Partikel an
   IV. Nehmt mit run1.mac und run2.mac jeweils eine Tiefendosiskurve auf (schaut euch zunächst die Geometrie des Scorers an). Achtet auf die Teilchenzahl (Dauer der Simulation)
   V. Führt das Skript „tiefendosiskurve.ipynb” aus

- Links:
  - Offizielle Dokumentation zu Geant4
  - Schönes Tutorial mit vielen Beispielaufgaben
  - Vortrag der Geant4 Programming School
  - Veröffentlichte CCB-Simulation