GEANT4 : overview

- Why use a Simulation
- What is GEANT4
- How it works
- How to write a simulation?
- Do I trust what comes out
- Where to get help

Conclusion

* some of the material is borrowed from a CERN G4 Workshop | Jan, 2020:

Why use simulation:

What simulation can help with:

Design phase of the experiment

Development/validation of the software like tracking, calibration, etc

Producing physical results

Estimate acceptance, estimate background rates, estimate expected energy loss

Pioneer a new technology | Feasibility study

What is a simulation:

Particles + Matter = ? | A GENERAL APPROACH

- Define the system components (Geometry)
- Define what enters the system (Particles)
- Define what happens within the system (Physics)

- Use software which will model the passage of particles that user defines through the system and model* interactions

* based on the Monte Carlo method of random sampling

What is G4?

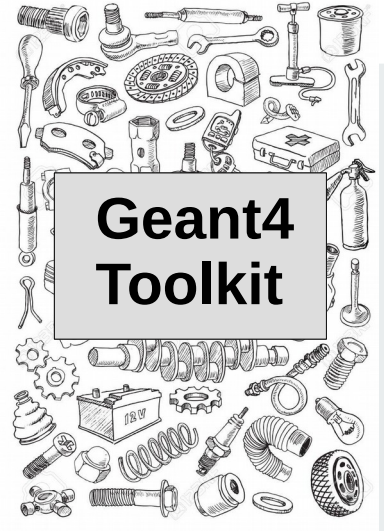
G4 is a toolkit that will help you with the following problem:

G4 DOES: step-by-step particle transport through the detector that user defines. Taking into account interactions (which user selects), and external fields (user defined) until the particle
- loses all its kinetic energy
- decays or disappears by interaction (photo effect for example)
- exits the simulation volume

Users can access the information as this transportation process occurs
- at each step
- when particle is in a certain volume
- when particle decays
- etc

This is denoted as “USER ACTIONS”

What is G4?



So G4 is an Open-Source Simulation Toolkit

Toolkit is NOT A READY-TO-USE program!

What it means, it’s an assortment of tools that you can use to solve the simulation problem(s) that we talked about

What is G4?

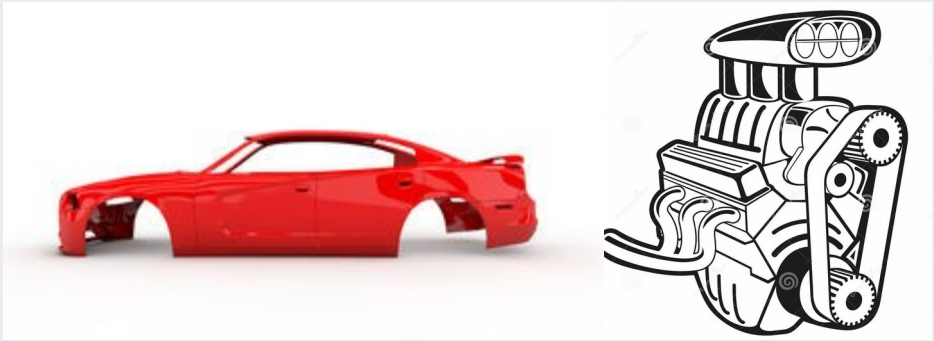
it’s an assortment of tools that you can use
to solve the simulation problem we talked about

Geometry

What is G4?

it’s an assortment of tools that you can use
to solve the simulation problem we talked about

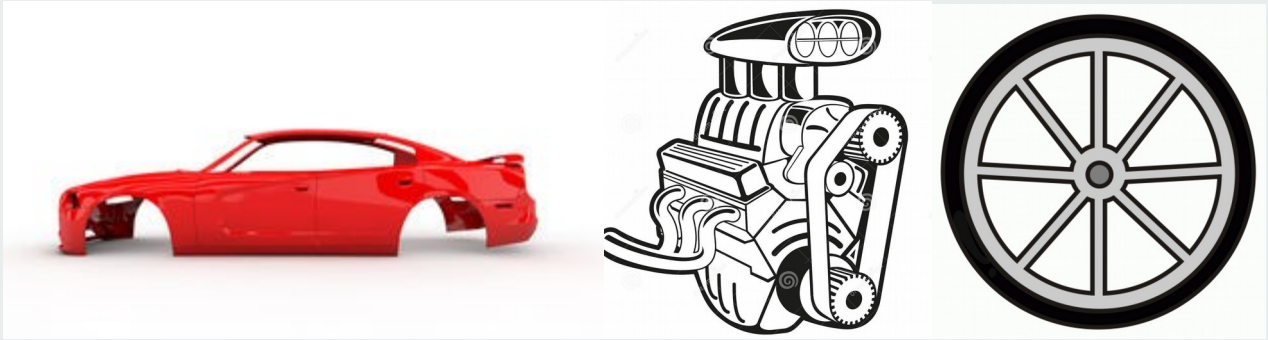
Geometry + Physics



What is G4?

it’s an assortment of tools that you can use
to solve the simulation problem we talked about

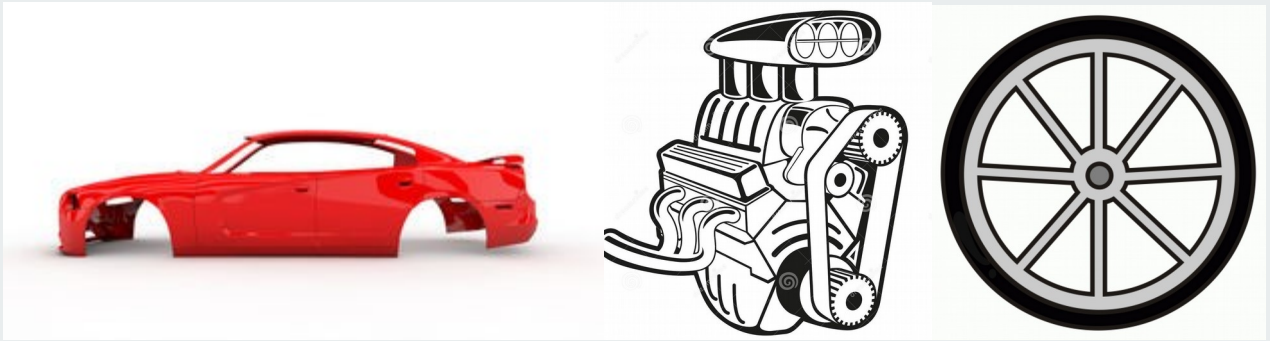
Geometry + Physics + Particles



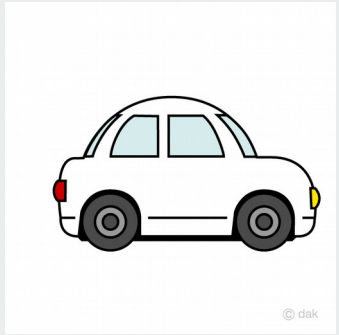
What is G4?

it’s an assortment of tools that you can use
to solve the simulation problem we talked about

Geometry + Physics + Particles



=



Your simulation

What is G4?

To recap: Geant4 is a toolkit developed:

- for simulating the passage of particles through matter

- provides all the necessary components needed to describe and to solve particle transport simulation problems

- **Input:** geometry, physics, particles etc.

- **Output**: step-by-step particle transport computation with information that accessible by user at every step (if there is need)

How it works...

Geant4 is implemented in C++

User: can operate a simulation through macro files or GUI can also do basic changes to the code with basic C++

Developer: needs substantial knowledge of C++ and Geant4 work-flow

Both need to understand how Geant4 works to avoid problems/conflicts

Materials & Geometry:

Material definition

- The Geant4 material model
- Material definition
- The NIST material data base

Geometry definition

- The Geant4 geometry description

  Concepts of: Solid, Logical - , Physical - volume

Materials & Geometry:

G4 Material model:

In nature, materials (chemical compounds, mixtures) are made of elements, and elements are made of isotopes. Geant4 has three main classes designed to reflect this organization. G4Element, G4Isotope, G4Material

Material documentation for reference.

Example:

```
#include "G4Isotope.hh"
#include "G4Element.hh"
#include "G4Material.hh"
#include "G4UnitsTable.hh"
int main() {
G4String name, symbol;             // a=mass of a mole;
G4double a, z, density;            // z=mean number of protons;
G4int iz, n;                       // iz=nb of protons  in an isotope;
                                   // n=nb of nucleons in an isotope;

G4int ncomponents, natoms;
G4double abundance, fractionmass;
G4double temperature, pressure;
G4UnitDefinition::BuildUnitsTable();
// define Elements
a = 1.01*g/mole;
G4Element* elH  = new G4Element(name="Hydrogen",symbol="H" , z= 1., a);
a = 12.01*g/mole;
G4Element* elC  = new G4Element(name="Carbon"  ,symbol="C" , z= 6., a);
a = 14.01*g/mole;
G4Element* elN  = new G4Element(name="Nitrogen",symbol="N" , z= 7., a);
a = 16.00*g/mole;
G4Element* elO  = new G4Element(name="Oxygen"  ,symbol="O" , z= 8., a);

// define an Element from isotopes, by relative abundance
G4Isotope* U5 = new G4Isotope(name="U235", iz=92, n=235, a=235.01*g/mole);
G4Isotope* U8 = new G4Isotope(name="U238", iz=92, n=238, a=238.03*g/mole);
G4Element* elU  = new G4Element(name="enriched Uranium", symbol="U", ncomponents=2);
elU->AddIsotope(U5, abundance= 90.*perCent);
elU->AddIsotope(U8, abundance= 10.*perCent);
```

G4Element

G4Isotope
G4Element

Materials & Geometry:

Simple Material:

```
// define simple materials
density = 2.700*g/cm3;
a = 26.98*g/mole;
G4Material* Al = new G4Material(name="Aluminum", z=13., a, density);
```

Compound material

```
density = 1.000*g/cm3;
G4Material* H2O = new G4Material(name="Water", density, ncomponents=2);
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```

All of this is useful, but there is an easier more consistent way:

Materials & Geometry:

NIST material database integrated into Geant4:

Use these pre-defined materials whenever possible:
- guaranties high accuracy for many derived parameters (consistency)

NIST and more pre-defined materials (318 (?) at the moment):
- single element NIST materials with Z = [1-98] and named after the atomic symbol:
  aluminum (“G4_Al”), silicon (“G4_Si”), gold (“G4_Au”), etc.
- compound NIST materials: “G4_AIR”, “G4_MUSCLE_SKELETAL_ICRP”, etc.
- HEP and nuclear materials: liquid argon “G4_lAr”, lead tungstate “G4_PbWO4”, etc.
- space materials: “G4_KEVLAR”, “G4_NEOPRENE”, etc.
- bio-chemical materials: the DNA bases “G4_ADENINE”, “G4_GUANINE”, etc.

Materials & Geometry:

NIST material database:

```
G4NistManager* man = G4NistManager::Instance();
man->SetVerbose(1);

// define elements
G4Element* C  = man->FindOrBuildElement("C");
G4Element* Pb = man->FindOrBuildMaterial("Pb");

// define pure NIST materials
G4Material* Al = man->FindOrBuildMaterial("G4_Al");
G4Material* Cu = man->FindOrBuildMaterial("G4_Cu");

// define NIST materials
G4Material* H2O  = man->FindOrBuildMaterial("G4_WATER");
G4Material* Sci  = man->FindOrBuildMaterial("G4_PLASTIC_SC_VINYLTOLUENE");
G4Material* SiO2 = man->FindOrBuildMaterial("G4_SILICON_DIOXIDE");
G4Material* Air  = man->FindOrBuildMaterial("G4_AIR");
```

Materials & Geometry:

• Geant4 detector geometry description is composed of three conceptual layers: Solid, Logical-Volume, Physical-Volume
• users need to construct them directly in their user code (Detector Construction) by “new”

• geometry description can be complex or simple (depends on what you are simulating, so maybe start with simpler approximations)

• more information on the detector geometry description can be found in the corresponding documentation: Detector Geometry

Materials & Geometry:

G4VSolid:

• the shape of the Geant4 detector geometry builds up from geometrical primitives, all derived from the G4VSolid base class that provides interface to:

– compute distances between the shape and a given point

– check whether a point is inside the shape

– compute the extent of the shape

– compute the surface normal to the shape at a given point

• Geant4 makes use of Constructed Solid Geometry (CSG) to define these geometrical primitives: G4Box, G4Tubes, G4Trd, G4Para, G4Trap, G4Torus, etc.. (special CSG-like solids e.g. G4Polycone, G4Polyhedra, G4Ellipsoid, etc., tessellated and boolean solids are also available. See the Geometry: Solids documentation).

• these three-dimensional primitives described by a minimal set of parameter to define the dimensions of the corresponding solid e.g. G4Box

• these implement the G4VSolid base class interface methods DOCUMENTATION

Materials & Geometry:

G4LogicalVolume:

• encapsulates all information of a detector volume element except its real physical position (position and rotation):
– the shape and dimensions of the volume i.e. a G4VSolid
– the material of the volume i.e. G4Material that is the minimally required additional information beyond the solid
– additional, optional information such as magnetic field (G4FieldManager) or user defined limits (G4UserLimits), etc.
• its NOT a base class! Its constructor:

• see the Geometry: Logical Volumes documentation DOCUMENTATION

Materials & Geometry:

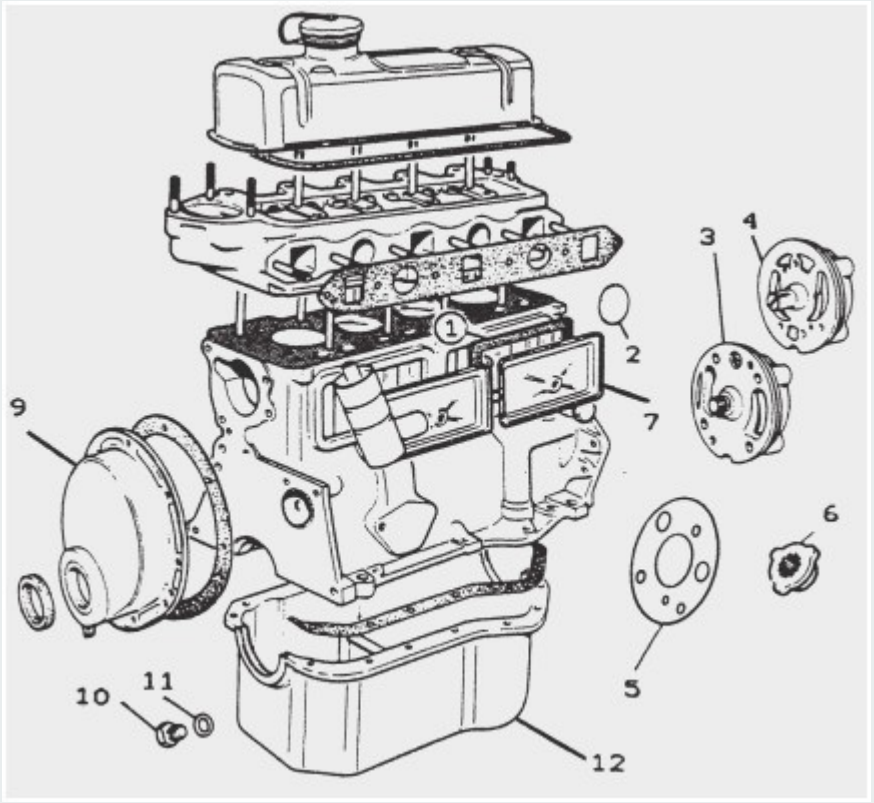
G4VPhysicalVolume:

• the abstract base class for representation of physically positioned volumes
• a volume is positioned in a mother volume relative to its coordinate system
• the positioning can be:
– placement volume: one positioned volume, i.e. one G4VPhysicalVolume object represents one “real” volume
– repeated volume: one volume positioned many times, i.e. one G4VPhysicalVolume object represents multiple copies of “real” volumes (reduces memory by exploiting symmetry)

Physical Volume DOCUMENTATION

PhysicsList

- What is a Physics List?
- The Geant4 Physics List interface
- Modular and reference Physics Lists



PhysicsList

Physics List is an object that is responsible to:
- specify all the particles that will be used in the simulation application
- together with the list of physics processes assigned to them

Provides a very flexible way to set up the physics environment
- the user can choose, specify the particles that they want to be used
- the user can choose the physics to assign to each particle

BUT: the user must have a very good understanding of the underlying physics

PhysicsList

Modular physics list:
- significantly easier way to obtain/compose a physics list?
- allows to use “physics modules”: a given physics module handles a well defined category of physics (e.g. EM, hadronic physics, decay, etc.)
- transportation is automatically added to all constructed particles

Reference physics list:
- even easier way to obtain a complete physics list
- these are “ready-to-use”, complete physics lists provided by the toolkit and constructed by expert developers
- each pre-packaged physics list includes different combinations of EM and hadronic physics
- given as it is: the user is responsible for validating them
- see more details can be found in the Guide for Physics Lists

PhysicsList

Reference physics list example:
- see more details can be found in the Guide for Physics Lists

```
// To be able to use and combine the refrence physics lists
#include "G4PhysListFactory.hh"
```

```
//
// 2. PhysicsList = G4VModularPhysicsList <- G4VUserPhysicsList
//    (use reference physics list through the G4PhysListFactory)
const G4String plName = "FTFP_BERT_EMZ";
G4PhysListFactory plFactory;
G4VModularPhysicsList *pl = plFactory.GetReferencePhysList( plName );
runManager->SetUserInitialization( pl );
```

Here an FTFP_BERT list is chosen with the EMZ option (EM Opt4)

PrimaryGeneratorAction

User needs to define: particle, position, momentum

```
{
  G4int n_particle = 1;
  fParticleGun  = new G4ParticleGun(n_particle);

  // default particle kinematic

  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle
                   = particleTable->FindParticle("chargedgeantino");
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.));
  fParticleGun->SetParticleEnergy(1*eV);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(1.,0.,0.));
}
```

And Finally needs to generate primaries:

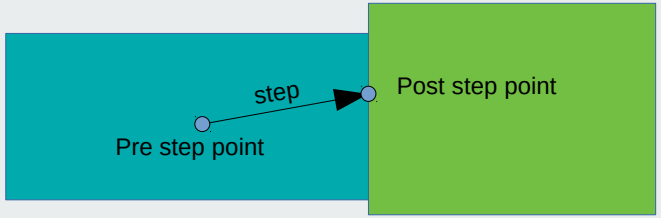
```
  //create vertex
  //
  fParticleGun->GeneratePrimaryVertex(anEvent);
```

How it Runs...

G4Track:
- a G4Track object represents/describes the state of a particle that is under simulation in a given instant of the time (i.e. a given time point)
It consists of:
- G4ParticleDefinition stores static particle properties (charge, mass, etc.) as it describes a particle type (e.g. G4Electron)
- G4DynamicParticle stores dynamic particle properties (energy, momentum, etc.)
- G4Track object is propagated in a step-by-step way during the simulation and the dynamic properties are continuously updated to reflect the current state

- manager: G4TrackingManager; optional user hook: G4UserTrackingAction

How it Runs...



G4Step:
- a G4Step object can provide the information regarding the change in the state of the particle (that is under tracking) within a simulation step (i.e. delta)
- has two G4StepPoint-s, pre- and post-step points, that stores information (position, direction, energy, material, volume, etc...) that belong to the corresponding point (space/time/step)
- (important) if a step is limited by the geometry (i.e. by a volume boundary), the post-step point:
• physically stands on the boundary (the step status of the post step point i.e. G4Step::GetPostStepPoint()->GetStepStatus() is fGeomBoundary)
• logically belongs to the next volume
• since these “boundary” G4Step-s have information both regarding the previous and the next volumes/materials, boundary processes (e.g. reflection, refractions) can be simulated
the G4Track object, that is under tracking i.e. generates information for the G4Step object, can be obtained from the step by the G4Step::GetTrack() method and the other way around G4Track::GetStep()
- manager: G4SteppingManager; optional user hook: G4UserSteppingAction

How it Runs...

• G4Event:
- a G4Event is the basic simulation unit that represents a set of G4Track objects
  - primary G4Track object(s) is(are) generated and pushed to a track-stack
  - one G4Track object is popped from this track-stack and transported/tracked/simulated:
    - the track object is propagated in a step-by-step way and its dynamic properties as well as the corresponding G4Step object are updated at each step
    - secondary G4Track-s, generated by these physics interactions, are also pushed to the track-stack
  - G4Track object is tracked until:
    - leaves the outermost (World) volume
    - participates in a destructive interaction (e.g. decay or photoelectric absorption)
    - loses all kinetic energy and doesn't have interaction that can happen “at-rest”
    - the user decided to (artificially) stop the tracking and kill the track
  - when one track object reaches its termination point, a new G4Track object (either secondary or primary) is popped from the stack for tracking
- Processing an event is done once the track stack is empty

How it Runs...

G4Run:

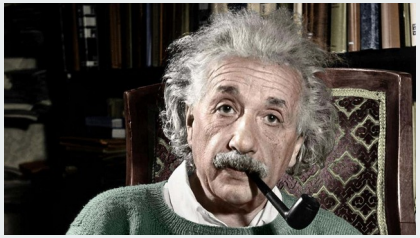
- G4Run is a collection of G4Event-s (a G4Event is a collection of G4Track-s)
- during a run, events are taken and processed one by one in an event-loop
- before the start of a run i.e. at run initialization (G4RunManager::Initialize()):
  The geometry is constructed and physics is initialized
- during the run, the user cannot modify neither the geometry (i.e. the detector setup)
  nor the physics settings (analogous with an experiment)

- manager: G4RunManager; optional user hook: G4UserRunAction

How / Where to start?

I hear and I forget,
I see and I remember,
I do and I understand

– Confucius



It’s not that I am smart,
it’s just that I stay with
problems longer

A. Einstein

G4 and your project

Writing you own simulation:
Before you start YOU NEED TO:
- understand your geometry (materials/volumes)
- understand your primary particles
- understand your physics
- know what output you need and in what format
  (as simple as ASCI file…. Or …. .root or even an .hdf5 data)

Write from Scratch

Use Examples and "mod" them

Examples Guide

Do I trust the output

Precision is case-by-case. Read up on validation of the G4

Always try to validate your simulation

- you can make mistakes
- G4 can have bugs
  (but if they are big, they get discovered very quickly due to large number of users)

Still, ideally stay away from the new releases. It’s safer to use previous release with the latest patch.

Use common sense to simulate some simple cases and check if results make sense.

Help!!!

Guides from Geant4

Available on the web and in .pdf form. And they are constantly improving, and getting refined.

G4 User Forum

A wealth of information. Search for answers, if nothing is found, then ask and be wait for an answer.

Reach out to the community

ALSO: If situation allows one can attend a Geant4 school.

Conclusion

Powerful tool

Strong team and a huge base of users (22 years and running)

Open source – a big plus in my mind

Do not need to be a C++ expert to use

Learn by doing, reach out in the forums for support

Be critical of results, if a bug is found → report it!!!

Conclusion

Powerful tool

Strong team and a huge base of users (2[illegible]ning)

Open source – a big plus in my mind

Do not need to be a C++ expert

Learn by [illegible] the [illegible] support

Be critical of results & bugs found → report it!!!