Introductory talk on Geant4

Raman Sehgal (BARC)

Things to be discussed

1) Quick Brushup of OOPs in C++

2) Geometries in Geant4

3) Structure of Geant4 application

Quick Brush up of C++

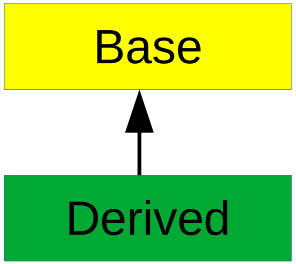
1) Class is basically a user-defined data type

2) The variables of class is known as objects

3) Class contains following

   (a) Data members : The variable defined inside the class

   (b) Member functions: The functions that operates on those variables



4) Constructor : A special function without any return type and is called automaticallyupon creation of objects of class

5) Construct can be default (without any parameters), or parameterized constructor.

6) Its always a good practice to define constructor. These are used to set the data member upon creation Of object.

```
class Base {
   int fDataMemberBaseClass;
public:
    Base() { std::cout << "Base Constructor\n"; }
    ~Base() { std::cout << "Base Destructor\n"; }
};

class Derived : public Base {
   int fDataMemberDerivedClass;
public:
    Derived() { std::cout << "Derived Constructor\n"; }
    ~Derived() { std::cout << "Derived Destructor\n"; }
    void Print();
};

int main() {
    Derived obj;
}
```

10) Two types of classes are there:

(a) **Abstract** classes : Objects **CAN'T** be instantiated

(b) **Concrete** classes : Object **CAN** be instantiated

13) It is mandatory to implement all the pure virtual function in derived class, otherwise the derived class itself become an Abstract class

14) Pointers of base class can hold the reference to the object of base class (A very important concept, which is extremely used while write Geant4 simulation code)

```
Base *ptr = new Derived; // base class holding object of derived class
Derived  *derivedPtr = static_cast<Derived*>(basePtr) //casting the base class pointer to derived class
```

Code snippet to demonstrate Abstract and Concrete class

```
class Shape {
public:

    virtual void Area() = 0;
    // Pure virtual function

};
```

```
class Square : public Shape {
double side;
public:
    Square(double s){
        side = s;
    }
    void Draw() {
       std::cout << "Drawing Circle\n";
    }
    void Area() {
        std::cout << "Area : " << (side*side)
<< std::endl;
    }
};
```

```
int main(){
 //Shape s ;
 //Not allowed as it is
a abstract class

Square s(4);
s.Area();

 return 0;
}
```

Geometry in Geant4 (Things to be discussed)

1) Steps involved to create the detector geometries

2) Some of the complex geometries available in Geant4

3) Discussion of “Materials” in brief.

4) Geometry hierarchy in a detector setup.

5) How to import / export the geometry

6) Use of GDML

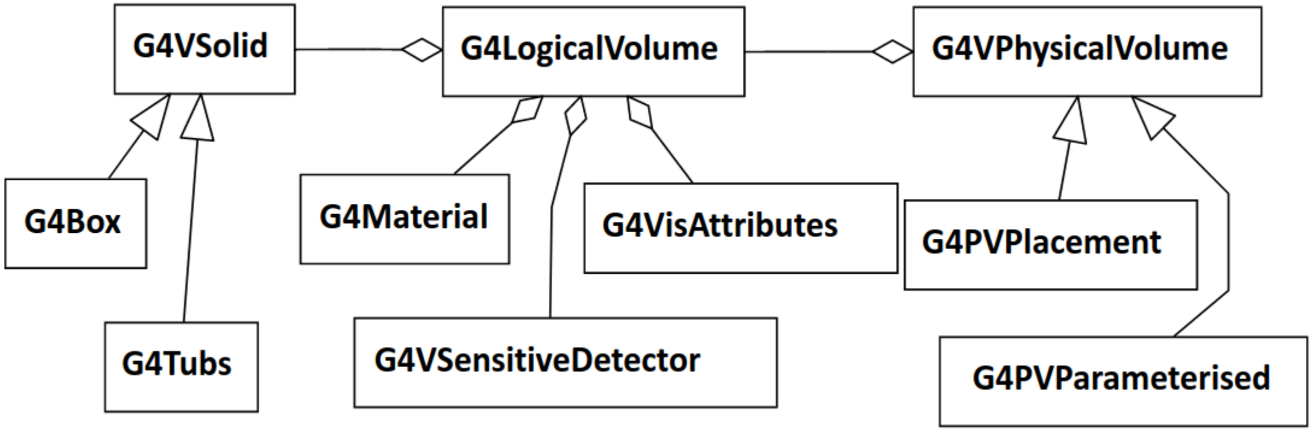
Software architecture of Detector geometry

Basically consist of three layers

1) Solid (Shape) :   G4VSolid : Defines the shape and size of the geometry

2) Logical Volume : G4LogicalVolume : material, sensitivity, visualization
attributes, physical placement of daughter volumes etc.

3) Physically place volume : G4VPhysicalVolume : defines position, rotatation,
and mother volume



Various Shapes available in Geant4

CSG (Constructed Solid Geometry) solids

- G4Box, G4Tubs, G4Cons, G4Trd, G4Para, G4Trap, G4Torus, G4CutTubs, G4Orb, G4Sphere

Specific solids (CSG like)

- G4Polycone, G4Polyhedra, G4Hype, G4Ellipsoid, G4EllipticalTube, G4Tet, G4EllipticalCone, G4Hype, G4GenericPolycone, G4GenericTrap, G4Paraboloid

Tessellated solids

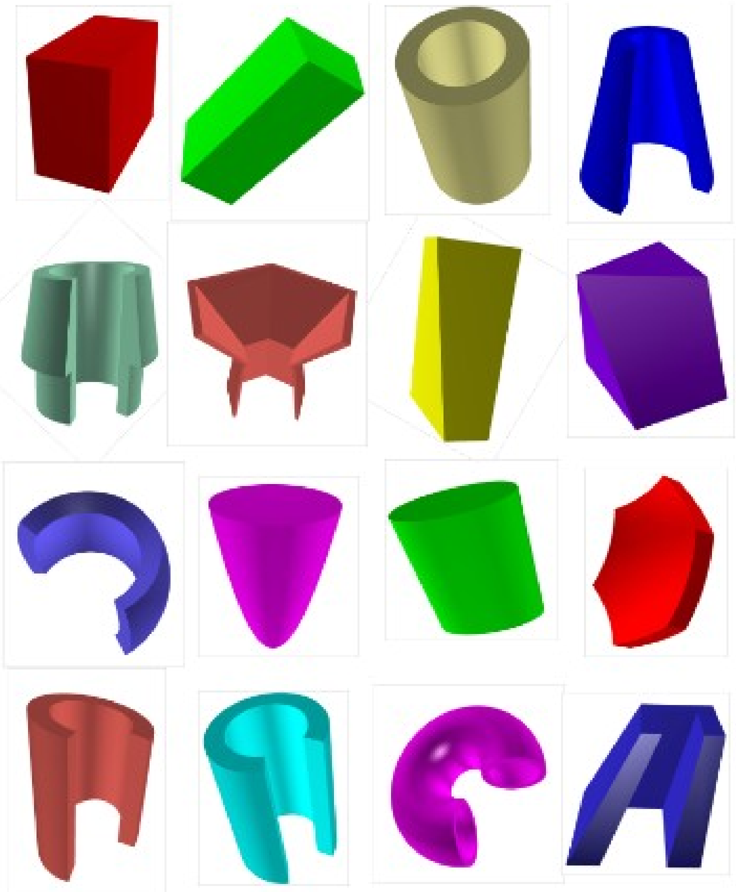
- G4TessellatedSolid, G4ExtrudedSolid

Boolean & scaled solids

- G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid, G4MultiUnion, G4ScaledSolid

Twisted shapes

- G4TwistedBox, G4TwistedTrap, G4TwistedTrd, G4TwistedTubs

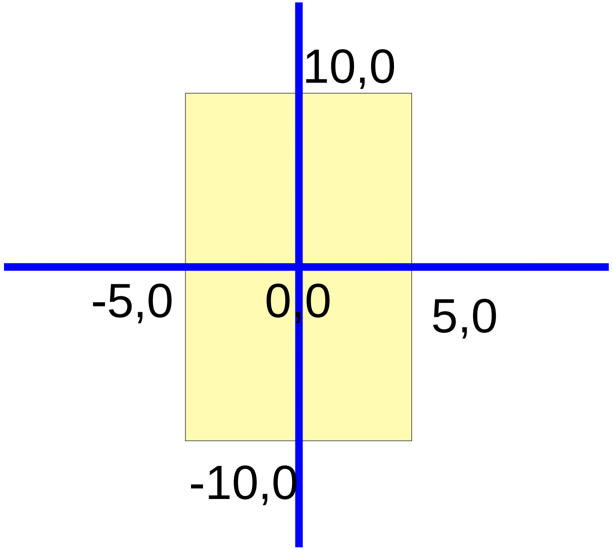


Concept of Half lengths

Geant4 geometry works on the concept of half lengths

Suppose we want to create the box of
[10 cm X 20 cm X 30 cm] : Required dimension
[5 cm X 10 cm X 15 cm] : Specified Half length

Same concept is applicable to all the geometries
Cone, Tube, etc.



Cone and Polycone

Worth discussing as they are used in frequently in experiment setup.

Cone is same as tube but having different lower and upper radius.

Polycone is something like connecting various cones and tubes one after the another.

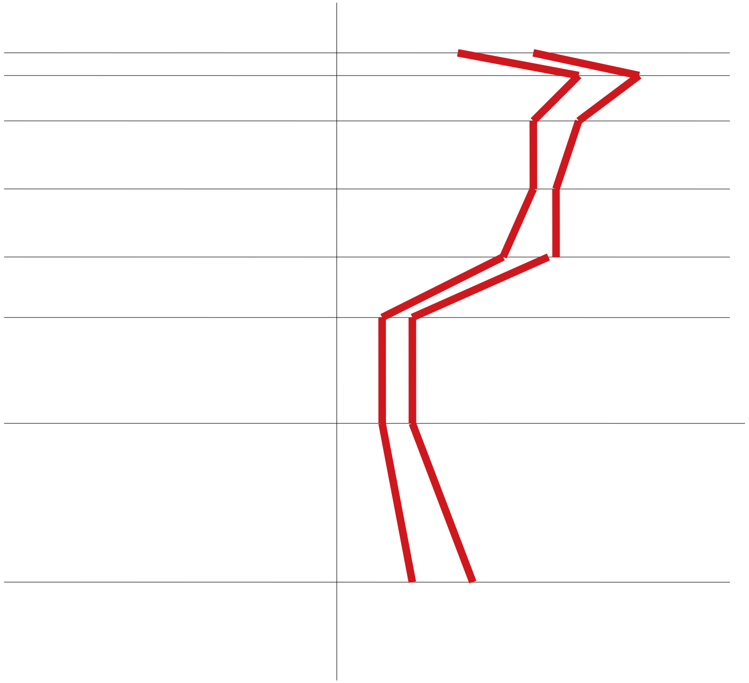
Different constructors exist

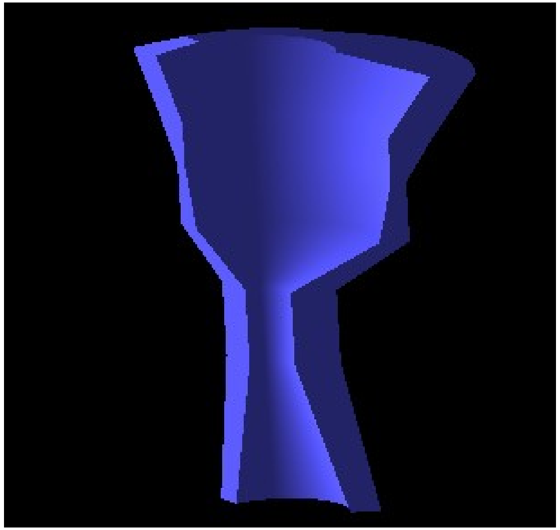
```
G4Polycone("MyPolycone", sPhi, dPhi, numZ, z,  rmin,  rmax);
```

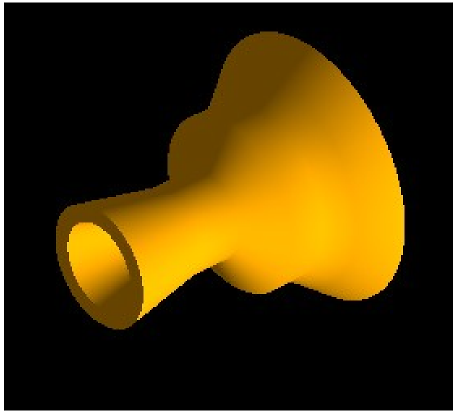
```
double z[8]       = {-10., 0., 5., 8., 12., 15., 19, 21};
double rmin[8]  = {5., 2., 2., 8., 9, 9, 12., 6};
double rmax[8] = {7., 5., 5., 10., 10, 12, 15, 8};
G4Polycone("LeadBlock",0.,  2*M_PI,  8,  z,  rmin,  rmax);
```

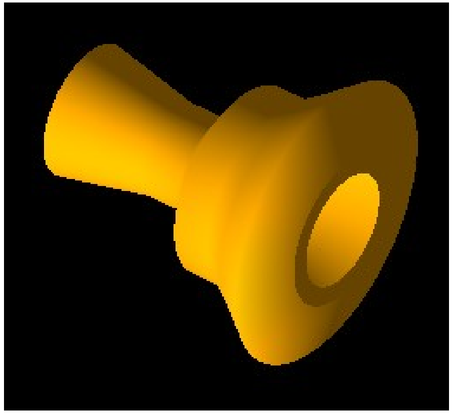
Polycone

```
double z[8]       = {-10., 0., 5., 8., 12., 15., 19, 21};
double rmin[8]  = {5., 2., 2., 8.,    9,   9, 12., 6};
double rmax[8] = {7., 5., 5., 10., 10, 12, 15, 8};
G4Polycone("LeadBlock",0.,  2*M_PI,  8,  z,  rmin,  rmax);
```









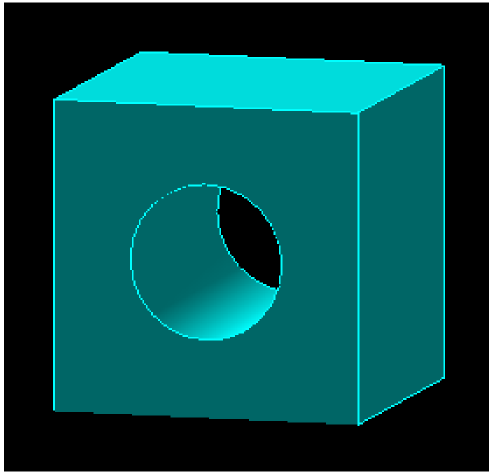
Boolean Operation

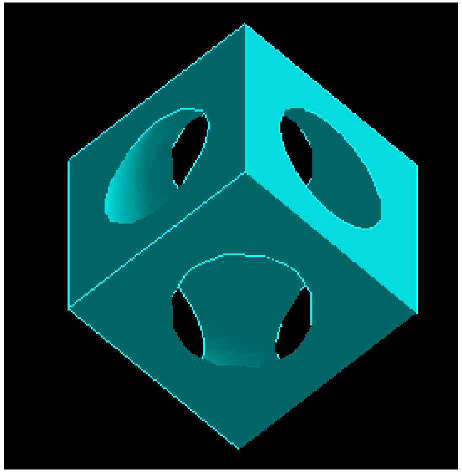
G4SubtractionSolid :
Subtraction of one shape from another.

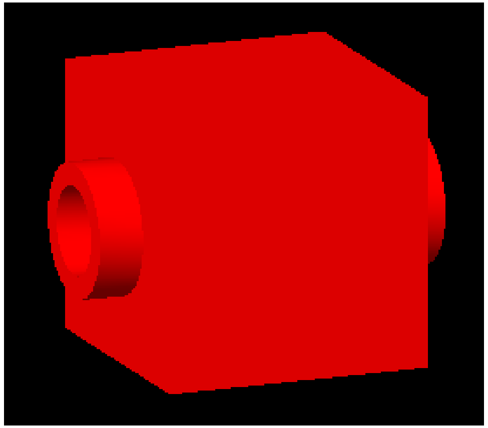
```
G4SubtractionSolid(  const G4String&pName,
                     G4VSolid* pSolidA ,
                     G4VSolid* pSolidB   ) ;

G4Box boxA("boxA",3*m,3*m,3*m);
G4Orb orb("orbB",4*m);

G4SubtractionSolid
subtracted("subtracted_boxes",&boxA,&orb);
```

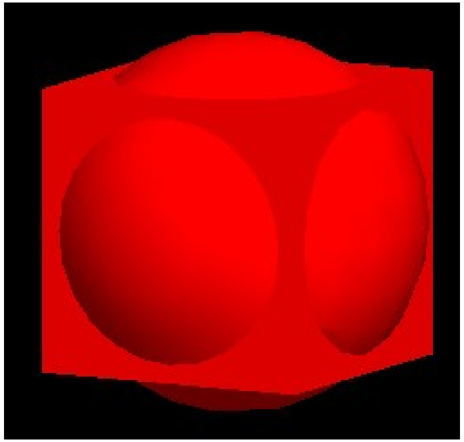
G4UnionSolid:
Union of two shapes.

```
G4UnionSolid(  const G4String&pName,
               G4VSolid* pSolidA ,
               G4VSolid* pSolidB   ) ;
```









Defining Materials

Material can be define in two ways :

1) Using the exising NIST database provided by Geant4

-- Contains a lot of material as elements, isotopes and compound.

-- Need an object of NistManger class

-- G4NistManager *nist = G4NistManager::Instance();

-- G4Material *world_mat = nist->FindOrBuildMaterial("G4_AIR");

(G4_Pb, G4_Al, G4_Mg, G4_Na .. etc.)

(G4_BAKELLITE, G4_ANTHRACENE etc..)

2) Making your own material that can be defined using the various classes available

-- Isotope	: G4Isotope
-- Element	: G4Element
-- Molecules	: G4Material
-- Compound and Mixture	: G4Material

Defining Materials

Let’s start with a single-element material:

```
G4double density = 4.506*g/cm3;
G4double a = 47.867*g/mole;
G4Material* ti = new G4Material(“pureTitanium”, z=22, a, density);
```

Creating Elements

```
G4double a = 1.01*g/mole;
G4Element* elH = new G4Element(“Hydrogen”, “H”, z=1, a);
a = 16.00*g/mole;
G4Element* elO =  new G4Element(“Oxygen”, “O”, z=8, a);
```

Finally creating material from elements

```
G4double density = 1.0*g/cm3;
G4int ncomp = 2;
G4Material* H2O = new G4Material(“Water”, density, ncomp);
G4int nAtoms;
H2O->AddElement(elH, nAtoms=2);
H2O->AddElement(elO, nAtoms=1);
```

Getting Water from NISTManager object

```
G4Material *world_mat =
nist->FindOrBuildMaterial("G4_WATER");
```

Creating Logical Volume and it Physical placement

```
G4LogicalVolume(G4VSolid* pSolid,
                G4Material* pMaterial,
                const G4String& name);
```

```
G4Box box("test", 5*cm, 5*cm, 5*cm);
G4Material Al;
G4LogicalVolume *logicalBox = new
G4LogicalVolume(box, Al, "LogicalBox");
```

```
G4PVPlacement(G4RotationMatrix *pRot,
              const G4ThreeVector &tlate,
              G4LogicalVolume *pLogical,
              const G4String& pName,
              G4LogicalVolume *pMotherLogical,
              G4bool pMany,
              G4int  pCopyNo,
              G4bool pSurfChk=false);
```

```
new G4PVPlacement(0,
                  G4ThreeVector(),
                  logicalBox,
                  "PhysicalVolume",
                   motherLogicalVol,
                   false,
                   0,
                   true);
```

Understanding the Geometry Hierarchy

Raw shapes never forms the part of geometry hierarchy.

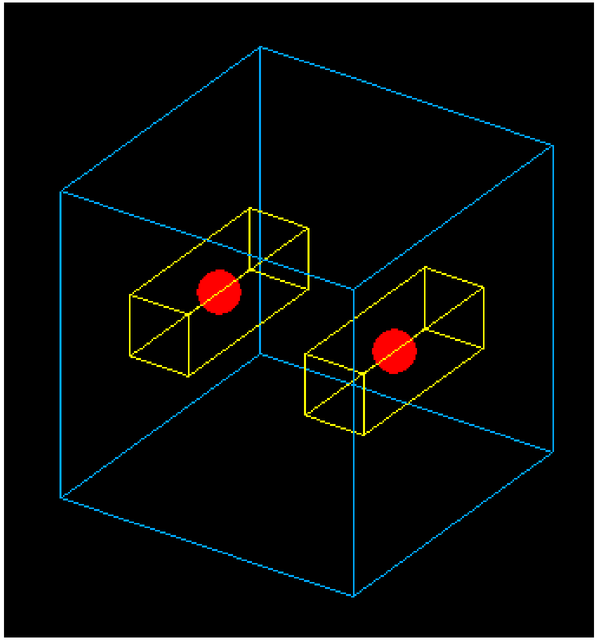
Physical placement is alway done for a logical volume.

The geometry hierarchy consist of Mother-Daughter relationship.

One Logical volume contains other Physical Volume daughter volumes

The mother logical volume forms the local coordinate system for all its daughter volumes.

If a mother volume is placed more than once, all its daughter volumes will be there in all physical volumes

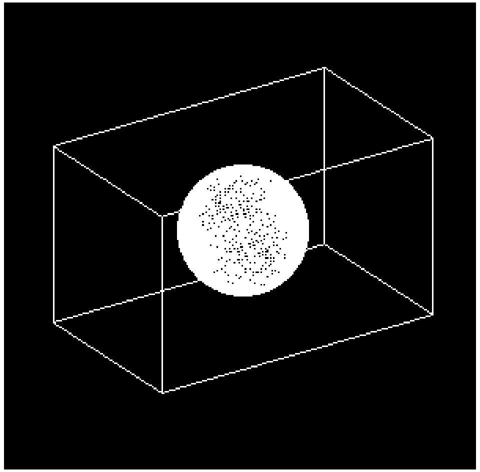


Only Exception : World Volume
Its a unique physical volume which contains all the other volume of your detector setup

World volume also forms the global coordinate system.

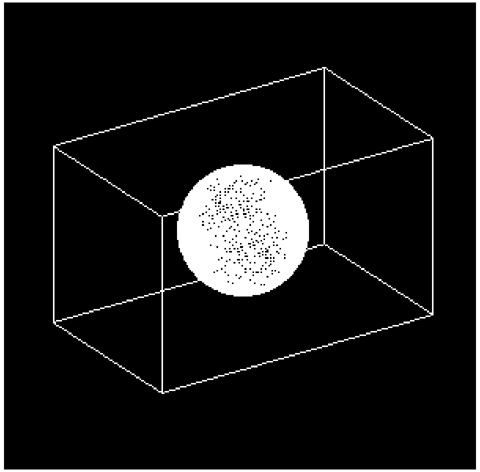
Understanding Physical placement in the Geometry Hierarchy

Both Box and Orb are placed with respect to **world reference frame**



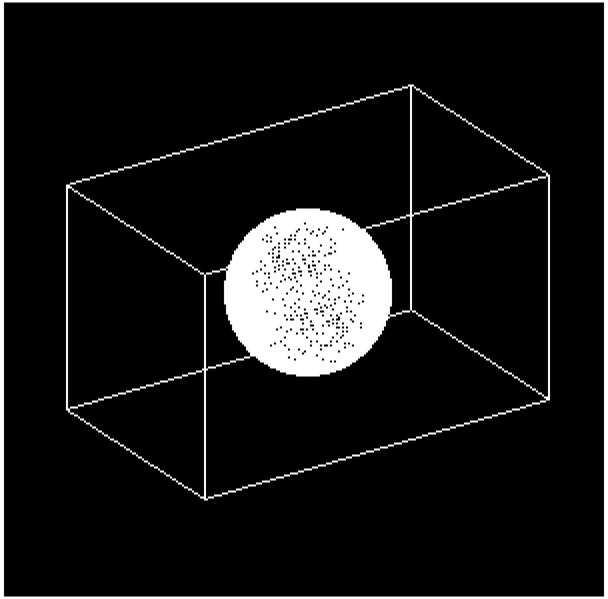
Box is placed with respect to **world reference frame**

Orb is place with respect to **box reference frame**



Both Box and Orb are placed with respect to **world reference frame**

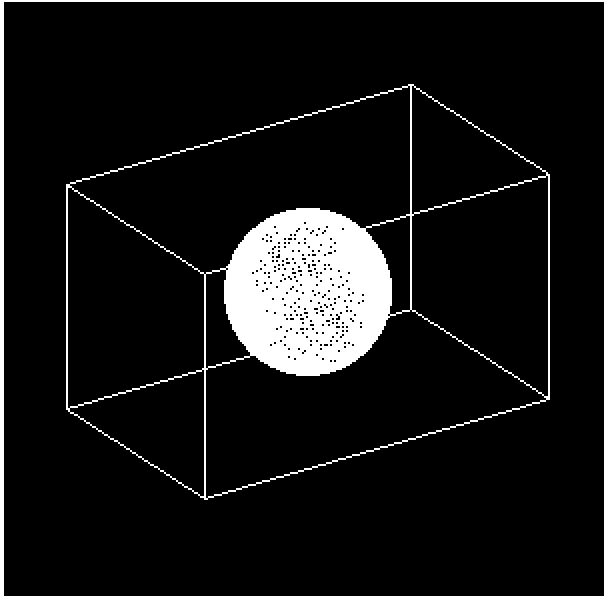
```
"/vis/list" to see available colours.
Checking overlaps for volume PhysicalLead ... OK!
Checking overlaps for volume PhysicalOrb ...
-------- WWWW ------- G4Exception-START -------- WWWW --
*** G4Exception : GeomVol1002
      issued by : G4PVPlacement::CheckOverlaps()
Overlap with volume already placed !
          Overlap is detected for volume PhysicalOrb:0
          with PhysicalLead:0 volume's
          local point (78.0177,-62.4168,4.16952), overla
pping by at least: 7.19823 cm
```



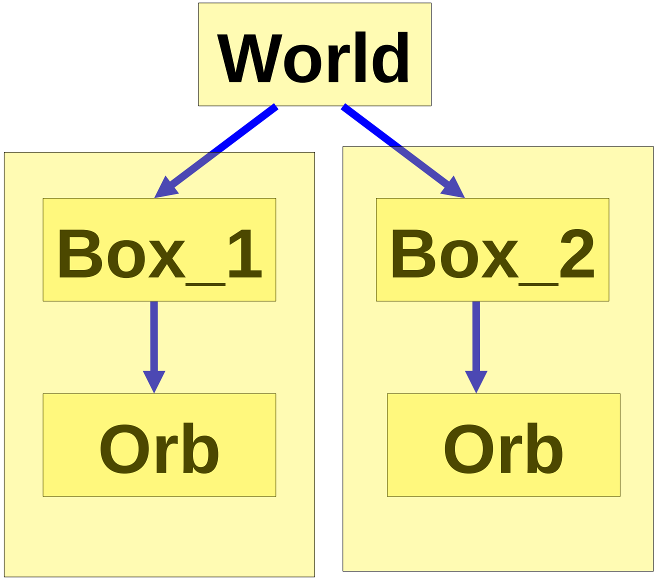
Box is placed with respect to **world reference frame**

Orb is place with respect to **box reference frame**

```
Some /vis commands (optionally) take a string to specify
 colour.
"/vis/list" to see available colours.
Checking overlaps for volume PhysicalLead ... OK!
Checking overlaps for volume PhysicalOrb ... OK!
G4GDML: Writing 'test.gdml'...
G4GDML: Writing definitions...
G4GDML: Writing materials...
G4GDML: Writing solids...
G4GDML: Writing structure...
G4GDML: Writing setup...
G4GDML: Writing surfaces...
G4GDML: Writing 'test.gdml' done !
```



Geometry Hierarchy Tree



Complete hierarchy contains 5 shapes

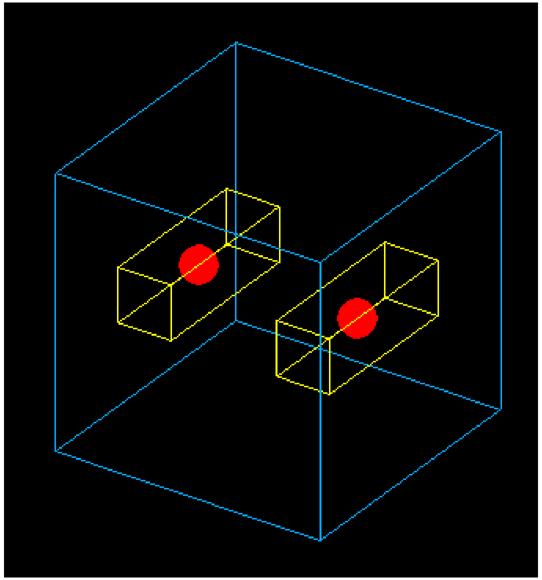
But you had created only 3 shapes.

World, Box and Orb

Mother box contains Orb daughter

Multiple placement of mother box contains all the daugter volumes

Make sure you give proper copy number and name to physical placement



Exporting Geometry

- If the same geometry setup needs to be used in multiple simulations or needs to be used by different people
- Geant supported geometry export format
- GDML (Graphics Description Markup language)
- A portable format, similar to XML.
- Can be read by standalone application
- Various XML reading libraries are present.
- Xerces-C is used by Geant4

Moving ahead

- Till now, whatever we understood

C++

- Defining shapes, logical volume and their physical placement
- Define material and attaching them to the shapes to convert them to logical volumes

Alternative way to achieve the same thing.

Instead of doing a the detector construction at compile that (as done above),
Idea is to generate it at run time.
Reading a text file (XML)
Benefits : Allows to quickly recreate the full detector construction with very few lines of code

GDML : Graphics Description Markup Language

- XML formatted text file.
- It implements hierarcy of volumes in a detector setup as the tree of geometries.
- Allows to define the material, and place the volumes.
- Makes the detector construction portable, and independent of the remaining simulation code.

Benefits of using GDML

- Language independent
- Containing user defined tags.
- Can be processed by any library that can process XML.
- Provides hierarchal structure, and mother daughter relationship can be easily maintained.
- Hierarchical structure make its suitable for object oriented programming.

Overview of GDML : Various Components

The flow of a default GDML file follows:

1) Definitions

2) Materials

3) Solids

4) Structures

5) Setup

```
-<gdml xsi:noNamespaceSchemaLocation="http://service-
  spi.web.cern.ch/service-spi/app/releases/GDML/schema
  /gdml.xsd">
    <define/>
   +<materials></materials>
   +<solids></solids>
   +<structure></structure>
   +<setup name="Default" version="1.0"></setup>
  </gdml>
```

NOTE : Your internet browser is a very useful tool to have a look at the XML file.

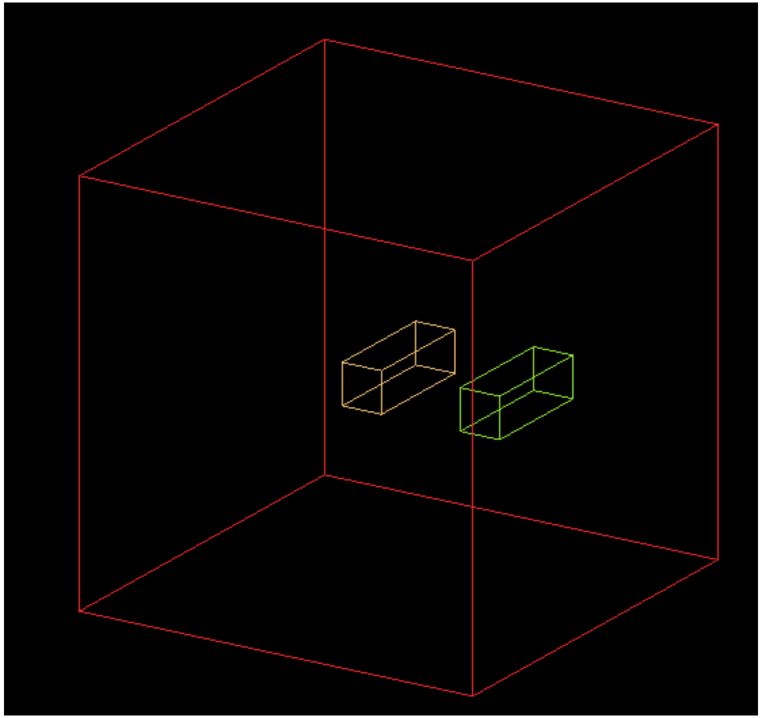
Various GDML Solids

GDML supports all the solids provided by Geant4

Box
Orb
Sphere
Cone
Tube
Parboloid
Ellipsoid
Polyhedro
Polycone
Torus

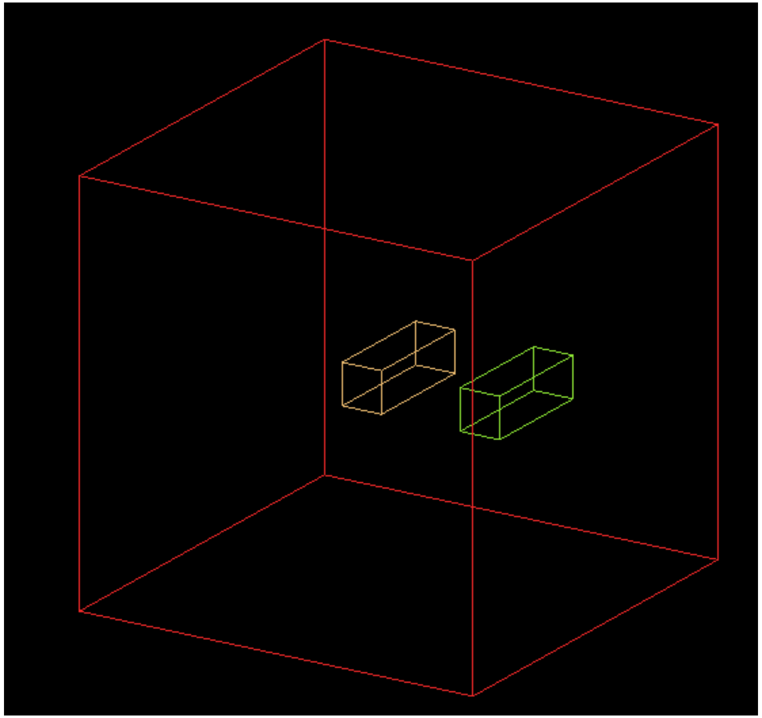
Trapezoid
Cut Tube
Segment of a Tube
Twisted tube
Extruded Solids
Tesellated Solids
Tetrahedrons
Twisted Generic
Trapezoid
Twisted Box

All the pieces of GDML



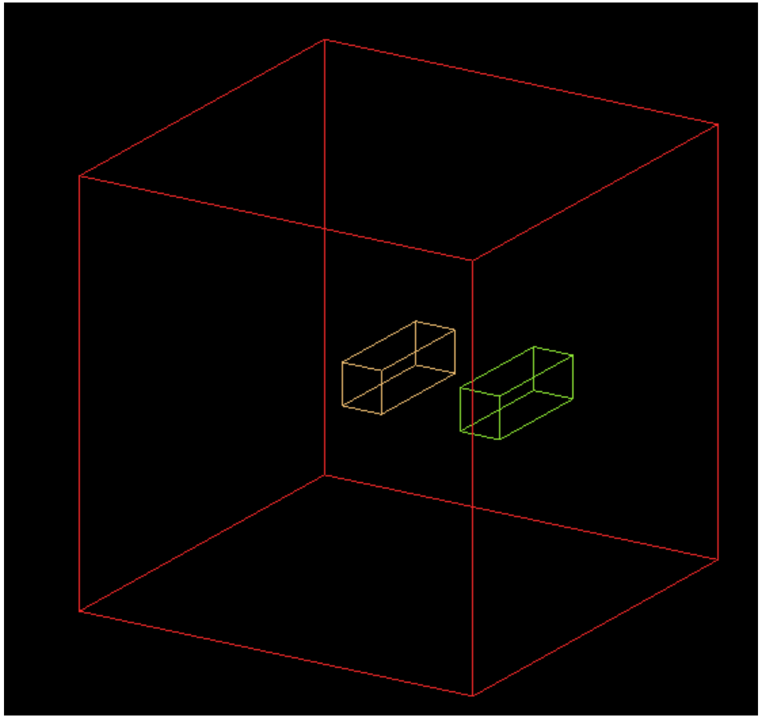
All the pieces of GDML

<solids> tag of GDML



```
-<solids>
   <box lunit="mm" name="LeadBlock" x="100" y="100" z="300"/>
   <box lunit="mm" name="AluminiiBlock" x="100" y="100" z="300"/>
   <box lunit="mm" name="WorldBlock" x="1000" y="1000" z="1000"/>
 </solids>
```

<materials> tag of GDML



```
-<materials>
  -<isotope N="204" Z="82" name="Pb204">
     <atom unit="g/mole" value="203.973"/>
   </isotope>
  -<isotope N="206" Z="82" name="Pb206">
     <atom unit="g/mole" value="205.974"/>
   </isotope>
  -<isotope N="207" Z="82" name="Pb207">
     <atom unit="g/mole" value="206.976"/>
   </isotope>
  -<isotope N="208" Z="82" name="Pb208">
     <atom unit="g/mole" value="207.977"/>
   </isotope>
  -<element name="Pb">
     <fraction n="0.014" ref="Pb204"/>
     <fraction n="0.241" ref="Pb206"/>
     <fraction n="0.221" ref="Pb207"/>
     <fraction n="0.524" ref="Pb208"/>
   </element>
  -<material name="G4_Pb" state="solid">
     <T unit="K" value="293.15"/>
     <MEE unit="eV" value="823"/>
     <D unit="g/cm3" value="11.35"/>
     <fraction n="1" ref="Pb"/>
   </material>
```

<structure> tag of GDML

Structure tag actually defines how different components of detector setup are arranged.
(Actually shows the mother-daughter relatioship)

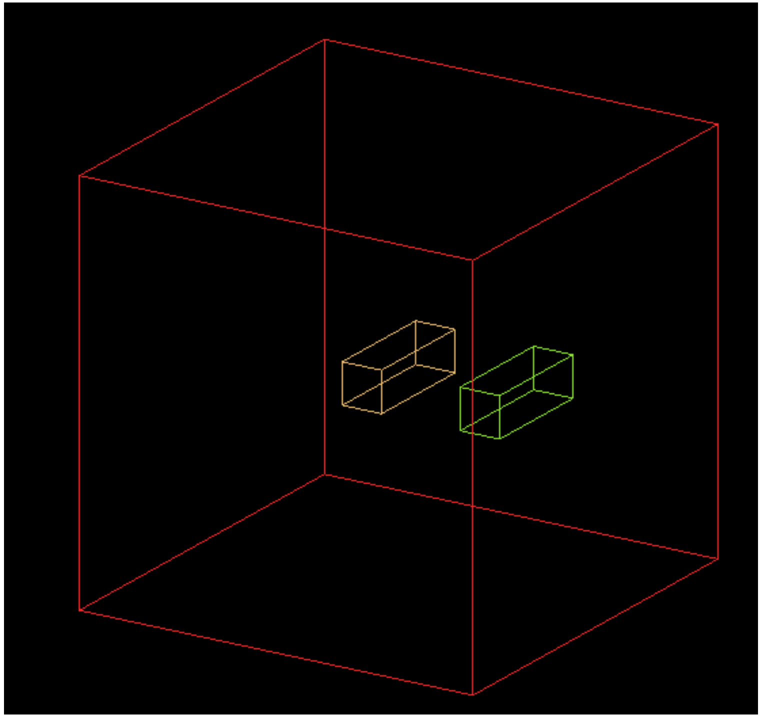
Both logical and physical volumes are defined in one structure

It show the hierarchy of detector components.

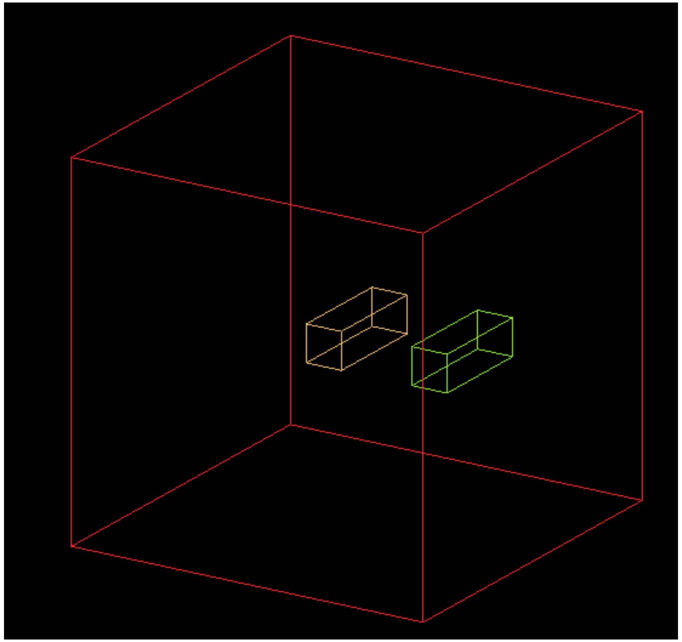
It consist of sequence of volumes tags, that define your logical volumes.

Each volume tag, keeps a pointer to the associated solid and the material.

Structure actually correponds to the your physical detector



Structures cont.



```
-<structure>
  -<volume name="LogicalLeadBlock">
      <materialref ref="G4_Pb"/>
      <solidref ref="LeadBlock"/>
    </volume>
  -<volume name="LogicalAlBlock">
      <materialref ref="G4_Al"/>
      <solidref ref="AluminiiBlock"/>
    </volume>
  -<volume name="World">
      <materialref ref="G4_Galactic"/>
      <solidref ref="WorldBlock"/>
    -<physvol name="Physical_PB_Block">
        <volumeref ref="LogicalLeadBlock"/>
      </physvol>
    -<physvol name="Physical_Al_Block">
        <volumeref ref="LogicalAlBlock"/>
        <position name="Physical_Al_Block_pos" unit="mm" x="300" y="0" z="0"/>
      </physvol>
    </volume>
  </structure>
```

Finally the <setup> tab

Setup contains the pointer to you world volume.

While creating a detector setup using gdml as an input file, we need to return a pointer to world volume from the **“Construct”** function of DetectorConstruction file.

It is possible to define multiple geometry setup, and chosing different volumes as world volume.

Moreover, we can actually split this geometry description in multiple file, which allows more granularity, and ease of maintainance.

```
-<setup name="Default" version="1.0">
   <world ref="World"/>
 </setup>
```

Exporting/ Importing GDML into Geant4

The GDML files can be imported directly into Geant4, in the detector construction class

Required class : **G4GDMLParser : (#include <G4GDMLParser.hh>)**

As usual this class contains various functions :

We will focus on Write and Read

An object of **“G4GDMLParser”** class is required.

G4GDMLParser myGDMLParser;

To export a full detector construction written in C++

myGDMLParser.Write(“geom.gdml”,pointerToPhyWorld);

To import a full detector setup, just do

myGDMLParser.Read(“geom.gdml);

Finally return the pointer to physical world volume to Geant.

return MyGDMLParser.GetWorldVolume()

NOTE : Geant4 needs to be compile with xercesC library

Building Complete GEANT application

-- Basic Structure of Geant4 Code

-- Where to write what

Things to be discussed

Geant4 Analogy of real experiment

Basic structure of the simulation code.

Writing a basic simulation code

Mandatory classes for your simulation code.
- -- Implementation of these mandatory classes

Getting the required information out of you simulation
- -- Optional classes
  - -- Implementation of these optional classes

Geant4 Analogy of the real experiment setup

Beam On : As in real experiment the Geant4 run starts with “Beam On”

A run is basically a collection of event.

As in experiment once the run start, user cannot change anything

--> Geometry Setup

--> Physics processes to study

Before starting the run, following things need to be initialized

--> Detector setup (geometry is optimized)

--> Physics List (cross-section tables are calculated, depending upon the materials used in the geometry creation)

Important user classes : Geant4 Program structure

Define your entry point : main() : There is no starting point provided by Geant4.
It is the place where you actually registers different component of you application.

Initialization classes : Classes whose **objects needs to initiated** before you simulation starts.

- Detector : **G4VUserDetectorConstruction**
- Physics : **G4VUserPhysicsList / Existing or Implemented**
- UserActions : **G4VUserActionInitialization**

Action classes :

- instantiated in the **G4VUserActionInitialization**
- The action classes are invoked during the event loop : ie. When you simulation is running.

G4VUserPrimaryGeneratorAction
G4UserRunAction
G4UserEventAction
G4UserStackingAction
G4UserTrackingAction
G4UserSteppingAction

The classes starting with **G4V** are abstract classes.
Their objects **can't** be created.
They are there to provide a skeleton required by Geant4
User needs to **inherit these classes**, and to implement few functions which are mandatory.

Creation of your DetectorConstruction : G4VUserDetectorConstruction

```
class G4VUserDetectorConstruction
{
 public:
  G4VUserDetectorConstruction();
  virtual ~G4VUserDetectorConstruction();

  virtual G4VPhysicalVolume* Construct() = 0;
};
```

(Pure virtual function)

The **Construct** method should return the pointer to the world physical volume, which represents your entire geometry setup.

```
class Sim01_DetectorConstruction : public
G4VUserDetectorConstruction
{
public:
        Sim01_DetectorConstruction(){}
        ~Sim01_DetectorConstruction(){}
        G4VPhysicalVolume* Construct(){
                //Write your stuff here
                //construct all your materials
                //construct all your volumes
                //declare you volume as sensitive
        }
};
```

Define your Physics

There is no default particles and physics process that comes automatically in your simulation code.

Not even particle transport.

Derive your own concrete class from **G4VUserPhysicsList** abstract base class.
– Define all necessary particles
– Define all necessary processes and assign them to proper particles
– Define all the required cut-off ranges

OR use the various physics lists that are already available in GEANT4.
FPFP_BERT **(add few more list)**

Primary Generator : G4VUserPrimaryGeneratorAction

The second mandatory user class : Controls the generation of primary particles.

--> This is again a abstract class

--> You cannot instantiate it : Will not do anything on its own

```
class G4VUserPrimaryGeneratorAction
{
 G4VUserPrimaryGeneratorAction();
  virtual ~G4VUserPrimaryGeneratorAction();
  virtual void GeneratePrimaries(G4Event* anEvent) = 0;
};
```

```
class Sim01_PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  G4ParticleGun *fParticleGun;
  Sim01_PrimaryGeneratorAction(){}
  ~Sim01_PrimaryGeneratorAction(){}

    void GeneratePrimaries(G4Event*){
      fParticleGun->GeneratePrimaryVertex();
    }

};
```

The generate primaries method is called at the beginning of every event.
Your primary generator will not generate any primary particle, until you call **GeneratePrimaryVertex()** function

Called only once

```
Sim01_PrimaryGeneratorAction::Sim01_PrimaryGeneratorAction() {

    int numOfParticle = 1;
    fParticleGun = new G4ParticleGun(numOfParticle);
    G4ParticleTable *particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition *particle = particleTable->FindParticle("mu-");
    fParticleGun->SetParticleDefinition(particle);
    fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,-1.));
    fParticleGun->SetParticleEnergy(3.*GeV);
    fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,30.*cm));
}
```

Called in the beginning of every event

```
void Sim01_PrimaryGeneratorAction::GeneratePrimaries(G4Event
*event) {
    fParticleGun->SetParticleMomentumDirection(G4RandomDirection());
    fParticleGun->GeneratePrimaryVertex(event);
}
```

Run Manager : G4RunManager

One of the manager class in Geant4 .

Helps in linking various objects and modules required during the initialization and run.

The program cannot run without the Run Manager.

User can inherit in their derived class to customize the behaviour

```
-------- EEEE ------- G4Exception-START ------- EEEE -------
*** G4Exception : Run0031
      issued by : G4RunManager::G4RunManager()
G4RunManager constructed twice.
*** Fatal Exception *** core dump ***
 **** Track information is not available at this moment
 **** Step information is not available at this moment

-------- EEEE -------- G4Exception-END --------- EEEE -------
```

G4RunManager or its Derived class must be singleton

--> Only one object should exist in the program's memory.

Singleton instance helps in accessing the same RunManager object in different locations in the code.

Action Initialization : G4VUserActionInitialization

Basically used to instantiate various classes required during event loop

```
class G4VUserActionInitialization
{
    G4VUserActionInitialization();
    virtual ~G4VUserActionInitialization();

    virtual void Build() const = 0;
}
```

```
class Sim01_ActionInitialization : public
G4VUserActionInitialization
{
  public:
    Sim01_ActionInitialization(){}
    virtual ~Sim01_ActionInitialization(){}

    virtual void BuildForMaster() const{}
    virtual void Build() const{
        // Link the objects of classes invoked
           during the event loop
           // EventAction, SteppingAction
    }
};
```

Structure of main() function

Define your entry point : main() :The place where you actually registers different components of your application.

Things TODO:
1) Instantiate your RunManager
2) Instantiate your DetectorConstruction
3) Instantiate your PhysicsList
4) Instantiate your ActionInitialization
5) Run your code
Optional
6) Instantiate your Visualization Manager

```
Int main(){
G4RunManager *runManager = new
G4RunManager;
DetectorConstruction *det = new
DetectorConstruction();
G4VModularPhysicsList *physicsList = new
FTFP_BERT;
ActionInitialization *actIni = new
ActionInitialization();
runManager->SetUserInitialization(det);
runManager->SetUserInitialization(physicsList);
runManager->SetUserInitialization(actIni);
G4UImanager *UImanager =
G4UImanager::GetUIpointer();
Uimanager->ApplyCommand(“/control/execute
Run.mac”);
```

Run.mac

```
/run/initialize
/run/beamOn 100
```

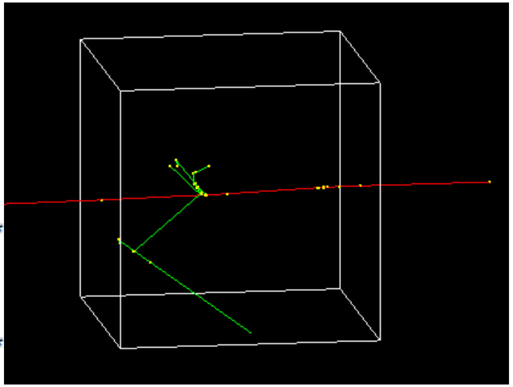
Our program is running : Where is the output ??

Geant4 runs the full simulation silently.

The required information needs to extracted.

Just to see what going on :
--> **use UI commands : */tracking/verbose 1***

This will basically start printing all the tracking information.
--> Particle information (location, direction etc.)
--> Step information
--> Energy loss
--> Associated volume
--> TrackId



```
****************************************************************************************************
****
* G4Track Information:   Particle = mu-,   Track ID = 1,   Parent ID = 0
****************************************************************************************************
****
Step#     X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
0         0         0       100    2e+03        0        0         0       World initStep
1         0         0        50    2e+03 1.49e-24       50        50 Physical_Lead_Block Transportation
2         0         0        42 1.99e+03     9.49     7.96        58 Physical_Lead_Block muIoni
3 -0.00409    -0.105      37.3 1.98e+03     5.58      4.7      62.7 Physical_Lead_Block CoulombScat
4  -0.0242    -0.159      35.8 1.98e+03     1.63     1.52      64.2 Physical_Lead_Block muIoni
5  -0.0527    -0.219      33.9 1.98e+03     2.27     1.95      66.1 Physical_Lead_Block muIoni
6   -0.517     -1.38      -1.2 1.93e+03     40.6     35.1       101 Physical_Lead_Block muIoni
7   -0.746     -1.51     -9.38 1.91e+03     8.89     8.18       109 Physical_Lead_Block muIoni
8    -1.37     -2.15       -50 1.86e+03     49.5     40.6       150       World Transportation
9     32.3       -19    -1e+03 1.86e+03 2.82e-23      951   1.1e+03  OutOfWorld Transportation
****************************************************************************************************
```

Geant4 Classes to get the information from the simulations

Information can be fetched at different levels, depending upon the requirements.

--> Run level information (G4UserRunAction)

--> Event level information (G4UserEventAction)

--> Step level information (G4UserSteppingAction)

--> Few more are also there.

These will be discussed in detail in the coming talks.

Thanks for your attention