Geant4 geometry and primary generator

Luciano Pandola

INFN

Part I: units and materials

Units in Geant4

- Geant4 has no default unit
- To feed the input data, unit must be "multiplied" to the number
  - for example :

```
G4double width = 12.5*m;
G4double density = 2.7*g/cm3;
```

  - Almost all commonly used units are available
  - The user can define new unit
- **`G4SystemOfUnits.hh`**
- To output the data you can **divide** a variable by a unit you want to get

```
G4cout << dE / MeV << " (MeV)" << G4endl;
```

System of Units in Geant4

- System of units are defined in CLHEP, based on:
  - millimetre (mm), nanosecond (ns), Mega eV (MeV), positron charge (eplus), degree Kelvin (kelvin), the amount of substance (mole), luminous intensity (candela), radian (radian), steradian (steradian)
  - All other units are computed from the basic ones
- Alternative way to output data: Geant4 can choose the most appropriate unit to use.
  - Just specify the category for the data (Length, Time, Energy, etc...):

```
G4cout << G4BestUnit(StepSize, “Length”);
```

  - StepSize will be printed in km, m, mm or ... fermi, depending on its actual value
- Custom units can be defined by the combination of the basic ones

```
G4UnitDefinition (“km/hour”, “km/h”, “Speed”,
km/3600*s);
```

Definition of materials

- Different kinds of materials can be defined:
  - isotopes <> G4Isotope
  - elements <> G4Element
  - molecules <> G4Material
  - compounds and mixtures <> G4Material
- Attributes associated:
  - temperature, pressure, state, density

- **G4Isotope** and **G4Element** describe properties of the atoms:

  - Atomic number, number of nucleons, atomic mass, , shell energies
- **G4Material** describes the macroscopic properties of the matter:

  - temperature, pressure, state, density, radiation length ...
- **G4Material** is the **only class used and visible to the toolkit**: it is used by tracking, geometry and physics

Making elements

- Isotopes can be assembled into elements

```
G4Isotope (const G4String& name,
                  G4int     z,     // atomic number
                  G4int     n,     // number of nucleons
                  G4double  a );   // mass of mole
```

- ... building elements as follows:

```
G4Element (const G4String& name,
           const G4String& symbol, // element symbol
                 G4int     nIso ); // n. of isotopes
G4Element::AddIsotope(G4Isotope* iso,  // isotope
                   G4double relAbund); // fraction of
                                      //atoms
                                     // per volume
```

Define a mono-element material

- Mono-element material:

```
G4double z;
G4double density = 1.390*g/cm3;
G4double a = 39.95*g/mole;
G4Material* lAr =
 new G4Material("liquidArgon", z=18, a,
  density);
```

Define a material as molecule or mixtures

```
a = 1.01*g/mole;
G4Element* elH  = new
  G4Element("Hydrogen",symbol="H",z=1.,a);
a = 16.00*g/mole;
G4Element* elO  =
   new G4Element("Oxygen",symbol="O",z=8.,a);
density = 1.000*g/cm3;
G4Material* H2O =
   new G4Material("Water",density,ncomponents=2);
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```

- Elemental composition can also be given by mass fraction (instead of number of atoms)

Compounds

- Compounds can also be made from *other materials* (given their mass fraction)

```
G4Element* elC  = …;   // define “carbon” element
G4Material* SiO2 = …;  // define “quartz” material
G4Material* H2O = …;   // define “water” material
density = 0.200*g/cm3;
G4Material* Aerog = new
 G4Material("Aerogel",density,ncomponents=3);
Aerog->AddMaterial(SiO2,fractionmass=62.5*perCent);
Aerog->AddMaterial(H2O ,fractionmass=37.4*perCent);
Aerog->AddElement (elC ,fractionmass= 0.1*perCent);
```

An example: a gas

- Necessary to specify temperature and pressure
  - affect dE/dx calculations, thermal scattering

```
G4double density = 27.*mg/cm3;
G4double temperature = 325.*kelvin;
G4double pressure = 50.*atmosphere;
G4Material* CO2 = new G4Material("CarbonicGas", density,
 ncomponents=2, kStateGas, temperature, pressure);
CO2->AddElement(C,natoms = 1);
CO2->AddElement(O,natoms = 2);
```

- Absolute vacuum does not exist: gas at very low ρ !
  - **Cannot** define materials with ρ=0

```
G4double rho = 1.e-25*g/cm3; G4double pr = 3.e-18*pascal;
G4Material* Vacuum = new G4Material("interGalactic",Z, A,
   rho, kStateGas, temperature, pr);
```

NIST Material Database in Geant4

- NIST database for elements materials imported in Geant4
  `http://physics.nist.gov/PhysRefData`
- Additional interfaces defined
- UI commands specific for handling materials
  `/material/nist/printElement`
  `/material/nist/listMaterials`
  - Print **available** elements and materials

Z		A	m	error	(%)	A_{eff}
14	Si	22	22.03453	(22)		28.0855(3)
		23	23.02552	(21)		
		24	24.011546	(21)		
		25	25.004107	(11)		
		26	25.992330	(3)		
		27	26.98670476	(17)		
		28	27.9769265327	(20)	92.2297 (7)	
		29	28.97649472	(3)	4.6832 (5)	
		30	29.97377022	(5)	3.0872 (5)	
		31	30.97536327	(7)		
		32	31.9741481	(23)		
		33	32.978001	(17)		
		34	33.978576	(15)		
		35	34.984580	(40)		
		36	35.98669	(11)		
		37	36.99300	(13)		
		38	37.99598	(29)		
		39	39.00230	(43)		
		40	40.00580	(54)		
		41	41.01270	(64)		
		42	42.01610	(75)		

- **Natural** isotope **compositions**
- More than 3000 isotope masses

NIST materials

- NIST elementary materials:
  - H -> Cf (Z = 1 -> 98)
- NIST compounds:
  - e.g. “G4_ADIPOSE_TISSUE_IRCP”
- HEP and Nuclear materials:
  - e.g. Liquid Ar, PbWO
- Possible to build mixtures of NIST and user-defined materials
- **Retrieve** materials from NIST manager:

```
G4NistManager* manager = G4NistManager::Instance();
G4Material* H2O = manager -> FindOrBuildMaterial(“G4_WATER”)
```

```
====================================
### Elementary Materials from the NIST Data
==================================
 Z Name  ChFormula        density(g/cm^3)  I(eV)
====================================
1   G4_H    H_2            8.3748e-05      19.2
2   G4_He                  0.000166322     41.8
3   G4_Li                  0.534           40
4   G4_Be                  1.848           63.7
5   G4_B                   2.37            76
6   G4_C                   2               81
7   G4_N    N_2            0.0011652       82
8   G4_O    O_2            0.00133151      95
9   G4_F                   0.00158029      115
10  G4_Ne                  0.000838505     137
11  G4_Na                  0.971           149
```

```
====================================
### Compound Materials from the NIST Data Base
====================================
 N Name     ChFormula      density(g/cm^3)  I(eV)
====================================
13  G4_Adipose_Tissue           0.92          63.2
            1     0.119477
            6     0.63724
            7     0.00797
            8     0.232333
            11    0.0005
            12    2e-05
            15    0.00016
            16    0.00073
            17    0.00119
            19    0.00032
            20    2e-05
            26    2e-05
            30    2e-05
4  G4_Air                       0.00120479    85.7
            6     0.000124
            7     0.755268
            8     0.231781
            18    0.012827
2  G4_CsI                       4.51          553.1
            53    0.47692
            55    0.52308
```

Part II: geometry and volumes

Describe your detector

- A detector geometry is made of a number of volumes
- The largest volume is called **World** volume
  - It **must** contain **all other volumes**
- Derive your own concrete class from **`G4VUserDetectorConstruction`** abstract base class
- Implementing the pure virtual method **`Construct()`**:
  - Define shapes/solids required to describe the geometry
  - Construct all necessary materials
  - Construct and place volumes of your detector geometry
  - (Define "sensitivity" properties associated to volumes)
  - (Associate magnetic field to detector regions)
  - (Define visualization attributes for the detector elements)

User Classes

Initialisation classes

Invoked at the initialization

- G4VUserDetectorConstruction
- G4VUserPhysicsList

Global: only one instance of them exists in memory, shared by all threads (**readonly**). Managed only by the master thread.

Action classes

Invoked during the execution loop

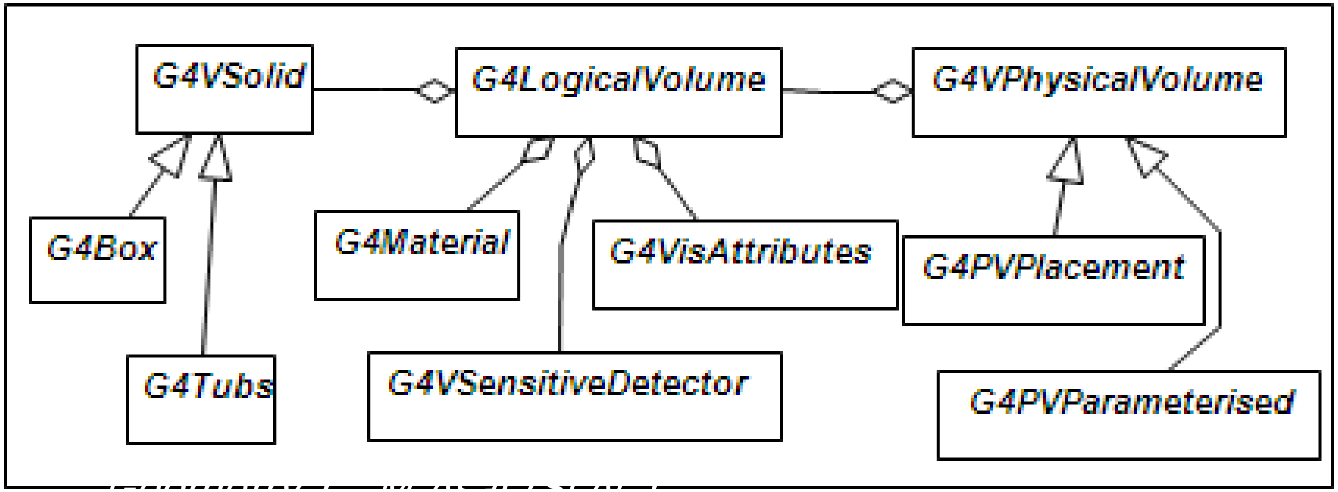
- G4VUserActionInitialization
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction (*)
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserStackingAction
  - G4UserSteppingAction

Local: an instance of each action class exists **for each thread**.

(*) Two RunAction's allowed: one for master and one for threads

Three conceptual layers

- **`G4VSolid`**
  - Shape, size
- **`G4LogicalVolume`**
  - Hierarchy of volumes, material, sensitivity, magnetic field
- **`G4VPhysicalVolume`**
  - Position, rotation. The same logical volume can be placed many times (repeated modules)

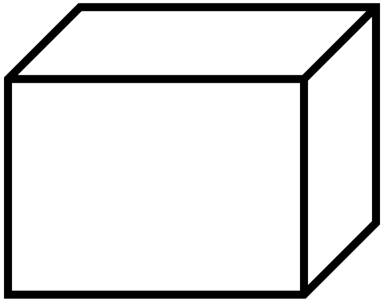


Define detector geometry

- Basic strategy

```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid",
        1.*m, 2.*m, 3.*m);
```

Solid : shape and size



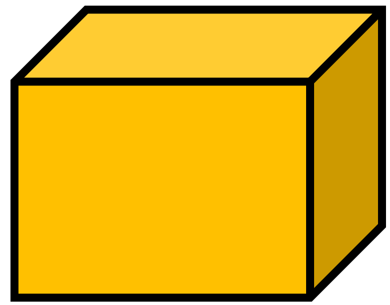
Step 1
Create the geom. object: box

Define detector geometry

Logical volume: + material, sensitivity, ...

- Basic strategy

```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid",
        1.*m, 2.*m, 3.*m);
G4LogicalVolume* pBoxLog =
 new G4LogicalVolume( pBoxSolid,
 pBoxMaterial, "aBoxLog", 0, 0, 0);
```



Step 1
Create the geom. object: box

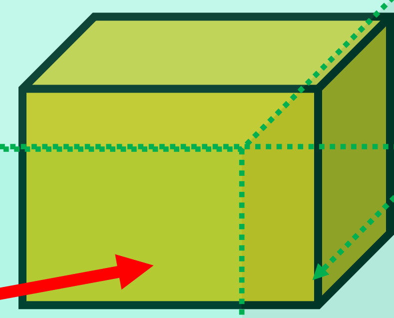
Step 2
Assign properties to object : material

Define detector geometry

Physical volume : + rotation and position

- Basic strategy

```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid",
        1.*m, 2.*m, 3.*m);

G4LogicalVolume* pBoxLog =
 new G4LogicalVolume( pBoxSolid,
 pBoxMaterial, "aBoxLog", 0, 0, 0);
G4VPhysicalVolume* aBoxPhys =
  new G4PVPlacement(pRotation,
  G4ThreeVector(posX, posY, posZ),
  pBoxLog,"aBoxPhys", pMotherLog, 0,
  copyNo);
```

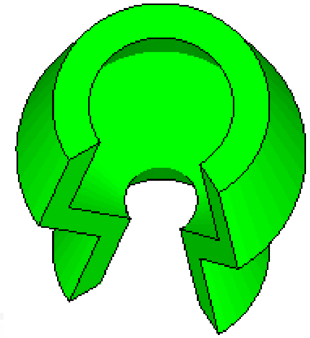


Step 1
Create the geom. object: box

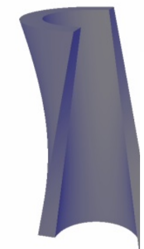
Step 2
Assign properties to object : material

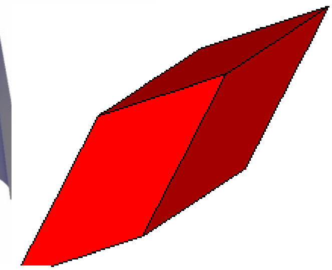
Step 3
Place it in the coordinate system of mother volume

Solids

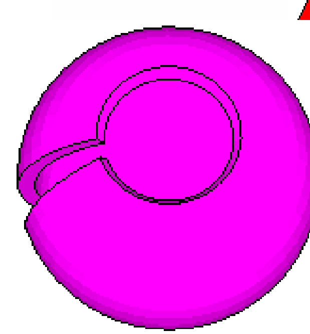


- CSG (Constructed Solid Geometry) solids
  - G4Box, G4Tubs, G4Cons, G4Trd, ...
  - Analogous to simple GEANT3 CSG solids

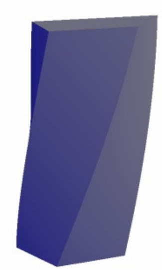


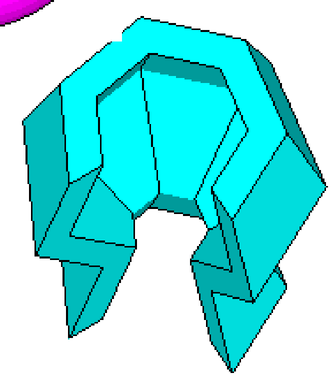


- Specific solids (CSG like)
  - G4Polycone, G4Polyhedra, G4Hype, ...
  - G4TwistedTubs, G4TwistedTrap, ...

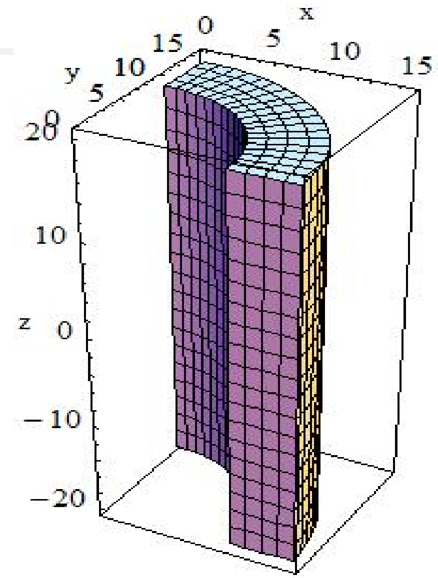


- BREP (Boundary REPresented) solids
  - G4BREPSolidPolycone, G4BSplineSurface, ...
  - Any order surface
- Boolean solids
  - G4UnionSolid, G4SubtractionSolid

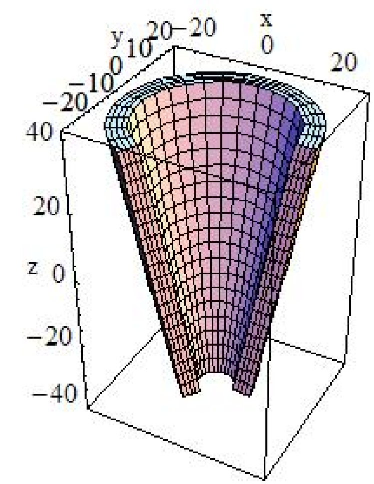




CSG: G4Tubs, G4Cons

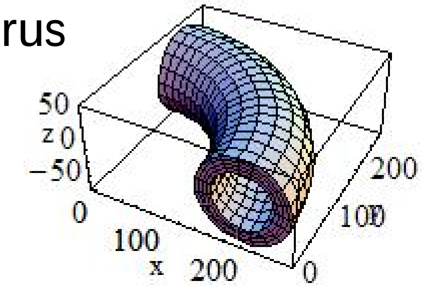


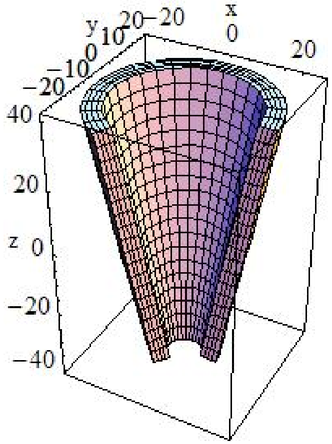

```
G4Tubs(const G4String& pname,   // name
             G4double  pRmin,   // inner radius
             G4double  pRmax,   // outer radius
             G4double  pDz,     // Z half length
             G4double  pSphi,   // starting Phi
             G4double  pDphi);  // segment angle

G4Cons(const G4String& pname,   // name
             G4double  pRmin1,  // inner radius -pDz
             G4double  pRmax1,  // outer radius -pDz
             G4double  pRmin2,  // inner radius +pDz
             G4double  pRmax2,  // outer radius +pDz
             G4double  pDz,     // Z half length
             G4double  pSphi,   // starting Phi
             G4double  pDphi);  // segment angle
```



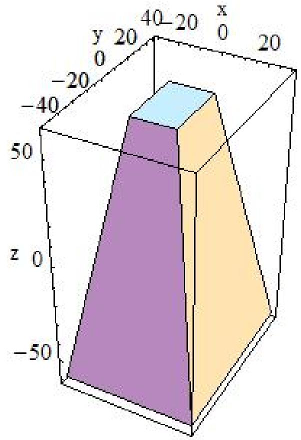
Other CSG solids

G4Torus

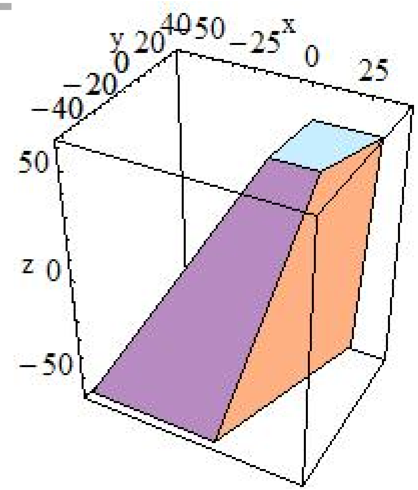




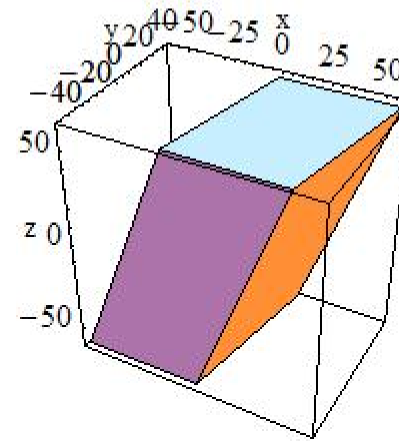
G4Cons



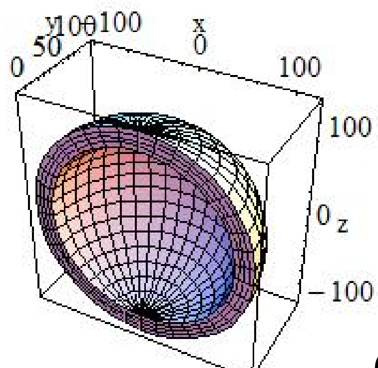
G4Trd



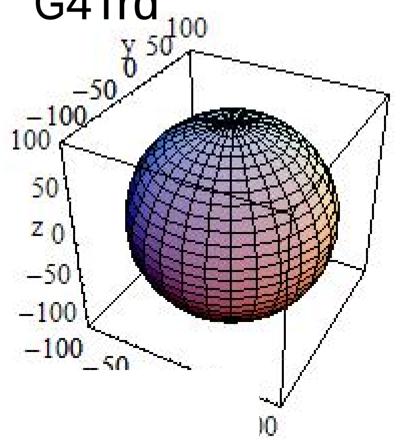
G4Trap



G4Para
(parallelepiped)



G4Sphere



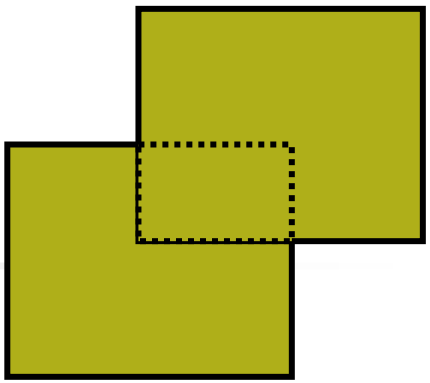
G4Orb
(full solid sphere)

Check Section 4.1.2 of Geant4 Application Developers Guide for all available shapes

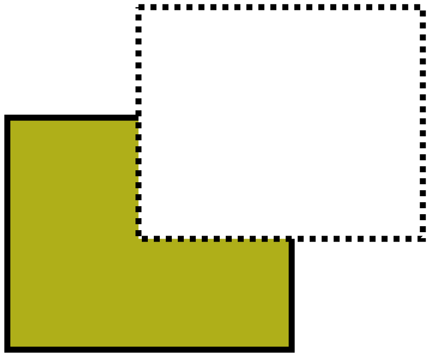
Boolean solids

- Solids can be **combined** using boolean operations:
  - **G4UnionSolid**, **G4SubtractionSolid**, **G4IntersectionSolid**
  - Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2nd solid
  - 2nd solid is positioned relative to the coordinate system of the 1st solid
  - Result of boolean operation becomes a solid → re-usable in a boolean operation
- Solids to be combined can be either CSG or other Boolean solids
- Note: tracking **cost** for the navigation in a complex Boolean solid is proportional to the number of constituent CSG solids

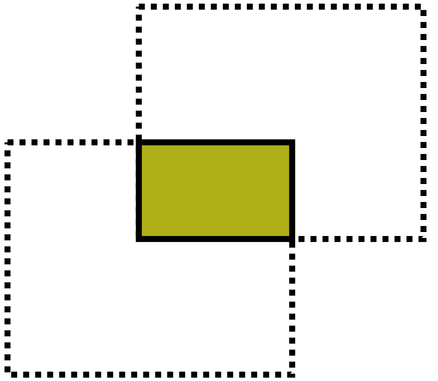
G4UnionSolid



G4SubtractionSolid



G4IntersectionSolid



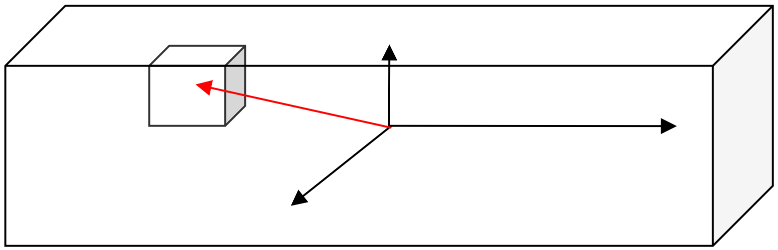
G4LogicalVolume

- Contains all information of volume **except position**:
  - Shape and dimension (**`G4VSolid`**)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits
- Physical volumes of same type **can share** a logical volume.
- The pointers to solid and material must be NOT null

```
G4LogicalVolume(G4VSolid* pSolid,
                G4Material* pMaterial,
                const G4String& name,
                G4FieldManager* pFieldMgr=0,
                G4VSensitiveDetector* pSDetector=0,
                G4UserLimits* pULimits=0,
                G4bool optimise=true);
```

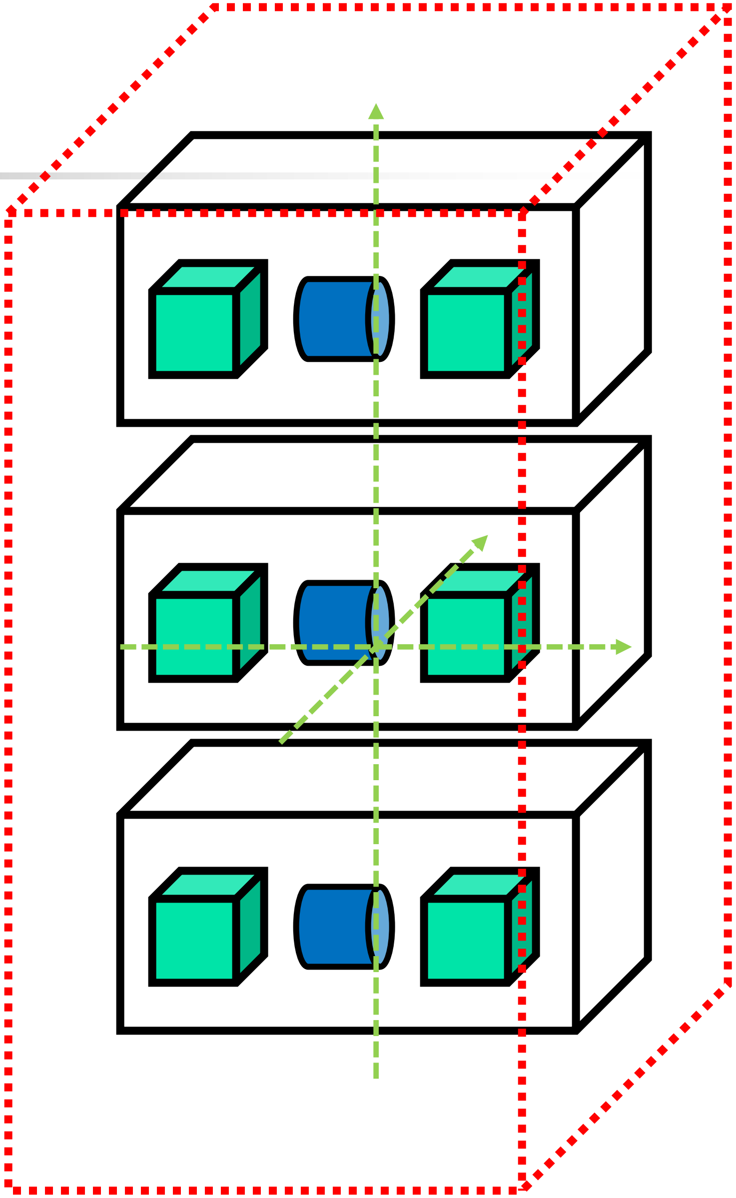
Geometry hierarchy

- A volume is **placed** in its **mother volume**
  - Position and rotation of the daughter volume is described with respect to the **local coordinate system** of the mother volume
  - The **origin** of the mother's local coordinate system is at the **center** of the mother volume
  - Daughter volumes cannot protrude from the mother volume
  - Daughter volumes cannot overlap
- The logical volume of mother knows the daughter volumes it contains
  - It is uniquely defined to be their mother volume



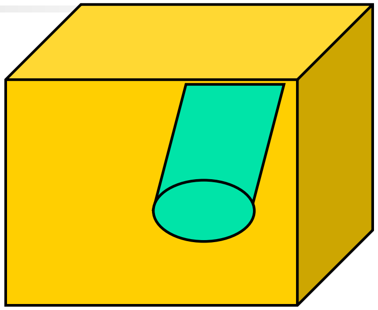
Geometry hierarchy

- One logical volume can be placed **more than once**. One or more volumes can be placed in a mother volume
- The mother-daughter relationship is an information of G4LogicalVolume
  - If the mother volume is placed more than once, all daughters by definition appear in each placed physical volume
- The **world volume** must be a unique physical volume which fully contains all other volumes (root volume of the hierarchy)
  - The world volume defines the global coordinate system. The origin of the global coordinate system is at the center of the world volume
  - Position of a track is given with respect to the global coordinate system



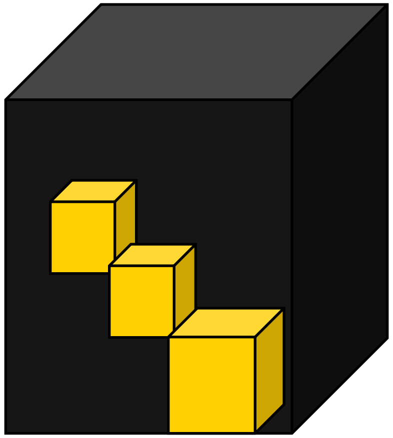
Physical volumes

- Placement: it is **one** positioned volume



placement

- Repeated: a volume placed **many times**
  - can represent any number of volumes
  - reduces use of memory
  - Replica
    - simple repetition, similar to G3 divisions
  - Parameterised



repeated

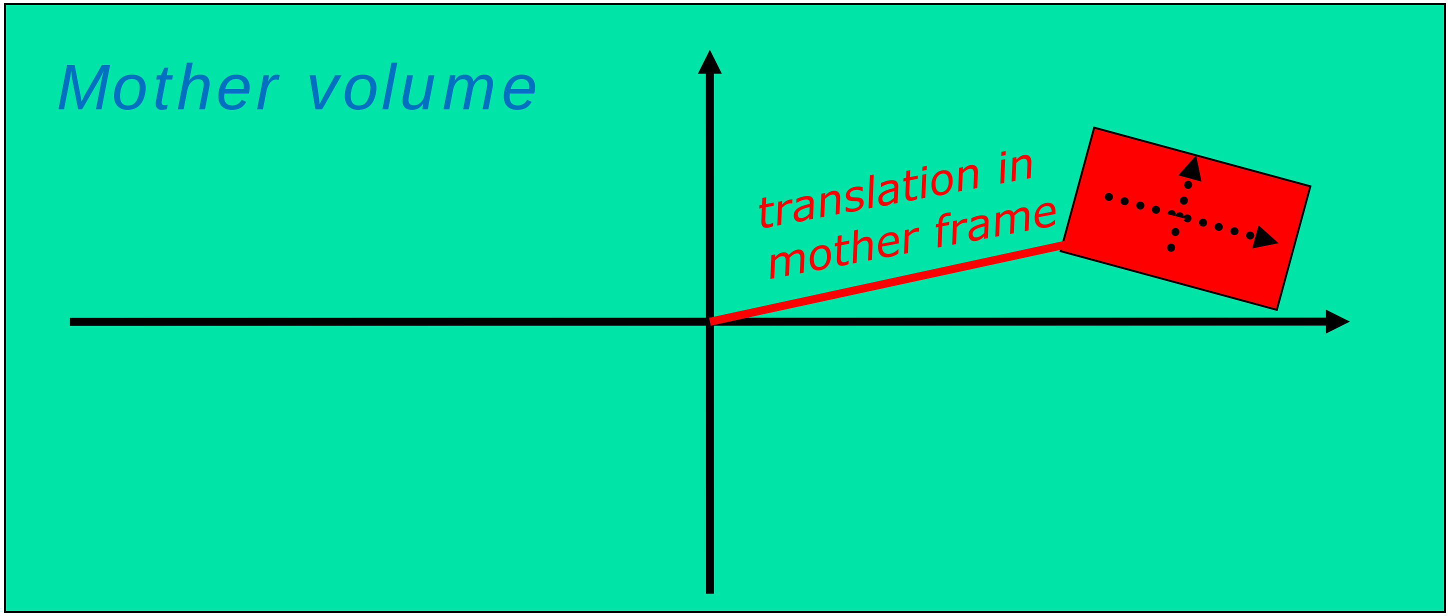
G4PVPlacement

- Single volume positioned **relatively** to the mother volume
  - In a frame rotated and translated relative to the coordinate system of the **mother volume**
- A few variants available:
  - Using **G4Transform3D** to represent the direct rotation and translation of the solid instead of the frame (*alternative constructor*)
  - specifying the mother volume as a pointer to its physical volume instead of its logical volume

```
G4PVPlacement(G4RotationMatrix* pRot, // rotation of mother frame
        const G4ThreeVector& tlate, // position in rotated frame
        G4LogicalVolume* pCurrentLogical,
        const G4String& pName,
        G4LogicalVolume* pMotherLogical,
        G4bool pMany,       // not used. Set it to false…
        G4int pCopyNo,           // unique arbitrary index
        G4bool pSurfChk=false);  // optional overlap check
```

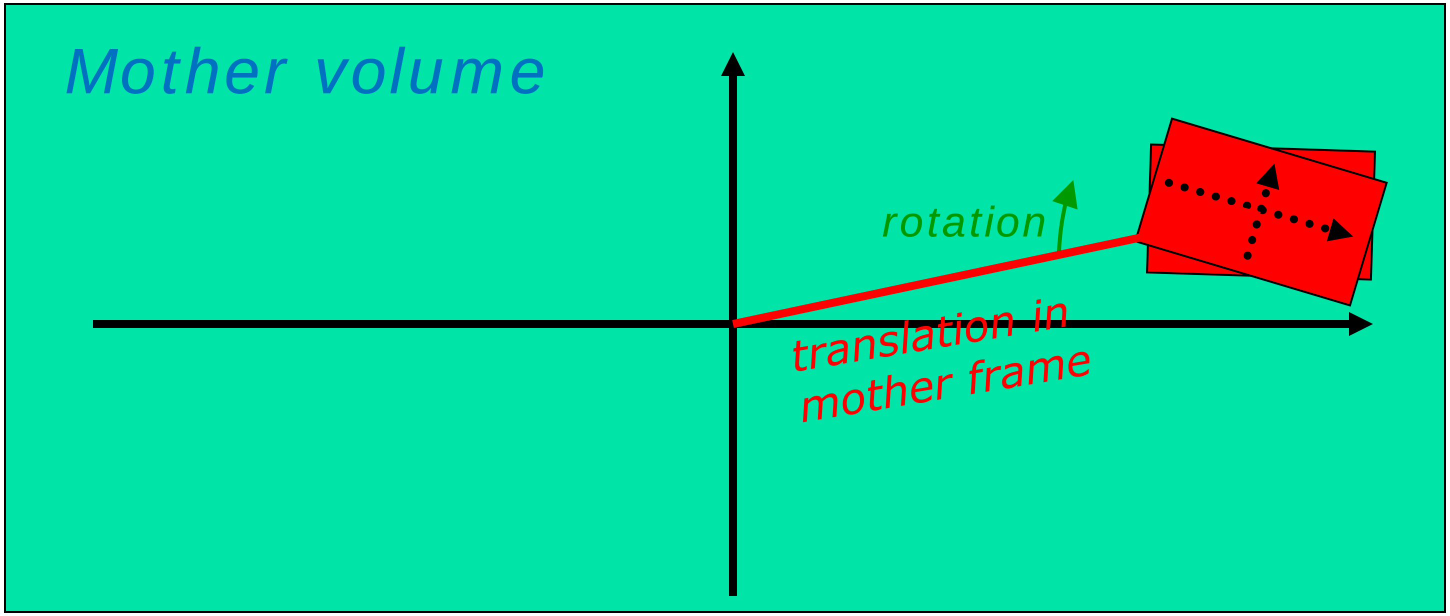
G4PVPlacement Rotation of mother frame …

```
G4PVPlacement(G4RotationMatrix* pRot, // rotation of mother frame
        const G4ThreeVector& tlate, // position in rotated frame
       G4LogicalVolume* pCurrentLogical,
        const G4String& pName,
        G4LogicalVolume* pMotherLogical,
        G4bool pMany,        // not used. Set it to false…
        G4int pCopyNo,          // unique arbitrary index
       G4bool pSurfChk=false);  // optional overlap check
```



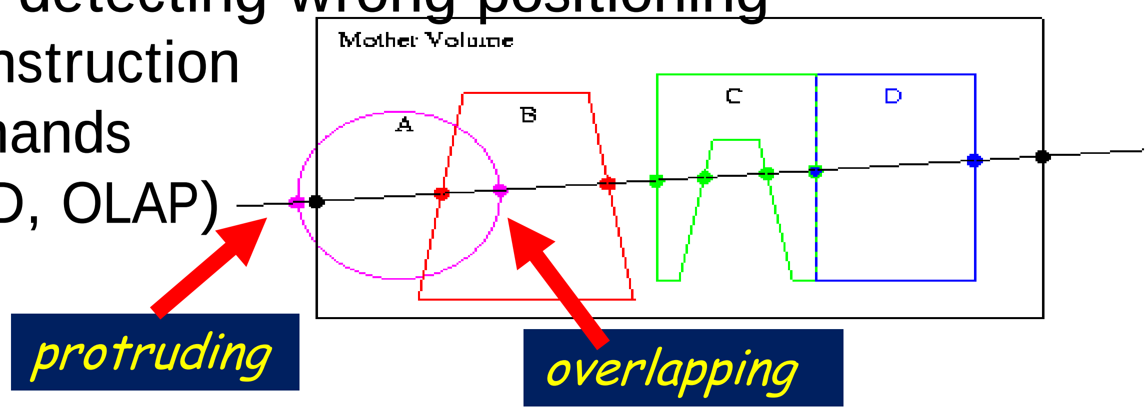
G4PVPlacement
Rotation <u>in</u> mother frame …

```
G4PVPlacement( G4Transform3D( G4RotationMatrix &pRot,
               const G4ThreeVector &tlate),
       G4LogicalVolume *pDaughterLogical, const G4String &pName,
       G4LogicalVolume *pMotherLogical,
       G4bool pMany, G4int pCopyNo, G4bool pSurfChk=false );
```



Tools for geometry check - 1

- A **protruding volume** is a contained daughter volume which actually protrudes from its mother volume
- When volumes in a common mother actually intersect themselves are defined as **overlapping**
- Geant4 **does not allow** for malformed geometries
  - The behavior of navigation is **unpredictable** for such cases
- The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description
- Utilities are provided for detecting wrong positioning
  - Optional checks at construction
  - Kernel run-time commands
  - Graphical tools (DAVID, OLAP)



Tools for geometry check - 2

- Constructors of **G4PVPlacement** and **G4PVParameterised** have an optional argument "**pSurfChk**"

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

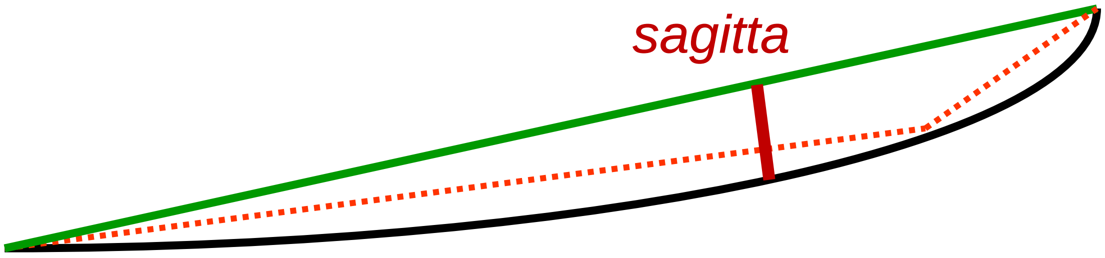
  - If this flag is true, **overlap check** is done at the construction
  - Some number of points are randomly sampled on the surface of creating volume
- This check requires lots of CPU time, but it is worth to **try** at least once

Region

- A **region** is a **sub-set** of the geometry
- It may have its specific
  - Production thresholds (cuts)
  - User limits
    - Artificial limits affecting to the tracking, e.g. max step length, max number of steps, min kinetic energy left, etc.
  - Field manager
  - …
- World logical volume is recognized as the default region. User is **not allowed** to define a region to the world logical volume

EM fields

- In order to propagate a particle inside a field (e.g. magnetic, electric or both), the **equation of motion** is numerically integrated
  - In general this is best done using a Runge-Kutta (RK) method for the integration of ordinary differential equations
  - Several RK methods are available
- Once a method is chosen for calculating the track's motion in a field, Geant4 breaks up this curved path into **linear chord segments**



- The chord segments are determined so that they closely approximate the curved path; they're chosen so that their **sagitta is small enough**

Create a magnetic field

- How to create Uniform field ?
  - Use the constructor of **G4UniformMagField**

```
G4ThreeVector  fieldV (0.1 * Tesla,1.0*Gauss,0.0 );
G4MagneticField *myField = new
G4UniformField(fieldV );
```

- Non-uniform field
  - Concrete class derived from **G4MagneticField**
- A field is packaged together with properties and accuracy parameters into a **field manager**:

```
G4FieldManager* localFieldMgr = new
G4FieldManager(myField);
```

  - One field manager is associated with the 'world'
  - Other volumes/regions in the geometry can override this

Part III: Primary generator

User Classes

Initialisation classes

Invoked at the initialization

- G4VUserDetectorConstruction
- G4VUserPhysicsList

Global: only one instance of them exists in memory, shared by all threads (**readonly**). Managed only by the master thread.

Action classes

Invoked during the execution loop

- G4VUserActionInitialization
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction (*)
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserStackingAction
  - G4UserSteppingAction

Local: an instance of each action class exists **for each thread**.

(*) Two RunAction's allowed: one for master and one for threads

G4VUserPrimaryGeneratorAction

- It is one of the **mandatory** user classes and it controls the generation of primary particles
  - This class does not directly generate primaries but invokes the **`GeneratePrimaryVertex()`** method of a **`generator`** to create the initial state
  - It registers the primary particle(s) to the G4Event object
- It has **`GeneratePrimaries(G4Event*)`** method which is purely virtual, so it **must** be implemented in the user class

G4ParticleGun

- (Simplest) concrete implementation of **`G4VPrimaryGenerator`**
  - It can be used for experiment specific primary generator implementation
- It shoots one primary particle of a given energy from a given point at a given time to a given direction
- Various "Set" methods are available (see ../source/event/include/G4ParticleGun.hh)

```
void SetParticleEnergy(G4double aKineticEnergy);
void SetParticleMomentum(G4double aMomentum);
void SetParticlePosition(G4ThreeVector aPosition);
void SetNumberOfParticles(G4int aHistoryNumber);
```

A "real-life" myPrimaryGenerator: constructor & destructor

```
myPrimaryGenerator::myPrimaryGenerator ()
: G4VUserPrimaryGeneratorAction(), fParticleGun(0)
{
  fParticleGun  = new G4ParticleGun();

  // set defaults
  fParticleGun->SetParticleDefinition(
       G4Gamma::Definition());
  fParticleGun->
       SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(6.*MeV);
}
```

Instantiate concrete generator

```
myPrimaryGenerator::~myPrimaryGenerator ()
{
  delete fParticleGun;
}
```

Clean it up in the destructor

A "real-life" myPrimaryGenerator: GeneratePrimaries(G4Event*)

```
myPrimaryGenerator::GeneratePrimaries(G4Event* evt)
{
  // Randomize event-per-event
  G4double cosT = -1.0 + G4UniformRand()*2.0;
  G4double phi = G4UniformRand()*twopi;

  G4double sinT = sqrt(1-cosT*cosT);
  G4ThreeVector direction(sinT*sin(phi),sinT*cos(phi),cosT);

  G4double ene = G4UniformRand()*6*MeV;

  fParticleGun->SetParticleDirection(direction);
  fParticleGun->SetParticleEnergy(ene);

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

Sample direction isotropically

Sample energy (flat distr.)

Shoot event

G4ParticleGun

- Commands can be also given interactively by **user interface**
  - But cannot do event-per-event randomization
- Allows to change primary parameters **between** one run and an other
  - Notice: parameters from the UI coud be overwritten in GeneratePrimaries()

```
/gun/energy 10 MeV
/gun/particle mu+
/gun/direction 0 0 -1
/run/beamOn 100
/gun/particle mu-
/gun/position 10 10 -100 cm
/run/beamOn 100
```

Change settings

Start first run

Change settings

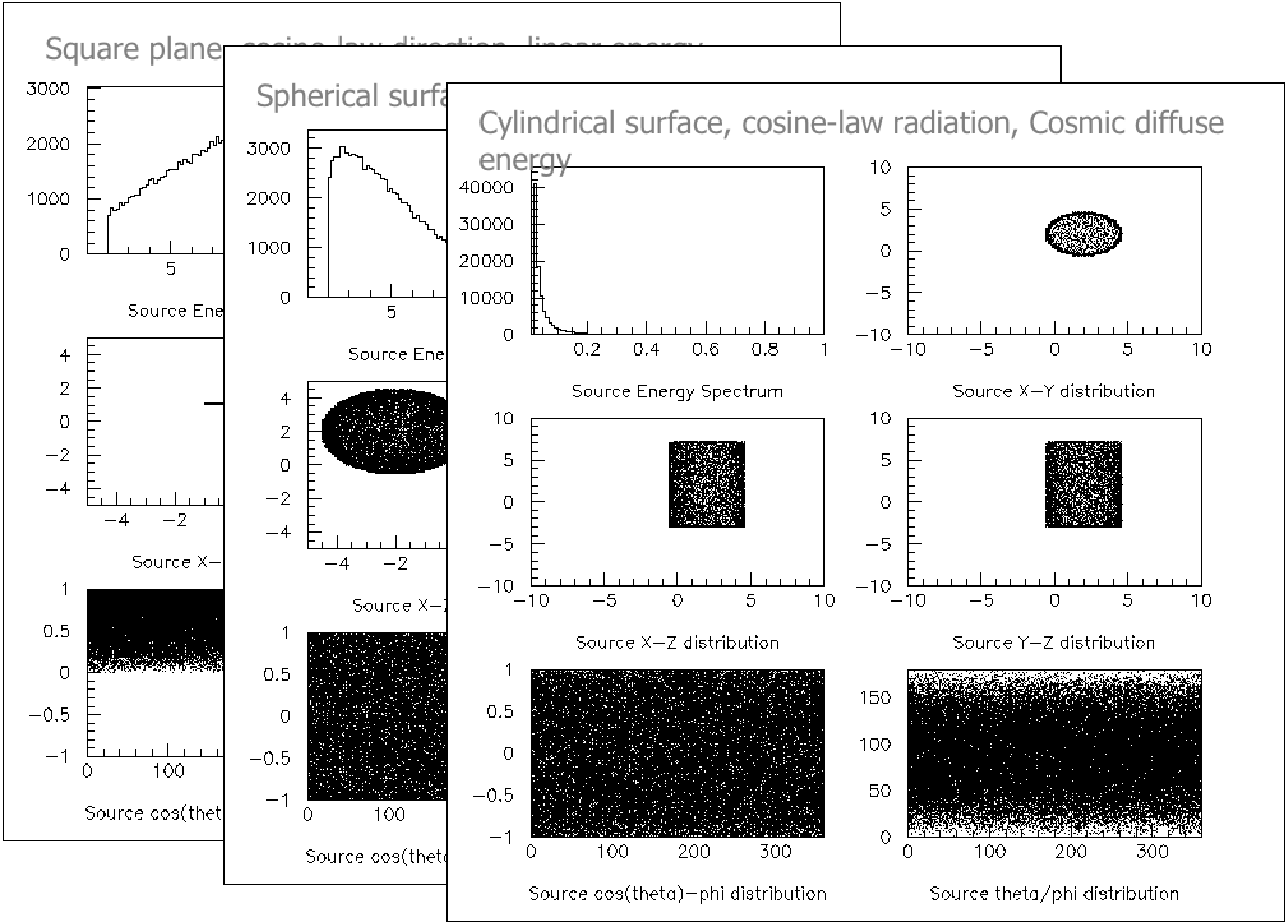
Start second run

G4GeneralParticleSource()

- **`source/event/include/G4GeneralParticleSource.hh`**
- Concrete implementation of **`G4VPrimaryGenerator`**
  `class G4GeneralParticleSource : public G4VPrimaryGenerator`
- Is designed to replace the G4ParticleGun class
- It is designed to allow specification of multiple particle sources each with independent definition of particle type, position, direction and energy distribution
  - Primary vertex can be randomly chosen on the surface of a certain volume, or within a volume
  - Momentum direction and kinetic energy of the primary particle can also be randomized
- Distribution defined by **UI commands**

G4GeneralParticleSource

- On line manual:
  - Section 2.7 of the Geant4 Application Developer Manual
- /gps main commands
  - **`/gps/pos/type`** (planar, point, etc.)
  - **`/gps/ang/type`** (iso, planar wave, etc.)
  - /**`gps/energy/type`** (monoenergetic, linear, User defined)
  - ....



Square plane
Source Ene
Source X–
Source cos(thet
Spherical surfa
Source Ene
Source X–
Source cos(theta
Cylindrical surface, cosine-law radiation, Cosmic diffuse energy
Source Energy Spectrum
Source X–Y distribution
Source X–Z distribution
Source Y–Z distribution
Source cos(theta)–phi distribution
Source theta/phi distribution

GPS documentation

2.7. Geant4 General Particle Source

2.7.1. Introduction

The `G4GeneralParticleSource` (GPS) is part of the Geant4 toolkit for Monte-Carlo, high-energy particle transport. Specifically, it allows the specifications of the spectral, spatial and angular distribution of the primary source particles. An overview of the GPS class structure is presented here. Section 2.7.2 covers the configuration of GPS for a user application, and Section 2.7.3 describes the macro command interface. Section 2.7.4 gives an example input file to guide the first time user.

Spectrum	Abbreviation	Functional Form	User Parameters
mono-energetic	Mono	$I \propto \delta(E-E_0)$	Energy E_0
linear	Lin	$I \propto I_0 + m \times E$	
exponential	Exp	$I \propto \exp(-E/E_0)$	
power-law	Pow	$I \propto E^{\alpha}$	
Gaussian	Gauss	$I = (2\pi\sigma)^{-1/2} \exp[-(E-E_0)^2 / \sigma^2]$	
bremsstrahlung	Brem	$I = \int 2E^2 [h^2c^2 (\exp(-E/kT$ $1)]^{-1}$	
black body	Bbody	$I \propto (kT)^{-1/2} E \exp(-E/kT)$	
cosmic diffuse gamma ray	Cdg	$I \propto [(E/E_b)^{\alpha 1} + (E/E_b)^{\alpha 2}]$	

2.7.3.3. Source position and structure

Command	Arguments	Description and restrictions
/gps/pos/type	dist	Sets the source positional distribution type: *Point* [default], *Plane, Beam, Surface, Volume*.
/gps/pos/shape	shape	Sets the source shape type, after `/gps/pos/type` has been used. For a Plane this can be *Circle, Annulus, Ellipse, Square, Rectangle*. For both Surface or Volume sources this can be *Sphere, Ellipsoid, Cylinder, Para* (parallelpiped).
/gps/pos/centre	X Y Z unit	Sets the centre co-ordinates (X,Y,Z) of the source [default (0,0,0) cm]. The units can only be micron, mm, cm, m or km.
/gps/pos/rot1	$R1_x$ $R1_y$ $R1_z$	Defines the first (x' direction) vector R1 [default (1,0,0)], which does not need to be a unit vector, and is used together with `/gps/pos/rot2` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/rot2	$R2_x$ $R2_y$ $R2_z$	Defines the second vector R2 in the xy plane [default (0,1,0)], which does not need to be a unit vector, and is used tohgether with `/gps/pos/rot1` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/halfx	len unit	Sets the half-length in x [default 0 cm] of the source. The units can only be micron, mm, cm, m or km.

ParticleGun vs. GPS

- G4ParticleGun
  - Simple and native
  - Shoots one track at a time
  - Easy to handle
- G4GeneralParticleSource
  - Powerful
  - Controlled by UI commands
    - G4GeneralParticleSourceMessenger.hh
    - Almost impossible to do with the naïve Set methods
  - Capability of shooting particles from a surface or a volume
  - Capability of randomizing kinetic energy, position, direction following a user-specified distribution (histogram)

> •If you need to shot primary particles from a surface of a complicated volume (outward or inward), GPS is the choice
>
> •If you need a complicated distribution, GPS is the choice

Hands-on session (tasks 1 and 2)

http://geant4.lngs.infn.it/TRISEP2014/introduction/index.html