

# GEANT4 advanced geometry

Luciano Pandola

INFN – Laboratori Nazionali del Sud

The 2<sup>nd</sup> Geant4 School in China,  
Shandong University, Qingdao,  
March 25<sup>th</sup>- 29<sup>th</sup>, 2019



# EM Fields

---

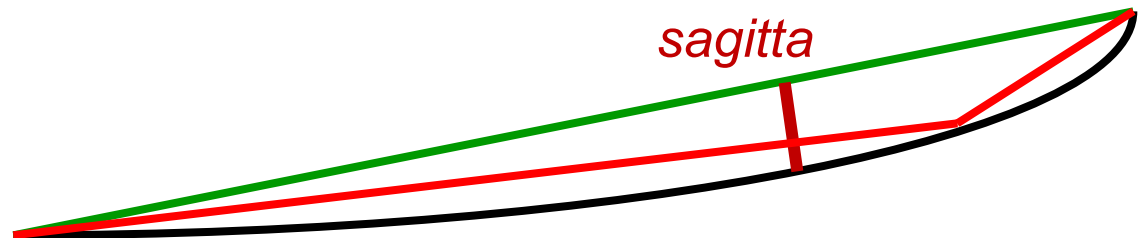
# Tracking in EM fields

- Divide the trajectory of the particle in "**steps**"
  - **Straight free-flight tracks** between consecutive physics interactions
- In presence of EM fields, the **free-flight** part between interactions is **not straight**
  - Change of *direction* (B-field) or *energy* (E-field)
  - Effect of fields must be incorporated into the **tracking** algorithm → **CPU-demanding**
- Notice: most codes handle only **weak fields**
  - An  $e^-$  **at rest will not accelerate**, no synchrotron radiation, **no avalanche**



# Tracking in fields

- In order to propagate a particle inside a field the **equation of motion** of the particle in the field is **integrated numerically**
- In general this is best done using a **Runge-Kutta (RK) method** for the integration of ordinary differential equations
  - Other methods are also available
- Once the curved path is calculated, Geant4 breaks it up into **linear chord segments**



- The **chord segments** are determined to closely approximate the curved path
  - In some cases, **one step** could be split in **several helix-turns**

# Example: how to create a magnetic field: uniform

- Uniform field in the entire world volume: easy recipe

```
G4ThreeVector field(0,1.*tesla,0);  
G4GlobalMagFieldMessenger* fMagFieldMessenger =  
    new G4GlobalMagFieldMessenger(field)
```

- In general, one can customize the precision of the stepper and method used for the numerical integration of the equations

```
G4UniformMagField* magField = new G4UniformMagField(field);  
G4FieldManager* fieldMgr =  
G4TransportationManager::GetTransportationManager()  
->GetFieldManager();  
fieldMgr->SetDetectorField(magField);  
fieldMgr->CreateChordFinder(magField);
```

} register

# Example: how to create a magnetic field: non-uniform

- Non-uniform field in the world volume
  - Create a class, derived from `G4MagneticField`, implementing  $\vec{B} = f(\vec{x}, t)$

```
void MyField::GetFieldValue (const double  
    Point[4], double *field) const
```

```
MyField* myField = new MyField();  
G4FieldManager* fieldMgr =  
G4TransportationManager::GetTransportationManager()  
->GetFieldManager();  
fieldMgr->SetDetectorField(myField);  
fieldMgr->CreateChordFinder(myField);
```

# Example: how to create a local magnetic field

- It is possible to define a **field** inside a **logical volume** (and its daughters)
  - This can be done creating a local **G4FieldManager** and attaching it to a **logical** volume

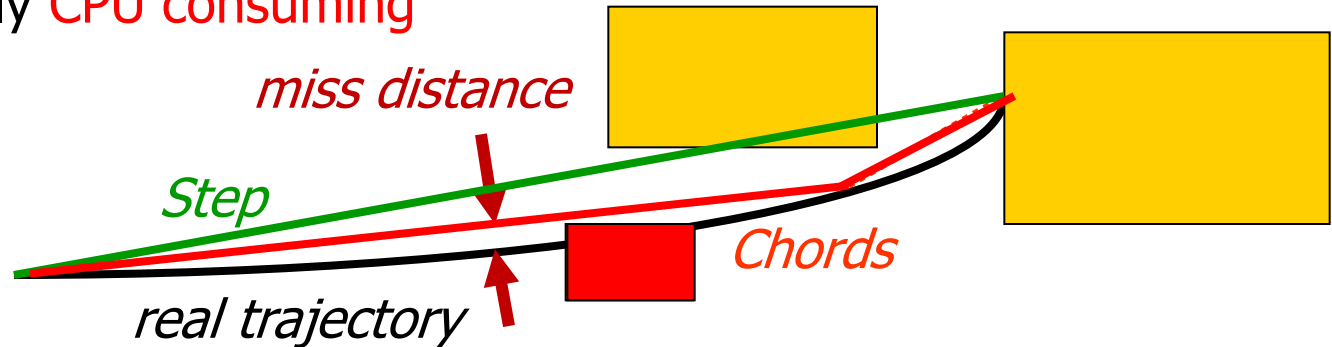
```
MyField* myField = new MyField();
G4FieldManager* localFieldMgr =
    new G4FieldManager(myField);
G4bool allLocal = true;
logicVolWithField
->SetFieldManager(localFieldMgr, allLocal);
```

If **true**, field assigned to **all daughters**

If **false**, field assigned only to daughters **w/o their own field manager**

# Customization

- A few parameters to customize the precision of the tracking in EM fields. Most critical: "miss distance"
  - Upper bound for the value of the sagitta (default: 3 mm)
  - May be highly CPU consuming



- Integration calculated by 4<sup>th</sup>-order Runge-Kutta (`G4ClassicalRK4`), robust and general purpose
  - If the field is not smooth (e.g. field map), lower-order (and faster) integrators can be appropriate
  - 3<sup>rd</sup> order `G4SimpleHeum`, 2<sup>nd</sup> order `G4ImplicitEuler`, 1<sup>st</sup> order `G4ExplicitEuler`



# (Bird's eye view) Replicas and parametrized volumes



---



# Physical volumes

---

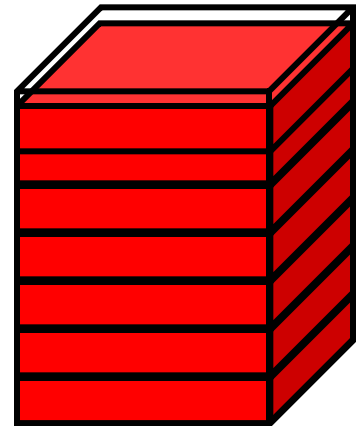
- Placement volume (**G4PVPPlacement**): one positioned volume
  - **One** physical volume represents **one "real"** volume
- **Repeated volume**: a volume placed **many times**
  - **One physical volume** represents **any number of "real"** volumes
    - Reduced use of memory
    - Very **convenient** for large voxelized geometries
  - Parametrized (repetitions w.r.t. copy number)
  - Replicas and Divisions
- Notice: a repeated volume is **not equivalent** to a loop of placements
  - All placements of the loop exists **individually** in the **memory**

# Replicated volumes (G4PVReplica)

- The mother volume is **completely filled** with replicas, all having **same size and shape**
  - If you need gaps, use **G4PVDivision** instead (less CPU-efficient)
- **Replication** may occur along:
  - **Cartesian** axes (kXAxis, kYAxis, kZAxis)
    - Coordinate system at the center of each replica
  - **Radial** axis (cylindrical polar) (kRho) - onion rings
    - Coordinate system same as the mother
  - **Phi** axis (cylindrical polar) (kPhi) – cheese wedges
    - Coordinate system rotated so that the X axis bisects the angle made by each wedge



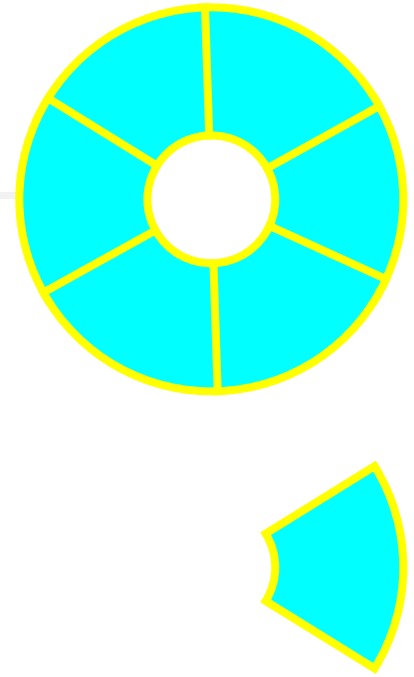
*a daughter  
logical volume to  
be replicated*



*mother volume*

# G4PVReplica

```
G4PVReplica(const G4String &pName,  
            G4LogicalVolume* pLogical,  
            G4LogicalVolume* pMother,  
            const EAxis pAxis,  
            const G4int nReplicas,  
            const G4double width,  
            const G4double offset=0.);
```



- Features and restrictions:
  - **CSG** solids only
  - **G4PVReplica** must be the **only** daughter
  - Replicas may be placed inside other replicas
  - Normal placement volumes may be placed inside replicas
  - No volume can be placed inside a radial replication
  - Parameterised volumes cannot be placed inside a replica

# Replica: axes, width and offset

Center of  $n^{\text{th}}$  daughter is given as

- Cartesian axes - kXaxis, kYaxis, kZaxis

$$-\text{width} * (\text{nReplicas} - 1) * 0.5 + \text{n} * \text{width}$$

Offset shall **not** be **used**

- Radial axis – kRho

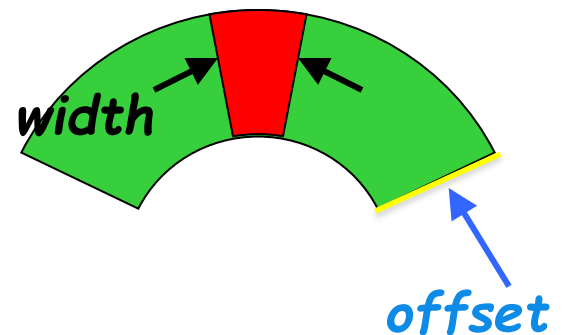
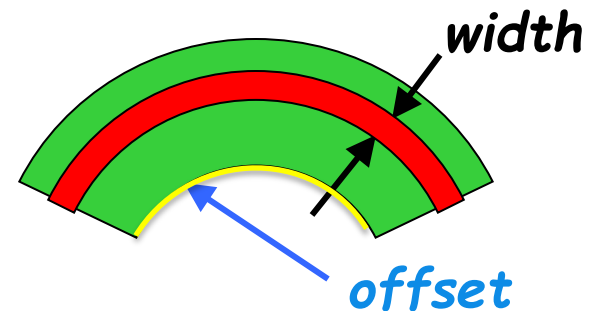
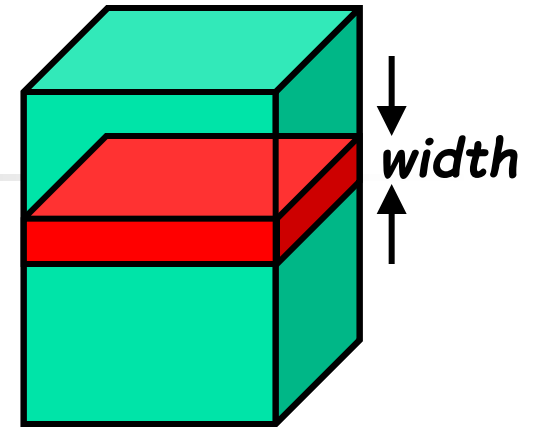
$$\text{width} * (\text{n} + 0.5) + \text{offset}$$

Offset must be the **inner radius of the mother**

- Phi axis – kPhi

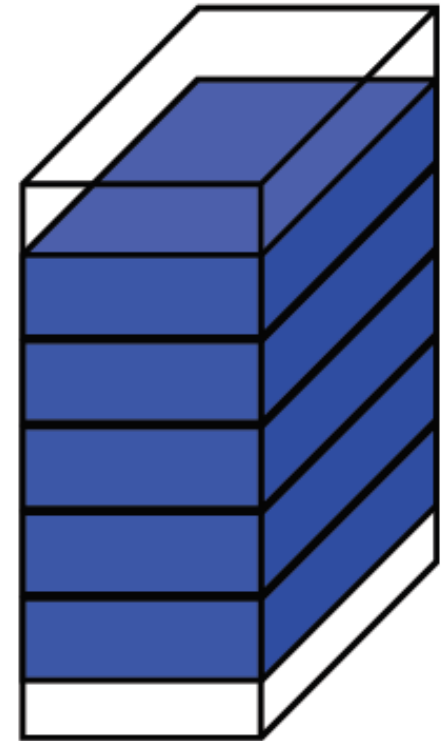
$$\text{width} * (\text{n} + 0.5) + \text{offset}$$

Offset must be the **starting angle of the mother**



# G4PVDivision

- The `G4PVDivision` is similar to the `G4PVReplica` **but**
  - Allows for **gaps** between mother and daughter volumes
  - **Less CPU-effective** than replica
- **Shape** of all daughter volumes **must be the same** as of the **mother** volume
- A number of shapes / axes patterns are supported, e.g.
  - `G4Box` : `kXAxis`, `kYAxis`, `kZAxis`
  - `G4Tubs` : `kRho`, `kPhi`, `kZAxis`
  - `G4Cons` : `kRho`, `kPhi`, `kZAxis`
  - ...



mother volume

# Parametrized volumes (G4VPVParameterisation)

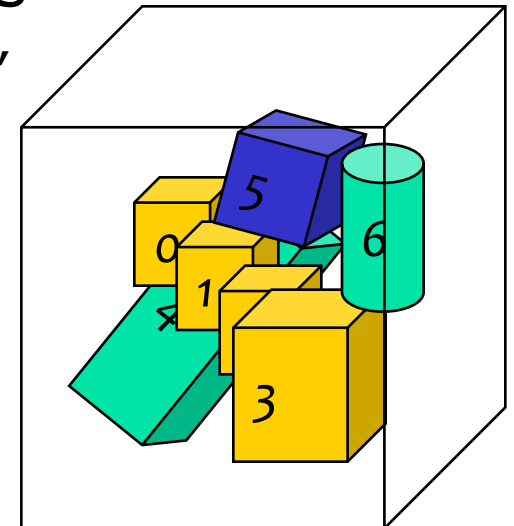
- Repeated volumes can differ by **size, shape, material** and **transformation matrix**, that can all be **parameterised** by the user as a function of the **copy number**

User is asked to derive her/his own **parameterisation class** from the G4VPVParameterisation class implementing the methods:

```
void ComputeTransformation(const G4int copyNo,  
                           G4VPhysicalVolume  
                           *physVol) const;  
void ComputeDimensions(G4Tubs& trackerLayer,  
                        const G4int copyNo, const  
                        G4VPhysicalVolume *physVol) const;
```

Optional methods:

```
ComputeMaterial (...)  
ComputeSolid (...)
```





# Parametrized volumes

---

- All daughters must be **fully contained** in the mother
  - Daughters **should not overlap** to each other
- Limitations:
  - Applies to simple **CSG solids only**
  - Grand-daughter volumes allowed only for special cases
- **Typical use-cases**
  - **Complex detectors** with large repetition of volumes, regular or irregular
  - **Medical applications**: the **material** in tissue is modeled as parametrized voxels with **variable density**
  - Limited **memory footprint**





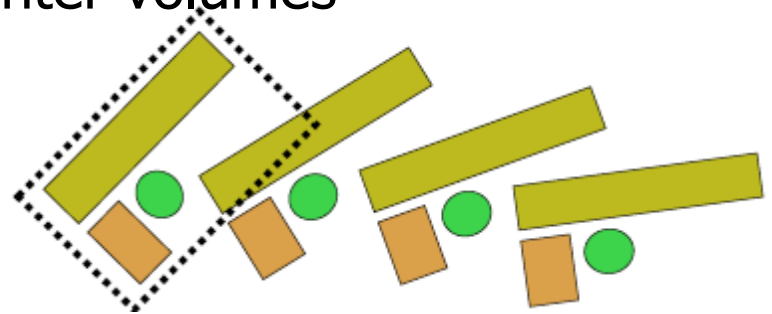
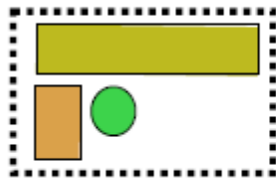
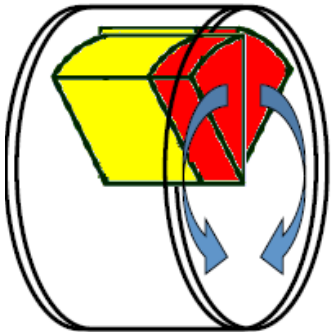
# G4PVParameterized

```
G4PVParameterised(const G4String& pName,  
                  G4LogicalVolume* pLogical,  
                  G4LogicalVolume* pMother,  
                  const EAxis pAxis,  
                  const G4int nReplicas,  
                  G4VPVParameterisation *pParam  
                  G4bool pSurfChk=false);
```

- Replicates the volume `nReplicas` times using the parameterization `pParam`, within the mother volume `pMother`
- `pAxis` specifies the **tracking optimisation** algorithm to apply:
  - `kXAxis`, `kYAxis`, `kzAxis` → 1D voxelisation algorithm
  - `kUndefined` → 3-D voxelisation algorithm
- Each replicated volume is a **touchable** detector element

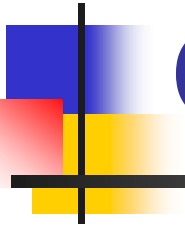
# Assembly & reflections

- Possible to represent a **regular pattern** of positioned volumes, composing a more or less complex structure
  - structures which are **hard** to describe with **simple replicas** or parameterised volumes
- **Assembly** volume (`G4AssemblyVolume`)
  - acts as an *envelope* for its daughter volumes



- `G4ReflectedSolid` (derived from `G4VSolid`)
  - Utility class representing a **solid shifted** from its **original reference frame** to a new **mirror** symmetric one

# GDMML exchange format and CAD





# GDML

Requires the **Xerces-C++** libraries

- **Geometry Description Markup Language**
  - application-independent **geometry description format** based on **XML**
  - Not limited to Geant4
- General **geometry description**
  - Implements "**geometry trees**" allowing for the description of **hierarchical** geometries
  - Contains **material** definitions and volume **placements**
- Profitably used for **geometry exchange** between:
  - Fluka and Geant4 → physics validation
  - Geant4 and ROOT → geometry visualization
  - CAD and Geant4 → geometry import (with care)
- Allows **running the same application** with different geometries

# GDMML document example

positions,  
rotations

materials

solids

geometry  
tree

'world'  
volume

```
<?xml version="1.0" encoding="UTF-8"?>
<gdmml xsi:noNamespaceSchemaLocation="GDMLSchema/gdmml.xsd">
  <define>
    ...
    <position name="TrackerinWorldpos" unit="mm" x="0" y="0" z="100"/>
  </define>
  <materials>
    ...
    <material formula=" " name="Air" >
      <D value="1.290" unit="mg/cm3"/>
      <fraction n="0.7" ref="Nitrogen" />
      <fraction n="0.3" ref="Oxygen" />
    </material>
  </materials>
  <solids>
    ...
    <box lunit="mm" name="Tracker" x="50" y="50" z="50"/>
  </solids>
  <structure>
    ...
    <volume name="World" >
      <materialref ref="Air" />
      <solidref ref="world" />
      <physvol>
        <volumeref ref="Tracker" />
        <positionref ref="TrackerinWorldpos"/>
        <rotationref ref="TrackerinWorldrot"/>
      </physvol>
    </volume>
  </structure>
  <setup name="Default" version="1.0" >
    <world ref="World" />
  </setup>
</gdmml>
```

◆ ASCII file: easy to create, read, debug, modify,...

◆ Similar to HTML, explicit tags for elements, ...



# GDMML solids

---

- Box
- Cone Segment
- Ellipsoid
- Elliptical Tube
- Elliptical Cone
- Orb
- Paraboloid
- Parallelepiped
- Polycone
- Polyhedron
- Sphere
- Torus Segment
- Trapezoid (x&y vary along z)
- General Trapezoid
- Tube with Hyperbolic Profile
- Cut Tube
- Tube Segment
- Twisted Box
- Twisted Trapezoid
- Twisted General Trapezoid
- Twisted Tube Segment
- Extruded Solid
- Tessellated Solid
- Tetrahedron

And booleans

# GDML components

## GDML Schema

- self-consistent definition of **GDML syntax**
- defines **document structure** and the list of legal elements

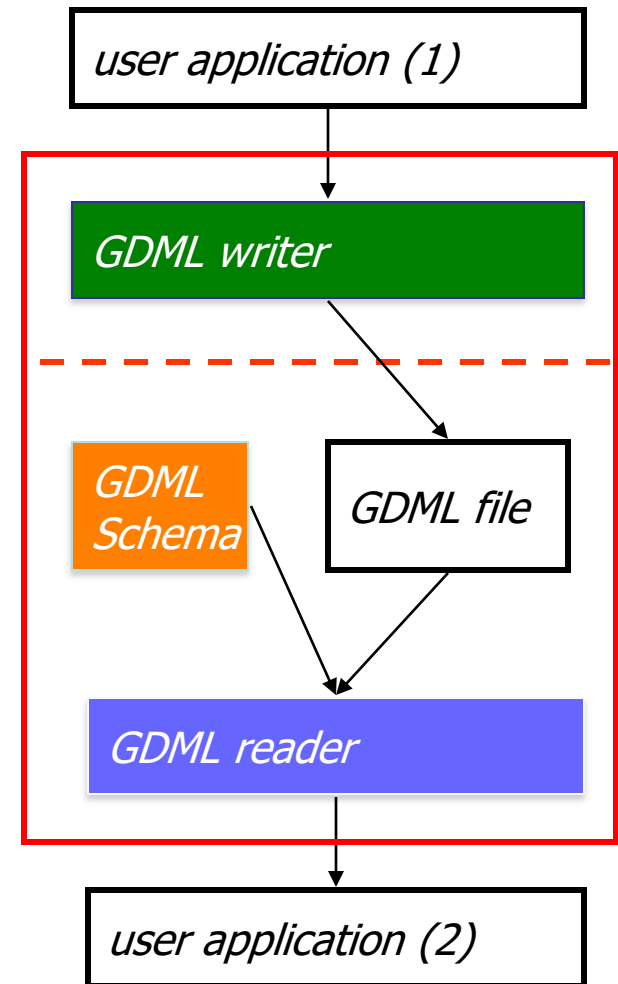
## GDML Reader

- Creates 'in-memory' representation of the geometry description

## GDML Writer

- Allows **exporting geometry** on a file
- Files can be also edited by hand

Reader and Writer are integrated in packages like Geant4 and ROOT providing **GDML compliant interfaces**





# Reading GDML files in Geant4

---

- Importing a geometry from a GDML file, only requires

```
DetectorConstructor::Construct() {  
  
    // gdml parser  
    #include "G4GDMLParser.hh"  
    G4GDMLParser fParser;  
  
    // importing geometry  
  
    fParser.Read("detectorgeometry.gdml"); // reads and stores  
                                           // in memory  
  
    G4VPhysicalVolume* fWorldPhysVol = fParser.GetWorldVolume();  
    // get world  
  
}
```

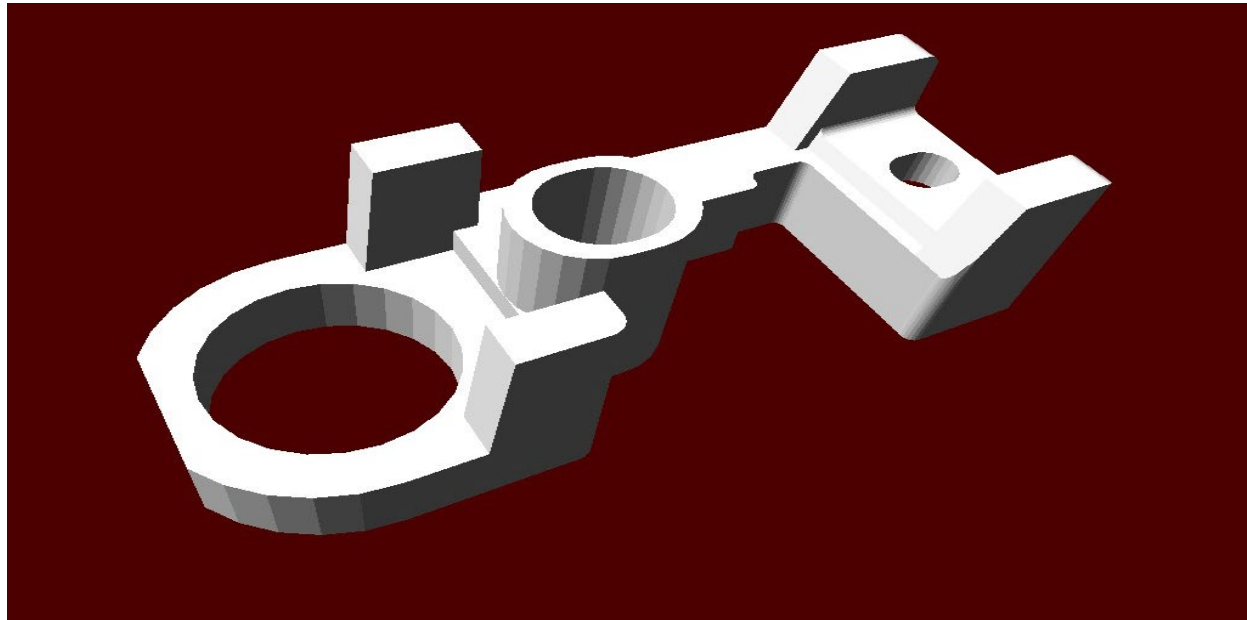


# Writing a GDML files from Geant4

- Conversely, one can **export** a Geant4 geometry (e.g. C++ coded) **in a GDML file**

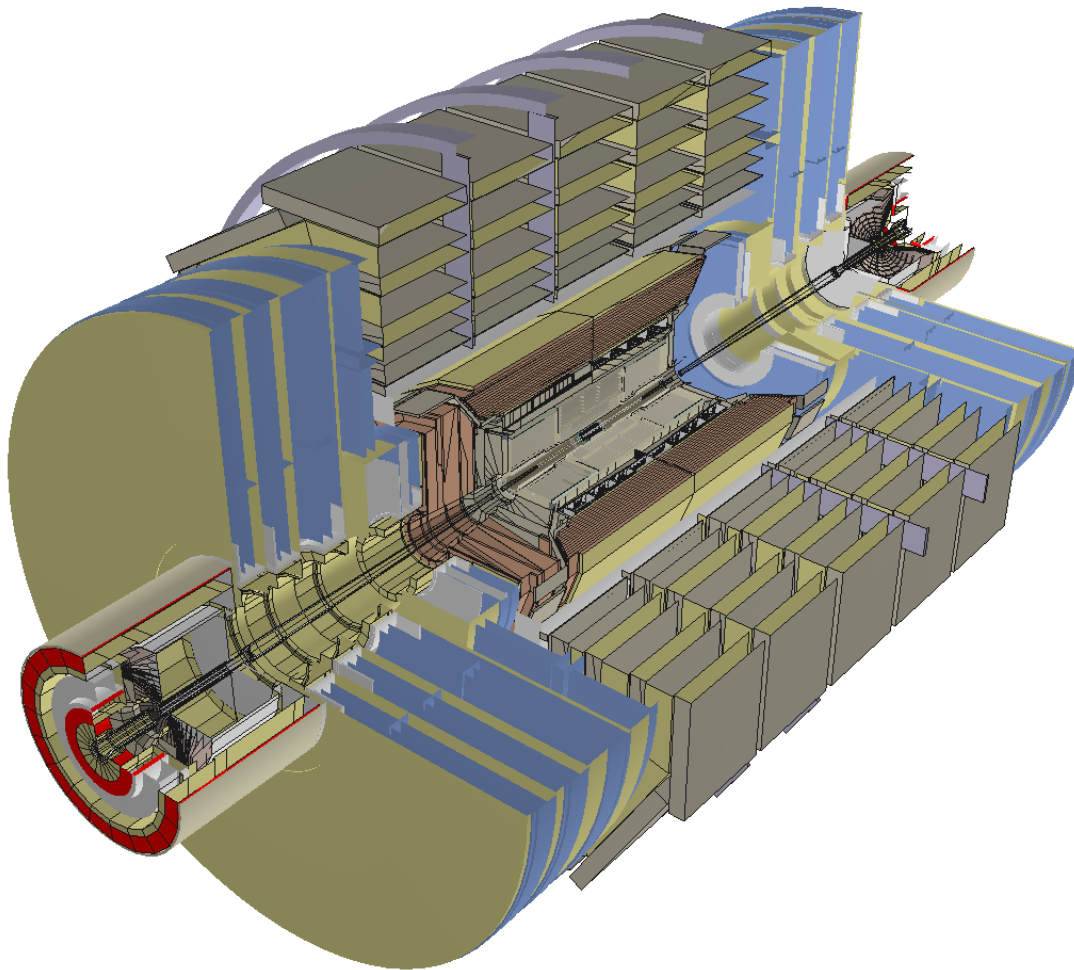
```
DetectorConstructor::Construct() {  
  
    // gdml parser  
    #include "G4GDMLParser.hh"  
    G4GDMLParser fParser;  
    G4VPhysicalVolume* fWorldPhysVol;  
  
    // exporting geometry  
    fParser.Write("geometrydump.gdml", fWorldPhysVol);  
  
}
```

# Tessellated solids



- The geometry imported from GDML will be made by **tessellated solids** `G4TessellatedSolid`
  - Generic solid defined by a **number of facets** (`G4VFacet`)
  - Facets can be triangular or quadrangular

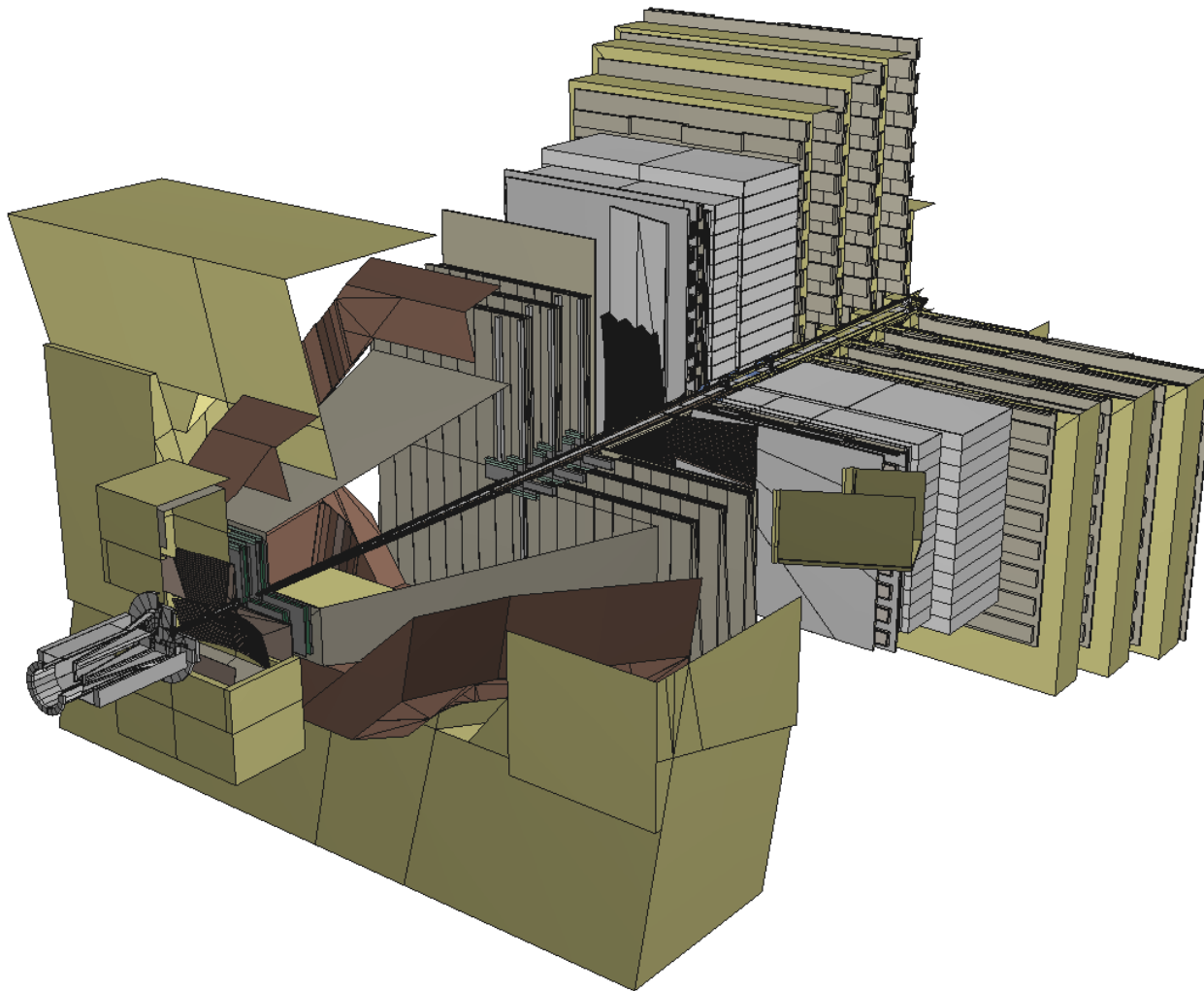
# CMS detector: G4→ GDML→ ROOT



*~19000 physical  
volumes*

*snapshot provided by  
R.Maunder*

# LHCb detector: G4→ GDML→ ROOT



*~5000 physical  
volumes*

*snapshot provided by  
R.Maunder*



# CAD import

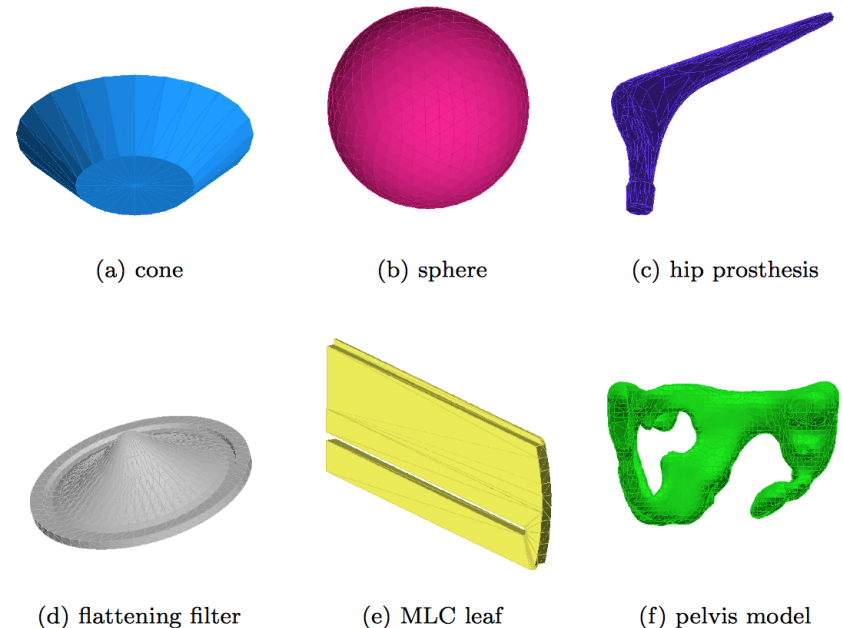
---

- Typical request: import **CAD technical drawings** as Geant4 geometries
- Difficulties:
  - **Proprietary, undocumented** or changing CAD formats
  - Usually no connection between **geometry** and **materials**
  - CAD is never as easy as you might think
- Possible solution (a lot of work!)
  - Convert **CAD into STEP** (no material information)
  - Convert **STEP into GDML** and restore manually material information
    - Needs **commercial software** (ST-viewer, FastRAD)

# CADMesh

- **CADMesh** is a **direct CAD model import interface** for Geant4 optionally leveraging VCGLIB, and ASSIMP by default.
  - Currently it supports the import of **triangular facet surface meshes** defined in formats such as STL and PLY
  - A **G4TessellatedSolid** is returned and can be included in a **standard user detector construction**

<https://code.google.com/p/cadmesh/>  
<http://arxiv.org/pdf/1105.0963.pdf>



**Fig. 3** Six test geometries loaded directly into GEANT4 using the proposed CAD interface and visualised using the GEANT4 OpenGL viewer.



# Parallel world

---



# Parallel world

---

- Parallel navigation functionality allows the user to define **more than one worlds** simultaneously
- New **G4Transportation** process sees **all worlds** simultaneously
  - A step is limited **not only by the boundary of the mass geometry** but also by the **boundaries of parallel geometries**: a step will never cross a boundary of any volume in any parallel world
  - Materials, production thresholds and EM field are used **only from the mass geometry**
- In a parallel world, the user can define volumes in arbitrary manner with **sensitivity**, regions with **shower parameterization**, and/or **importance field for biasing**
  - Volumes in different worlds *may overlap*





# Parallel world and navigation

---

- **G4VUserParallelWorld** is the new base class where the user implements a parallel world.
  - The world physical volume of the parallel world is provided by G4RunManager as a **clone of the mass geometry**
    - The same world volume applies to all parallel worlds
  - All UserParallelWorlds must be **registered** to UserDetectorConstruction.
  - Each parallel world has its dedicated **G4Navigator** object, that is automatically assigned
- The user has to have **G4ParallelWorldProcess** in his physics list.

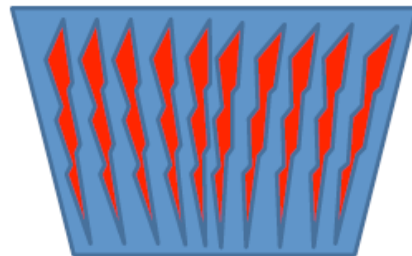
# example/extended/runAndEvent/RE06

- Mass geometry
  - sandwich of rectangular absorbers and scintillators
- Parallel scoring geometry
  - Cylindrical layers

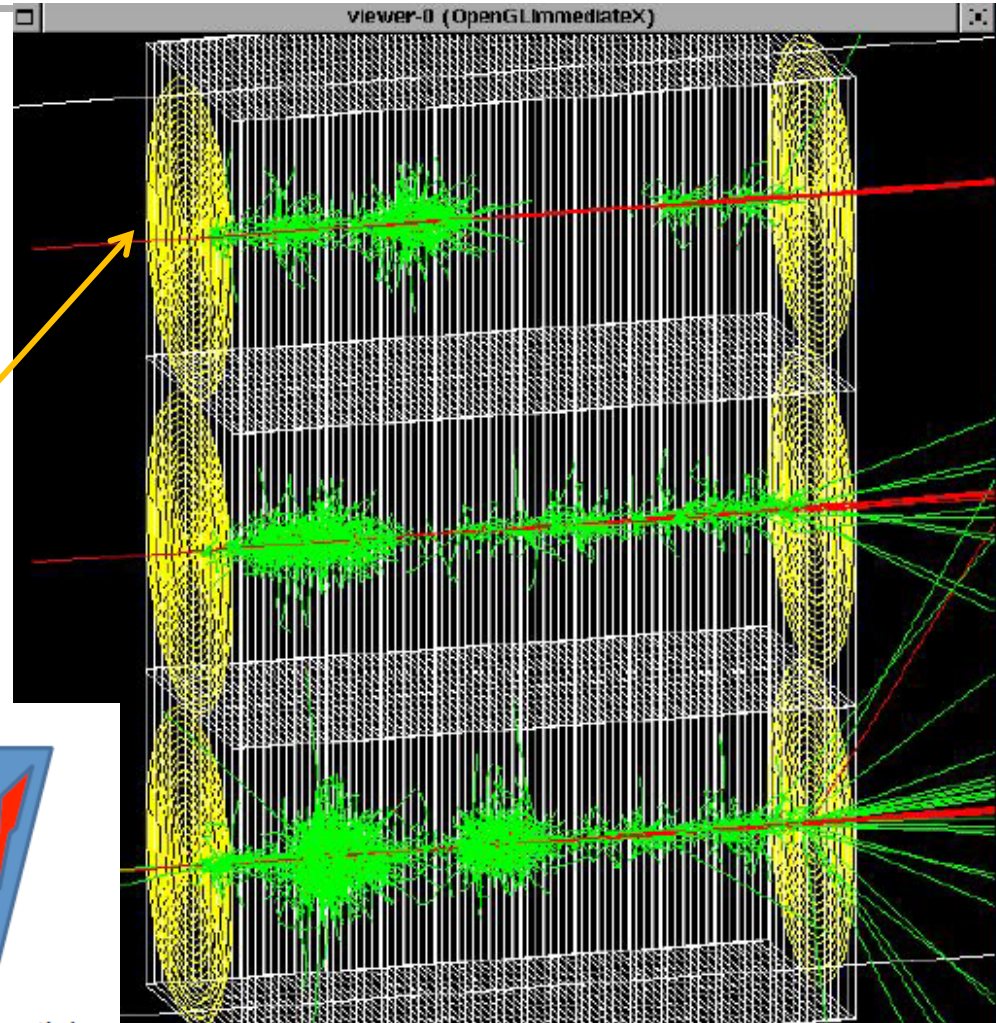
Shower parametrization



Geometry seen by  $e^+$ ,  $e^-$ ,  $\gamma$



Geometry seen by other particles





# Hands-on session

---

- Task1c
  - Magnetic fields
- `http://202.122.35.42/task1`



# Backup

---



# Parallel world

---

- The possibility to define a **scoring volume** different from the **physical volumes** available since the old times of Geant4 (**ROGeometry**)
- Occasionally, it is **not straightforward to define sensitivity**, importance or envelope to be assigned to volumes in the mass geometry.
  - Typically a geometry built machinery by CAD, GDML, DICOM, etc. has this difficulty. Mass geometry is composed by voxels or tessels (difficult to be treated individually for sensitivity)
- Other concurrent/similar requirements emerged since then
  - Ghost volume for shower parameterization
  - Importance field geometry for geometry importance biasing assigned to importance biasing process
  - Scoring geometry assigned to scoring process
- New design → everything merged into **G4ParallelWorld**