GEANT4
A SIMULATION TOOLKIT

Version 10.5

Geometry I

Gabriele Cosmo (CERN)

Geant4 Beginners Course

Adapted/extended from original material by M.Asai (SLAC)

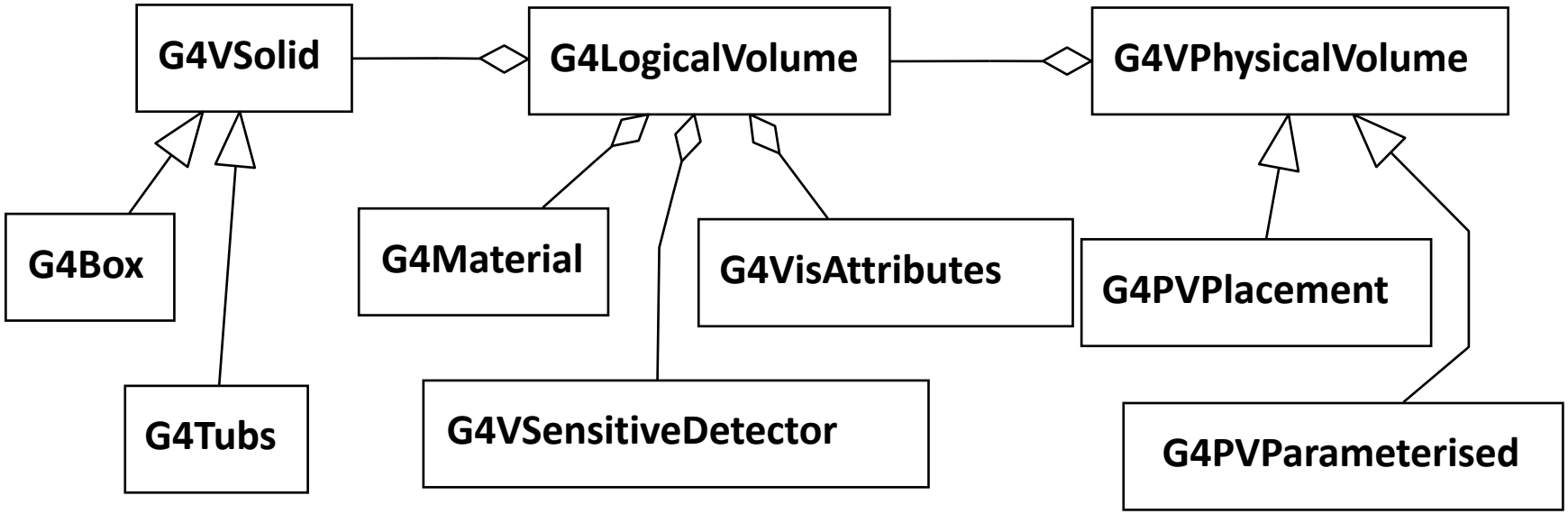
Contents

- Introduction
- User Detector Construction class
- Solids/shapes & constructs
- Logical volumes
- Physical volumes
- Placements, replica & parameterized volumes
- Touchables
- Geometry checking tools
- GDML persistency
- Basics of magnetic field (John)

Introduction

Detector geometry

- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes, material, sensitivity, user limits, etc.*
  - G4VPhysicalVolume -- *position, rotation*

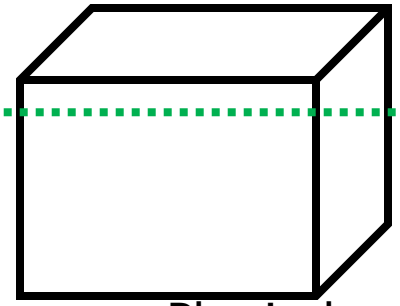


Define detector geometry

- 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);
```

Logical volume :
"Solid" : shape and size
+ material, sensitivity, etc.

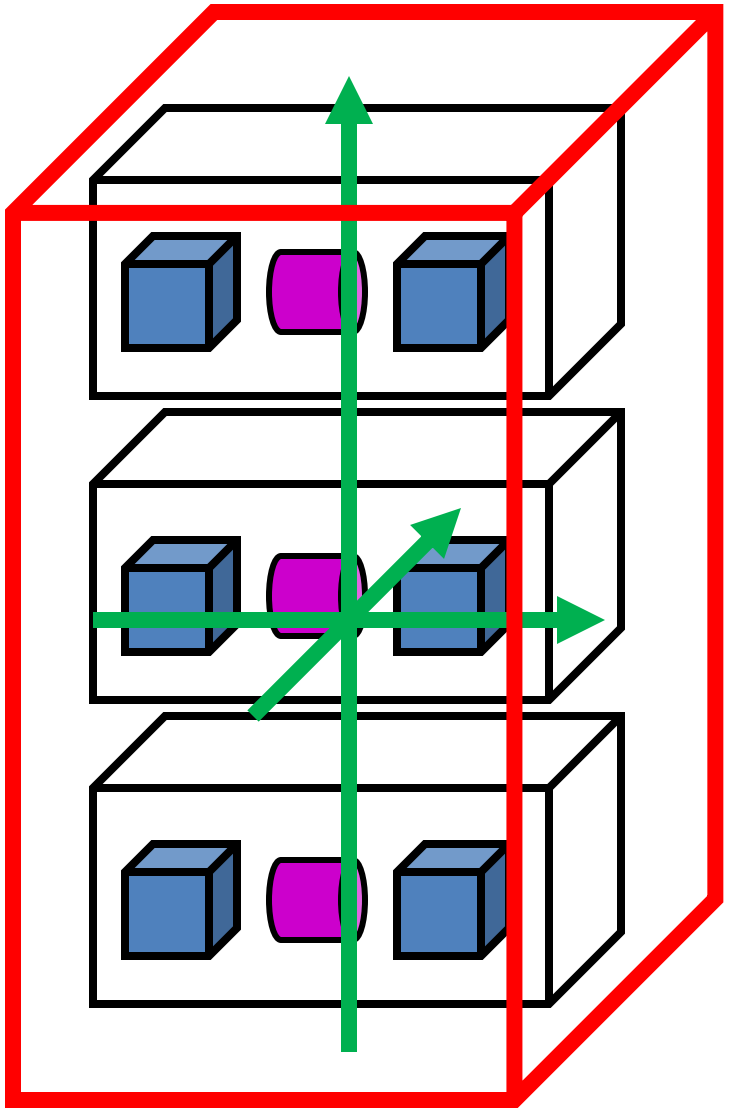


Physical volume :
+ rotation and position

- 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 mother volume's local coordinate system is at the center of the mother volume
  - Daughter volume cannot protrude from mother volume

Geometrical hierarchy

- One logical volume can be placed more than once. One or more volumes can be placed in a mother volume
- Note that the mother-daughter relationship is an information of G4LogicalVolume
  - If the mother volume is placed more than once, all its daughters are by definition appearing in all placed physical volumes
- The world volume must be a unique physical volume which fully contains with some margin all other volumes
  - 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



User Detector Construction

User classes

- main()
  - Geant4 does not provide *main()*

  Note: classes written in red are mandatory
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to enable the user classes
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserActionInitialization
- Action classes
  - Instantiated in G4VUserActionInitialization.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

G4VUserDetectorConstruction

```
class G4VUserDetectorConstruction
{
  public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();
  public:
    virtual G4VPhysicalVolume* Construct() = 0;
    virtual void ConstructSDandField();
 public:
    void RegisterParallelWorld(G4VUserParallelWorld*);
```

The *Construct()* method should return the pointer of the world physical volume. The world physical volume represents your whole geometry setup

Sensitive detector and field should be instantiated and set to logical volumes in the *ConstructSDandField()* method

Your Detector Construction class

```
#ifndef MyDetctorConstruction_h
#define MyDetctorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"

class MyDetectorConstruction
     : public G4VUserDetectorConstruction
{
  public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();
    virtual G4VPhysicalVolume* Construct();
    virtual void ConstructSDandField();
  public:
   // set/get methods if needed
  private:
   // granular private methods if needed
   // data members if needed
};
#endif
```

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class
- Implement Construct() and ConstructSDandField() methods
  1) Construct all necessary materials
  2) Define shapes/solids
  3) Define logical volumes
  4) Place volumes of your detector geometry
  5) Associate (magnetic) field to geometry *(optional)*
  6) Instantiate sensitive detectors / scorers and set them to corresponding logical volumes *(optional)*
  7) Define visualization attributes for the detector elements *(optional)*
  8) Define regions *(optional)*
- Set your construction class to G4RunManager or G4MTRunManager

Exercise – 1

Inspect examples/basic/B1 – B1DetectorConstruction

Task-J : Inspect example B1

- Build and run example B1

```
cd $HOME/geant4
mkdir work/task_j
mkdir work/task_j/B1_build
cp -r $G4EXAMPLES/basic/B1 .
cd work/task_j/B1_build
cmake -DGeant4_DIR=$G4COMP ../../../B1
make -j4
./exampleB1
```

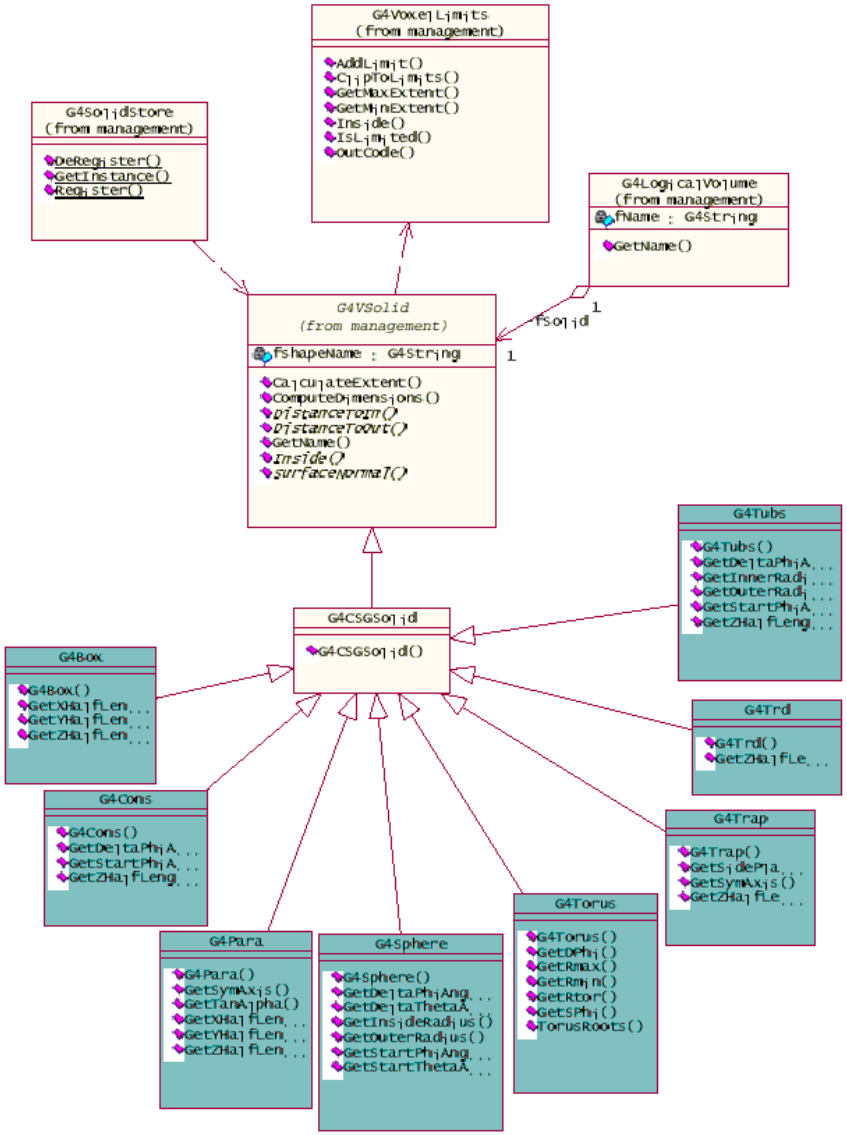
- Edit and inspect the B1DetectorConstruction class

```
cd $HOME/geant4/B1
ls -F
ls -F include
ls -F src
atom */B1DetectorConstruction*
```

Solids/shapes & constructs

G4VSolid

- Abstract class. All solids in Geant4 are derived from it
- It defines but not implement all functions required 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
- User can create his/her own solid class



Solids

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

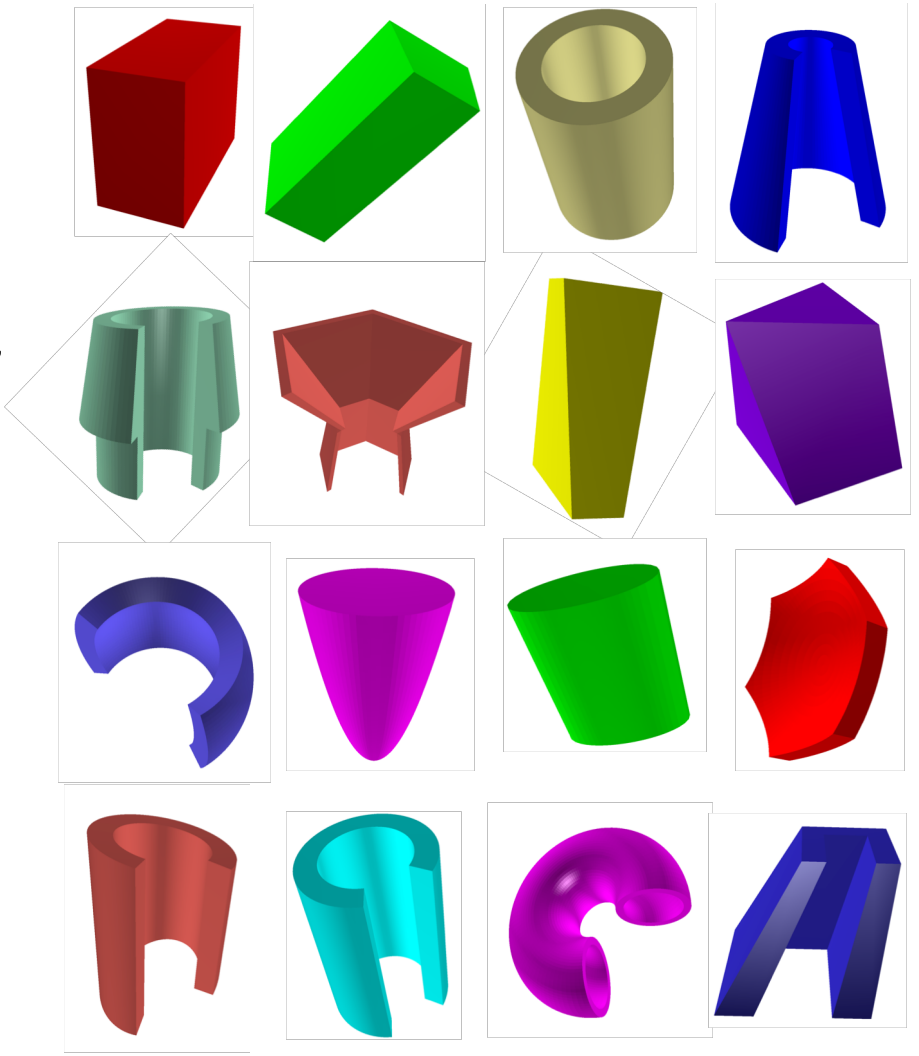
- G4TessellatedSolid, G4ExtrudedSolid

Boolean & scaled solids

- G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid, G4MultiUnion, G4ScaledSolid

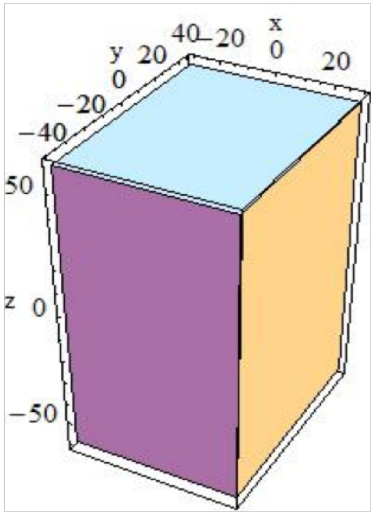
Twisted shapes

- G4TwistedBox, G4TwistedTrap, G4TwistedTrd, G4TwistedTubs

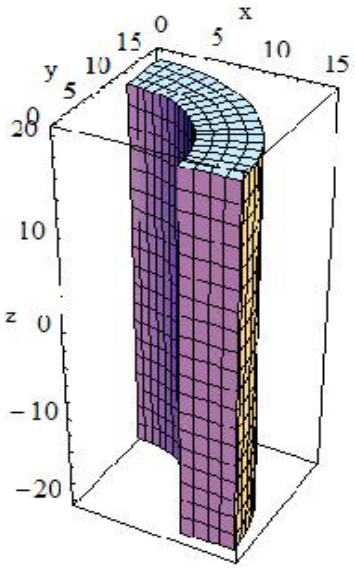


CSG: G4Box, G4Tubs

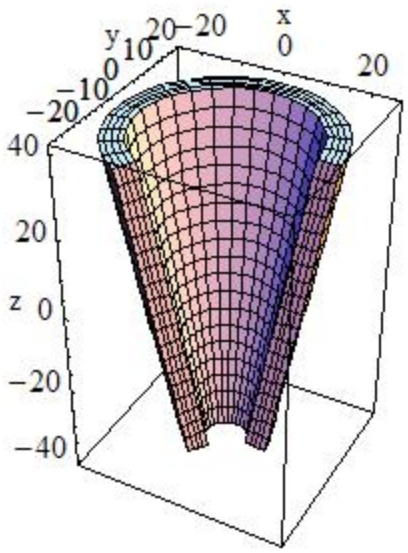
```
G4Box(const G4String &pname,    // name
                G4double half_x,    // X half size
                G4double half_y,    // Y half size
                G4double half_z);   // Z half size
```



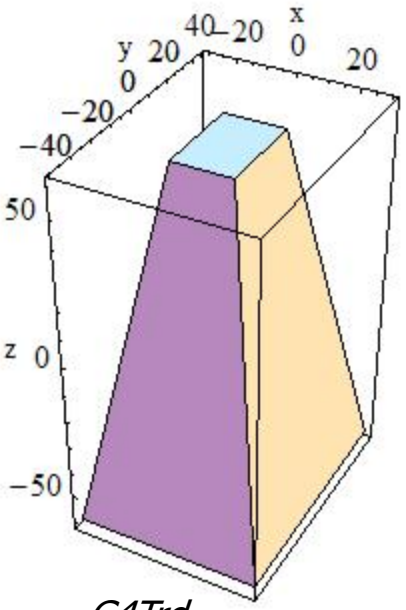

```
G4Tubs(const G4String &pname,   // name
                 G4double  pRmin,   // inner radius
                 G4double  pRmax,   // outer radius
                 G4double  pDz,     // Z half length
                 G4double  pSphi,   // starting Phi
                 G4double  pDphi); // segment angle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



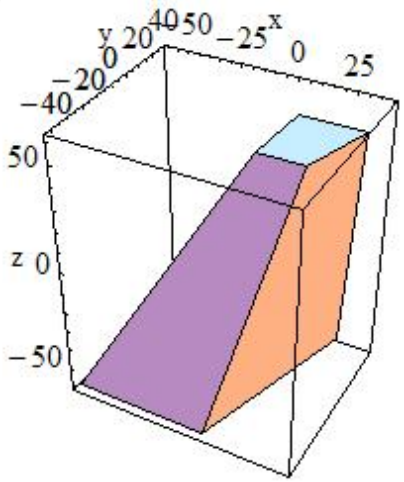
Other CSG Solids



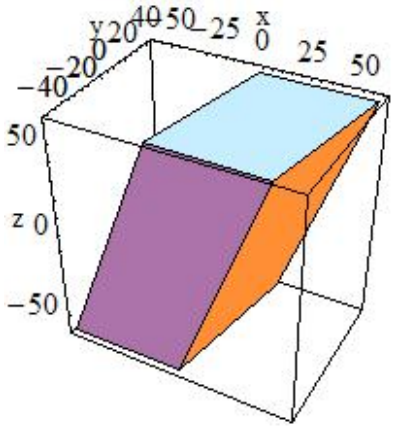
G4Cons



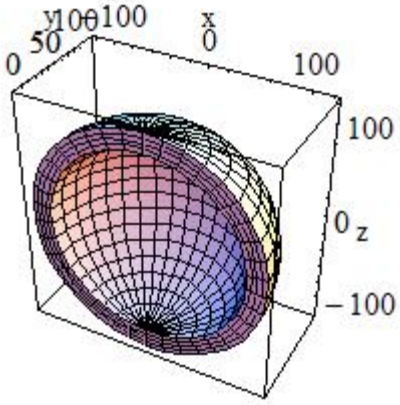
G4Trd



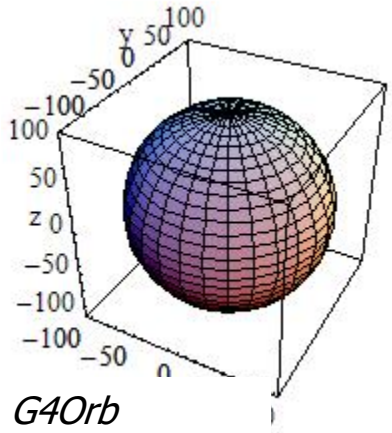
G4Trap



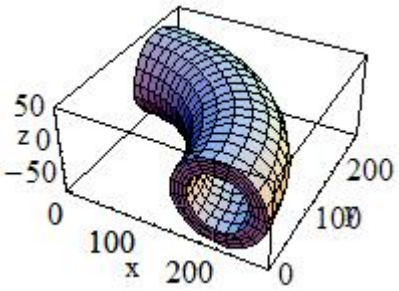
G4Para
(parallelepiped)



G4Sphere



G4Orb
(full solid sphere)



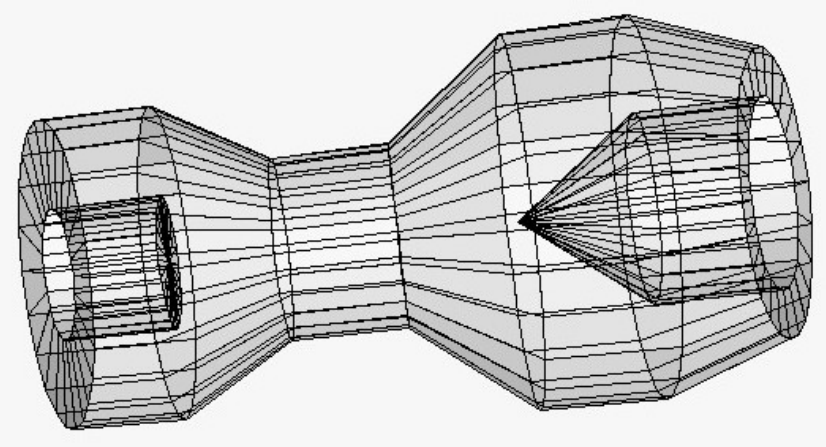
G4Torus

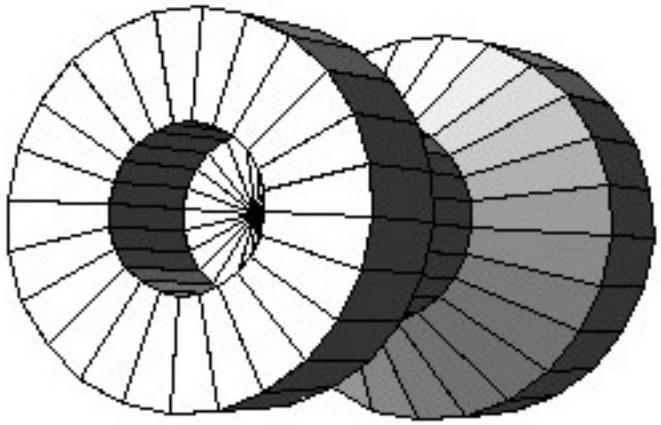
Consult the Geant4 Application Developers Guide for all available shapes

Specific CSG Solids: Polycone

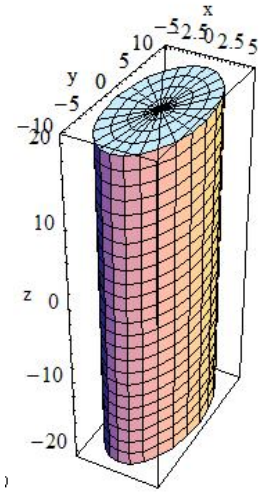
```
G4Polycone(const G4String& pName,
           G4double phiStart, G4double phiTotal,
           G4int numZPlanes,
           const G4double zPlane[],
           const G4double rInner[]
           const G4double rOuter[]);
```

- **`numZPlanes`** - numbers of **`Z`** planes
- **`zPlane`** – position of **`Z`** planes, with **`Z`** in increasing order
- **`rInner, rOuter`** – tangent distances to inner/outer surface

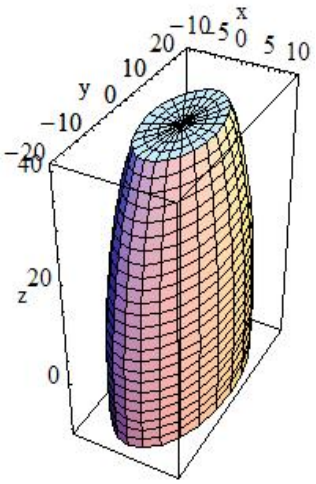




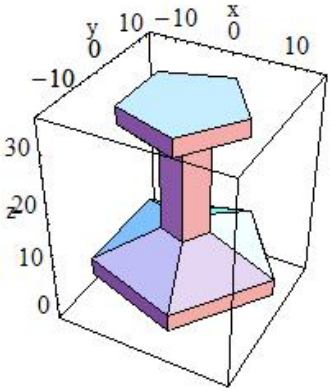
Other specific CSG solids



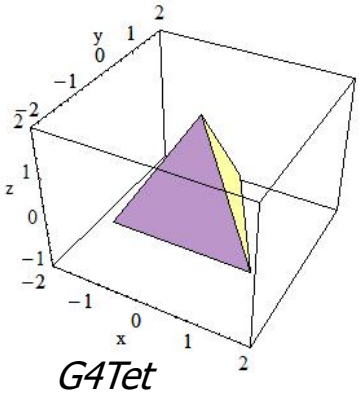
G4EllipticalTube



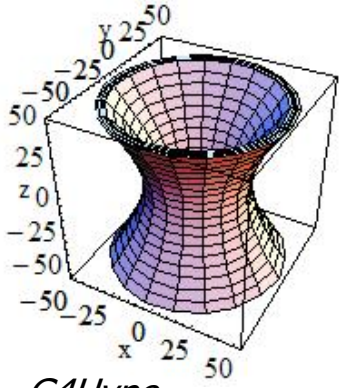
G4Ellipsoid



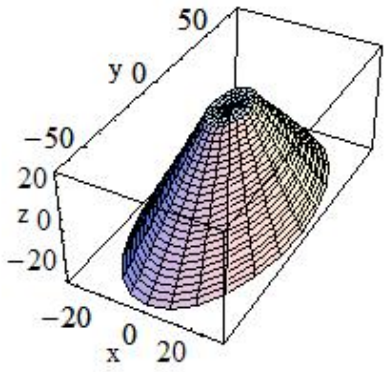
G4Polyhedra



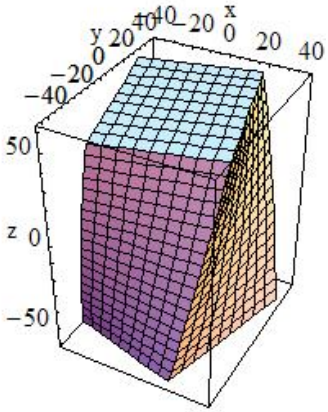
G4Tet
(tetrahedra)



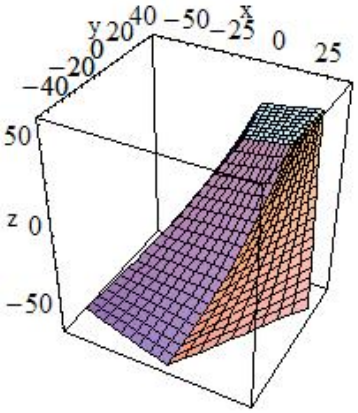
G4Hype



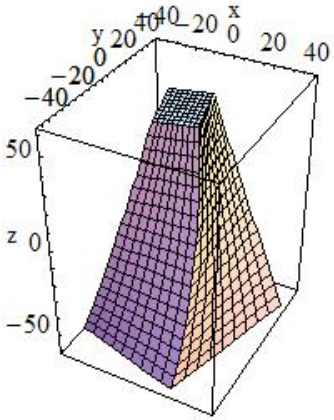
G4EllipticalCone



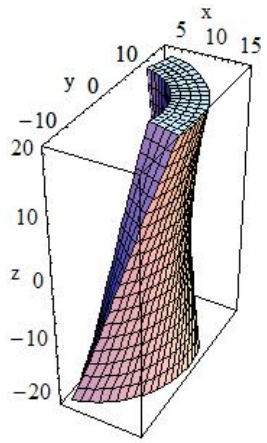
G4TwistedBox



G4TwistedTrap



G4TwistedTrd

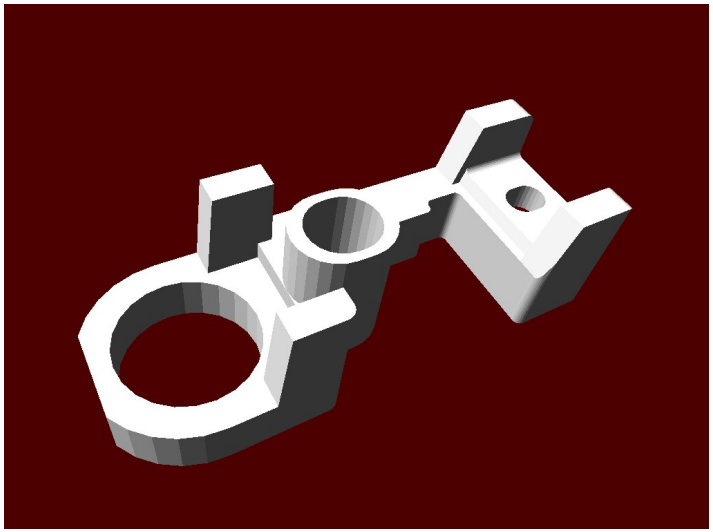


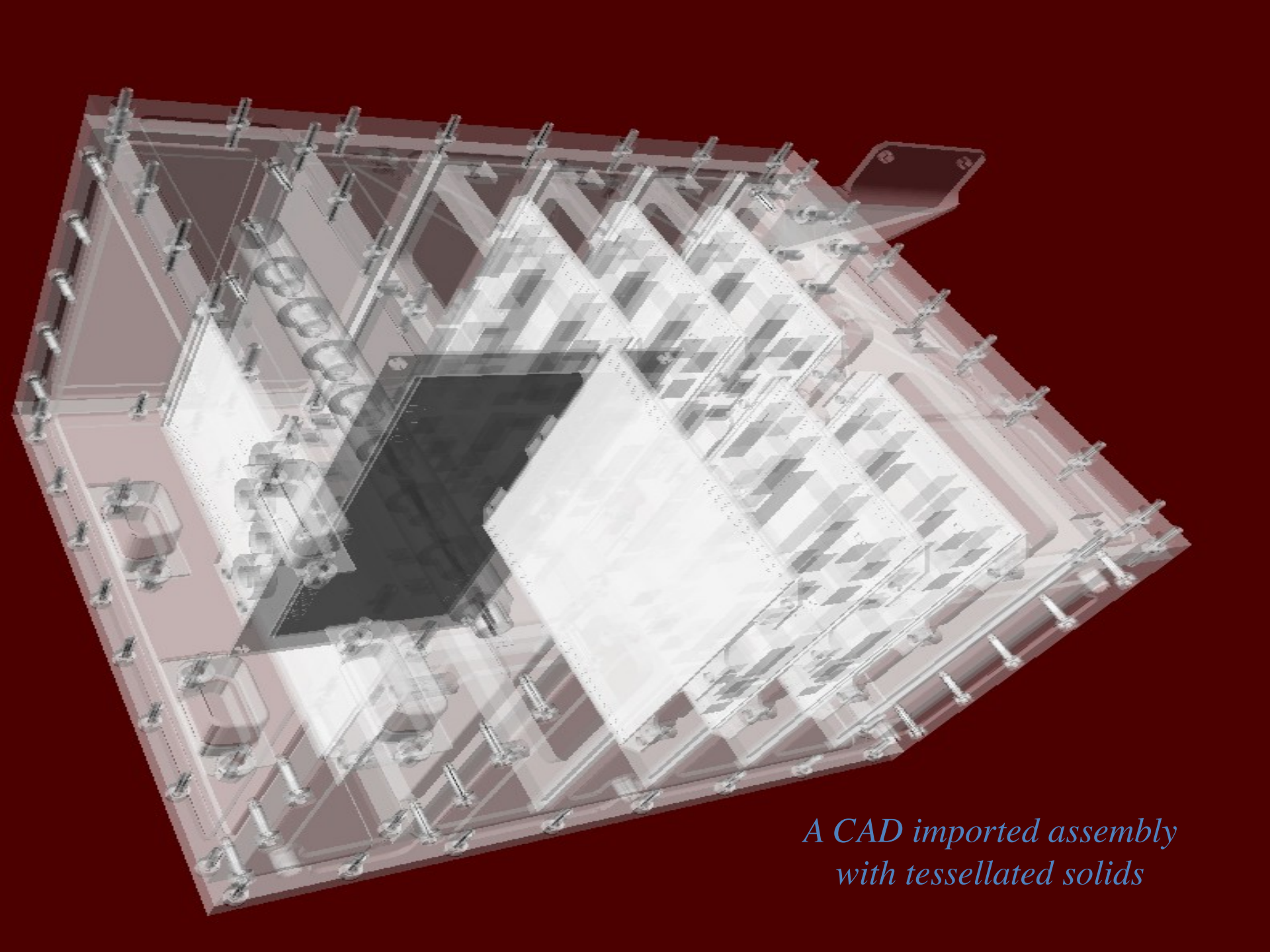
G4TwistedTubs

Consult the Geant4 Application Developers Guide for all available shapes

Tessellated Solid

- **`G4TessellatedSolid`**
  - Generic solid defined by a number of facets (`G4VFacet`)
    - Facets can be triangular (`G4TriangularFacet`) or quadrangular (`G4QuadrangularFacet`)
  - Constructs especially important for conversion of complex geometrical shapes imported from CAD systems
  - Can also be explicitly defined:
    - By providing the vertices of the facets in *anti-clock wise* order, in *absolute* or *relative* reference frame
  - GDML binding

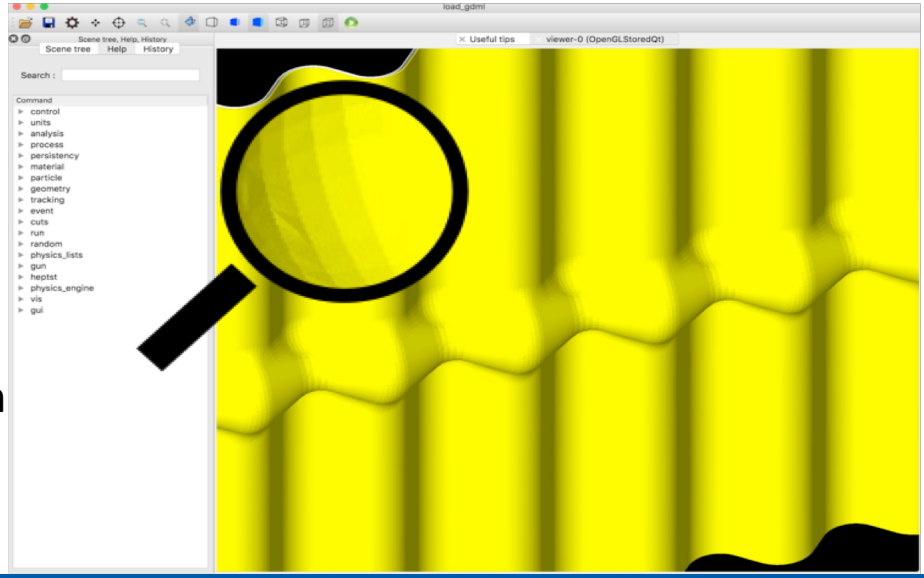




A CAD imported assembly with tessellated solids

VecGeom – New geometry modelling library

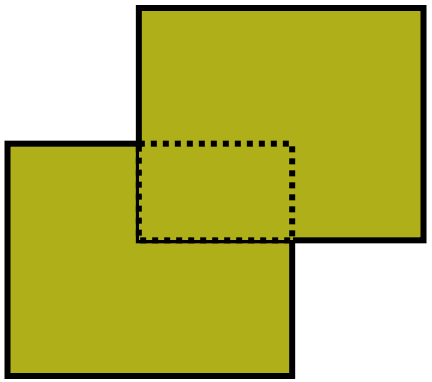
- An important effort started in the recent years to write a new modelling library, reviewing at the algorithmic level most of the primitives, providing an enhanced, optimized and well-tested implementation to be shared among software packages
- In most cases considerable performance improvement was achieved
  - For example, the time required to compute intersections with the tessellated solid was considerably reduced with the adoption of spatial partitioning for composing facets into a 3D grid of voxels, grouping them using vector SIMD instructions
- Such technique allows considerable speedup for relatively complex structures of the order of 100k to millions of facets, which is typical for geometry descriptions imported from CAD drawings
  - It is now possible to use tessellated geometries for tuning the precision in simulation by increasing the mesh resolution, something which was hardly possible before
- https://gitlab.cern.ch/VecGeom/VecGeom
- Can be used as alternative implementation for replacing original Geant4 primitives (version v01.01.00), with same API



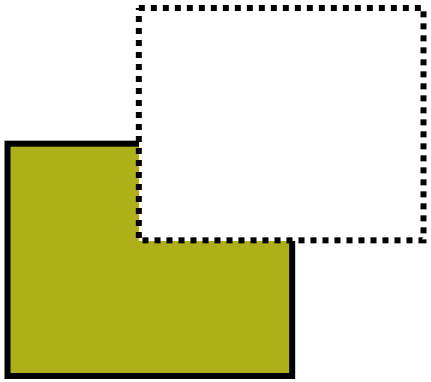
Boolean solids

- 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
    - A third solid can be combined to the resulting solid of first operation
    - So solids can be combined with other Boolean solids

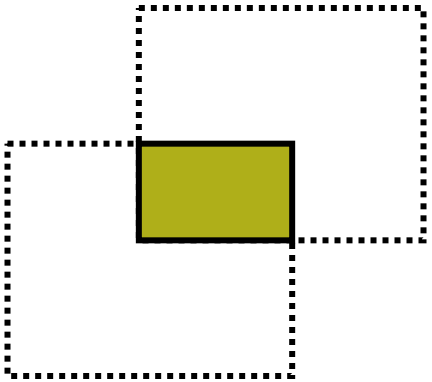
G4UnionSolid



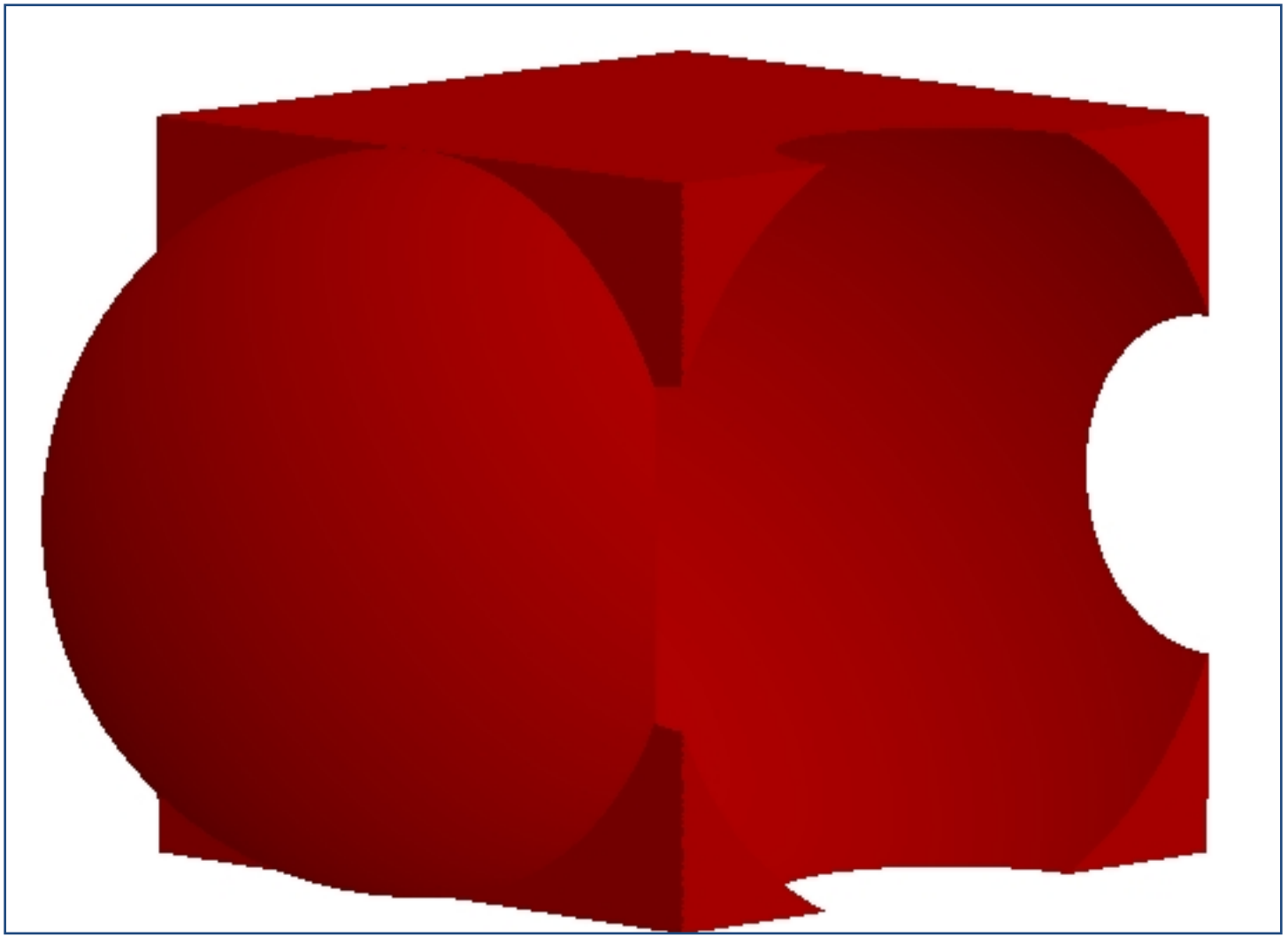
G4SubtractionSolid



G4IntersectionSolid



Boolean solid



Boolean solids - example

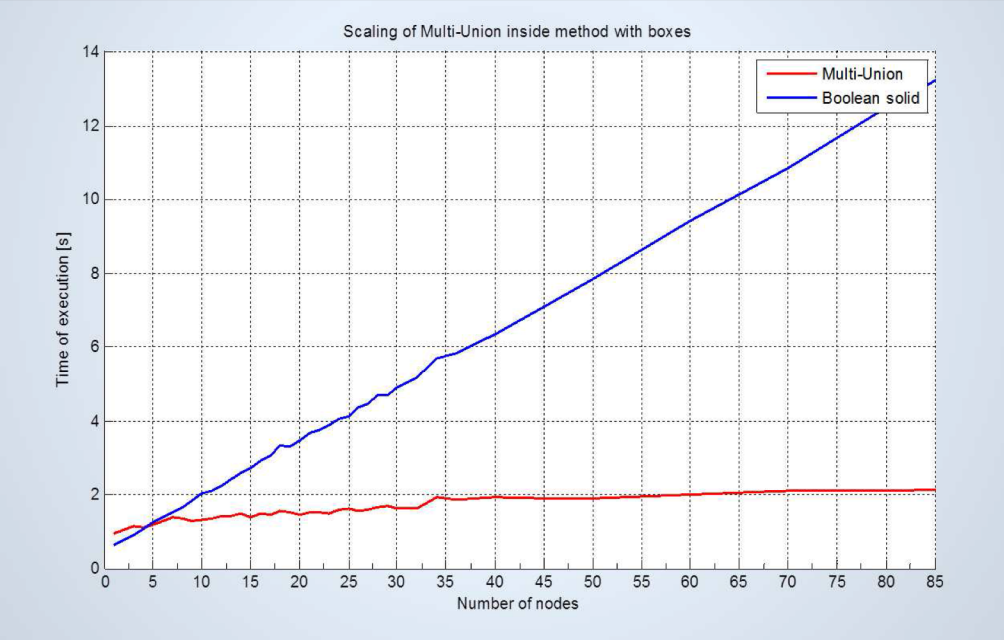
```
G4VSolid* box = new G4Box("Box",50*cm,60*cm,40*cm);
G4VSolid* cylinder
 = new G4Tubs("Cylinder",0.,50.*cm,50.*cm,0.,2*M_PI*rad);
G4VSolid* union
 = new G4UnionSolid("Box+Cylinder", box, cylinder);
G4VSolid* subtract
 = new G4SubtractionSolid("Box-Cylinder", box, cylinder,
       0, G4ThreeVector(30.*cm,0.,0.));
G4RotationMatrix* rm = new G4RotationMatrix();
rm->RotateX(30.*deg);
G4VSolid* intersect
   = new G4IntersectionSolid("Box&&Cylinder",
       box, cylinder, rm, G4ThreeVector(0.,0.,0.));
```

- The origin and the coordinates of the combined solid are those of the first solid

Multi-Union construct

- In addition to a full set of highly optimized primitives and Boolean combinations, the modeller includes a “multi-union” construct implementing a composite set of many solids to be placed in 3D space
- This differs from the simple technique based on Boolean unions, with the aim of providing excellent scalability on the number of constituent solids
- The multi-union adopts a similar voxelization technique to partition 3D space used for the tessellated solid, allowing to dramatically improve speed and scalability over the original implementation based on Boolean unions

```
// Initialise a MultiUnion structure
   G4MultiUnion* munion_solid = new
            G4MultiUnion("Boxes_Union");
// Add the shapes to the structure
   munion_solid->AddNode(*box1,tr1);
   munion_solid->AddNode(*box2,tr2);
     :
// Finally close the structure
   munion_solid->Voxelize();
// Associate it to a logical volume as normal solid
   G4LogicalVolume* lvol = new
     G4LogicalVolume(munion_solid, // its solid
                     munion_mat,   // its material
                     "Boxes_Union_LV"); // its name
```



Logical Volumes

G4LogicalVolume

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

- Contains all information of a volume except position and rotation
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits, Region
- Physical volumes of same type can share the common logical volume object
- The pointer to the solid must NOT be null
- The pointer to the material must NOT be null for the tracking geometry
- It is not meant to act as a base class

Computing Capacity and Weight

- The geometrical capacity of a generic solid or Boolean composition can be computed from the solid:

```
G4double GetCubicVolume();
```

  - Exact capacity is mathematically calculated for most of the solids, while estimation based on Monte Carlo integration may be given for other solids

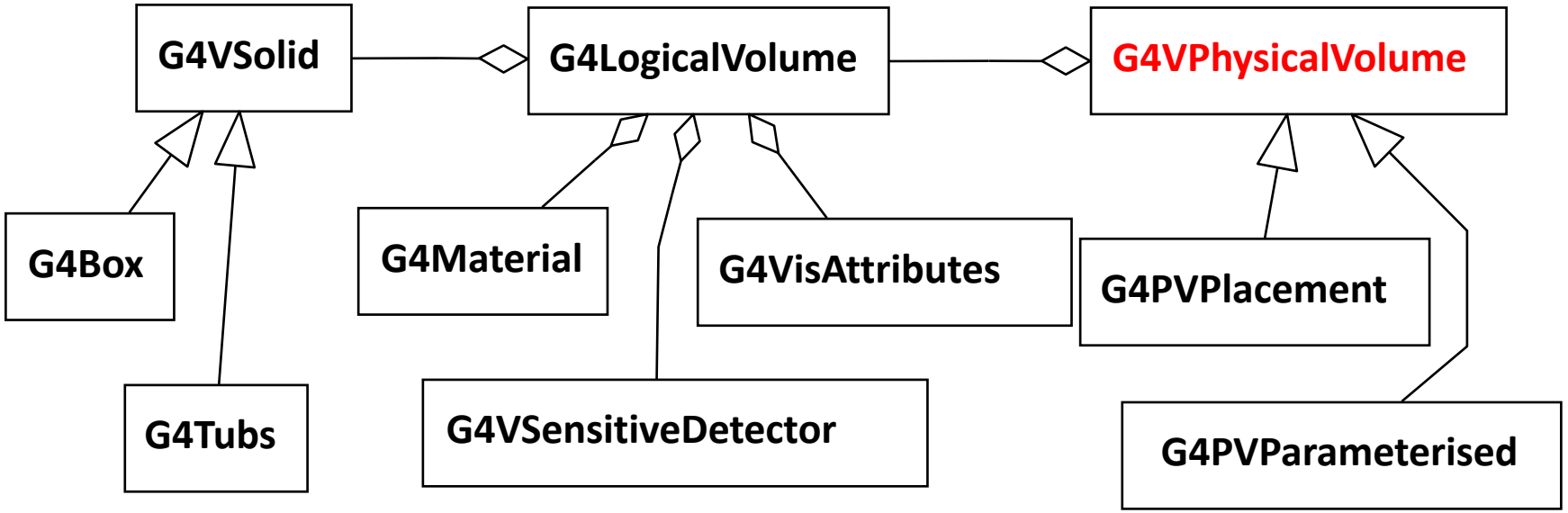
- Overall weight of a geometry setup can be computed from the logical volume:

```
G4double GetMass(G4bool forced=false,
                 G4bool propagate=true,
                 G4Material* pMaterial=0);
```

  - The computation may require a considerable amount of time, depending on the complexity of the geometry
  - The return value is cached and reused until ***forced*=true**
  - Daughter volumes will be neglected if ***propagate*=false**

Physical Volumes

- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes,*
    material, sensitivity, user limits, etc.
  - G4VPhysicalVolume -- *position, rotation*



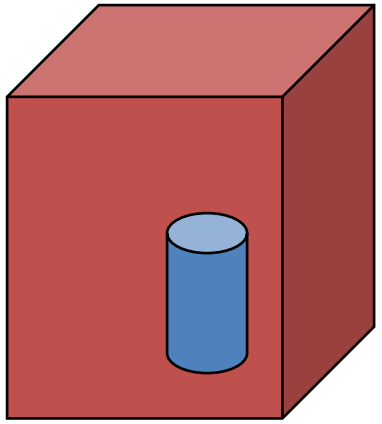
Define a detector geometry

- Basic strategy

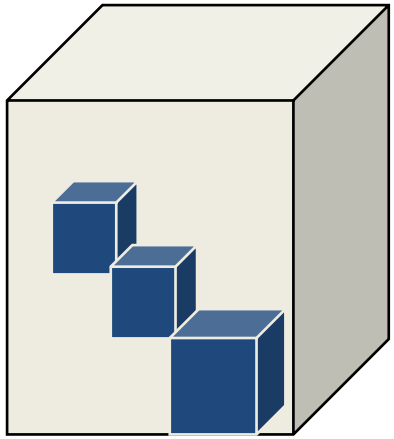
```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
G4LogicalVolume* pBoxLog =
  new G4LogicalVolume( pBoxSolid, pBoxMaterial,
                       "aBoxLog", 0, 0, 0);
G4VPhysicalVolume* aBoxPhys =
  new G4PVPlacement( pRotation,
    G4ThreeVector(posX, posY, posZ), pBoxLog,
    "aBoxPhys", pMotherLog, 0, copyNo);
```

Physical Volumes

- Placement volume - it is one positioned volume
  - One physical volume object represents one "real" volume
- Repeated volume - a volume placed many times
  - One physical volume object represents any number of "real" volumes
  - reduces use of memory
  - *Parameterised*
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterized by the copy number
  - *Replica and Division*
    - simple repetition along one axis
- A mother volume can contain either
  - many placement volumes
  - or, one repeated volume



placement

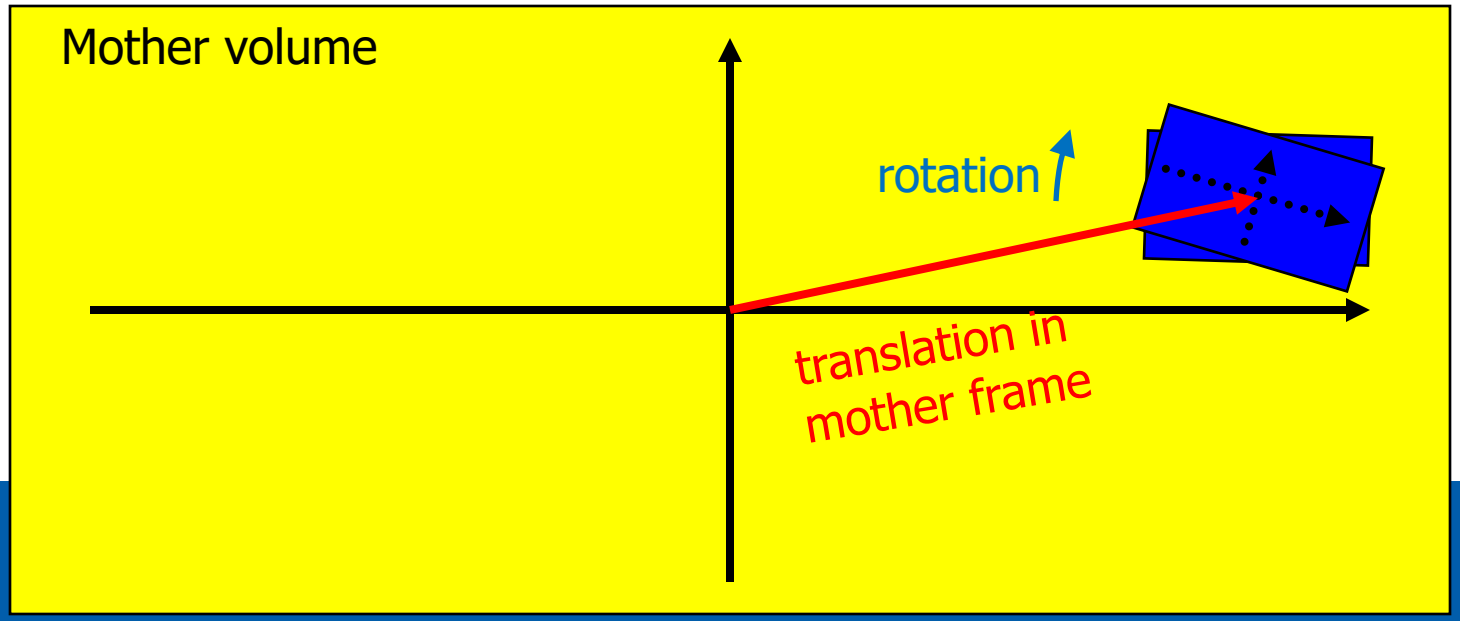


repeated

G4PVPlacement

```
G4PVPlacement(
  G4Transform3D(G4RotationMatrix &pRot, // rotation of daughter volume
                const G4ThreeVector &tlate), // position in mother frame
  G4LogicalVolume *pDaughterLogical,
  const G4String &pName,
  G4LogicalVolume *pMotherLogical,
  G4bool pMany, // not supported/for future use…
  G4int pCopyNo, // unique arbitrary integer
  G4bool pSurfChk=false); // optional boundary check
```

- Single volume positioned relatively to the mother volume



Alternative G4PVPlacement construct

```
G4PVPlacement(G4RotationMatrix* pRot,   // rotation of mother frame
              const G4ThreeVector &tlate, // position in mother frame
              G4LogicalVolume *pDaughterLogical,
              const G4String &pName,
              G4LogicalVolume *pMotherLogical,
              G4bool pMany, // not supported/for future use…
              G4int pCopyNo, // unique arbitrary integer
              G4bool pSurfChk=false); // optional boundary check
```

- Single volume positioned relatively to the mother volume frame

Note:

- This G4PVPlacement is identical to the previous one if there is no rotation
  - For power-users: previous one is much easier to understand
- The advantage of this second constructor is setting the pointer of the rotation matrix rather than providing the values of the matrix
  - You may change the matrix without accessing to the physical volume

Exercise – 2

Simple placements of volumes
Mother-daughter relationship…

Task-J : Placement of volumes

- Modify B1DetectorConstruction class in example B1
  1. Remove definition of “shape1” and “shape2” from Construct()
  2. Define a orb ‘Shape1’ – 0.2 cm radius – material air (G4_AIR)
  3. Define a box ‘Shape2’ - 1x2x1 cm – material aluminum (G4_Al)
  4. Place ‘Shape1’ volume in ‘Shape2’ volume at the origin (0,0,0)
  5. Place ‘Shape2’ volume 5 times inside “Envelope” at:
     - origin ; (-2.5, -2.5, -7.5)cm ; (2.5, 2.5, -7.5)cm ; (-2.5, -2.5, 7.5)cm ; (2.5, 2.5, 7.5)cm

- Solution

```
// Shape 1
#include "G4Orb.hh"
G4Orb* shape1 = new G4Orb("Shape1", 0.2*cm);
G4LogicalVolume* logicShape1 = new G4LogicalVolume(shape1, world_mat, "Shape1");
// Shape 2
G4Material* shape2_mat = nist->FindOrBuildMaterial("G4_Al");
G4Box* shape2 = new G4Box("Shape2", 0.5*cm, 1*cm, 0.5*cm);
G4LogicalVolume* logicShape2 = new G4LogicalVolume(shape2, shape2_mat, "Shape2");
// Place orb (shape1) in box (shape2)
new G4PVPlacement(0, G4ThreeVector(0,0,0), logicShape1, "Hole", logicShape2, false, 0, false);
// Place five boxes (shape2) in envelope
G4ThreeVector pos1(-2.5*cm,-2.5*cm,-7.5*cm), pos2(2.5*cm,2.5*cm,-7.5*cm),
              pos3(-2.5*cm,-2.5*cm,7.5*cm), pos4(2.5*cm,2.5*cm,7.5*cm);
new G4PVPlacement(0, G4ThreeVector(0,0,0), logicShape2, "Block", logicEnv, false, 1, false);
new G4PVPlacement(0, pos1, logicShape2, "Block", logicEnv, false, 2, false);
new G4PVPlacement(0, pos2, logicShape2, "Block", logicEnv, false, 3, false);
new G4PVPlacement(0, pos3, logicShape2, "Block", logicEnv, false, 4, false);
new G4PVPlacement(0, pos4, logicShape2, "Block", logicEnv, false, 5, false);
```

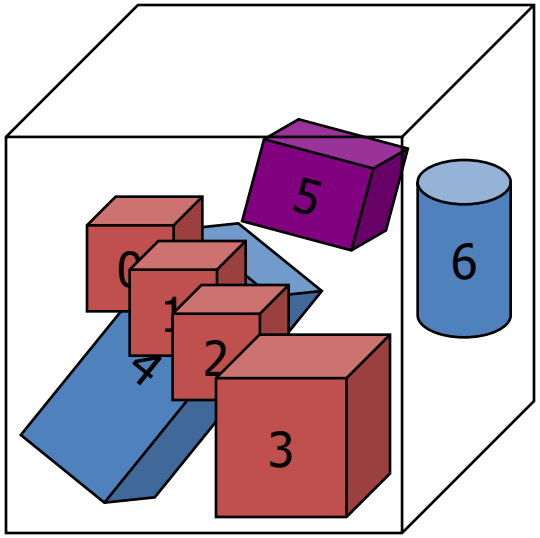
G4PVParameterised

```
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 parameterisation **pParam**, within the mother volume **pMother**
- **pAxis** is a "suggestion" to the navigator along which Cartesian axis replication of parameterized volumes dominates
  - kXAxis, kYAxis, kZAxis : one-dimensional optimization
  - kUndefined : three-dimensional optimization

Parameterised Physical Volumes

- User should implement a class derived from G4VPVParameterisation abstract base class and define the following as a function of copy number:
  - where it is positioned (transformation, rotation)
- Optionally:
  - the size of the solid (dimensions)
  - the type of the solid, material, sensitivity, vis attributes
- All daughters must be fully contained in the mother
- Daughters should not overlap to each other
- Limitations/suggestions:
  - Applies to a limited set of solids only
    - Box, Tube, Trd, Simple Trapezoid, Cone, Sphere, Orb, Ellipsoid, Torus, Parallelepiped, Polycone, Polyhedron, Hyperboloid
  - Granddaughter volumes allowed only for special cases
  - Consider parameterised volumes as “leaf” volumes
- Typical use-cases
  - Complex detectors
    - with large repetition of volumes, regular or irregular
  - Medical applications
    - the material in animal tissue measured as cubes with varying material



Exercise – 3

Inspect example basic/B2/B2b

Task-J : Inspect example B2b

- Build and run example B2b

```
cd $HOME/geant4
mkdir work/task_j/B2_build
cp -r $G4EXAMPLES/basic/B2/B2b .
cd work/task_j/B2_build
cmake -DGeant4_DIR=$G4COMP ../../../B2b
make -j4
./exampleB2b
```

- Edit and inspect B2DetectorConstruction and B2bChamberParameterisation classes

```
cd $HOME/geant4/B2b
ls -F
ls -F include
ls -F src
atom */B2DetectorConstruction* */B2bChamberPar*
```

G4VPVParameterisation : example 1/3

```
class ChamberParameterisation : public G4VPVParameterisation
{
  public:
   ChamberParameterisation();
   virtual ~ChamberParameterisation();
   virtual void ComputeTransformation // position, rotation
     (const G4int copyNo, G4VPhysicalVolume* physVol) const;
   virtual void ComputeDimensions // size
     (G4Box& trackerLayer, const G4int copyNo,
      const G4VPhysicalVolume* physVol) const;
   virtual G4VSolid* ComputeSolid // shape
     (const G4int copyNo, G4VPhysicalVolume* physVol);
   virtual G4Material* ComputeMaterial // material, sensitivity, visAtt
     (const G4int copyNo, G4VPhysicalVolume* physVol,
      const G4VTouchable *parentTouch=0);
       // G4VTouchable should not be used for ordinary parameterization
};
```

G4VPVParameterisation : example – 2/3

```
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
  G4double Zposition = … // w.r.t. copyNo
  G4ThreeVector origin(Xposition,Yposition,Zposition);
  physVol->SetTranslation(origin);
  physVol->SetRotation(0);
}

void ChamberParameterisation::ComputeDimensions
(G4Box& trackerChamber, const G4int copyNo,
 const G4VPhysicalVolume* physVol) const
{
  G4double XhalfLength = … // w.r.t. copyNo
  G4double rmax = … // w.r.t. copyNo
      :
  trackerChamber.SetOuterRadius(rmax);
      :
}
```

G4VPVParameterisation : example – 3/3

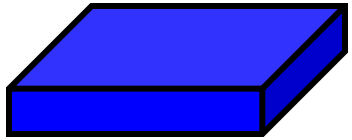
```
G4VSolid* ChamberParameterisation::ComputeSolid
      (const G4int copyNo, G4VPhysicalVolume* physVol)
{
  G4VSolid* solid;
  if(copyNo == …) solid = myBox;
  else if(copyNo == …) solid = myTubs;
  …
  return solid;
}

G4Material* ComputeMaterial // material, sensitivity, visAtt
      (const G4int copyNo, G4VPhysicalVolume* physVol,
       const G4VTouchable *parentTouch=0);
{
  G4Material* mat;
  if(copyNo == …)
  {
    mat = material1;
    physVol->GetLogicalVolume()->SetVisAttributes( att1 );
  }
  …
  return mat;
}
```

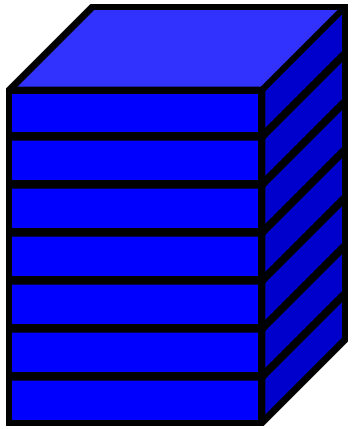
Physical Volumes: replicas

Replicated Volumes

- The mother volume is completely filled with replicas, all of which are the same size (width) and shape
- Replication may occur along:
  - Cartesian axes (X, Y, Z) – slices are considered perpendicular to the axis of replication
    - Coordinate system at the center of each replica
  - Radial axis (Rho) – cons/tubs sections centered on the origin and un-rotated
    - Coordinate system same as the mother
  - Phi axis (Phi) – phi sections or wedges, of cons/tubs form
    - Coordinate system rotated such as 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.);
```

- `offset` may be used only for tube/cone segment
- Features and restrictions:
  - Replicas can be placed inside other replicas
  - Normal placement volumes can be placed inside replicas, assuming no intersection/overlaps with the mother volume or with other replicas
  - No volume can be placed inside a radial replication
  - Parameterised volumes cannot be placed inside a replica

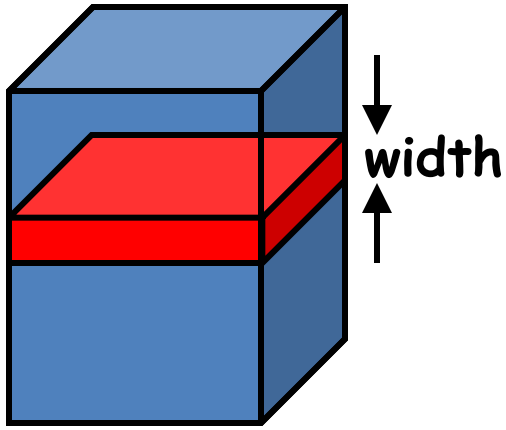
Replica - axis, width, offset

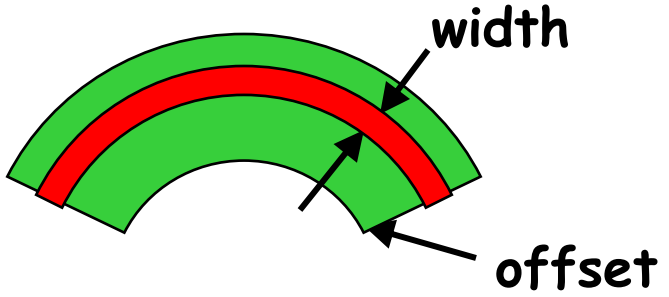
- Cartesian axes - **`kXaxis, kYaxis, kZaxis`**
  - Center of n-th daughter is given as

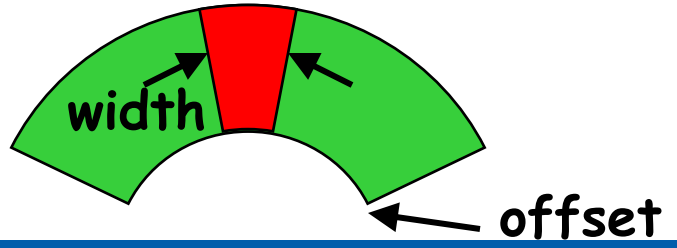
    `-width*(nReplicas-1)*0.5+n*width`
  - Offset shall not be used
- Radial axis - **`kRaxis`**
  - Center of n-th daughter is given as

    `width*(n+0.5)+offset`
  - Offset must be the inner radius of the mother
- Phi axis - **`kPhi`**
  - Center of n-th daughter is given as

    `width*(n+0.5)+offset`
  - Offset must be the starting angle of the mother

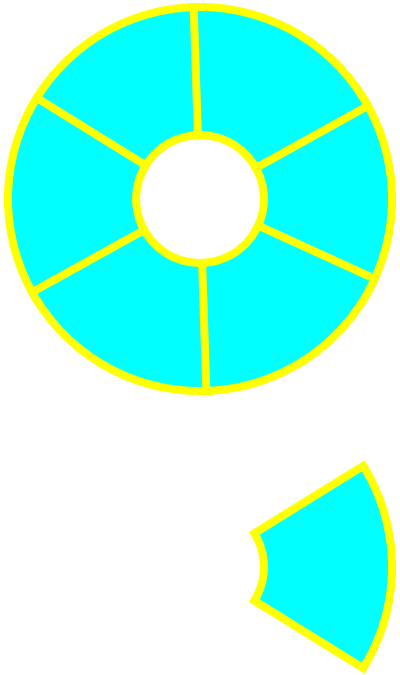






G4PVReplica : example

```
G4double tube_dPhi = 2.* M_PI * rad;
G4VSolid* tube =
   new G4Tubs("tube",20*cm,50*cm,30*cm,0.,tube_dPhi);
G4LogicalVolume * tube_log =
   new G4LogicalVolume(tube, Air, "tubeL", 0, 0, 0);
G4VPhysicalVolume* tube_phys =
   new G4PVPlacement(0,G4ThreeVector(-200.*cm,0.,0.),
             "tubeP", tube_log, world_phys, false, 0);
G4double divided_tube_dPhi = tube_dPhi/6.;
G4VSolid* div_tube =
   new G4Tubs("div_tube", 20*cm, 50*cm, 30*cm,
        -divided_tube_dPhi/2., divided_tube_dPhi);
G4LogicalVolume* div_tube_log =
   new G4LogicalVolume(div_tube,Pb,"div_tubeL",0,0,0);
G4VPhysicalVolume* div_tube_phys =
   new G4PVReplica("div_tube_phys", div_tube_log,
   tube_log, kPhi, 6, divided_tube_dPhi);
```



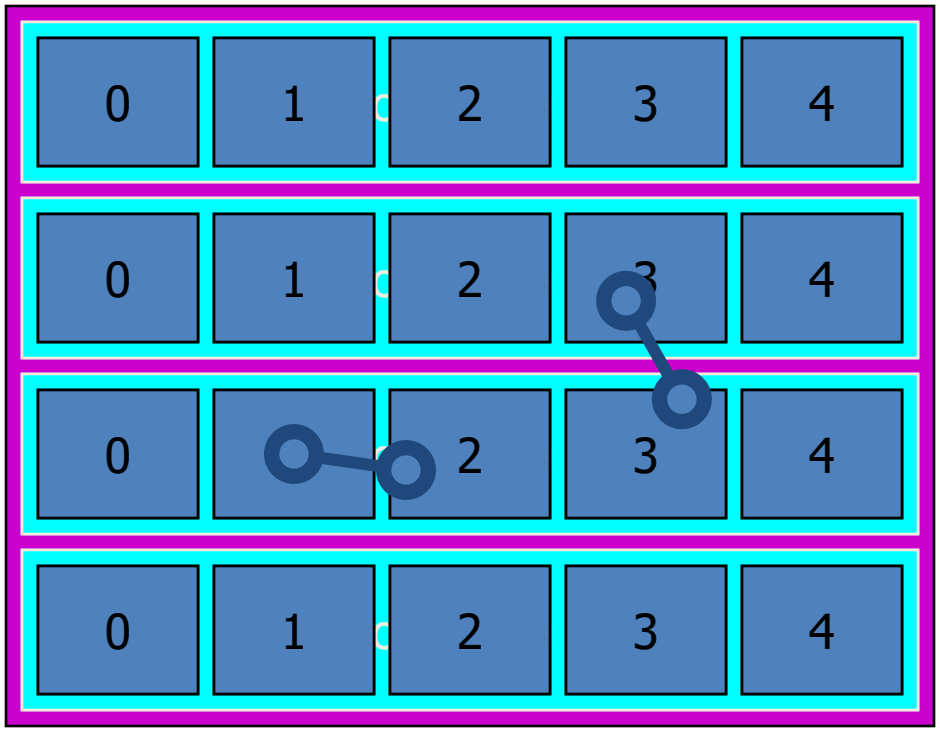
Touchables

Step Point & Touchable

- A track in Geant4 is composed of steps; **`G4Step`** has two **`G4StepPoint`** objects as its starting and ending points. All the geometrical information of the particular step should be taken from the "PreStepPoint"
  - The geometrical information associated with G4Track is identical to the "PostStepPoint"
- Each **`G4StepPoint`** object provides:
  - Position in world coordinate system
  - Global and local time
  - Material
  - **`G4TouchableHistory`** for geometrical information
- The **`G4TouchableHistory`** object is a vector of information for each geometrical hierarchy, including:
  - copy number
  - transformation / rotation to its mother
- *Handles* (or *smart-pointers*) to touchables are intrinsically used in Geant4. Touchables are reference counted objects

Copy number

- Suppose a calorimeter is made of 4x5 cells
  - and it is implemented by two levels of replica
- In reality, there is only one physical volume object for each level. Its position is parameterised by its copy number
- To get the copy number of each level, suppose what happens if a step belongs to two cells



  - Remember geometrical information in **`G4Track`** is identical to "PostStepPoint"
  - You cannot get the correct copy number for "PreStepPoint" if you directly access to the physical volume
- Use touchable to get the proper copy number, transform matrix, etc.

How to get information from a touchable?

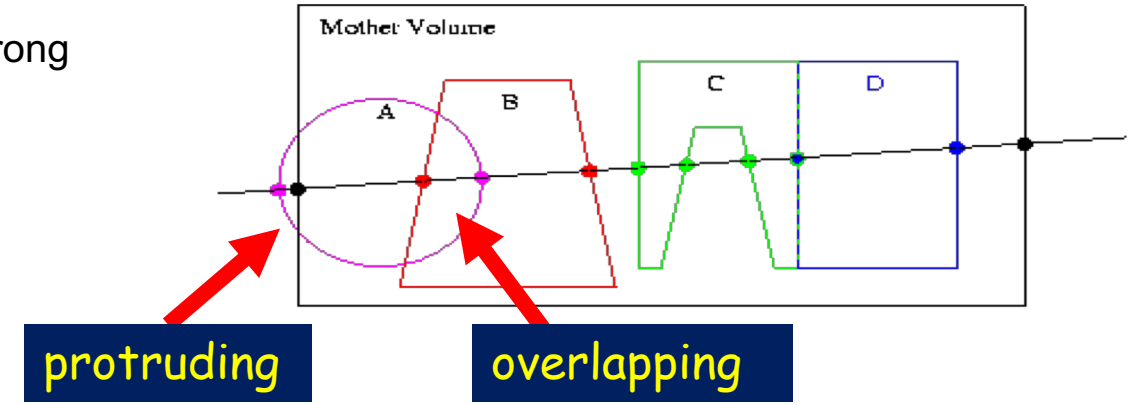
G4TouchableHistory has information on the geometrical hierarchy of the point

```
G4Step* aStep;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHistory* theTouchable =
    (G4TouchableHistory*)(preStepPoint->GetTouchable());
G4int copyNo = theTouchable->GetVolume()->GetCopyNo();
G4int motherCopyNo
            = theTouchable->GetVolume(1)->GetCopyNo();
G4int grandMotherCopyNo
            = theTouchable->GetVolume(2)->GetCopyNo();
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()
    ->GetTopTransform().TransformPoint(worldPos);
```

Geometry checking tools

Debugging geometries

- A protruding volume is a contained daughter volume which actually protrudes from its mother volume
- Volumes are also often positioned in a same volume with the intent of not provoking intersections between themselves. When volumes in a common mother actually intersect themselves are defined as overlapping
- Geant4 does not allow for malformed geometries, neither protruding nor overlapping
  - The behavior of navigation can be unpredictable in such cases
- The problem of detecting overlaps between volumes is bounded by the complexity of the solids model description
- Utilities are provided for detecting wrong positioning:
  - Optional checks at construction
  - Kernel run-time commands
  - Graphical tools (vis commands, DAVID)



Optional checks at construction

- 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 (10000 by default) are randomly sampled on the surface when creating the volume
  - Each of these points are examined
    - If it is outside of the mother volume, or
    - If it is inside of already existing other volumes in the same mother volume
- This may require lots of CPU time depending on the geometry complexity, it is therefore suggested to apply it to portions of a setup progressively
- Alternatively, one can use explicitly the overlaps check for a simple volume:

```
G4bool CheckOverlaps(G4int res=1000, G4double tol=0., G4bool verbose=true)
```

Debugging run-time commands

- Built-in run-time commands to activate verification tests for the user geometry are defined
  - to start verification of geometry for overlapping regions recursively through the volumes tree:

    `geometry/test/run`
  - To set the starting depth level in the volumes tree from where checking for overlaps. Default is level '0' (i.e. the world volume):

    `geometry/test/recursion_start [int]`
  - To set the total depth in the volume tree for checking for overlaps. Default is '-1' (i.e. checking the whole tree). Recursion will stop after having reached the specified depth":

    `geometry/test/recursion_depth [int]`
  - To define tolerance by which overlaps should not be reported. Default is '0':

    `geometry/test/tolerance [double] [unit]`
  - To set verbosity mode. Default is 'true':

    `geometry/test/verbosity [bool]`
  - To establish the number of points on surface to be generated and checked for each volume. Default is '10000':

    `geometry/test/resolution [int]`
  - To fix the threshold for the number of errors to be reported for a single volume. By default, for each volume, reports stop after the first error reported:

    `geometry/test/maximum_errors [int]`

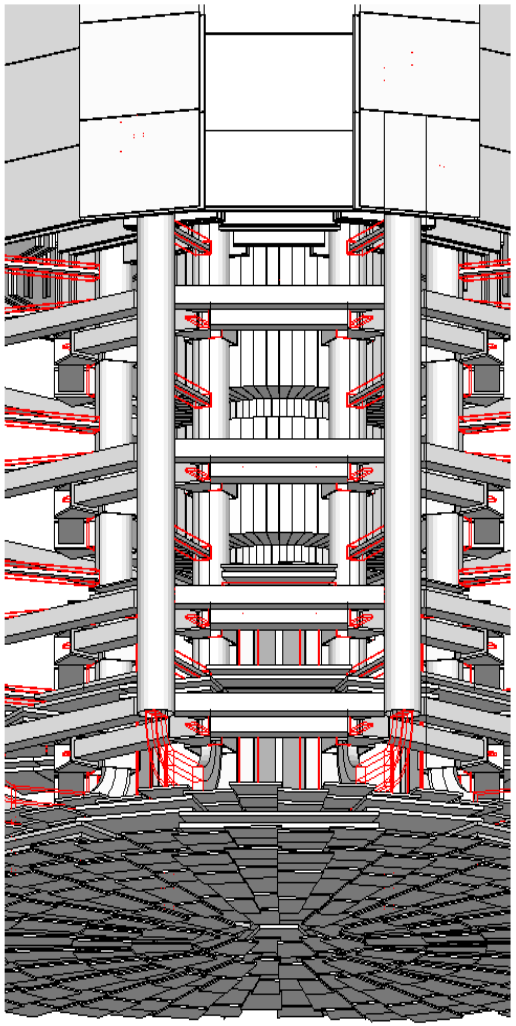
Debugging run-time reports

Example layout:

```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    240           -240      -145.5      -145.5     0       -145.5      -145.5
Which in the mother coordinate system are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Tracker[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Overlap[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
```

Debugging tools through visualization

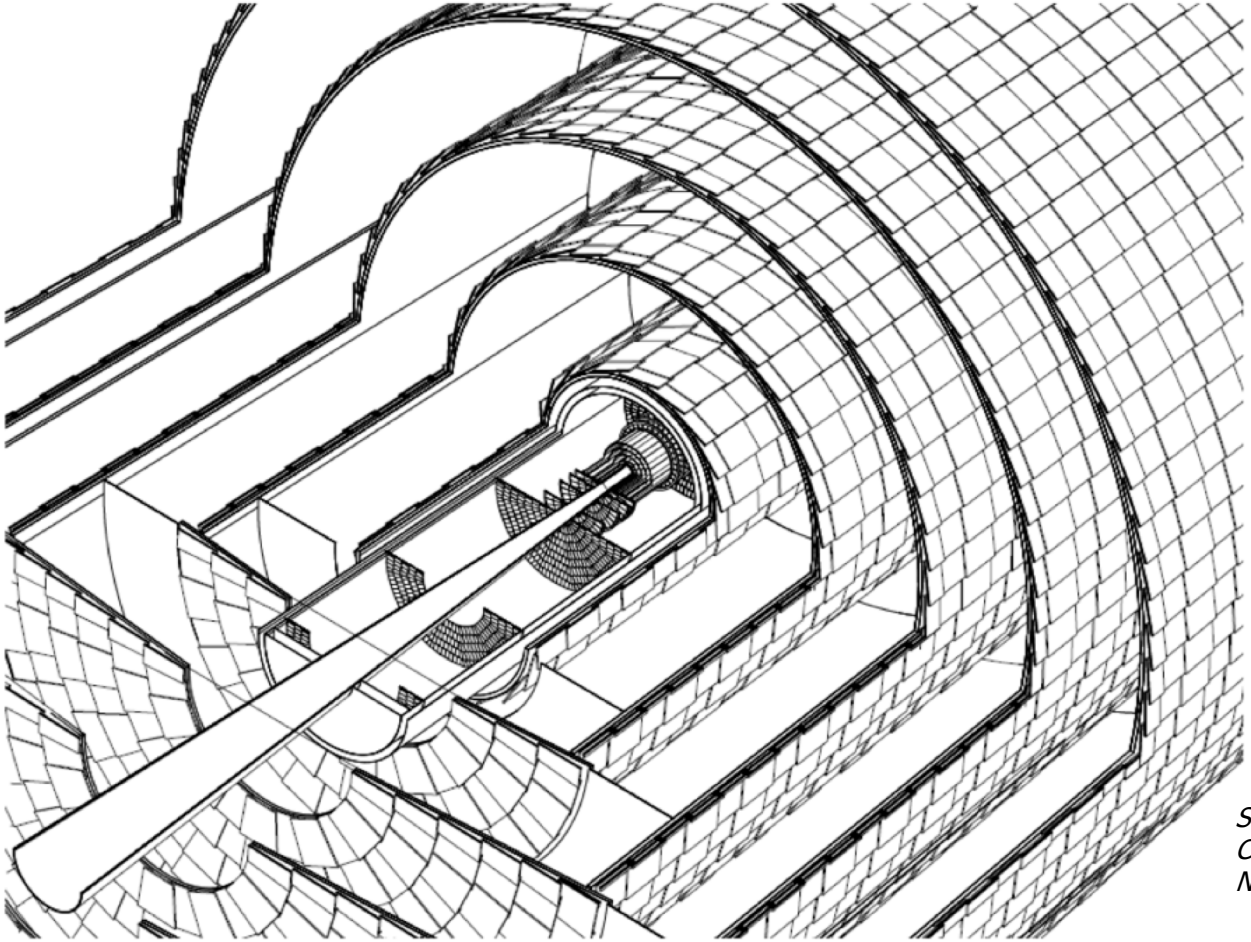
- DAVID is a graphical debugging tool for detecting potential intersections of volumes
  - Accuracy of the graphical representation can be tuned to the exact geometrical description.
  - physical-volume surfaces are automatically decomposed into 3D polygons
  - intersections of the generated polygons are parsed.
  - If a polygon intersects with another one, the physical volumes associated to these polygons are highlighted in color (red is the default).
  - DAVID can be downloaded from the Web as external tool for Geant4
    - http://geant4.kek.jp/~tanaka/
- Since release 10.5 visualization UI commands are available for dumping overlapping data
  - Will be soon integrated in the visualization system, to graphically display intersections



GDML persistency

GDML: importing geometries from files

- A geometry can be defined at runtime by providing a file-based detector description
- GDML defines a standard for describing detector geometries persistently to/from files



Silicon Pixel & Microstrip Tracker for Collider Detector
N. Graf, J. McCormick, LCDD, SLAC

GDML : Geometrical Description Markup Language

- An XML-based language designed as an application independent persistent format for describing the geometries of detectors.
  - Implements “geometry trees” which correspond to the hierarchy of volumes a detector geometry can be composed of
  - Allows materials to be defined and solids to be positioned
- Because it is pure XML, GDML can be used universally
  - Not just for Geant4
  - Can be format for interchanging geometries among different applications.
  - Can be used to translate CAD geometries to Geant4
- XML is simple
  - Rigid set of rules, self-describing data validated against schema
- XML is extensible
  - Easy to add custom features, data types
- XML is Interoperable to OS’s, languages, applications
- XML has hierarchical structure
  - Appropriate for Object-Oriented programming
  - Detector/sub-detector relationships

Importing a GDML file

- GDML files can be directly imported into Geant4 geometry, using the GDML plug-in facility:

```
#include "G4GDMLParser.hh"
```

- Generally you will want to put the following lines into your DetectorConstruction class:

```
G4GDMLParser parser;
parser.Read("geometryFile.gdml");
G4VphysicalVolume* W=parser.GetWorldVolume();
```

- The GDML parser also allows to export geometry to file
- To include the Geant4 module for GDML:
  - Install the XercesC parser (version 3.2.2 or higher)

    http://xerces.apache.org/xerces-c/
  - Set appropriate CMake configuration flag **`GEANT4_USE_GDML`** when configuring Geant4 and specify path to the XercesC library installation
- Examples available in:

  examples/extended/persistency/gdml

Magnetic Fields: basics