Geant 4

Detector Description - Basics

Anton Lechner, CERN

Acknowledgements:
Slides produced by J. Apostolakis, G. Cosmo, M. Asai, A. Howard

http://cern.ch/geant4

Introduction

Basic concepts of setting up a detector geometry

Detector Description

- Derive your own concrete class from the G4VUserDetectorConstruction abstract base class.
- Implementing the method Construct():
  - Modularize it according to each detector component or sub-detector:
    - Construct all necessary materials
    - Define shapes/solids required to describe the geometry
    - Construct and place volumes of your detector geometry
    - Define sensitive detectors and identify detector volumes which to associate them (optional)
    - Associate magnetic field to detector regions (optional)
    - Define visualization attributes for the detector elements (optional)
    - Define regions (optional)

Example:

```
G4VUserDetectorConstruction {
public:
  virtual G4VPhysicalVolume* Construct() = 0;
....
};
```

pure virtual method

```
// header file: MyDetectorConstruction.hh

#include "G4VUserDetectorConstruction.hh"

class MyDetectorConstruction : public G4VUserDetectorConstruction {

 public:
   ...

    G4VPhysicalVolume* Construct();  // Construct() must return the pointer of
                                     // the world physical volume
  ...
};

// source file: MyDetectorConstruction.cc

G4VPhysicalVolume* MyDetectorConstruction::Construct() {
   ...
   // Setup your detector here (as shown in the following)
}
```

Example (cont.):

```
//  main() of your Geant4 application:

#include "G4RunManager.hh"
#include "MyDetectorConstruction.hh"

... // other header files

int main() {

  G4RunManager* runManager = new G4RunManager;

  G4VUserDetectorConstruction* detector =  new MyDetectorConstruction();

  runManager -> SetUserInitialization(detector);

  ... // instantiate other obligatory initialization (G4VUserPhysicsList) and action
    //  classes (G4VUserPrimaryGeneratorAction) and assign them to run manag.

  ... // instantiate other optional action classes (G4UserRunAction,
    // G4UserEventAction,G4UserStackingAction, G4UserSteppingAction,
    // G4UserTrackingAction) and assign them to run manager
}
```

obligatory initialization class

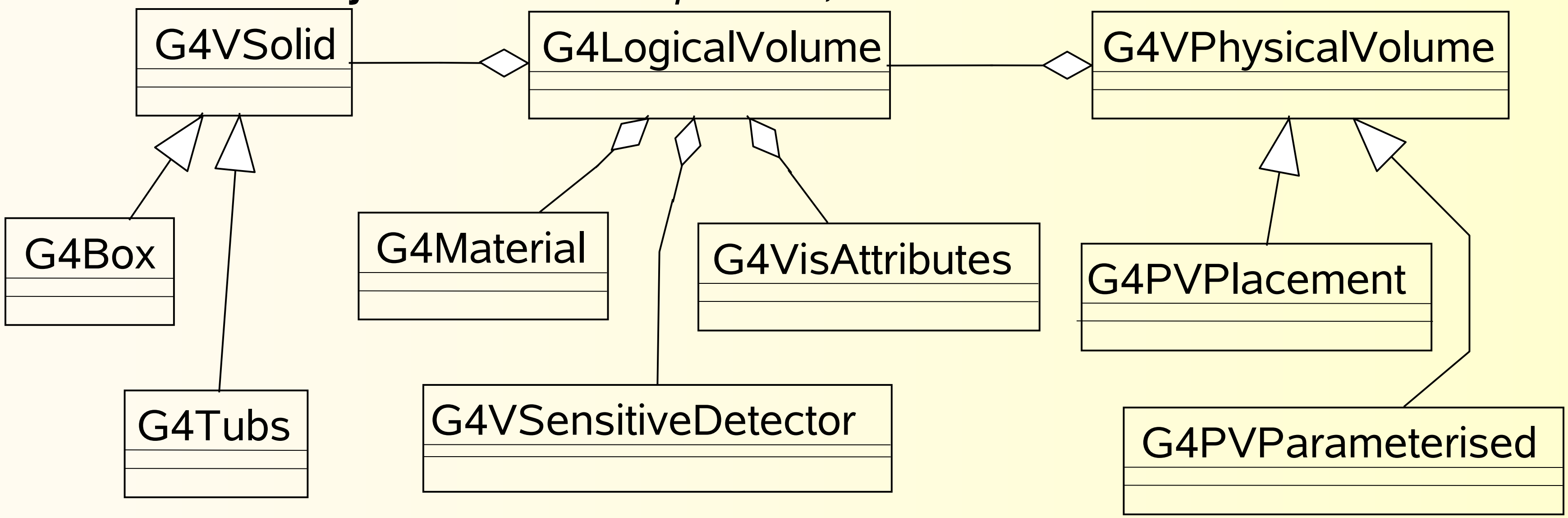
Creating a Detector Volume

- Start with its Shape & Size
  - Box 3x5x7 cm, sphere R=8m
- Add properties:
  - material, B/E field,
  - make it sensitive
- Place it in another volume
  - in one place
  - repeatedly using a function

- *Solid*
- *Logical-Volume*
- *Physical-Volume*

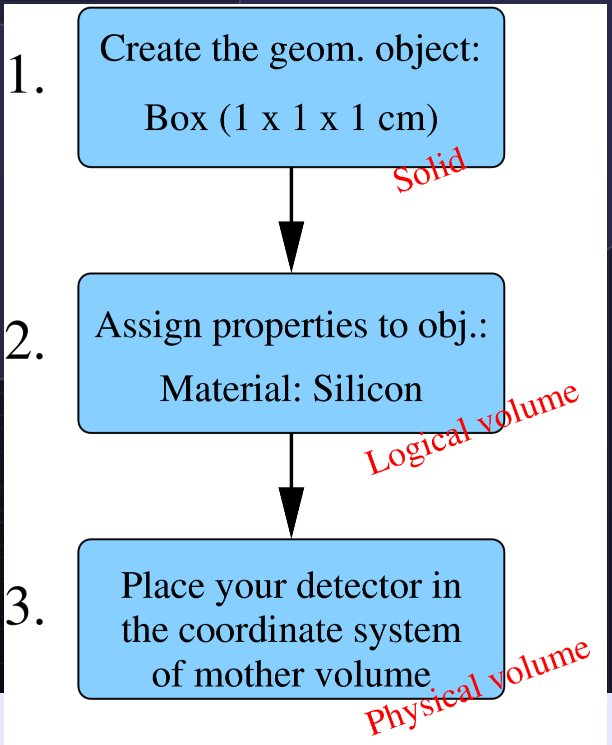
Defining the Detector Geometry

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



Example:

- Assume, your detector is a cube
  - size: 1 x 1 x 1 cm
  - material: silicon




```
G4VSolid* boxSolid =  new G4Box( “aBoxSolid“, 1.0 * cm, 1.0 * cm, 1.0 * cm);


G4LogicalVolume* boxLog =
    new  G4LogicalVolume( boxSolid, matSilicon, “aBoxLog“, 0, 0, 0);
```

create a material as shown in the previous presentation

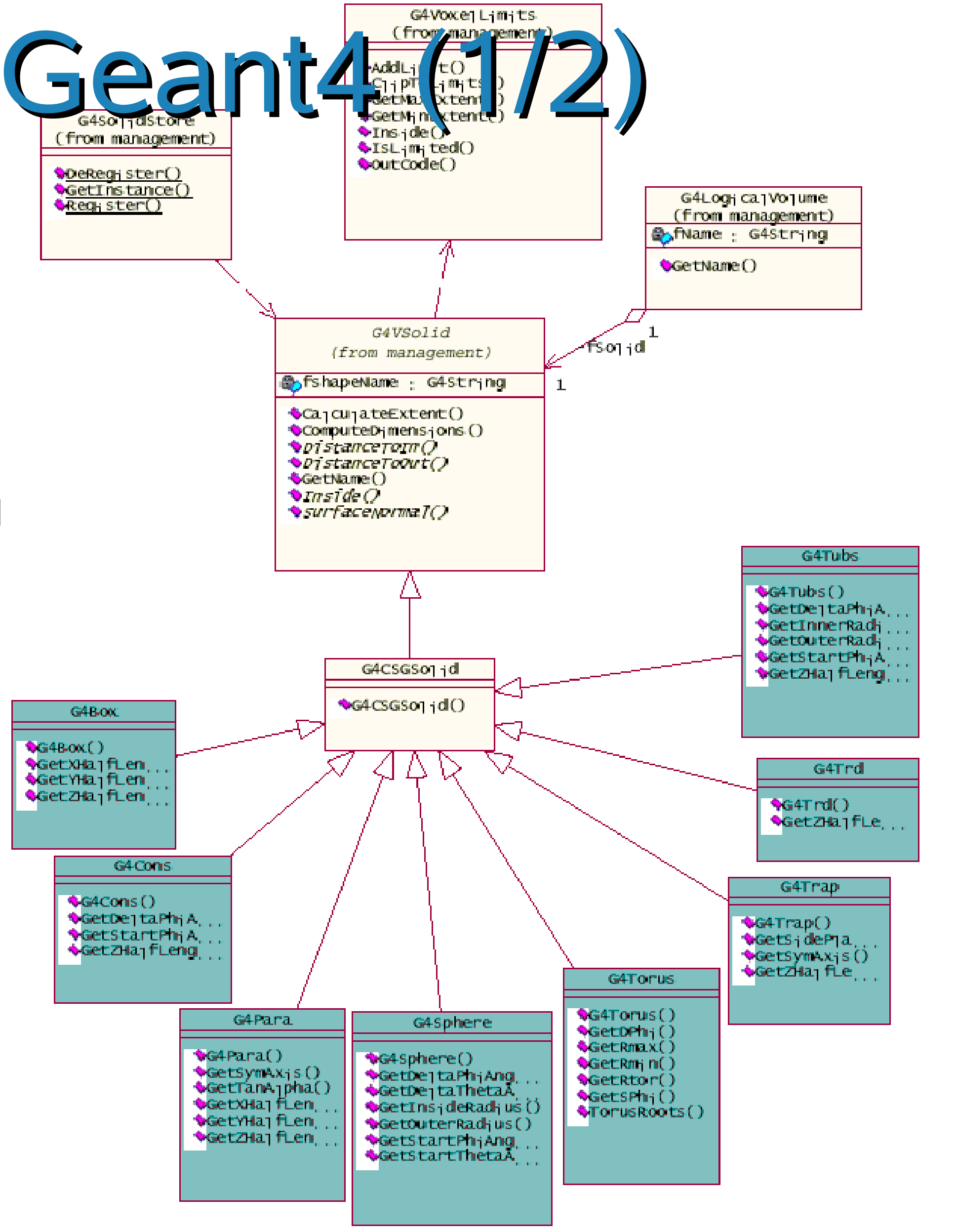
```
G4VPhysicalVolume* boxPhys =
    new  G4PVPlacement( rotation, G4ThreeVector(posX, posY, posZ), boxLog,
                        “aBoxPhys“, motherLog, 0, copyNo);
```

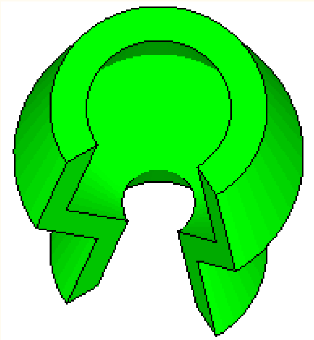
Solids

Geometrical objects

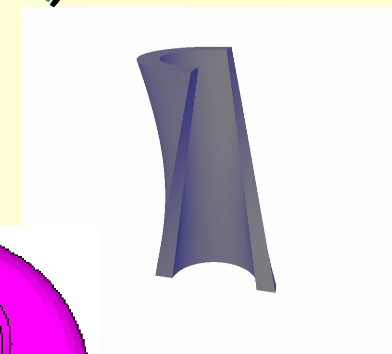
Solids in Geant4 (1/2)

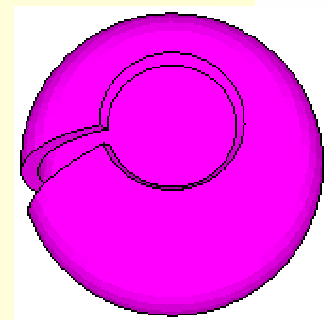
- All solids derive from the abstract base class G4VSolid
- Once constructed, each solid is automatically registered in a specific solid store

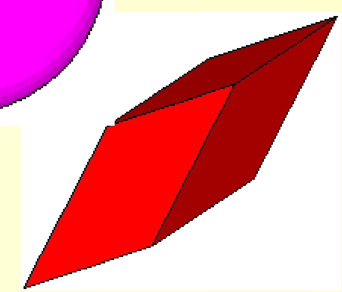




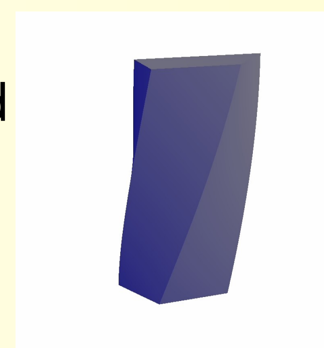
Solids in Geant4 (2/2)

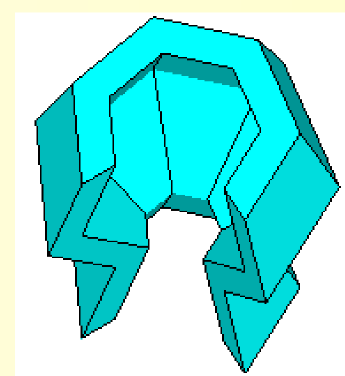




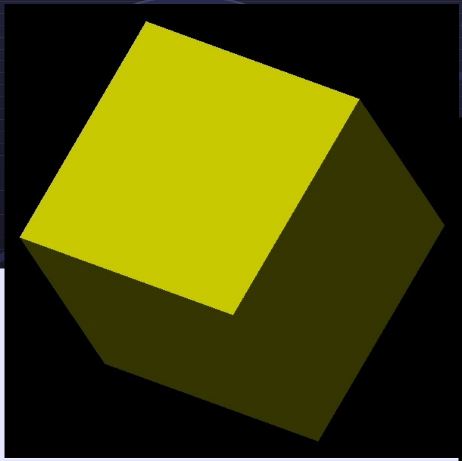


- CSG (Constructed Solid Geometry):
  - G4Box, G4Tubs, G4Cons, G4Trd, ...
  - Analogous to simple G3 CSG solids
- Specific solids (CSG like):
  - G4Polycone, G4Polyhedra, G4Hype, ...
  - G4TwistedTubs, G4TwistedTrap,...
- BREP (Boundary REPresented) and Boolean solids
  - See presentation on advanced geometries

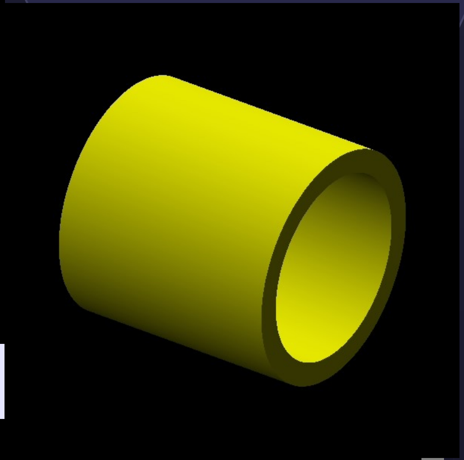


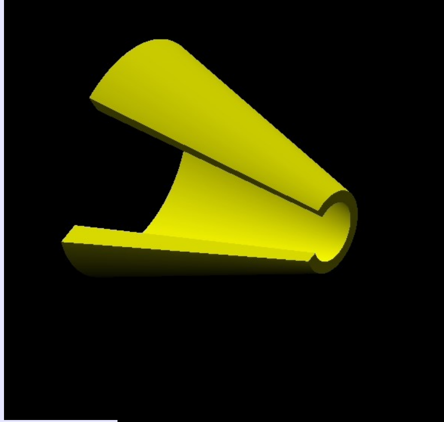


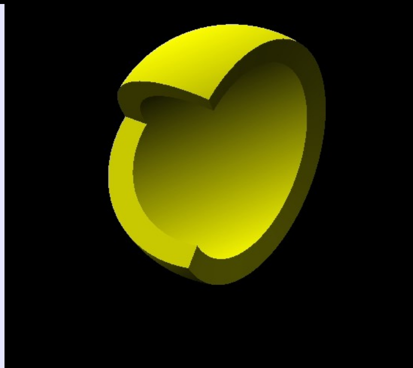
Examples (CSG):



```
G4VSolid* boxSolid =
   new G4Box( “aBoxSolid“,
              1.0 * cm, 1.0 * cm, 1.0 * cm);

G4VSolid* tubeSolid =
   new G4Tubs("aTubeSolid",
              1.6 * cm,    // inner radius
              2.0 * cm,    // outer radius
              2.0 * cm,    // height
              0.0 * deg,  360.0 * deg);  // segment angles

G4VSolid* coneSolid =
   new G4Cons( “aConeSolid”,
              0.5 * cm, 0.7 * cm, // inner & outer radius 1
              1.6 * cm, 2.0 * cm, // inner & outer radius 2
              2.0 * cm,           // height
              0.0 * deg,  285. *deg);  // segment angles

G4VSolid* sphereSolid =
    new G4Sphere(“aSphereSolid “,
              1.6 * cm, 2.0 * cm,       // inner & outer radius
              0.0 * deg, 270.0 *deg,  // segment angles phi
              0.0 * deg, 100.0 *deg); // segment angles theta
```







Example (specific CSG):

```
G4int nmbRZ = 10;

G4double r[] = {0.0 * mm, 2.0 * mm, 2.0 * mm, 3.0 * mm, 3.0 * mm,
                1.0 * mm, 1.0 * mm, 3.0 * mm, 3.0 * mm, 0.0 * mm};

G4double z[] = {0.5 * mm, 0.5 * mm, 0.0 * mm, 0.0 * mm, 2.0 * mm,
                5.0 * mm, 8.0 * mm, 11.0 * mm, 13 * mm, 13.0 * mm};

G4VSolid* polyconeSolid =
    new  G4Polycone("aPolyconeSolid",
                    0.0 * deg,      // start angle phi
                    360.0 * deg,  // total angle phi
                    nmbRZ, // Numbers of corners in the r,z space
                    r,          //  r-coordinates of corners
                    z);         //  z-coordinates of corners
```

Logical Volumes

Volumes with attributes: Material, Sensitivity, ...

Logical Volumes (1/2)

- A logical volume contains all information of the volume except position and rotation:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Magnetic field, User Limits
  - Shower parameterisation
  - Position of daughter volumes
  - Region
- Physical volumes of the same type can share a logical volume

Logical Volumes (2/2)

- To create a logical volume for a given material and solid, the user must instantiate G4LogicalVolume:

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

- Note:
  - The pointers to solid and material must be NOT null
  - Once created it is automatically entered in the LV store
  - It is not meant to act as a base class

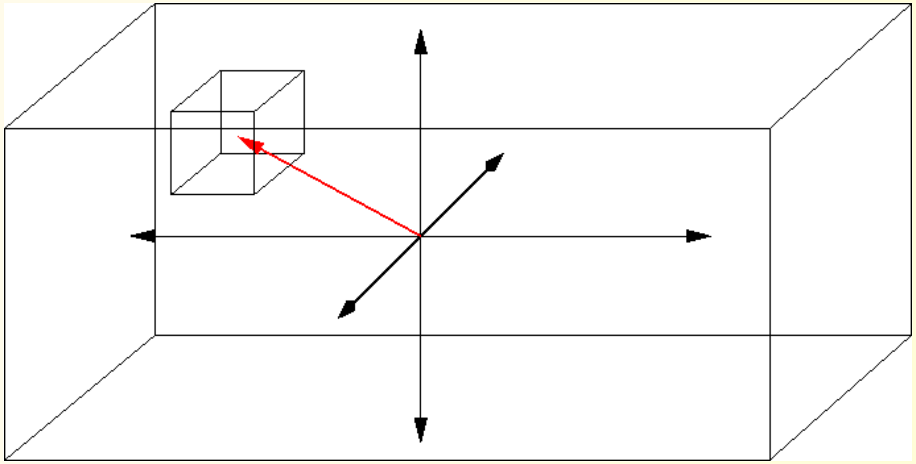
Physical Volumes

Placing Logical Volumes

- Physical volumes are placed instances of logical volumes
  - One logical volume can be placed more than once
  - Several techniques can be used:
    - Single placement
    - Repeated placement (Replicas, parametrization, ...)
  - Volumes are part of a geometrical hierarchy:
    - Volumes always have a mother volume (except the root volume)
    - Volumes may have several daughter volumes
- G4VPhysicalVolume is the base class of physical volumes
  - Use the inherited classes to place your logical volumes

Geometrical hierarchy (1/2)

- Mother and daughter volumes
  - 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
  - One or more volumes can be placed to a mother volume

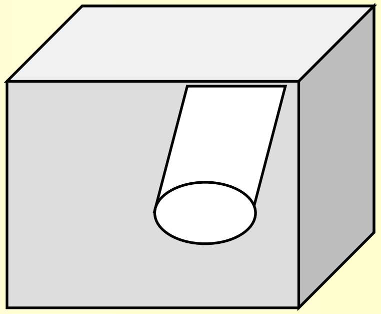


Geometrical hierarchy (2/2)

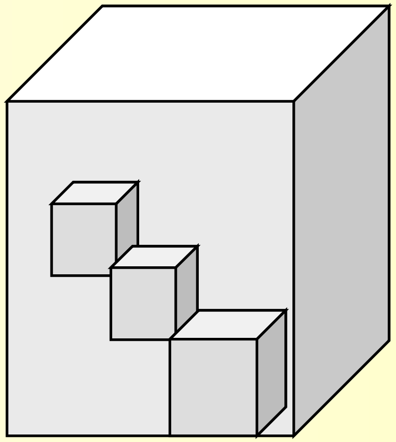
- Mother and daughter volumes (cont.)
  - The *logical volume* of the mother knows the physical volumes it contains
    - It is uniquely defined to be their mother volume
    - If the logical volume of the mother is placed more than once, all daughters appear by definition in all these physical instances of the mother
- World volume = root volume of the hierarchy
  - The world volume must be a unique physical volume which fully contains all other volumes
    - The world defines the global coordinate system
    - The origin of the global coordinate system is at the center of the world volume
    - The position of a track is given with respect to the global coordinate system
  - The most simple shape to describe the world is a box

Physical Volumes

- Placement: it is one positioned volume
  - One physical volume represents one “real” volume
- Repeated: a volume placed many times
  - On physical volume <u>represents</u> any number of “real” volumes
  - reduces use of memory
  - <u>Replica</u> and <u>Division</u>
    - simple repetition along one axis
  - <u>Parameterised</u>
    - repetition with respect to copy number
- A **mother** volume can contain **either**
  - **many placement** volumes **<u>OR</u>**
  - **one repeated** volume



placement



repeated

G4VPhysicalVolume

- **G4PVPlacement** 1 Placement = One Volume
  - A volume instance positioned once in a mother volume
- **G4PVParameterised** 1 Parameterised = Many Volumes
  - Parameterised by the copy number
    - Shape, size, material, position and rotation can be parameterised, by implementing a concrete class of G4VPVParameterisation.
  - Reduction of memory consumption
    - Currently: parameterisation can be used only for volumes that either a) have no further daughters <u>or</u> b) are identical in size & shape.
- **G4PVReplica** 1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)

Single Physical Volume (1/2)

- To place a single instance of a logical volume in a mother volume use G4PVPlacement:

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

- The volume is positioned in a frame, which is rotated and translated relative to the mother volume

Single Physical Volume (2/2)

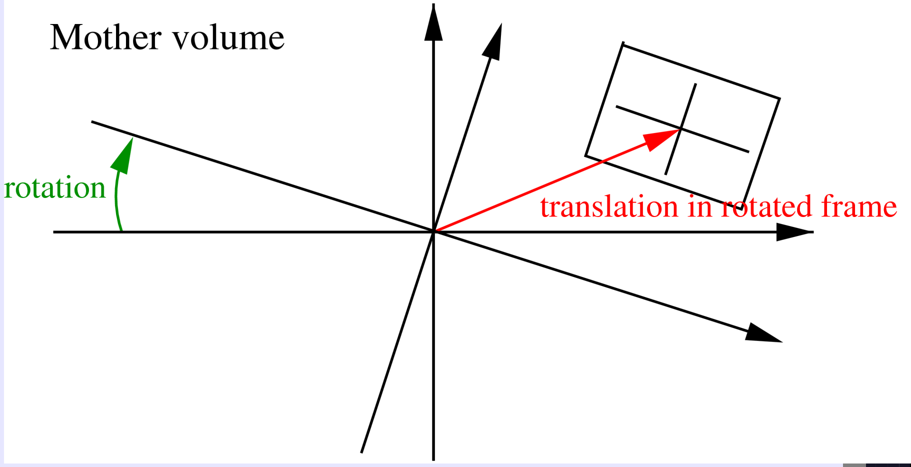
- Alternative constructor for G4PVPlacement: Use G4Transform3D to represent the direct rotation and translation of the solid instead of the frame

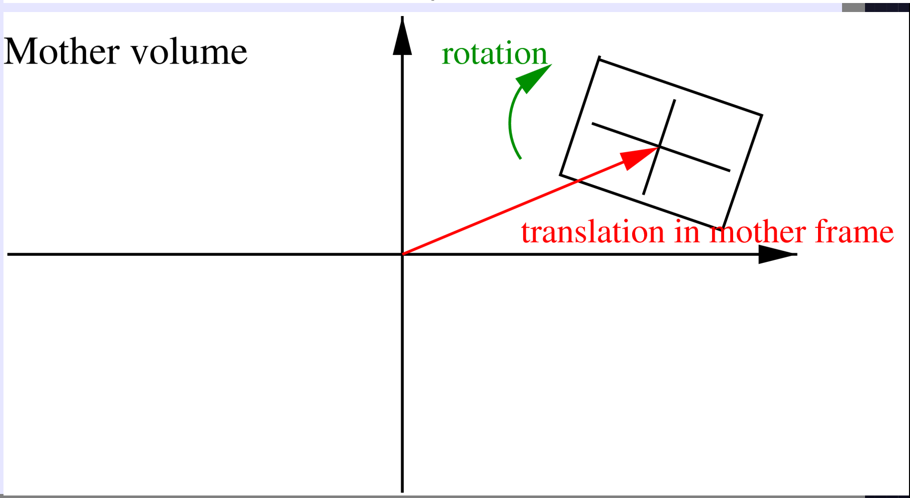
```
G4PVPlacement(G4Transform3D(G4RotationMatrix& pRot,
                            const G4ThreeVector& tlate),
              G4LogicalVolume* pCurrentLogical,
              const G4String& pName,
              G4LogicalVolume* pMotherLogical,
              G4bool pMany,
              G4int pCopyNo);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

- Two additional constructors are available:
  - Same arguments as for the two previous constructors, except that you can specify the mother volume by its pointer to the physical volume instead of its logical volume

Example:

```
G4RotationMatrix* yRot = new G4RotationMatrix; // Rotates X and Z axes only
yRot -> rotateY(M_PI/4.*rad);                  // Rotates 45 degrees

G4ThreeVector zTrans(0, 0, 5 * cm);

// Constructor 1:
G4VPhysicalVolume* boxPhys =
    new  G4PVPlacement( yRot, zTrans,
                        boxLog,
                        “aBoxPhys“,
                        motherLog,
                        0, copyNo);
// Constructor 2:
G4VPhysicalVolume* boxPhys =
    new  G4PVPlacement(
        G4Transform3D(*yRot, zTrans),
        boxLog,
        “aBoxPhys“,
        motherLog,
        0, copyNo);
```





Parameter. Physical Volume

- Using G4PVParamterised, one can place a logical volume multiple times, where position and dimension are parametrised w.r.t. the copy number:

```
G4PVParameterised( const G4String& pName,
                   G4LogicalVolume* pCurrentLogical,
                   G4LogicalVolume* pMotherLogical,
                   const EAxis pAxis,
                   const G4int nReplicas,
                   G4VPVParameterisation* pParam);
```

see next page

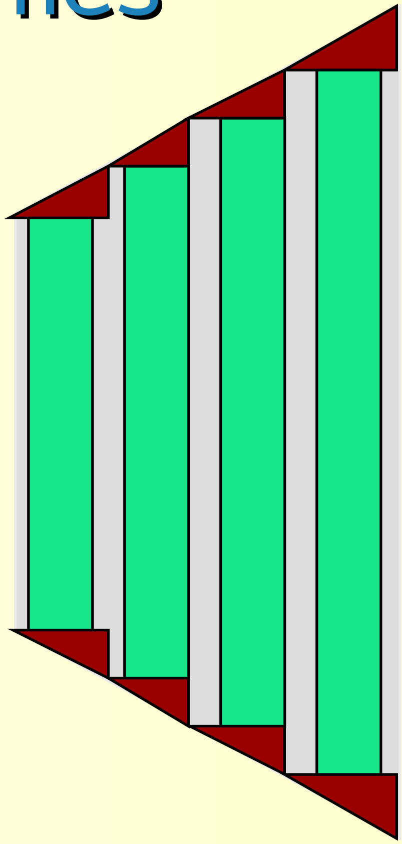
- Note:
  - Replicates the volume nReplicas times (nReplicas touchables differing in position and size).
  - The positioning of the replicas is dominant along the specified Cartesian axis (if kUndefined is used, 3D voxelisation for optimisation of the geometry is adopted).

Parameter. Physical Volume

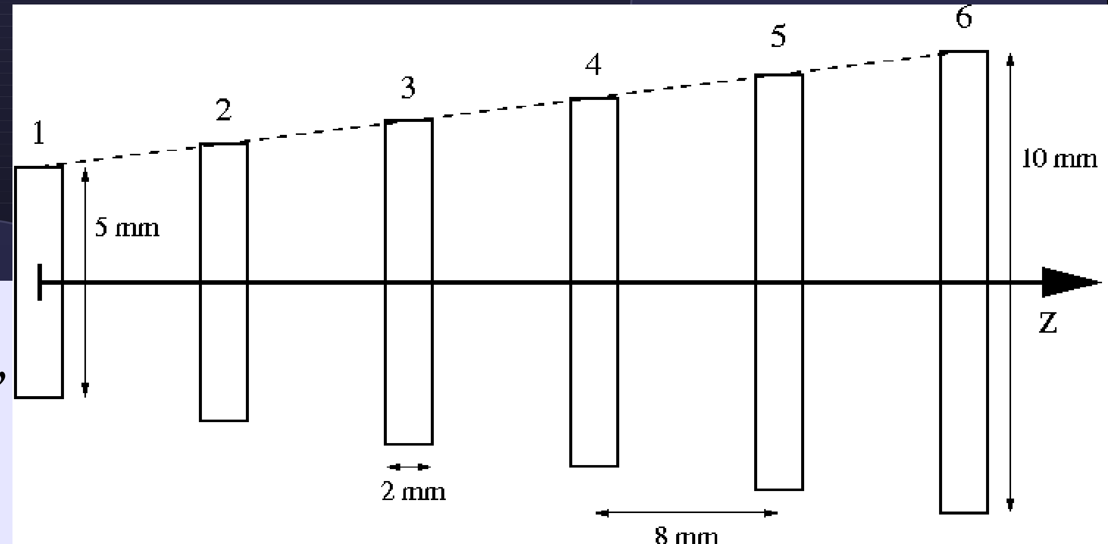
- The dimension and position of a replica for a given copy number are calculated by means of the concrete user implementation of G4VPVParameterisation:
  - The user must implement the methods:
    - ComputeDimensions() (for calculating the dimensions)
    - ComputeTransformations() (for calculating the position)
  - Optional methods are:
    - ComputeSolid() (for the type of solid)
    - ComputeMaterial() (for the material)
- Limitations:
  - For simple CSG solids only
  - Daughter volumes only allowed in special cases

Uses of Parameterised Volumes

- Complex detectors
  - with large repetition of volumes
    - regular or irregular
- Medical applications
  - the material in animal tissue is measured
    - cubes with varying material



Example:




```
G4VSolid* chamberSolid =
   new G4Box( “chamberSolid“,
                          1. * cm,
                          1. * cm,
                          1. * cm);

G4LogicalVolume* chamberLog =
     new G4LogicalVolume(chamberSolid, chamberMat, “chamberLog“,0,0,0);

G4VPVParameterisation* chamberParam =
             new ChamberParameterisation(6,            // Number of chambers
                                       0.0 * mm, // Z of centre of first
                                       8.0 * mm, // Z spacing of centres
                                       2.0 * mm, // Width chamber
                                       5.0 * mm, // length first chamber
                                       10.0 * mm); // length last chamber

G4VPhysicalVolume* chamberPhys =
             new G4PVParameterised(“chamber”, chamberLog, motherLog,
                                  kZAxis,
                                  6,         // Number of chambers
                                  chamberParam);
```

User defined class, see next slide

Example:

```
class ChamberParameterisation : public G4VPVParameterisation {

public:
      ChamberParameterisation(G4int NoChambers,
                              G4double startZ,
                              G4double spacingZ,
                              G4double widthChamber,
                              G4double lengthFirstChamber,
                              G4double lengthLastChamber);
      ~ChamberParameterisation();

      void ComputeTransformation(const G4int copyNo,
                                 G4VPhysicalVolume* physVol) const;

      void ComputeDimension(G4Box& chamber,
                            const G4int copyNo
                            const G4VPhysicalVolume* physVol) const;

  private:
       ... // data members
};
```

Values initialize corresp. data members,which are used in the Compute... methods

Example:

```
void ChamberParameterisation::ComputeTransformation(
                    const G4int copyNo,
                    G4VPhysicalVolume* physVol) const {

  G4double positionZ = fStartZ + (copyNo + 1) * fSpacingZ;
  G4ThreeVector origin(0, 0, positionZ);
  physVol -> SetTranslation(origin);
  physVol -> SetRotation(0);
}

void ChamberParameterisation::ComputeDimension(
                                        G4Box& chamber,
                                        const G4int copyNo,
                                        const G4VPhysicalVolume* physVol) const{

   G4double fLengthChamber = fLengthFirstChamber + copyNo *
               ((fLengthLastChamber - fLengthFirstChamber) / fNoChambers);

   chamber.SetXHalfLength(fLengthChamber * 0.5);
   chamber.SetYHalfLength(fLengthChamber * 0.5);
   chamber.SetZHalfLength(fWidthChamber * 0.5);
}
```