Geant4 Geometry

Slides kindly prepared by Makoto Asai (SLAC) and slightly reorganized by Sebastien Incerti (CNRS)

KIT Tutorial, October 25-26, 2011, Karlsruhe
Introduction
Detector geometry

- Three conceptual layers
  - G4VSolid -- shape, size
  - G4LogicalVolume -- daughter physical volumes, material, sensitivity, user limits, magnetic field, visualization attributes, 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);

• 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 daughters are by definition to appear in all of mother physical volumes.
- The world volume must be a unique physical volume which fully contains all the 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.
G4VUserDetectorConstruction
User classes

- **main()**
  - Geant4 does not provide *main()*.
  
  Note: classes written in red are mandatory.

- **Initialization** classes
  - Use G4RunManager::SetUserInitialization() to define in *main()*
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList

- **Action** classes
  - Use G4RunManager::SetUserAction() to define in *main()*
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction
Construct() should return the pointer of the world physical volume. The world physical volume represents all of your geometry setup.
Describe your detector

• Derive your own concrete class from `G4VUserDetectorConstruction` abstract base class.

• Implement the method `Construct()`
  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 volumes *(optional)*
  7) Define visualization attributes for the detector elements *(optional)*
  8) Define regions *(optional)*

• Set your construction class to `G4RunManager`

• It is suggested to modularize `Construct()` method w.r.t. each component or sub-detector for easier maintenance of your code.
1) Shape of volume
G4VSolid

- Abstract class. All solids in Geant4 are derived from it.
- It defines but does 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

- Solids defined in Geant4:
  - CSG (Constructed Solid Geometry) solids
    - G4Box, G4Tubs, G4Cons, G4Trd, ...
    - Analogous to simple GEANT3 CSG solids
  - Specific solids (CSG like)
    - G4Polycone, G4Polyhedra, G4Hype, ...
  - BREP (Boundary REPresented) solids
    - G4BREPSolidPolycone, G4BSplineSurface, ...
    - Any order surface
  - Boolean solids
    - G4UnionSolid, G4SubtractionSolid, ...
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

G4Torus

G4Sphere

G4Orb (full solid sphere)

G4Para (parallelepiped)

Consult Section 4.1.2 of Geant4 Application Developers Guide for all available shapes.
Specific CSG Solids: G4Polycone

G4Polycone(const G4String& pName,
            G4double phiStart,
            G4double phiTotal,
            G4int numRZ,
            const G4double r[],
            const G4double z[]);

• **numRZ** - numbers of corners in the \( r, z \) space
• **\( r, z \)** - coordinates of corners
Other Specific CSG solids

- G4Polyhedra
- G4Tet (tetrahedra)
- G4Hype
- G4EllipticalTube
- G4Ellipsoid
- G4EllipticalCone
- G4TwistedBox
- G4TwistedTrap
- G4TwistedTrd
- G4TwistedTubs

Consult Section 4.1.2 of Geant4 Application Developers Guide for all available shapes.
**BREP Solids**

- **BREP = Boundary REPresented Solid**
- Listing all its surfaces specifies a solid
  - e.g. 6 planes for a cube
- Surfaces can be
  - planar, 2nd or higher order
    - elementary BREPS
  - Splines, B-Splines,
    - NURBS (Non-Uniform B-Splines)
    - advanced BREPS
- Few elementary BREPS pre-defined
  - box, cons, tubs, sphere, torus, polycone, polyhedra
- Advanced BREPS built through CAD systems
Boolean Solids

- Solids can be combined using boolean operations:
  - `G4UnionSolid`, `G4SubtractionSolid`, `G4IntersectionSolid`
  - Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2\textsuperscript{nd} solid
  - 2\textsuperscript{nd} solid is positioned relative to the coordinate system of the 1\textsuperscript{st} solid
  - Result of boolean operation becomes a solid. Thus the third solid can be combined to the resulting solid of first 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
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 the same as those of the first solid.
Tessellated solids

- **G4TessellatedSolid** (since 8.1)
  - 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
  - But 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 - release 8.1
2) Logical Volume
G4LogicalVolume

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

• Contains all information of 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 a common logical volume object.
• The pointers to solid must NOT be null.
• The pointers to material must NOT be null for tracking geometry.
• It is not meant to act as a base class.
Computing volumes and weights

- Geometrical volume of a generic solid or boolean composition can be computed from the **solid volume**:
  
  \[
  \text{G4double GetCubicVolume();}
  \]
  
  - Exact volume is determinatively calculated for most of CSG solids, while estimation based on Monte Carlo integration is given for other solids.

- Overall weight of a geometry setup (sub-geometry) can be computed from the **logical volume**:
  
  \[
  \text{G4double GetMass(G4bool forced=false, G4bool propagate=true, G4Material* pMaterial=0);}\]
  
  - The computation may require a considerable amount of time, depending on the complexity of the geometry.
  
  - The return value is cached and reused until \texttt{forced}=true.
  
  - Daughter volumes will be neglected if \texttt{propagate}=false.
Logical volume
Region
Region

• A region may have its unique
  – Production thresholds (cuts)
    • If a region in the mass geometry does not have its own production thresholds, those of the default region are used (i.e., may not be those of the parent region).
  – User limits
    • Artificial limits affecting to the tracking, e.g. max step length, max number of steps, min kinetic energy left, etc.
    • You can set user limits directly to logical volume as well. If both logical volume and associated region have user limits, those of logical volume wins.
  – User region information
    • E.g. to implement a fast Boolean method to identify the nature of the region.
  – Fast simulation manager
  – Regional user stepping action
  – Field manager
• Please note :
  – World logical volume is recognized as the default region. User is not allowed to define a region to the world logical volume.
Root logical volume

- A logical volume can be a region. More than one logical volumes may belong to a region.
- A region is a part of the geometrical hierarchy, i.e. a set of geometry volumes, typically of a sub-system.
- A logical volume becomes a root logical volume once a region is assigned to it.
  - All daughter volumes belonging to the root logical volume share the same region, unless a daughter volume itself belongs to another root.
- Important restriction:
  - No logical volume can be shared by more than one regions, regardless of root volume or not.
G4Region

- A region is instantiated and defined by
  
  ```
  G4Region* aRegion = new G4Region("region_name");
  aRegion->AddRootLogicalVolume(aLogicalVolume);
  ```
  
  - Region propagates down to all geometrical hierarchy until the bottom or another root logical volume.
- Production thresholds (cuts) can be assigned to a region by
  
  ```
  G4Region* aRegion
  = G4RegionStore::GetInstance()->GetRegion("region_name");
  G4ProductionCuts* cuts = new G4ProductionCuts;
  cuts->SetProductionCut(cutValue);
  aRegion->SetProductionCuts(cuts);
  ```
G4Region class

- G4Region class may take following quantities:
  - `void SetProductionCuts(G4ProductionCuts* cut);`
  - `void SetUserInformation(G4VUserRegionInformation* uri);`
  - `void SetUserLimits(G4UserLimits* ul);`
  - `void SetFastSimulationManager(G4FastSimulationManager* fsm);`
  - `void SetRegionalSteppingAction(G4UserSteppingAction* rusa);`
  - `void SetFieldManager(G4FieldManager* fm);`

- Please note:
  - If any of the above properties are not set for a region, properties of the world volume (i.e. default region) are used. Properties of mother region do not propagate to daughter region.
3) Physical volume
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
    - repetition w.r.t. copy number
  - **Replica and Division**
    - simple repetition along one axis
- A mother volume can contain either
  - many placement volumes
  - or, one repeated volume
Physical volume

- **G4PVPlacement**  1 Placement = One Placement Volume
  - A volume instance positioned once in its mother volume
- **G4PVParameterised**  1 Parameterized = Many Repeated Volumes
  - Parameterized by the copy number
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterized by the copy number.
    - You **have to** implement a concrete class of G4VPVParameterisation.
  - Reduction of memory consumption
  - Currently: parameterization can be used only for volumes that either
    a) have **no** further daughters, **or**
    b) are identical in size & shape (so that grand-daughters are safely fit inside).
  - By implementing G4PVNestedParameterisation instead of G4VPVParameterisation, material, sensitivity and vis attributes can be parameterized by the copy numbers of ancestors.
Physical volume

- **G4PVReplica**  
  1 Replica = Many Repeated Volumes  
  - Daughters of same shape are aligned along one axis  
  - Daughters fill the mother completely without gap in between.
- **G4PVDivision**  
  1 Division = Many Repeated Volumes  
  - Daughters of same shape are aligned along one axis and fill the mother.  
  - There can be gaps between mother wall and outmost daughters.  
  - No gap in between daughters.
- **G4ReflectionFactory**  
  1 Placement = a pair of Placement volumes  
  - generating placements of a volume and its reflected volume  
  - Useful typically for end-cap calorimeter
- **G4AssemblyVolume**  
  1 Placement = a set of Placement volumes  
  - Position a group of volumes
Physical volume
1) G4PVPlacement
G4PVPlacement

G4PVPlacement(G4RotationMatrix* pRot,  // rotation of mother frame
               const G4ThreeVector &tlate, // position in mother frame
               G4LogicalVolume *pDaughterLogical,
               const G4String &pName,
               G4LogicalVolume *pMotherLogical,
               G4bool pMany, // 'true' is not supported yet...
               G4int pCopyNo, // unique arbitrary integer
               G4bool pSurfChk=false); // optional boundary check

- Single volume positioned relatively to the mother volume.
Physical volume

2) **G4PVParameterised**
G4PVPParameterised

G4PVPParameterised(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 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
Parameterized 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)
- Optional:
  - 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:
  - Applies to simple CSG solids only
  - 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 is measured as cubes with varying material
Geometry checking tools
Debugging geometries

- An **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 they are defined as **overlapping**.
- Geant4 **does not allow** for malformed geometries, neither protruding nor overlapping.
  - 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)

![Diagram](image-url)

**protruding** **overlapping**
Optional checks at construction

- Constructors of `G4PVPlacement` and `G4PVParameterised` have an optional argument “pSurfChk”.
  ```cpp
  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.
  - 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 check requires lots of CPU time, but it is worth to try at least once when you implement your geometry of some complexity.
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 based on a standard grid setup, limited to the first depth level
    - `geometry/test/run` or `geometry/test/grid_test`
  - applies the grid test to all depth levels (may require lots of CPU time!)
    - `geometry/test/recursive_test`
  - shoots lines according to a cylindrical pattern
    - `geometry/test/cylinder_test`
  - to shoot a line along a specified direction and position
    - `geometry/test/line_test`
  - to specify position for the `line_test`
    - `geometry/test/position`
  - to specify direction for the `line_test`
    - `geometry/test/direction`
Debugging tools: DAVID

- **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 an external tool for Geant4.
Debugging tools: OLAP

- Stand-alone batch application
  - Provided as extended example
  - Can be combined with a graphical environment and GUI

**Geant4 Macro:**

```plaintext
/vis/scene/create
/vis/sceneHandler/create VRML2FILE
/vis/viewer/create
/olap/goto ECalEnd
/olap/grid 7 7 7
/olap/trigger
/vis/viewer/update
```

**Output:**

```
delta=59.3416
vol 1: point=(560.513,1503.21,-141.4)
vol 2: point=(560.513,1443.86,-141.4)
A -> B:
[0]: ins=[2] PVName=[NewWorld:0] Type=[N] ...
[1]: ins=[0] PVName=[ECalEndcap:0] Type=[N] ...
B -> A:
[0]: ins=[2] PVName=[NewWorld:0] Type=[N] ...
```

- red: mother
- blue: daughters
- daughters are protruding their mother
- NavigationHistories of points of overlap (including: info about translation, rotation, solid specs)
Material scanner

- Measures material thickness in units of geometrical length, radiation length and interaction length.
  - It can be region sensitive, so that you can measure the thickness of one particular region.
- `/control/matScan`
  - scan - Start material scanning.
  - theta - Define theta range.
  - phi - Define phi range.
  - singleMeasure - Measure thickness for one particular direction.
  - eyePosition - Define the eye position.
  - regionSensitive - Set region sensitivity.
  - region - Define region name to be scanned.
THANK YOU
To learn more
Your detector construction

```cpp
#ifndef MyDetectorConstruction_h
#define MyDetectorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"
class MyDetectorConstruction
  : public G4VUserDetectorConstruction
{
  public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();
    virtual G4VPhysicalVolume* Construct();
  public:
    // set/get methods if needed
  private:
    // granular private methods if needed
    // data members if needed
};
#endif
```
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, // ‘true’ is not supported yet...
    G4int pCopyNo, // unique arbitrary integer
    G4bool pSurfChk=false); // optional boundary check

• Single volume positioned relatively to the mother volume.
G4PVParameterized : example

G4VSolid* solidChamber =
    new G4Box("chamber", 100*cm, 100*cm, 10*cm);

G4LogicalVolume* logicChamber =
    new G4LogicalVolume
    (solidChamber, ChamberMater, "Chamber", 0, 0, 0);

G4VPVParameterisation* chamberParam =
    new ChamberParameterisation();

G4VPhysicalVolume* physChamber =
    new G4PVParameterised("Chamber", logicChamber,
        logicMother, kZAxis, NbOfChambers, chamberParam);
G4VPVParameterisation : example

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

```cpp
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Xposition = ... // 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
    trackerChamber.SetXHalfLength(XhalfLength);
    trackerChamber.SetYHalfLength(YhalfLength);
    trackerChamber.SetZHalfLength(ZhalfLength);
}
```
G4VPVParameterisation : example

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;
}
Divided volume
G4PVDivision

• G4PVDivision is a special kind of G4PVParameterised.
  – G4VPVP Parameterisation is automatically generated according to the parameters given in G4PVDivision.
• G4PVDivision is similar to G4PVR eplica but
  – It currently allows gaps in between mother and daughter volumes
  – We are extending G4PVDivision to allow gaps between daughters, and also gaps on side walls. We plan to release this extension in near future.
• Shape of all daughter volumes must be same shape as the mother volume.
  – G4VSolid (to be assigned to the daughter logical volume) must be the same type, but different object.
• Replication must be aligned along one axis.
• If your geometry does not have gaps, use G4Replica.
  – For identical geometry, navigation of G4Replica is faster.
G4PVDivision(const G4String& pName,  
   G4LogicalVolume* pDaughterLogical,  
   G4LogicalVolume* pMotherLogical,  
   const EAxis pAxis,  
   const G4int nDivisions, // number of division is given  
   const G4double offset);

• The size (width) of the daughter volume is calculated as
  
  \[
  \frac{(\text{size of mother}) - \text{offset}}{n\text{Divisions}}
  \]
G4PVDivision - 2

G4PVDivision(const G4String& pName,
        G4LogicalVolume* pDaughterLogical,
        G4LogicalVolume* pMotherLogical,
        const EAxis pAxis,
        const G4double width,   // width of daughter volume is given
        const G4double offset);

• The number of daughter volumes is calculated as
  int( ( (size of mother) - offset ) / width )
  – As many daughters as width and offset allow
G4PVDivision - 3

G4PVDivision(const G4String& pName,
             G4LogicalVolume* pDaughterLogical,
             G4LogicalVolume* pMotherLogical,
             const EAxis pAxis,
             const G4int nDivisions,
             const G4double width,  // both number of division and width are given
             const G4double offset);

• \textit{nDivisions} daughters of \textit{width} thickness
G4PVDivision

- G4PVDivision currently supports following shapes / axes.
  - G4Box : kXAxis, kYAxis, kZAxis
  - G4Tubs : kRho, kPhi, kZAxis
  - G4Cons : kRho, kPhi, kZAxis
  - G4Trd : kXAxis, kYAxis, kZAxis
  - G4Para : kXAxis, kYAxis, kZAxis
  - G4Polycone : kRho, kPhi, kZAxis
    - kZAxis - the number of divisions has to be the same as solid sections, (i.e. numZPlanes-1), the width will not be taken into account.
  - G4Polyhedra : kRho, kPhi, kZAxis
    - kPhi - the number of divisions has to be the same as solid sides, (i.e. numSides), the width will not be taken into account.
    - kZAxis - the number of divisions has to be the same as solid sections, (i.e. numZPlanes-1), the width will not be taken into account.
- In the case of division along kRho of G4Cons, G4Polycone, G4Polyhedra, if width is provided, it is taken as the width at the -Z radius; the width at other radii will be scaled to this one.
G4ReplicatedSlice

- New extension of G4Division introduced with version 9.4.
- It allows gaps in between divided volumes.

\[
\begin{align*}
\text{G4PVDivision}(& \text{const G4String& pName, G4LogicalVolume* pDaughterLogical,} \\
& \text{G4LogicalVolume* pMotherLogical, const EAxis pAxis,} \\
& \text{const G4int nDivisions, const G4double half_gap, const G4double offset);} \\
\text{G4PVDivision}(& \text{const G4String& pName, G4LogicalVolume* pDaughterLogical,} \\
& \text{G4LogicalVolume* pMotherLogical, const EAxis pAxis,} \\
& \text{const G4double width, const G4double half_gap, const G4double offset);} \\
\text{G4PVDivision}(& \text{const G4String& pName, G4LogicalVolume* pDaughterLogical,} \\
& \text{G4LogicalVolume* pMotherLogical, const EAxis pAxis,} \\
& \text{const G4int nDivisions, const G4double width,} \\
& \text{const G4double half_gap, const G4double offset);} \\
\end{align*}
\]
Physical volume

Replicated volume: G4PVReplica
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
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);
Debugging run-time commands

• 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)

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>start position (cm)</th>
<th>end position (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>-240</td>
<td>-145.5</td>
</tr>
<tr>
<td></td>
<td>-145.5</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-145.5</td>
<td>-145.5</td>
</tr>
</tbody>
</table>

Which in the mother coordinate system are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>start position (cm)</th>
<th>end position (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which in the coordinate system of Tracker[0] are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>start position (cm)</th>
<th>end position (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which in the coordinate system of Overlap[0] are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>start position (cm)</th>
<th>end position (cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>. . .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>