Geometry II

Geant4 Collaboration

KEK/CRC

Contents

Replicated volume

Parameterization

Advanced ways of placement

Region

Defining magnetic field

REPLICATED VOLUME

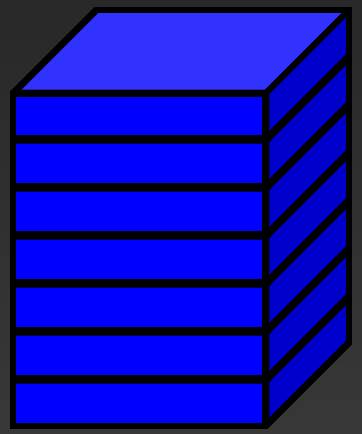
Replicated volumes

The mother volume is completely filled with replicas, all of which are the same size (width) and shape.

Replication is allowed 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

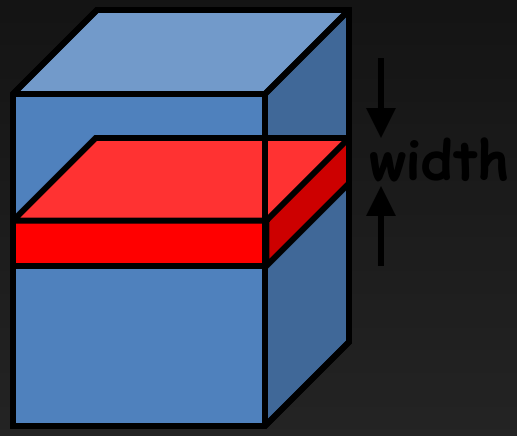
Replica - axis, width, offset

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

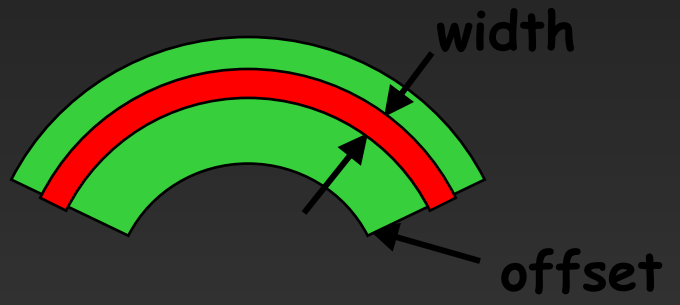


Radial axis - **`kRaxis`**

- Center of n-th daughter is given as

  `width*(n+0.5)+offset`

- Offset must be the inner radius of the mother

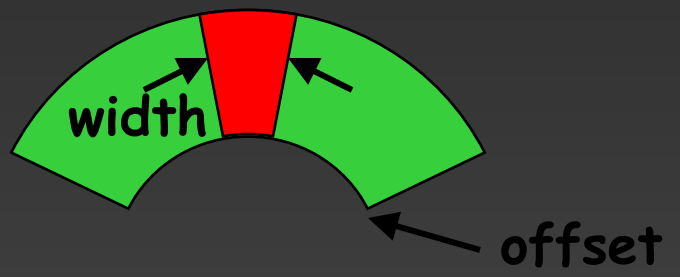


Phi axis - **`kPhi`**

- Center of n-th daughter is given as

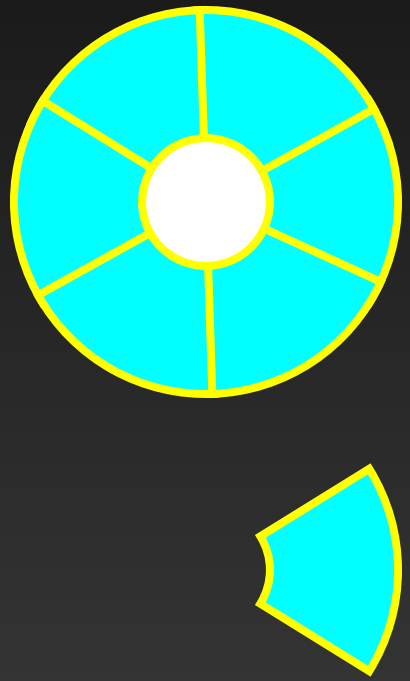
  `width*(n+0.5)+offset`

- 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);
```



PARAMETERIZED VOLUME

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 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

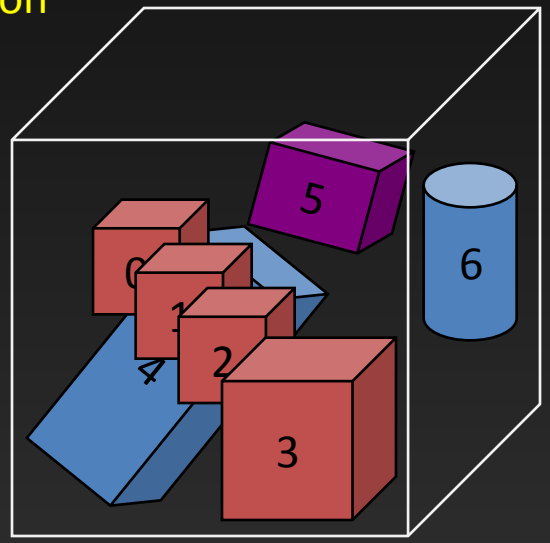
Parameterized Physical Volumes

User should implement a class derived from G4VPVParameterisation and define the followings 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



Limitations:

- Applies to simple CSG solids only
- Granddaughter volumes allowed only for special cases

Typical use-cases

- Complex detectors with large repetition of volumes, regular or irregular

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

```
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);
}
```

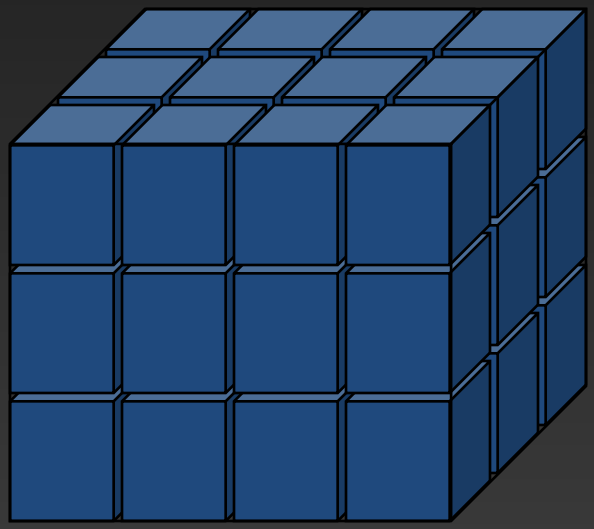
G4VPVParameterisation : example

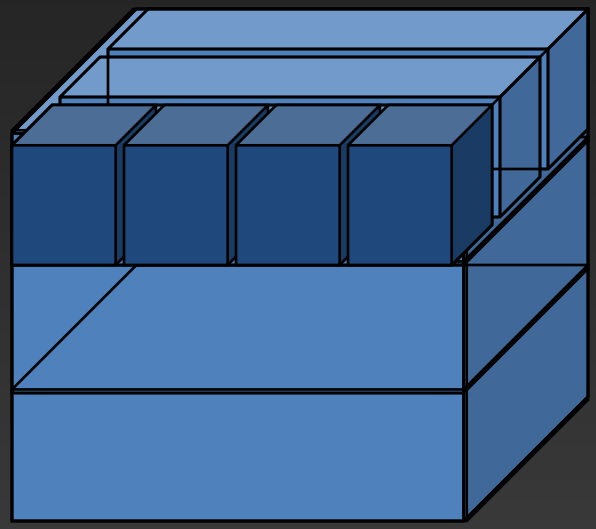
```
G4VSolid* ChamberParameterisation::ComputeSolid
     (const G4int copyNo, G4VPhysicalVolume* physVol)
{
  G4VSolid* solid;
  if(copyNo == …) solid = myBox;
  else if(copyNo == …) solid = myTubs;
  …
  return solid;
}

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;
}
```

Nested parameterization

- Suppose your geometry has three-dimensional regular reputation of same shape and size of volumes without gap between volumes. And material of such volumes are changing according to the position.
  - E.g. voxels made by CT Scan data (DICOM)
- Instead of direct three-dimensional parameterized volume,

  use replicas for the first and second axes sequentially, and then use one-dimensional parameterization along the third axis.





- It requires much less memory for geometry optimization and gives much faster navigation for ultra-large number of voxels.

Divisions

Assembly volumes

Reflected volumes

ADVANCED WAYS OF PLACEMENT

G4PVDivision

G4PVDivision is a special kind of G4PVParameterised.

- G4VPVParameterisation is automatically generated according to the parameters given in G4PVDivision.

G4PVDivision is similar to G4PVReplica but

- It allows gaps between mother and daughter volumes
- also allow gaps between daughters, and also gaps on side walls.

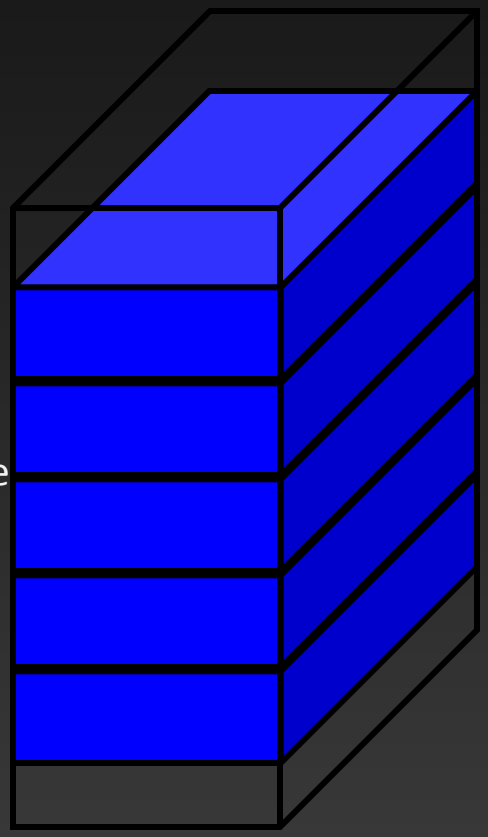
Shape of all daughter volumes must be same shape as the mother volume.

- 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.

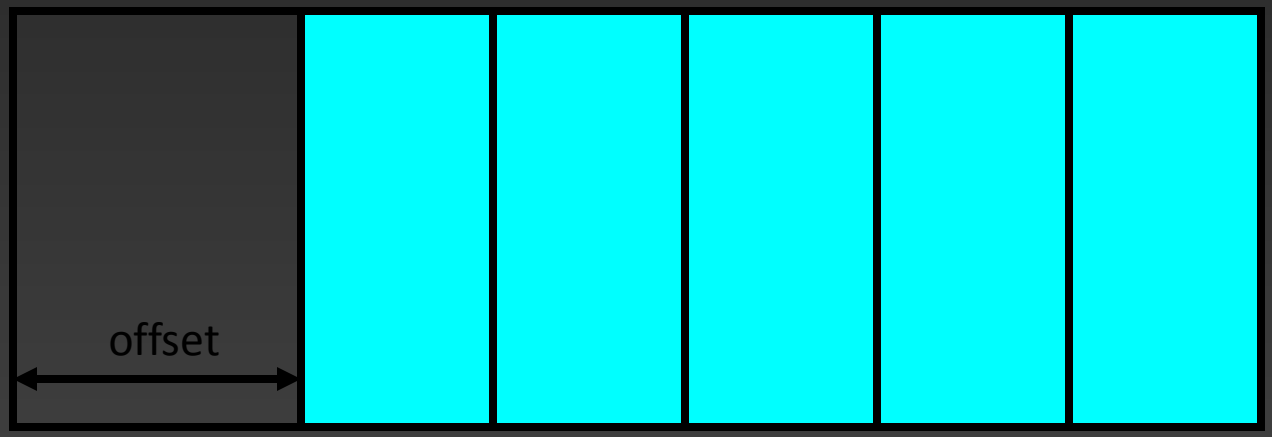


G4PVDivision - 1

```
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

```
( (size of mother) - offset ) / nDivisions
```



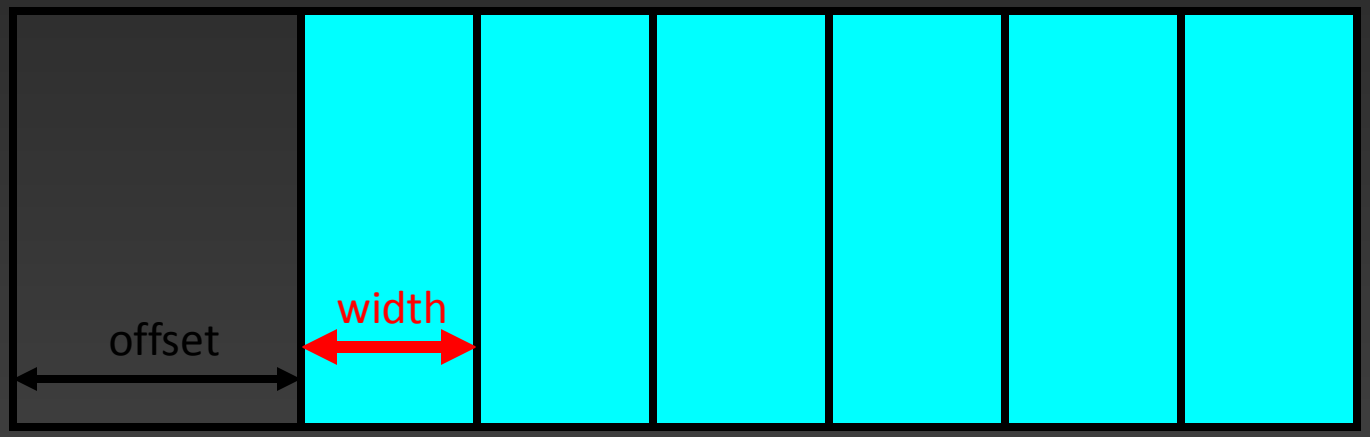
G4PVDivision - 2

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 )`

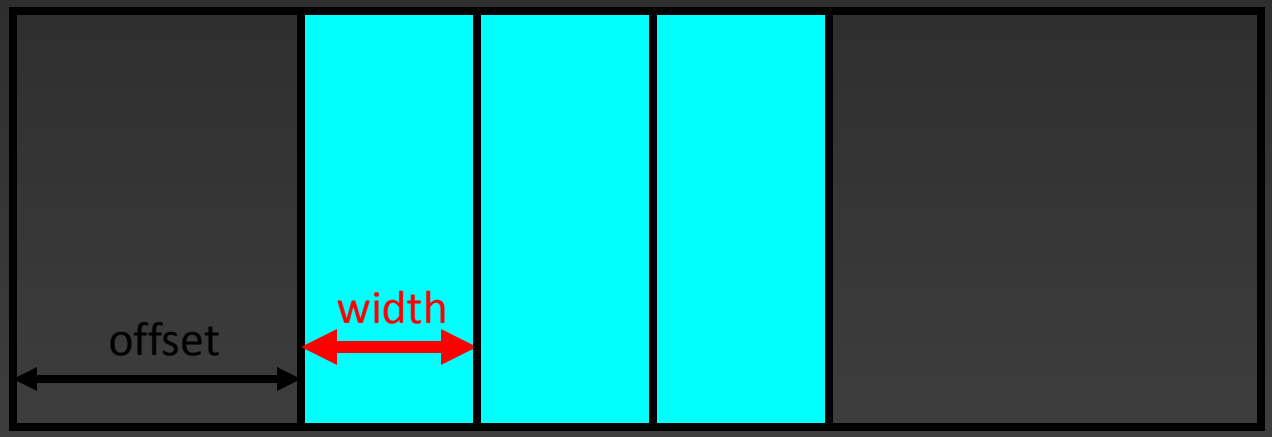
- As many daughters as width and offset allow



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);
```

nDivisions daughters of *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.

Grouping volumes

To represent a regular pattern of positioned volumes, composing a more or less complex structure

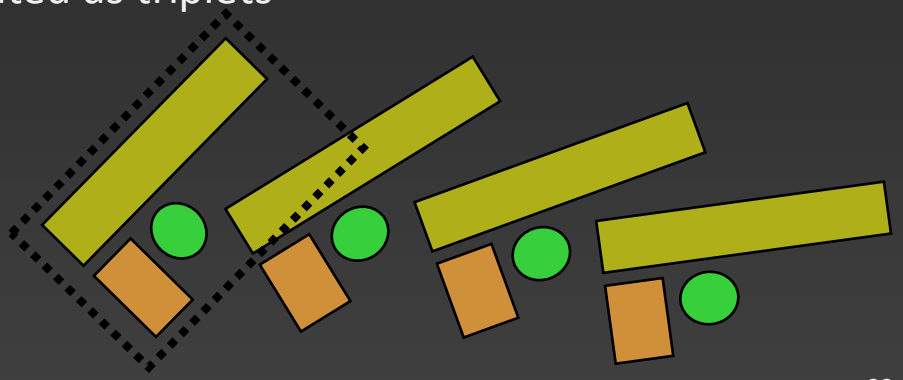
- structures which are hard to describe with simple replicas or parameterised volumes
- structures which may consist of different shapes
- Too densely positioned to utilize a mother volume

Assembly volume

- acts as an *envelope* for its daughter volumes
- its role is over once its logical volume has been placed
- daughter physical volumes become independent copies in the final structure

Participating daughter logical volumes are treated as triplets

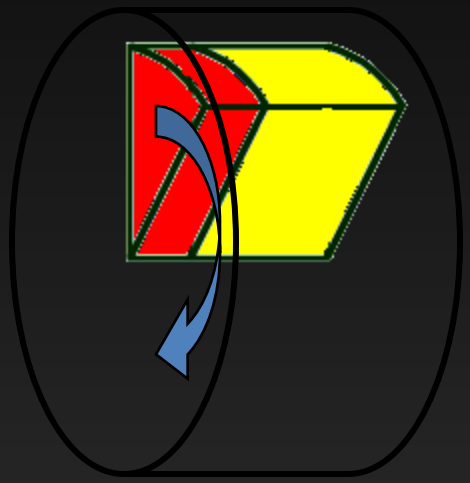
- logical volume
- translation w.r.t. envelop
- rotation w.r.t. envelop



Reflecting solids

- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation

or 180 degree rotation.



G4ReflectedSolid (derived from G4VSolid)

- Utility class representing a solid shifted from its original reference frame to a new mirror symmetric one
- The reflection (G4Reflect[X/Y/Z]3D) is applied as a decomposition into rotation and translation

G4ReflectionFactory

- Singleton object using G4ReflectedSolid for generating placements of reflected volumes

Reflections are currently limited to simple CSG solids.

- will be extended soon to all solids

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
  - ✓ 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
  - ✓ Shower envelope
- Regional user stepping action (new with version 9.0)

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.

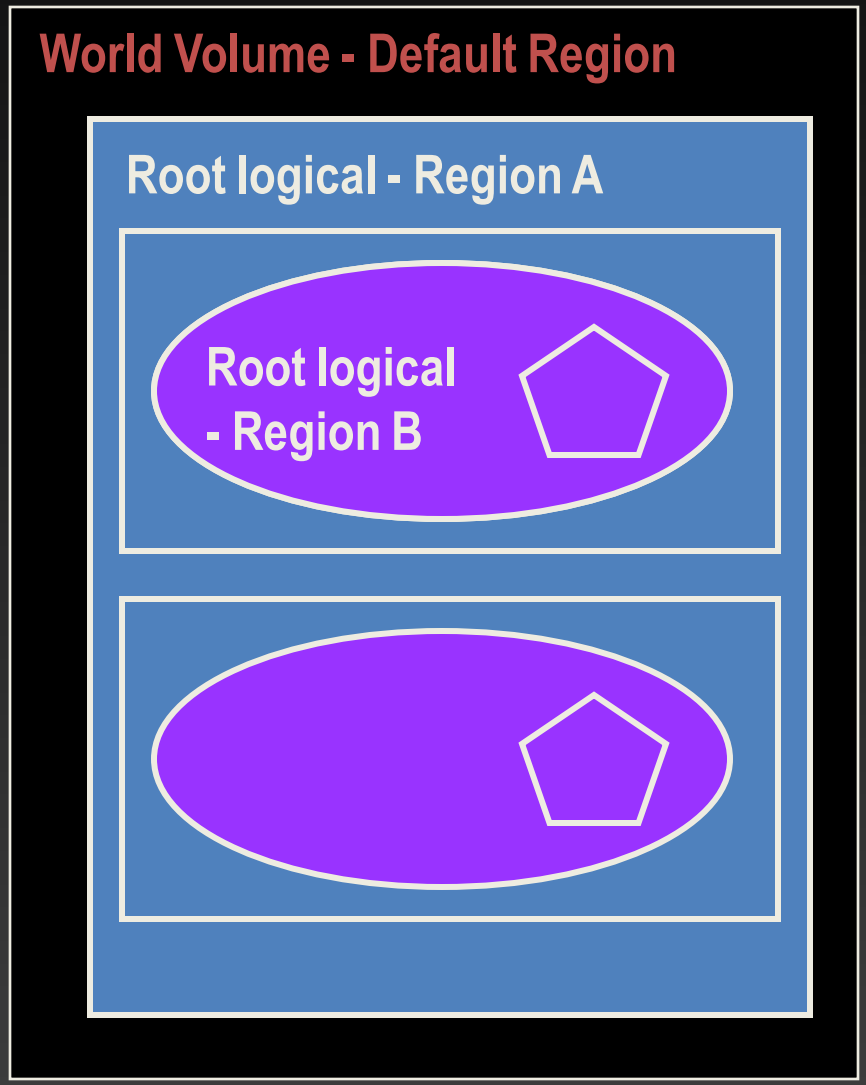
A region is a part of the geometrical hierarchy, i.e. a set of geometry volumes, typically of a sub-system.

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 becomes 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);
```

DEFINING A MAGNETIC FIELD

Magnetic field (1)

Create your Magnetic field class

- Uniform field :
  - ✓ Use an object of the G4UniformMagField class

```
G4MagneticField* magField =
    new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.);
```

- Non-uniform field :
  - ✓ Create your own concrete class derived from G4MagneticField and implement **`GetFieldValue`** method.

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

  - ✓ Point[0..2] are position in global coordinate system, Point[3] is time
  - ✓ field[0..2] are returning magnetic field

Magnetic field (2)

Tell Geant4 to use your field

1. Find the global Field Manager

```
G4FieldManager* globalFieldMgr =
    G4TransportationManager::GetTransportationManager()
    ->GetFieldManager();
```

2. Set the field for this FieldManager,

```
globalFieldMgr->SetDetectorField(magField);
```

3. and create a Chord Finder.

```
globalFieldMgr->CreateChordFinder(magField);
```

/example/novice/N04/ExN04 is a good starting point

Global and local fields

One field manager is associated with the 'world' and it is set in G4TransportationManager

Other volumes can override this

- An alternative field manager can be associated with any logical volume
  - ✓ The field must accept position in global coordinates and return field in global coordinates
- By default this is propagated to all its daughter volumes

```
G4FieldManager* localFieldMgr
      = new G4FieldManager(magField);
logVolume->setFieldManager(localFieldMgr, true);
```

where '**true**' makes it push the field to all the volumes it contains, unless a daughter has its own field manager.

Customizing the field propagation classes

- Choosing an appropriate stepper for your field
- Setting precision parameters

Field integration

In order to propagate a particle inside a field, we solve the equation of motion of the particle in the field.

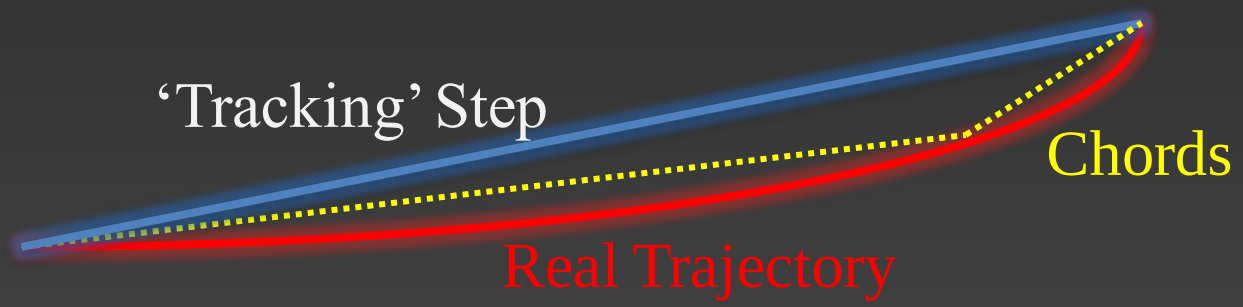
We use a Runge-Kutta method for the integration of the ordinary differential equations of motion.

- *Several Runge-Kutta 'steppers'* are available.

In specific cases other solvers can also be used:

- In a uniform field, using the analytical solution.
- In a smooth but varying field, with RK+helix.

Using the method to calculate the track's motion in a field, Geant4 breaks up this curved path into linear *chord segments*.



Tracking in field

We use the chords to interrogate *G4Navigator*, to see *whether the track has crossed a volume boundary*.

- One tracking step can create several chords.
- Users can set the accuracy of the volume intersection,
  - ✓ By setting a parameter called the "*miss distance*"
  - ✓ It is a measure of the error in whether the approximate track intersects a volume.
  - ✓ It is quite expensive in CPU performance to set too small "*miss distance*".