Geant 4
Detector Description – basic concepts

Gabriele Cosmo, CERN/IT

Geant4 Users’ Workshop, CERN 11-15 November 2002
Detector Description

Part I  The Basics
Part II Logical and physical volumes
Part III Solids, touchables
Part IV Optimisation technique & Advanced features
PART 1

Detector Description: the Basics
Describe your detector

- Derive your own concrete class from `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
      - Associate magnetic field to detector regions
      - Define visualization attributes for the detector elements
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
Define 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

  ```cpp
  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 unique physical volume which represents the experimental area must exist and fully contains all other components

  ➢ The world volume
PART II

Detector Description:
Logical and Physical Volumes
G4LogicalVolume

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

- Contains all information of volume except position:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits
  - Shower parameterisation

- Physical volumes of same type can share a logical volume.
- 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
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 or b) are identical in size & shape.
- G4PVReplica 1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)
**Physical Volumes**

- **Placement**: it is one positioned volume
- **Repeated**: a volume placed many times
  - can represent any number of volumes
  - reduces use of memory.
- **Replica**
  - simple repetition, similar to G3 divisions
- **Parameterised**

- A **mother** volume can contain **either**
  - **many placement** volumes **OR**
  - **one repeated** volume
G4PVPlacement

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

- Single volume positioned relatively to the mother volume
  - In a frame rotated and translated relative to the coordinate system of the mother volume
- Three additional constructors:
  - A simple variation: specifying the mother volume as a pointer to its physical volume instead of its logical volume.
  - Using G4Transform3D to represent the direct rotation and translation of the solid instead of the frame
  - The combination of the two variants above
Parameterised Physical Volumes

- User written functions define:
  - the size of the solid (dimensions)
    - Function `ComputeDimensions(…)`
  - where it is positioned (transformation)
    - Function `ComputeTransformations(…)`

- Optional:
  - the type of the solid
    - Function `ComputeSolid(…)`
  - the material
    - Function `ComputeMaterial(…)`

- Limitations:
  - Applies to simple CSG solids only
  - Daughter volumes allowed only for special cases

- Very powerful
  - Consider parameterised volumes as “leaf” volumes

G.Cosmo, Detector Description - CERN, November 2002
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
G4PVPVParameterised

G4PVPVParameterised(const G4String& pName,
    G4LogicalVolume* pCurrentLogical,
    G4LogicalVolume* pMotherLogical,
    const EAxis pAxis,
    const G4int nReplicas,
    G4VPVPVParameterisation* pParam);

- Replicates the volume nReplicas times using the parameterisation pParam, within the mother volume pMother.
- The positioning of the replicas is dominant along the specified Cartesian axis
  - If kUndefined is specified as axis, 3D voxelisation for optimisation of the geometry is adopted
- Represents many touchable detector elements differing in their positioning and dimensions. Both are calculated by means of a G4VPVPVParameterisation object
- Alternative constructor using pointer to physical volume for the mother
Parameterisation example - 1

```cpp
G4VSolid* solidChamber = new G4Box("chamber", 100*cm, 100*cm, 10*cm);
G4LogicalVolume* logicChamber =
    new G4LogicalVolume(solidChamber, ChamberMater, "Chamber", 0, 0, 0);
G4double firstPosition = -trackerSize + 0.5*ChamberWidth;
G4double firstLength = fTrackerLength/10;
G4double lastLength = fTrackerLength;
G4VPVParameterisation* chamberParam =
    new ChamberParameterisation( NbOfChambers, firstPosition,
                                ChamberSpacing, ChamberWidth,
                                firstLength, lastLength);
G4VPhysicalVolume* physChamber =
    new G4PVParameterised( "Chamber", logicChamber, logicTracker,
                           kZAxis, NbOfChambers, chamberParam);
```

Use `kUndefined` for activating 3D voxelisation for optimisation.
class ChamberParameterisation : public G4VPVParameterisation
{
    public:

    ChamberParameterisation( G4int NoChambers, G4double startZ, 
                            G4double spacing, G4double widthChamber, 
                            G4double lenInitial, G4double lenFinal );

    ~ChamberParameterisation();

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

    void ComputeDimensions ( G4Box& trackerLayer, const G4int copyNo, 
                             const G4VPhysicalVolume* physVol ) const;

};
Parameterisation example - 3

```cpp
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Zposition= fStartZ + (copyNo+1) * fSpacing;
    G4ThreeVector origin(0, 0, Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}

void ChamberParameterisation::ComputeDimensions
(G4Box& trackerChamber, const G4int copyNo,
 const G4VPhysicalVolume* physVol) const
{
    G4double halfLength= fHalfLengthFirst + copyNo * fHalfLengthIncr;
    trackerChamber.SetXHalfLength(halfLength);
    trackerChamber.SetYHalfLength(halfLength);
    trackerChamber.SetZHalfLength(fHalfWidth);
}
```

G.Cosmo, Detector Description - CERN, November 2002
Replicated Physical Volumes

- The mother volume is sliced into replicas, all of the same size and dimensions.
- Represents many touchable detector elements differing only in their positioning.
- 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* pCurrentLogical,
 G4LogicalVolume* pMotherLogical,
 const EAxis pAxis,
 const G4int nReplicas,
 const G4double width,
 const G4double offset=0);

- Alternative constructor: using pointer to physical volume for the mother
- An offset can only be associated to a mother offset along the axis of replication
- 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
Replication example

G4double tube_dPhi = 2.* M_PI;
G4VSolid* tube =
    new G4Tubs("tube", 20*cm, 50*cm, 30*cm, 0., tube_dPhi*rad);
G4LogicalVolume * tube_log =
    new G4LogicalVolume(tube, Ar, "tubeL", 0, 0, 0);
G4VPhysicalVolume* tube_phys =
    new G4PVPlacement(0,G4ThreeVector(-200.*cm, 0., 0.*cm),
                        "tubeP", tube_log, world_phys, false, 0);
G4double divided_tube_dPhi = tube_dPhi/6.;
G4VSolid* divided_tube =
    new G4Tubs("divided_tube", 20*cm, 50*cm, 30*cm,
                -divided_tube_dPhi/2.*rad, divided_tube_dPhi*rad);
G4LogicalVolume* divided_tube_log =
    new G4LogicalVolume(divided_tube, Ar, "div_tubeL", 0, 0, 0);
G4VPhysicalVolume* divided_tube_phys =
    new G4PVReplica("divided_tube_phys", divided_tube_log, tube_log,
                    kPhi, 6, divided_tube_dPhi);
PART III

Detector Description:
Solids & Touchables
G4VSolid

- Abstract class. All solids in Geant4 derive from it
  - Defines but does not implement all functions required to:
    - compute distances to/from the shape
    - 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

- Once constructed, each solid is automatically registered in a specific solid store
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, ...
  - STEP interface
    - to import BREP solid models from CAD systems - STEP compliant solid modeler
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

G4Cons(const G4String& pName, // name
        G4double pRmin1, // inner radius -pDz
        G4double pRmax1, // outer radius -pDz
        G4double pRmin2, // inner radius +pDz
        G4double pRmax2, // outer radius +pDz
        G4double pDz,   // Z half length
        G4double pSphi, // starting Phi
        G4double pDphi); // segment angle
Specific CSG Solids: G4Polycone

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

- \text{numRZ} - numbers of corners in the \text{r, z} space
- \text{r, z} - coordinates of corners
- Additional constructor using planes
**BREP Solids**

- **BREP** = *Boundary REPresented Solid*
- Listing all its surfaces specifies a solid
  - e.g. 6 squares for a cube
- Surfaces can be
  - planar, 2\textsuperscript{nd} 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
BREPS: G4BREPSolidPolyhedra

G4BREPSolidPolyhedra(const G4String& pName,
                      G4double phiStart,
                      G4double phiTotal,
                      G4int sides,
                      G4int nZplanes,
                      G4double zStart,
                      const G4double zval[],
                      const G4double rmin[],
                      const G4double rmax[]);

- **sides** - numbers of sides of each polygon in the $x$-$y$ plane
- **nZplanes** - numbers of planes perpendicular to the $z$ axis
- **zval[]** - $z$ coordinates of each plane
- **rmin[], rmax[]** - Radii of inner and outer polygon at each plane
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

- Example:
  ```cpp
g4Box box("Box", 20, 30, 40);
g4Tubs cylinder("Cylinder", 0, 50, 50, 0, 2*M_PI); // r: 0 -> 50
                                               // z: -50 -> 50
                                               // phi: 0 -> 2 pi
  
g4UnionSolid union("Box+Cylinder", &box, &cylinder);
g4IntersectionSolid intersect("Box Intersect Cylinder", &box, &cylinder);
g4SubtractionSolid subtract("Box-Cylinder", &box, &cylinder);
```

- Solids 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 solids
How to identify a volume uniquely?

- Need to identify a volume uniquely
- Is a physical volume pointer enough? **NO!**

• **Touchable**

G.Cosmo, Detector Description - CERN, November 2002
What can a touchable do?

- All generic touchables can reply to these queries:
  - positioning information (rotation, position): `GetTranslation()`, `GetRotation()`

- Specific types of touchable also know:
  - (solids) - their associated shape: `GetSolid()`
  - (volumes) - their physical volume: `GetVolume()`
  - (volumes) - their replication number: `GetReplicaNumber()`
  - (volumes hierarchy or touchable history):
    - info about its hierarchy of placements: `GetHistoryDepth()`
      - At the top of the history tree is the world volume
    - modify/update touchable: `MoveUpHistory()`, `UpdateYourself()`
      - take additional arguments
Benefits of Touchables in track

- Permanent information stored
  - to avoid implications with a “live” volume tree
- Full geometrical information available
  - to processes
  - to sensitive detectors
  - to hits

![Diagram showing points A1 and A2 connected by a line]
Touchable - 1

- G4Step has two G4StepPoint objects as its starting and ending points. All the geometrical information of the particular step should be got from “PreStepPoint”
  - Geometrical information associated with G4Track is basically same as “PostStepPoint”
- Each G4StepPoint object has:
  - position in world coordinate system
  - global and local time
  - material
  - G4TouchableHistory for geometrical information
- Since release 4.0, handles (or smart-pointers) to touchables are intrinsically used. Touchables are reference counted
Touchable - 2

- G4TouchableHistory has information of geometrical hierarchy of the point

```cpp
G4Step* aStep = ..;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHistoryHandle theTouchable =
    preStepPoint->GetTouchableHandle();
G4int copyNo = theTouchable->GetReplicaNumber();
G4int motherCopyNo = theTouchable->GetReplicaNumber(1);
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()->
    GetTopTransform().TransformPoint(worldPos);
```