Geometry - 2

I. Hrivnacova, IPN Orsay

Credits: T. Nikitina, J.Apostolakis, G.Cosmo, A. Lechner (CERN), M. Asai (SLAC) and others

Geant4 School at IFIN-HH, Bucharest,
14 - 18 November 2016

Outline

- Defining magnetic field
- Tunable parameters of propagation in magnetic field
- Geometry checking tools

Defining Magnetic Field

Describe Your Detector

- To describe your detector you have to derive your own concrete class from G4VUserDetectorConstruction abstract base class.
- Implement the virtual method Construct(), where you
  - Instantiate all necessary materials
  - Instantiate volumes of your detector geometry
- Optionally, implement the virtual method ConstructSDandField(), where you
  - Instantiate your sensitive detector classes and set them to the corresponding logical volumes
  - Instantiate magnetic (or other) field
- Optionally you can define
  - Regions for any part of your detector
  - Visualization attributes (color, visibility, etc.) of your detector elements

---

Before 10.x

- Instantiate your sensitive detector and magnetic (or other) field in Construct()

Field Manager

- The magnetic field is applied to geometry with means of G4FieldManager
- One field manager is associated with the ‘world’ and it is set in G4TransportationManager, it handles the **global field**
  - The global field manager need not to be created by the user

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

- An alternative field manager can be associated with any logical volume, it handles then the **local field**
  - By default this is propagated to all its daughter volumes
  - The field must accept position in global coordinates and return field in global coordinates

```
G4FieldManager* fieldManager = new G4FieldManager(magField);
logVolume->SetFieldManager(fieldManager, true);
```

  – Where ‘true’ means to propagate field to all the volumes it contains

Magnetic field

- Magnetic field class defines the strength of magnetic field within the world (global field) or within a given volume (local field)
- Magnetic field class:
  - Users can define their own concrete class derived from G4MagneticField and implement GetFieldValue method:

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

  - where point[0..2] represents the position in global coordinate system and point[3] time
  - field[0..2] return the field value in the given position
- To define a **uniform magnetic field**, users need not to define their own class, but can use G4UniformMagField:

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

Global Magnetic Field

```
void MyDetectorConstruction::CreateSDandField
{
   // Magnetic field
   MyMagneticField* myField = new MyMagneticField();

   // Field manager
   G4FieldManager* fieldManager
      = G4TransportationManager::GetTransportationManager()
        ->GetFieldManager();
   fieldManager->SetDetectorField(myField);
   fieldManager->CreateChordFinder(myField);

   // Register the field for deleting
   G4AutoDelete::Register(myField);
}
```

Local Magnetic Field

```
void MyDetectorConstruction::CreateSDandField
{
   // Magnetic field
   MyMagneticField* myField = new MyMagneticField();

   // Field manager
   G4Fieldmanager* fieldManager = new G4FieldManager();
   fieldManager->SetDetectorField(myField);
   fieldManager->CreateChordFinder(myField);

   // Set field to a logical volume
   G4bool forceToAllDaughters = true;
   magneticLogical
     ->SetFieldManager(fieldManager, forceToAllDaughters);

   // Register the field and its manager for deleting
   G4AutoDelete::Register(myField);
   G4AutoDelete::Register(fieldManager);
}
```

See basic example B5

Global Field Messenger

- A helper class, G4GlobalMagFieldMessenger, is available since Geant4 10.00
  - It creates **the global uniform magnetic field**
  - **The field** is **activated** (set to the G4TransportationManager object) only when its fieldValue is non zero vector.
  - It can be also used to change the field value (and activate or inactivate the field again

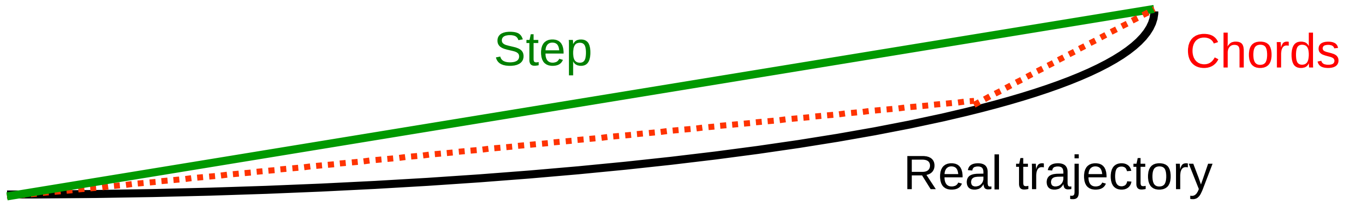
```
void MyDetectorConstruction::CreateSDandField
{
   // Global magnetic field & its messenger
   G4ThreeVector fieldValue = G4ThreeVector();
   G4GlobalMagFieldMessenger* magFieldMessenger
      = new G4GlobalMagFieldMessenger(fieldValue);

   // Register the messenger for deleting
   G4AutoDelete::Register(myFieldMessenger);
}
```

See basic examples B2 and B4

Propagation in Field
Tunable Parameters

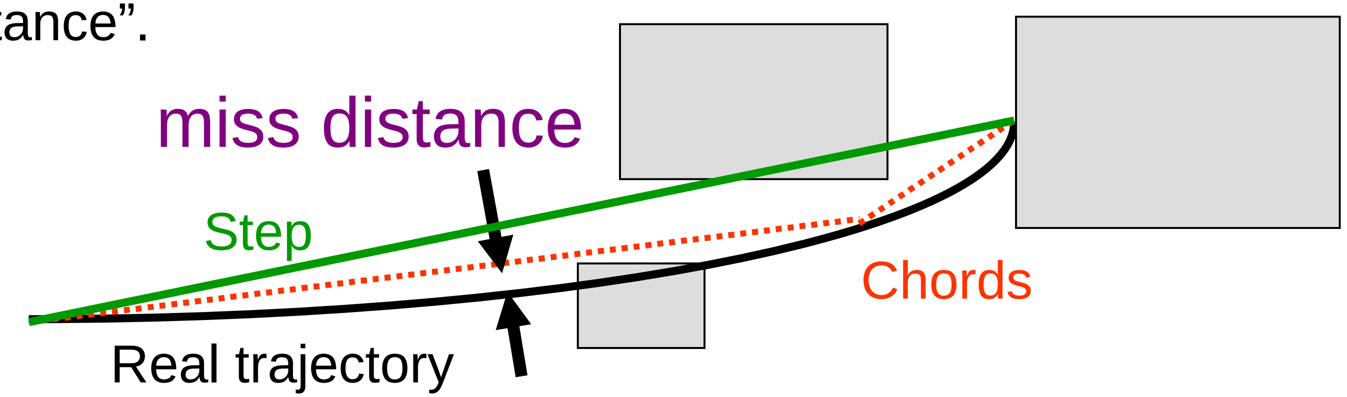
Propagation in Field (1)

- In order to propagate a particle inside a field (e.g. magnetic, electric or both), 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 analytic 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.
  - We determine the chord segments so that they closely approximate the curved path.



Tunable Parameters

- We use the chords to interrogate the G4Navigator, to see whether the track has crossed a volume boundary.
- One physics/tracking step can create several chords.
  - In some cases, one step consists of several helix turns.
- User 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".



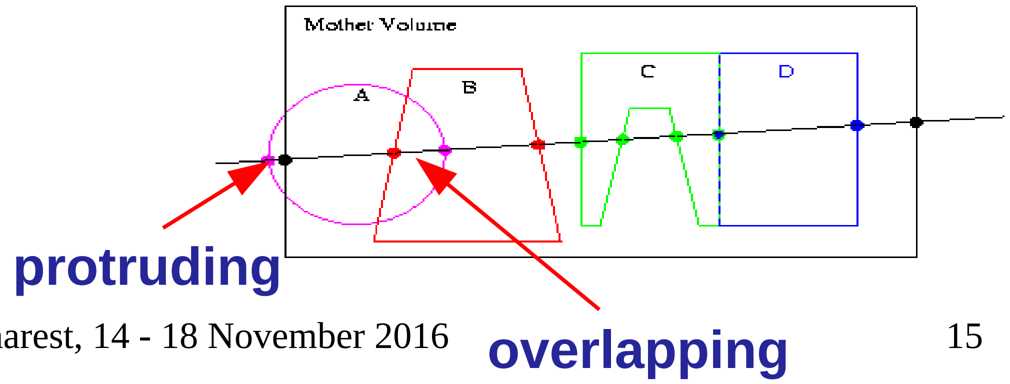
Tunable Parameters (2)

- In addition to the “miss distance” there are two more parameters which the user can set in order to adjust the accuracy (and performance) of tracking in a field.
  - These parameters govern the accuracy of the intersection with a volume boundary and the accuracy of the integration of other steps.
- The “delta intersection” parameter is the accuracy to which an intersection with a volume boundary is calculated.
  - This parameter is especially important because it is used to limit a bias that our algorithm (for boundary crossing in a field) exhibits.
- The “delta one step” parameter is the accuracy for the endpoint of 'ordinary' integration steps, those which do not intersect a volume boundary.
  - This parameter is a limit on the estimation error of the endpoint of each physics step.

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 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)



Optional Checks at Construction

- The option to check overlaps at geometry construction can be activated in the G4PVPlacement constructor :

```
 G4PVPlacement(
   G4RotationMatrix* rotation,        // rotation
   const G4ThreeVector& translation, // translation
   G4LogicalVolume* currentLV,        // volume being placed
   const G4String& name,              // physical volume name
   G4LogicalVolume* motherLV,         // mother logical volume
   G4bool many,                       // not used
   G4int copyNumber,                  // position (copy) number
   G4bool surfaceCheck = false);     // option to activate
                                      // overlap checking
```

- Some number of points are then 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

- Verification of geometry for overlapping regions recursively through the volumes tree can be a run with a built-in run-time command:
  - `geometry/test/run`
- Volumes are recursively asked to verify for overlaps for points generated on the surface against their respective mother and volumes at the same level, performing for daughters and daughters of daughters etc.
  - It may take a very long time in complex geomteries
  - Parameters which can be tuned:
    - `recursion_start` – starting depth level (default 0)
    - `recursion_depth` – the total depth level for checking overlaps (default -1, which mean all levels)
    - `tolerance` – tolerance by which overlaps should not be `reported.`
    - `resolution` - the number of points on surface to be generated and checked for each volume (default is '10000')
    - `maximum_errors` - the threshold for the number of errors to be reported for a single volume (default is 1)

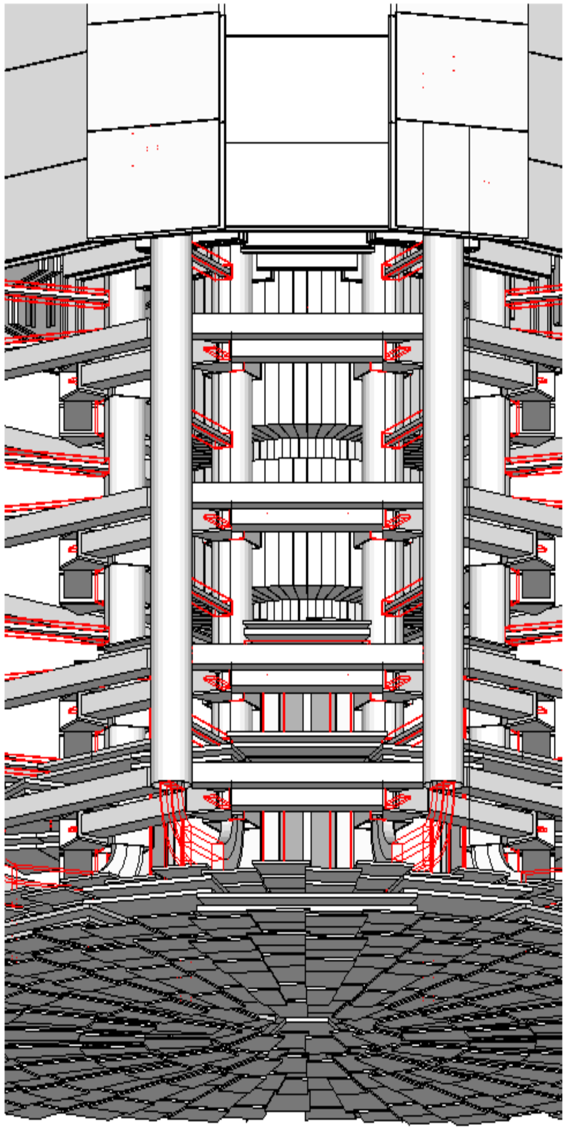
Debugging Run-time Commands (2)

- Example of a test output

```
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: 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
  - http://geant4.kek.jp/~tanaka/



Summary

- Several classes can be used to define a repeated placement: G4PVReplica, G4PVDivisin and G4PVParameterisedVolume
- The volumes can be grouped together in a G4AssemblyVolume object and the whole group can be placed as a single "virtual" volume
- The volume hierarchies can be reflected using G4ReflectionFactory
- Geant4 does not allow for malformed geometries, neither protruding nor overlapping
  - Geometry checking tools are available to detect such cases