Geometry 5

I.Hrivnacova[1], J.Apostolakis[2]
[1]IPN, Orsay; [2]CERN

Cours Geant4 @ Paris 2007
4 - 8 June 2007

Contents

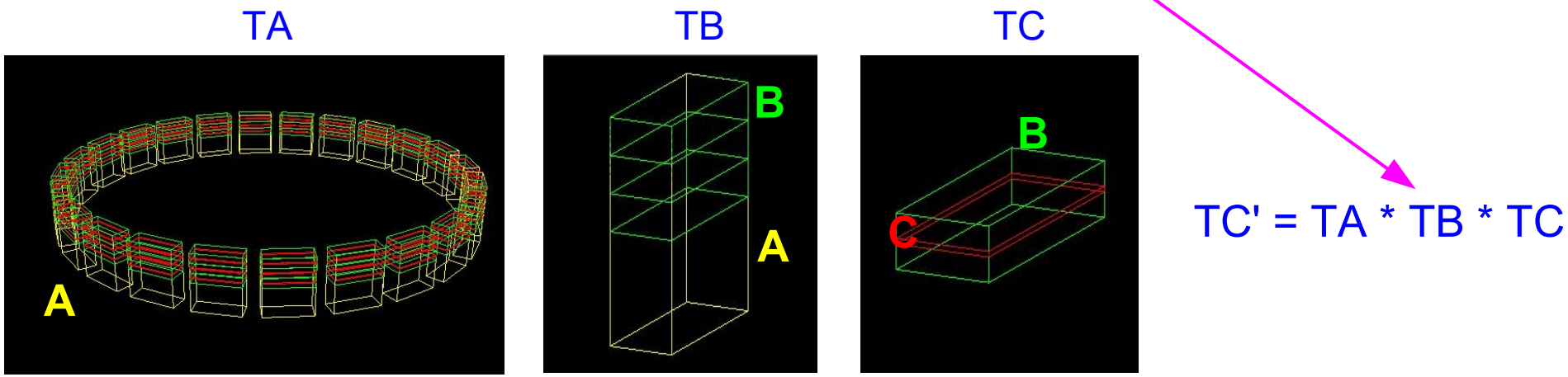
- Navigator's view of the geometry
  - Geometry navigator and its utilities
  - Navigation & Tracking
  - Locating points in a user application
- Best practice in creating a geometry
  - Things to avoid, recipes to use

Geometry Navigator

- Navigation through the geometry at tracking time is implemented by the class G4Navigator
  - It locates a point in the geometry
  - It computes the distance to the next geometry boundary
  - It is the key point of interaction between the tracking and the geometry

Geometry Navigator (cont.)

- The Navigator sees the user's geometry definition, which provides a 'logical' hierarchy of volumes with relative positions mother <-> daughter
  - The set of volume starting from the world, going down the volume tree to the current volume is stored in the Navigator – and provided as the touchable.
  - Its relative position is defined by a multiplication of all transformations.
  - To avoid recomputing of the final positions of touchable every time, a utility class G4NavigationHistory is defined



Touchable (and Navigation) History

- The information in the touchable/navigation history object, at each level of the hierarchy to the current location (depth):
  - the compounded transformations,
  - the volume types - whether normal (placement), replicated or parameterised.
  - replication/parameterisation information
  - volume pointers

Touchable (and Navigation) History (cont.)

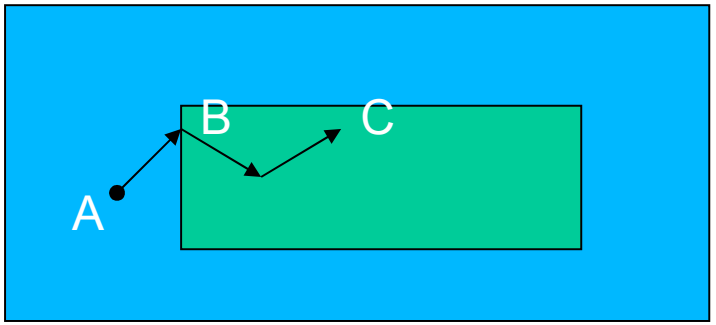
- The Navigator owns a private G4NavigationHistory object,
  - Which enables it to go up a level with no effort, and to retrieve the transformation at the current and all other levels.
- The same functionality is made available for the user using the G4TouchableHistory,
  - Which the Navigator creates (it is requested by the G4Transportation process, which passes it to the G4Track).

The Navigator State

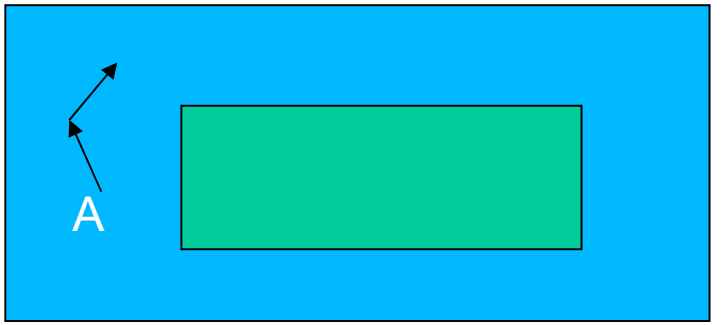
- With each locating a point, the navigator 'remembers' this and updates its state
- Besides the navigation history (G4NavigationHistory), the navigator state includes various other information like
  - The last located local point
  - Whether entering a daughter volume with this step, whether existing the mother volume
  - Whether the last step was limited by geometry
  - Etc.
- This information is then used when locating the next point to speed up this process

Locate a point with the Navigator

- The Navigator can locate a point either relative to the previous location or starting from the world volume
  - With each new step a track can move only to a neighboring volume in space.
    - Locating relative to the previous location offers better performance
  - Even most new tracks are close to previous tracks
    - Yet if there is an overlap in the geometry this can cause difficulties.



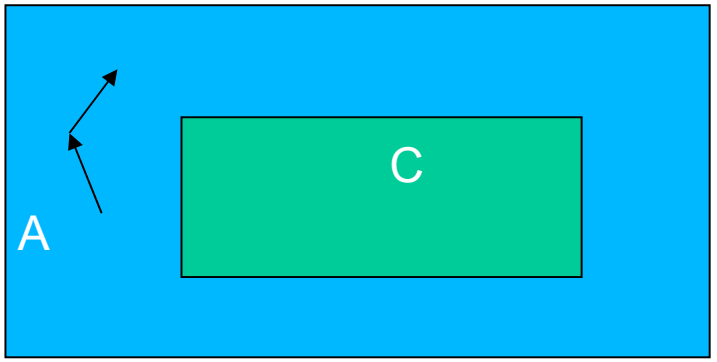
When track arrives at point B, the Navigator locates the green volume by searching only for sub-volumes of A.



After the track died at C, the tracking starts the next track at A. It can find it quickly looking 'up' to the blue volume.

How the Navigator Locates a point

- At each location the Navigator will examine whether the new point is inside the current volume:
  - If the point is not inside the current volume it will exit into its mother volume
  - It will check again – continuing to do this until it finds a volume that contains it. Else if it is not contained even in the world, it is done.
  - Once it has located a containing volume (it could be the original one), it will check to see whether the point is contained by any of the its daughter volumes. (At every point it will use the local coordinates for the right level.)
  - It will continue to descend, checking to see whether the point is inside any daughter at each level



In this figure, starting from the green volume, and locating at A requires checking only two volumes: the green one (outside) and the blue one (inside).

Navigation Utilities

- The concrete way of computing the location and distances depends strongly on the type of the physical volume; that's why there are introduced the following navigation utility classes

G4NormalNavigation	*For geometries containing 'placement' volumes, with no voxels.*
G4VoxelNavigation	*For geometries containing 'placement' volumes, with voxels.*
G4ParameterisedNavigation	*For geometries containing parameterised volumes, with voxels.*
G4ReplicaNavigation	*For replicated geometries*

Navigation Utilities (cont.)

- G4VoxelNavigation
  - Internally a stack of voxel information is maintained.
  - Private functions allow for isotropic distance computation to voxel boundaries and for computation of the 'next voxel' in a specified direction.
- G4ParameterisedNavigation
  - Voxel information is maintained similarly to G4VoxelNavigation, but computation can also be simpler by adopting voxels to be one level deep only (unrefined, or 1D optimisation)
  - If you have many sub-volumes in a 2D or 3D pattern (eg. lattice) then you must ensure that the navigator uses full voxelisation (to do this use 'kUndefined' as axis.)
  - If you have volumes placed along an axis so that only 3 or less exist in any one 'slice' along this axis, then you can inform the navigator by specifying this axis

Navigation & Tracking

- The Navigator must be told the 'root' or first volume of the geometry.
  - SetWorldVolume() - sets the first volume in the hierarchy. It must be unrotated and untranslated from the origin.
- The main functions required for tracking in the geometry:
  - LocateGlobalPointAndSetup() - locates the volume containing the specified global point. To improve efficiency a G4TouchableHistory can be used.
  - LocateGlobalPointAndUpdateTouchableHandle() - as previous, but then use the volume found and its navigation history to update the touchable.
  - ComputeStep() - computes the distance to the next boundary intersected along the specified unit direction from a specified point. The point must have been located prior to calling ComputeStep().

Navigation & Tracking (cont.)

- SetGeometricallyLimitedStep() - informs the navigator that the last computed step was taken in its entirety.
  - In a running application it is the responsibility of the Transportation process to call this method.
- CreateTouchableHistory() - creates a G4TouchableHistory object, for which the caller has deletion responsibility. The 'touchable' volume is the volume returned by the last Locate operation. The object includes a copy of the current NavigationHistory, enabling the efficient relocation of points in/close to the current volume in the hierarchy.
  - User code or other processes can create a touchable history.
  - They are now responsible to delete the object they get – or they can use a handle to enable automatic deletion – using eg. CreateTouchableHistoryHandle()

Locating Points in a User Application

- G4Navigator functions can be used in a user application to locate points in geometry
- The 'main' navigator object
  - The navigator instantiated automatically at the startup of a simulation program,
  - It is used for all the tracking (locating at the start of the track, and each step)
  - Can be accessed at any stage of the application:

```
G4Navigator* tracking_navigator
  = G4TransportationManager::GetInstance()
      ->GetNavigatorForTracking();
```

- **But the 'main'navigator also takes care of relocating the actual particle tracked**
  - ***if user code uses it during tracking, the tracking actual particle can be affected by such a user call !***
- **Note:**
  - ***it is not safe to use either the 'tracking' Navigator or another one during tracking*** **if the geometry contains a replica, division or parameterised volume – since their state is part of the state of the navigation.**

Locating Points
During Tracking

- Use the touchable handle available via current step:

```
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHandle theTouchable
  = preStepPoint->GetTouchableHandle();
G4ThreeVector worldPosition = preStepPoint->GetPosition();
```

Locating Points
In the |Idle> State

- Create an alternative navigator object first and assign it to the world volume:

```
G4Navigator* myNavigator = new G4Navigator();
myNavigator->SetWorldVolume(worldVolumePointer);
```

- To locate a given point in global coordinates:

```
G4ThreeVector myPoint(10*cm, 10*cm., 10*cm);
myNavigator->LocateGlobalPointAndSetup(myPoint);
G4TouchableHistoryHandle  myTouchable
  = myNavigator->CreateTouchableHistoryHandle();
// Convert point in local coordinates (local to the current volume)
G4ThreeVector localPosition
  = aTouchable->GetHistory()
      ->GetTopTransform().TransformPoint(myPoint);
```

Run-time commands

- To print debugging information user can set the navigator verbose level

  `/geometry/navigator/verbose [verbose_level]`

  - Geant4 has to be compiled with G4VERBOSE set (what is the default)

- To force the navigator to run in check mode

  `/geometry/navigator/check_mode [true/false]`

  - More strict and less tolerant checks in step/safety computation to verify the correctness of the solids' response in the geometry

Examples

- By combining check_mode with verbosity level-1, additional verbosity checks on the response from the solids can be activated
- To get verbose information from each solid asked, including problem issues, use:

```
/geometry/navigator/verbose 1
/geometry/navigator/check_mode true
```

- To get checking, but without verboseness for each solid, use instead:

```
/geometry/navigator/verbose 0
/geometry/navigator/check_mode true
```

Contents

Navigator's view of the geometry
Geometry navigator and its utilities
Navigation & Tracking
Locating points

Best practice in creating a geometry
Things to avoid, recipes to use

Creating Geometry

- The define the geometry setup in Geant4, you have to derive your own concrete class from G4VUserDetectorConstruction abstract base class and implement the method Construct()
- This includes:
  - Construct all necessary materials
  - Define shapes/solids required to describe the geometry
  - Construct and place volumes of your detector geometry
  - Define sensitive detectors
  - Associate magnetic field to detector region
  - Define visualization attributes for the detector elements
- In this lecture, we will concentrate on the items in red

Modularizing Geometry Construction

- Definition of geometry is often an iterative process
  - The first simulation starts from a rough geometry which is successively replaced with more detailed pieces
  - Such a process allows geometry verifications step by step
- Big experiment setups
  - The whole detector is usually composed from several sub-detector, which are often constructed by different teams
  - It is natural to decompose geometry construction in several classes per sub-systems and let the main geometry construction operate as their manager
- Small experiment setups
  - If one class can do it (without exceeding thousands of lines), it may turn useful to define member functions per detector parts and call them from the ConstructGeometry() method

Choosing/Verifying a solid

- Choose the simplest solid that is close enough to your case
  - Eg. G4Box to G4BREPBox, G4Trd to G4Trap if its symmetry allows it
- Check solids parameters
  - All solids have implemented streaming operator:

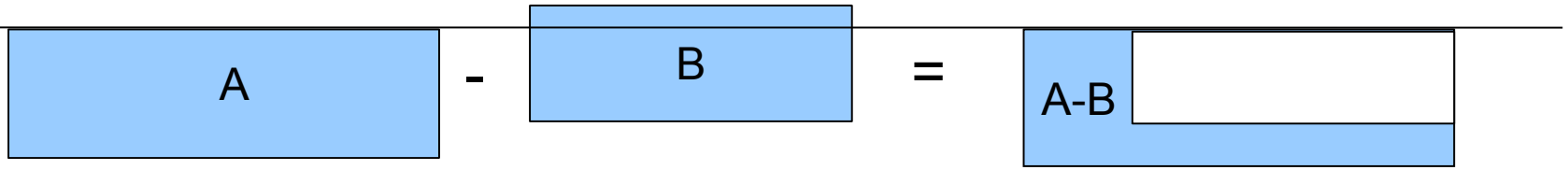
```
G4cout << *mySolid << G4endl;
```

- Verification of solid by shooting geantino in suitable directions
  - Useful for more complex solids (general trapezoids, polyhedra, tessellated solids, ...)
- Verification with visualization

Choosing/Verifying a solid (cont.)

- **Boolean solids**
  - Avoid too many levels of boolean operations (time consuming)
    - The time for location and intersection operations can/will rise proportionally to the number of operations used
  - To aid visualization (and verification), avoid having coincident surfaces between constituent solids, if possible. It should be easily possible to avoid these in particular when subtracting solids
    - This causes difficulty in visualization:

    - Whereas extending the volume B outside can avoid the problem:



Assembling volumes

- The defined solid can be placed in geometry in various ways:
  - Via simple placement (G4PVPlacement)
  - Via multiple placements (as a parameterised volume, replica, division, ...)
- There are different trade-offs when utilising placements, and parameterised volumes:
  - Using placements means that no computation is needed to see each copy, but that each copy uses memory
  - If memory is not an issue, then placements would be the fastest solution
  - Otherwise a parameterised volume is a solution which will reduce the size of memory but will increase computation time

Assembling volumes (cont.)

- Memory is used in two ways:
  1) First for the volumes themselves and also
  2) for the voxel information that is used to optimise locating a point and deciding which volumes to intersect in a step computation.
- Implementing a geometry with a lot of volumes at the same level ("flat") is possible in Geant4.
  - It must be done with care in order to avoid requiring more memory than is available (in particular for the voxelisation information.)
  - Parameters exist (smartless) that aid to tune the amount of memory required (See in Geometry 4: Geometry optimisation);
    - `G4LogicalVolume::SetSmartless(G4double);`

Assembling volumes (cont.)

- For geometries that can be divided hierarchically it is better to put fewer volumes in one mother.
  - For example six physical copies of the same sixth of a cylinder will consume less memory than a whole cylinder.
  - This can be done easily only when the pieces can be subdivided, without sub-volumes protruding from the parent.
- If possible, prefer replica to parameterised volume or division
  - Replica can be used if the slices fill the whole mother
  - Navigation in replica is faster than in parameterised volume

Verification Tools

- Switch on geometry checking in the G4LogicalVolume constructor
- Eliminating overlaps/protruding
  - Several methods have been presented
- Visualization
- See in detail in Geometry 4: Geometry checking tools