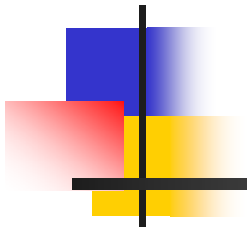
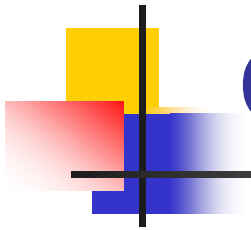


Ideas for G4 navigation interface using ROOT geometry

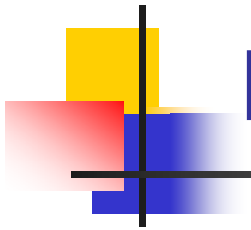
A.Gheata

VMC Workshop, 29-30 Nov. 04



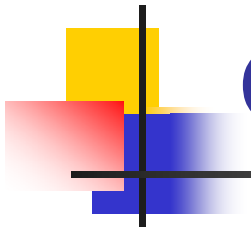
Outline

- Motivations
- Requirements and observations
- Correspondence between G4-TGeo geometry objects/features
- A possible strategy
- Conclusions



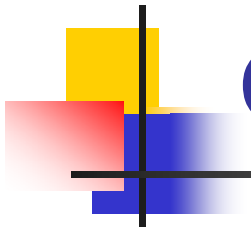
Motivations

- Possibility to compare G3,G4 and FLUKA simulations having the same geometry model behind
  - We have it for G3/FLUKA – it will insure consistency at navigation level
  - Allows usage of TGeo as a simulation engine - neutral geometry in the reconstruction framework
- VMC provides already an interface to G4 and geometry converters ROOT⇔G4 (see talk from Ivana)
  - What is existing is a big step forward without any doubt, but:
  - Some limitations in mapping certain features available in VMC to G4,
  - Geometry conversions limited to the common denominator of candidates,
  - Possibility of ROOT geometry usage within TGeant4 will certainly give more flexibility
- Possibility of a cross-check between navigation algorithms G4-ROOT



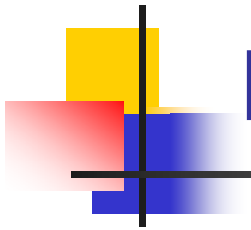
G4Navigator requirements

- Pure geometrical queries – taking point,vector,flags as input and returning distance/flags
  - ComputeStep(), ComputeSafety(), GetLocalExitNormal()
- Geometrical queries requiring a geometrical state as input (G4VTouchable - derived objects)
  - Local-to-Global and Global-to-local transformations



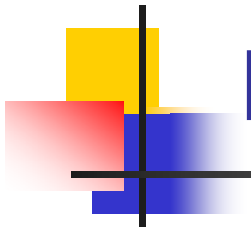
G4Navigator requirements (cont)

- Geometry queries finding a state and/or acting on a state
  - ResetHierarchyAndLocate(), LocateGlobalPointAndXXX()
- Geometrical state management and handles, utilities
  - CreateTouchableHistory(), CreateTouchableHandle()
  - CreateGRSVolume/Solid(), Set/GetWorldVolume()



Preliminary observations (1)

- *G4Navigator* is an abstract base class, but besides computing pure geometrical parameters, it provides/handles/acts on G4 native geometrical objects
  - This is natural for any OO framework, besides – geometry is not just a set of numbers giving back distances or in/out flags, but also objects embedding information required at tracking time
  - It does not make life easier compared to interfacing a FOTRAN navigator, it just introduces an additional dimension to the problem that HAVE to be dealt with



Preliminary observations (2)

- Once the previous fact is established, we have to look on:
  - Which are the G4 geometrical classes that are really required for navigation ?
  - What is the mapping between G4 objects and TGeo ones – is there a 1/1 correspondence ?
  - Are the methods purely related to navigation corresponding to what is offered by TGeo ?
- Knowing all this, what is the best strategy to follow ?
  - Requiring as less as possible development effort, but providing needed functionality
  - Optimizing performance at low memory cost – what is the "good compromise"

Correspondence between G4⇔TGeo geometrical objects

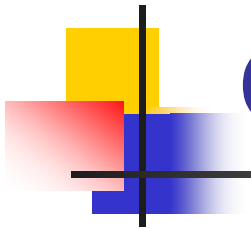
- **G4VSolid ⇔ TGeoShape**
  - Both abstract base classes with several implementations
  - Same quantities computed: In/Out, distance to boundary, safety, normal to exit point
  - One-to-one correspondence for all G4 solids to TGeo shapes (HYPE was missing but now implemented)
  - Some extra shapes with very low usage (so far) in TGeo: TGeoArb8, TGeoXtru, TGeoParaboloid
- Interface class: **TG4Solid : public G4VSolid**
  - Implementation is mandatory
  - Data member: TGeoShape *fShape
  - All query methods can be mapped
  - One limitation: point classification as ON BOUNDARY – missing in TGeo. It will be implemented if required by navigation.

Correspondence G4⇔TGeo (2)

- **G4VPhysicalVolume ⇔TGeoNode**
  - A volume positioned relative to its container
  - Same functionality
  - Slightly different structures and some differences in parameterization (divisions) treatment – not a stumbling block in the VMC approach
- Interface class: **TG4PhysicalVolume** : public **G4VPhysicalVolume**
  - Data members: TGeoNode *fNode, TG4LogicalVolume *fVolume , *fMother ! (see later)
  - A mapping TGeoNode => TG4PhysicalVolume absolutely needed since TGeoNode is the object provided by TGeo navigation methods

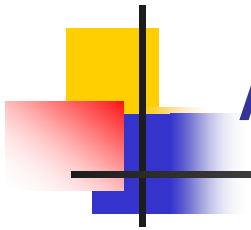
Correspondence G4⇔TGeo (3)

- **G4VTouchable ⇔TGeoCacheState**
  - Representing a geometrical "touchable" unique object, e.g. a branch in the logical volume hierarchy
  - Created by the navigation interface, ref.-counted handles can be asked also by users
  - Temporary object during TGeo navigation, but supports push/pop mechanism
- Interface: TG4StatePool – a pool of TG4VTouchable pre-built objects + ref-count handle mechanism
  - TG4VTouchable : public G4VTouchable, holding the current branch of TG4PhysicalVolume objects



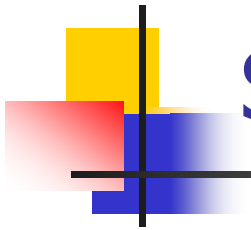
Other G4 object needed

- **G4LogicalVolume ⇔ TGeoVolume**
  - None abstract, both key elements in the logical hierarchy !
  - Not directly manipulated by TG4Navigator, but **required** from G4VTouchable/G4VPhysicalVolume by physics processes.
- Replicas, division, parameterisations
  - First 2 more or less the same in TGeo, third different
  - Tracking requires/acts according this information
  - May affect only when converting parameterized G4 geometries to TGeo



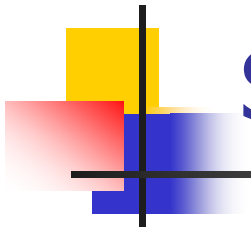
A possible strategy

- Step 1: TGeoShape acting as G4VSolid
  - Most easy to implement, probably the very first step to do
  - Instead of creating a native G4 solid make rather an object having a pointer to the corresponding ROOT shape.
  - Does not need full ROOT geometry to be built
  - Requires just the new derived class TG4Solid + few modifications in the existing GEANT4 VMC
  - Allows immediately a direct testing/cross check for query/classification algorithms at the level of solids
- Step2: Implementation of the mechanism of handling geometry states in G4 style as an addition to the current *stack* style in TGeo
  - Can be done at the level of the interface, but also direcly in TGeo
  - Not a tremendous effort – can be plugged in the interface once ready



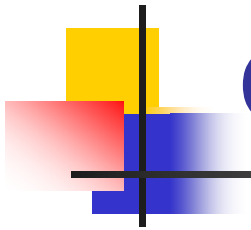
Strategy (cont)

- Step 3: Interfacing/mapping G4 and TGeo logical hierarchies
  - G4 needs its G4LogicalVolume objects => we have to provide them
  - Basically 2 ways for doing this:
    - Pool of limited number of objects of this type, as in the case of touchables
      - Several complications related to the fact that these are not virtual objects + heavy interface management
    - Just create and store full G4 logical tree in memory, in parallel with TGeo ones
      - Size less than 10 MBytes for geometries like ALICE or ATLAS
      - Will surely make the implementation easier
  - Connect the physical volume list as vector<TG4PhysicalNode*> and create the mapping TGeoNode->TG4PhysicalNode



Strategy (last)

- Step 4: Once we have all the infrastructure, implement all required navigation methods in TG4RootNavigator : public G4Navigator
  - Most methods have a 1/1 correspondence with TGeoManager methods, or are just derived queries that can be factorized in a manageable way
- One need to have in mind getting to this interface either from TGeo representation or even from G4 native



Conclusions

- Interfacing G4 navigation with TGeo is a challenge, but can be implemented in a reasonable amount of time (6 months)
  - Created as an option within the current TGeant4, will require less effort and benefit of the existing interface
- Step by step operation, may come-up with some first results much earlier than expected even if full validation will definitely be longer
- GEANT4 team is supporting this – will give more flexibility and use cases both to VMC and G4 users
- No major stumbling block: G4 and TGeo geometries are alike
  - Good policy: provide support for what is incompatible/missing in TGeo but required by G4Navigator, minimize additional structures to be managed at the level of the interface