ALICE misalignment framework

Raffaele Grosso

ALICE Offline Group

LHC Detector Alignment Workshop
September 5^{th} 2006

Outline

Introduction

The ALICE misalignment framework offers a set of tools to be used in ALICE by

- simulation and reconstruction
- survey procedures
- alignment procedures

to handle alignment information.
This presentation is about the functionality provided by the framework for creating, storing, retrieving, checking and applying alignment constants.

Outline

Alignment constants

The misalignment framework is based on objects each holding the alignment constants for a single alignable volume and the information to access that volume:

1. unique volume identifier
2. unique global index
3. delta transformation

Unique volume identifier

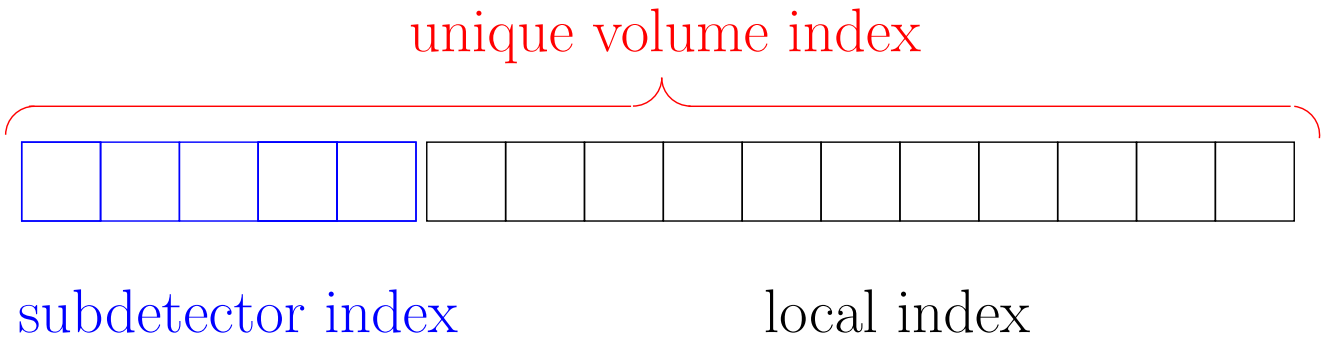
The final unique volume identifier is its path, used by TGeo to make a physical node out of it. We store a “symbolic name” which is univocally, but dynamically, associated to the volume path to allow the alignment constants to keep their validity in the case of modified volume path.

alignment constants		TGeo
symbolic volume name	$\Leftrightarrow$	volume path

alignment constants are then applicable to every ALICE geometry (should be valid till the end of the experiment with no need to reference a specific geometry version).

Unique index

A unique index is assigned to each volume; it's built up from the index of the "layer" or subdetector to which the volume belongs to and the "local index", i.e. the index of the volume itself inside the subdetector.



Indexes are useful for fast access to volumes during alignment procedures. The framework allows to easily browse the lookuptable mapping indexes to symbolic volume names.

Delta transformation

For us the default global transformation is the matrix $\mathcal{G}$ which, as in TGeo, transforms the local vector $\vec{l}$ (giving the position in the local reference system) into the global vector $\vec{g} = \mathcal{G}\vec{l}$ (giving the position in the global (or master) reference system, "MARS")

The default local transformation matrix is the matrix $\mathcal{L}$ which transforms a local vector $\vec{l}$ into the corresponding vector in the mother volume RS, $\vec{m} = \mathcal{L}\vec{l}$. Thus: $\vec{g} = \mathcal{G}\vec{l} = \mathcal{M}\vec{m} = \mathcal{M}\mathcal{L}\vec{l}$

Let's use the superscript ^{a} to denote the global $\mathcal{G}^a$ and local $\mathcal{L}^a$ matrices in the aligned geometry.

For the global delta transformation $\mathcal{A}$ – the correction to be applied to the default global transformation in order to get the aligned global transformation – we have:

$$\mathcal{G}^a = \mathcal{A}\mathcal{G} = \mathcal{A}\mathcal{M}\mathcal{L}$$

And similarly for the local delta transformation $\mathcal{A}^l$ – the transformation to be applied in the local RS in order to obtain aligned positions from default positions:

$$\mathcal{G}^a = \mathcal{G}\mathcal{A}^l = \mathcal{M}\mathcal{L}\mathcal{A}^l = \mathcal{M}\mathcal{L}^a$$

.

The alignment constant stored by the framework are the global delta transformation $\mathcal{A}$.

Why the global delta $\mathcal{A}$ instead of the local delta $\mathcal{A}^l$?:

- for direct application of the matrix during digitization and reconstruction
- to avoid ambiguities
  - there is sometimes more than one local RS (one in simulation, one in reconstruction ...)
  - the local RS can change as the geometry does

Nevertheless the alignment constants can also be passed referring to the local delta transformation. This interface is much more frendly in most use cases (e.g. rotation around fixed point in local RS) but requires users to be aware of the default local RS.

The system will then use the relation: $\mathcal{A} = \mathcal{G}\mathcal{A}^l\mathcal{G}^{-1}$
to convert the constants to the global delta transformation and store them.

Delta transform

The user can choose to use either:

- the TGeoMatrix or
- the 6 parameters (3 shifts, 3 euler angles)

defining the delta transformation, both

- as arguments of the constructor or setter method and
- as data members to be stored by the object (two alignment constant classes deriving the functionality from the same base class).

The 3 angles stored are the 3 euler angles in the xyz convention (roll-pitch-yaw).

Outline

Underlying ROOT geometry functionality

The misalignment framework of ALICE is based on the ROOT geometry package ("TGeo") and takes advantage of most of its functionality. In particular:

- physical nodes: allow redefinition of the local positioning matrix of the last node in the branch
  - TGeo stores and accesses the complete list of transformations for the whole branch only for volumes declared as "physical nodes"
  - Alignment of the node is done by applying the local delta transformation to the local default matrix of the last logical node in the path; the default matrix is still kept.
- the overlap checker
- geometry IO (read geometry from/dump geometry to file)
- functionality and performance in building, checking and navigating the geometry.

Underlying ROOT geometry functionality

- assemblies: unions of several volumes positioned with respect to a common local frame $\rightarrow$ have no shape container, need no material/medium (a point inside the assembly is always inside one of the components)
  - used not only during geometry creation, but also to optimize navigation
  - allow to create an additional hierarchical level in a flat structure of objects without defining a physical container
  - possibility to convert to assemblies existing volumes
  - powerful tool to avoid proliferation of unphysical overlaps (“MANY volumes”, geant3 unions ...)

Outline

Protection against overlaps

- After applying the complete set of alignment constants the framework checks for overlaps/extrusions exceeding a fixed threshold by means of the overlap checker of TGeo.
- This is presently a partial protection, due to some rare overlap topologies not detected.
- Check of complete geometry takes $\sim$10 seconds.

Several levels' alignment

Coherent application of alignment constants referring to nested volumes is achieved by making sure that misalignment is applied to container volumes first and only later to contained volumes, looking at the level of the volume in the hierarchy.

Outline

Interface to the Conditions DB

- The conditions DB automatically takes care of versioning of objects which are put in and, if not otherwise specified, automatically returns, for the query of the objects of a given subdetector, the last version available.
- Each alignment object is retrieved by specifying
  - “three-levels” subdirectory (subdetector - alignment - type of alignment)
  - run range validity
  - alignment conditions version
- Conditions DB stores also file with geometry for reference purposes

- By default simulation and reconstruction load set of alignment constants found on default DB instance
- Alignment constants are stored in arrays per subdetector
- Access to the conditions DB is done only once per simulation or reconstruction run
- The user can ask to load alignment conditions from different DBs for different subdetectors
- The user can choose to directly pass a given set of alignment constants in memory, without CDB queries ...
- System presently under stress for PDC06 with three different sets of alignment constants (zero-, residual- and full-misalignment).

Outline

Conclusions

The framework provides the functionality described in this presentation aiming at:

- being consistent and unambigous while
- offering a flexible user interface

With this in mind the framework's main goal is simplicity:

- one framework for different purposes (same type of information stored as objects of the same type);
- use of available underlying functionality (TGeo);
- no duplication of geometry (transport, simulation, tracking, alignment procedures all see the same geometry);
- transparent and centralized storing and versioning of alignment related objects.