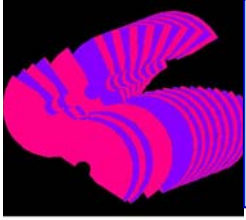


# Geometry framework

- **Getting misalignments into the geometry service**
- **Using misalignments in LHCb SW**
- **Using CondDB and UpdateManager**
- **Requirements to users**
- **Next steps**
- **Open questions**

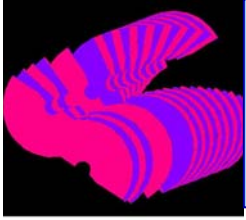
**Juan P. Palacios**  
**CERN**



# Misalignments in DetDesc

## - Recap of the problem

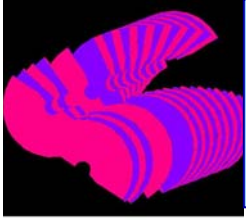
- **So far, all geometry information in LHCb DetDesc framework based on ideal positions of volumes within a hierarchy**
  - No way of applying misalignments in a consistent manner.
  - Misalignments not foreseen in framework so no easy way to do it
- **Need to Provide users of detector geometry description with geometrical information derived from two places:**
  - Nominal alignment of all the sub-components of LHCb
    - Placements of logical volumes within hierarchy (XMLDDDB)
  - Deviations from the nominal alignment
    - Thought of as conditions: could change during job



# Misalignments in DetDesc

## - Towards a solution

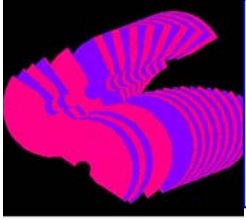
- **Already one attempt to fix this: multiple data store approach (W. Bell)**
  - Different detector data stores containing ideal, delta and combined geometries
  - Technically very difficult to implement
  - Have to really fight the framework
- **I wasn't up for a fight so I tried a more peaceful approach...**
  - Let detector elements do the work ***BEFORE*** data store is populated



# Misalignments in DetDesc

## - Detector element approach

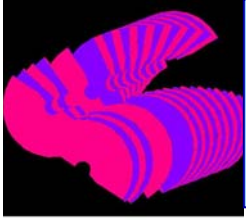
- **Basic principle:**
  - Make transformations accessible to the *Detector Element* and let it do the work
  - Keep single data store. Store misaligned detectors (and ability to provide ideal geometry information)
- **Two limits:**
  1. **ALIGNMENT:** One detector element per alignable object. Needs basically one transformation.
  2. **OTHER CONDITIONS (?):** A detector element that knows enough about its PVolume daughters (and their daughters, and their daughters' daughters, and their daughters' daughters' daughters, and ...) to assign the correct transformations to each. Interesting for conditions that are simple and numerous (calibration constants)



# Detector element approach

## - Outline

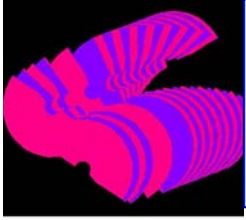
- **One detector element per alignable object**
  - Example: whole VELO, VELO halves,  $r$ - $\phi$  pairs, individual sensors
- **One “delta” transformation matrix per alignable object**
  - one to one mapping between detector elements and delta transformations
- **No changes to LHCb detector geometry description philosophy**
  - Physical volumes are placed logical volumes
  - Physical volumes know only of position within mother logical volume



## Detector element approach

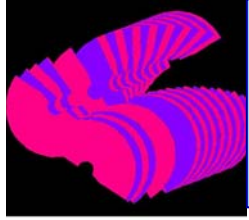
- One detector element per alignable object

- Up to sub detector SW people or alignment specialists to decide granularity
- Detector elements do not **HAVE** to be specialised
  - Not necessary to write dedicated detector elements
  - Enough to have “**detelemref**” in XML structure
  - Need to have associated logical volume ***but not necessary to have associated solid***
- Detector element has pointer to IGeometryInfo
  - As its name indicates, this holds all the geometry information related to that detector element



## Detector element approach - Aside: IGeometryInfo

- **Pointer accessible through DetectorElement::geometry() methods**
- **Initialised from XMLDDDB <geometryinfo/>**
  - Logical volume name, physical volume support and path,...
- **Has local to global and global to local transformations and a host of other services**
  - See [http://lhcb-release-area.web.cern.ch/LHCB-release-area/LHCB/doc/html/class\\_i\\_geometry\\_info.html](http://lhcb-release-area.web.cern.ch/LHCB-release-area/LHCB/doc/html/class_i_geometry_info.html)
- **Natural place to incorporate misalignments**
  - Each detector element will now have ideal and misaligned geometry information
  - User code should remain the same

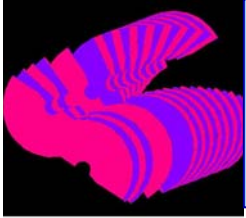


# Detector element misalignments

## - Implementation

- **Conditions catalogue**
  - Need paths of conditions plus information to convert stored information into transformation matrices (and whatever else is needed)
- **Condition class: *AlignmentCondition***
  - Class to hold an alignment delta transformation and related info.
- **New *GeometryInfo* implementation**
  - Must use alignment deltas wisely
- **Example: the VELO**





# Implementation

## - Conditions catalogue

- **Added “condition” attribute to geometryinfo in structure.dtd**

- Only necessary to add an

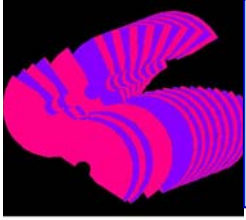
```
<geometryinfo condition="somePath" />
```

in each <detelem .... /> definition

- Then store the condition somewhere:

```
<condition classID="6" name="somePath">  
  <paramVector name="dPosXYZ" type="double"> 0. 0. 0.</paramVector>  
  <paramVector name="dRotXYZ" type="double"> 0. 0. 0.</paramVector>  
</condition>
```

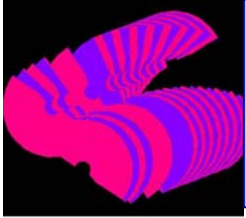
- **Information necessary to construct tranf. matrix**
- **VELO example Written in test XML files**
- **Can store as XML strings in CondDB: keep format the same!**



# Implementation

## - AlignmentCondition

- **Very simple class holding delta transformation related to an alignable object**
  - In object's mother's frame
  - Also contains inverse transformation
  - Is a ValidDataObject so knows about validity ranges
- **At the moment ONLY contains transformation + inverse, and a method to set a new transformation**
- **A basic building block, but obviously could evolve...**



# Implementation

## - AlignmentCondition creation

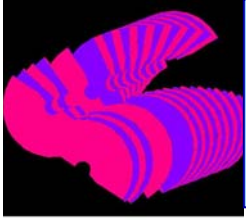
- **Constructed from condition path and dedicated XML “converter”**
- **classID = 6 maps to XmlAlignmentConditionCnv**
  - This is a little template which simply instantiates the right kind of condition
- **AlignmentCondition has access to the parameters and constructs the matrices**
  - ***Could add more information if requested!***

```
<condition classID="6" name="/dd/Conditions/LHCb/myDetector/Module67">  
  <paramVector name="dPosXYZ" type="double"> 0. 0. 0.</paramVector>  
  <paramVector name="dRotXYZ" type="double"> 0. 0. 0.</paramVector>  
</condition>
```

- **Constructed and accessed via data service:**

```
SmartDataPtr<AlignmentCondition> cond(datasvc(),  
  "/dd/Conditions/LHCb/myDetector/Module67");
```

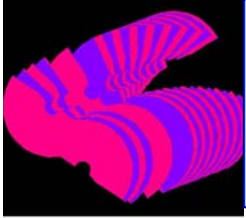
- **In practice, constricted in new GeometryInfo**



# Implementation

## - New GeometryInfo

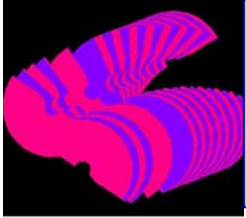
- **Handles both ideal and delta transformations:**
  - DetectorElement public interface stays the same.
  - IGeometryInfo toGlobal and toLocal methods ***now deal with combined ideal + delta transformations***
  - Methods to get ideal geometry have been added
  - Possible for users to update delta matrix
  - Easy to “refresh” state of new GeometryInfo. All caching controlled from one method, aptly named ***cache()***
  - GeometryInfo accesses CondDB (or test XML file), generates all necessary matrices
- **VELO example implementation ready and tested with XML conditions catalogue**



# Implementation

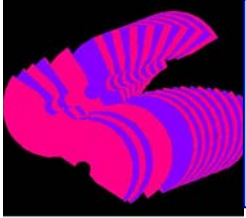
## - GeometryInfo details

- **Constructor uses path to AlignmentCondition**
  - Gets condition from data store (XML file or CondDB)
- **Scans down tree picking up parent transformations**
  - Iterative procedure finding GeometryInfos of support detector elements
  - Stores ideal and delta transformations for each level in vectors
    - Allow to re-calculate after updates
  - Combines ideal matrices to get ideal case local to global
  - Combines all matrices to get local to global with misalignments
- **“Local” misalignment matrix can be updated by user**
  - Re-calculates “global” matrix automatically
  - At the moment this is de-coupled from automatic update mechanism



# Using misalignment information

- **Keep currently used `DetectorElement` interface**
  - `DetectorElement::geometry()` points to new `GeometryInfo`
  - Standard transformation methods now deal with misaligned geometry
    - User code will automatically get misaligned geometry transformations if unchanged
  - New methods allow to perform ideal transformations
  - Delta matrix also available – and can be modified
- **BEWARE: code using `DetectorElement::geometry()` will now get MISALIGNED GEOMETRY automatically!**
  - **To get misaligned geometry, new code must use detector elements**



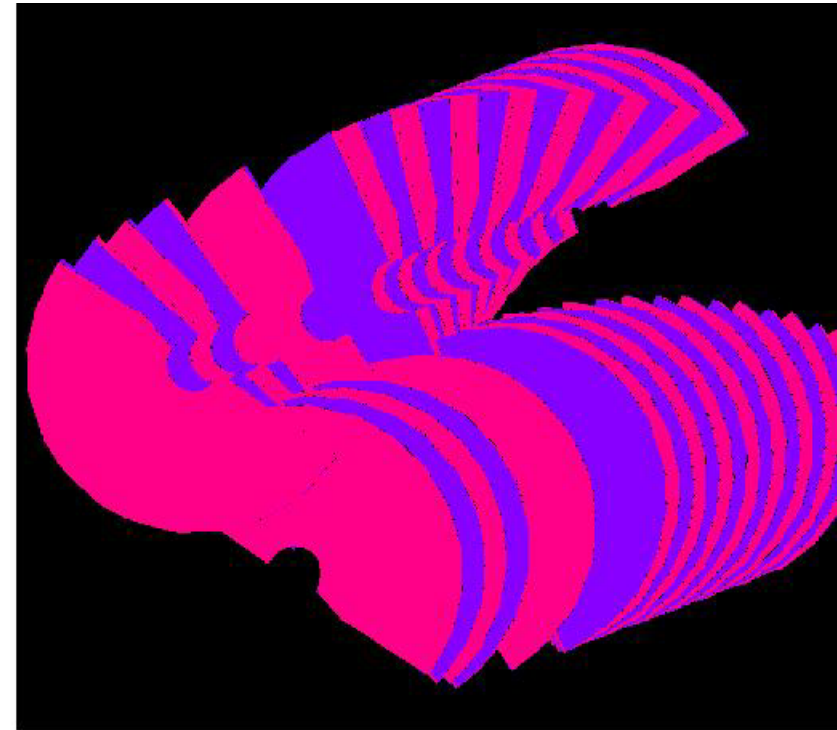
# Using misalignments in LHCb SW

## - Panoramix case

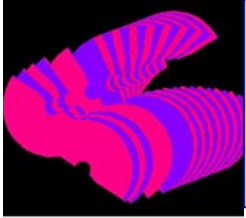
- **Uses detector element to plot volume**
  - Automatic access to misalignment
- **BUT does not get misalignments for daughters**
  - Must specify *each* detector element

```
/dd/Structure/LHCb/Velo/VeloLeftModule13  
/dd/Structure/LHCb/Velo/VeloLeftModule15  
/dd/Structure/LHCb/Velo/VeloLeft/Module17
```

**Plots with misalignments up to the module level but NOT sensor misalignments**



**Open VELO with Z-dependent  $\phi$  misalignment from test XML conditions file**



# Using misalignments in LHCb SW

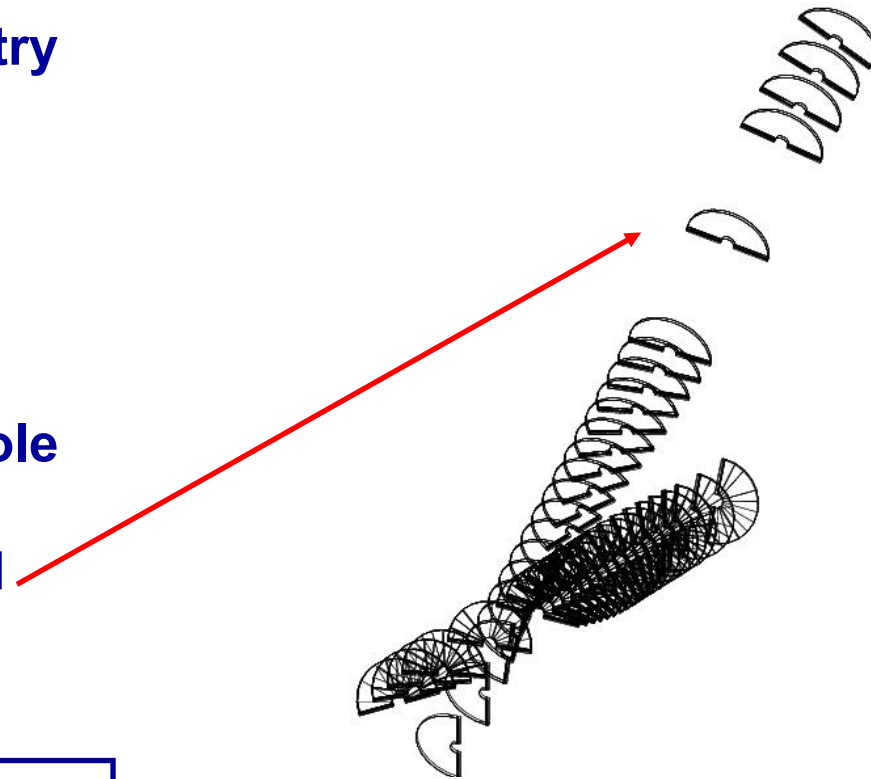
## - Gauss case

- In Gauss add detector geometry stream in options file:

```
Geo.StreamItems +=  
  {"/dd/Structure/LHCb/Velo"}
```

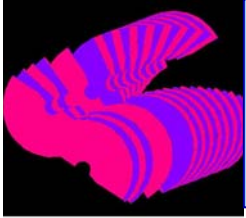
- This would disregard any misalignments other than whole VELO
- To get all misalignments need to add leaves explicitly as in Panoramix

**But what do do about PV without detector element? Or PV in wrond place in hierarchy? Will need to write some new GiGa code...**



**Geant4 Open VELO with Z-dependent  $\phi$  misalignemnt from test XML conditions file**

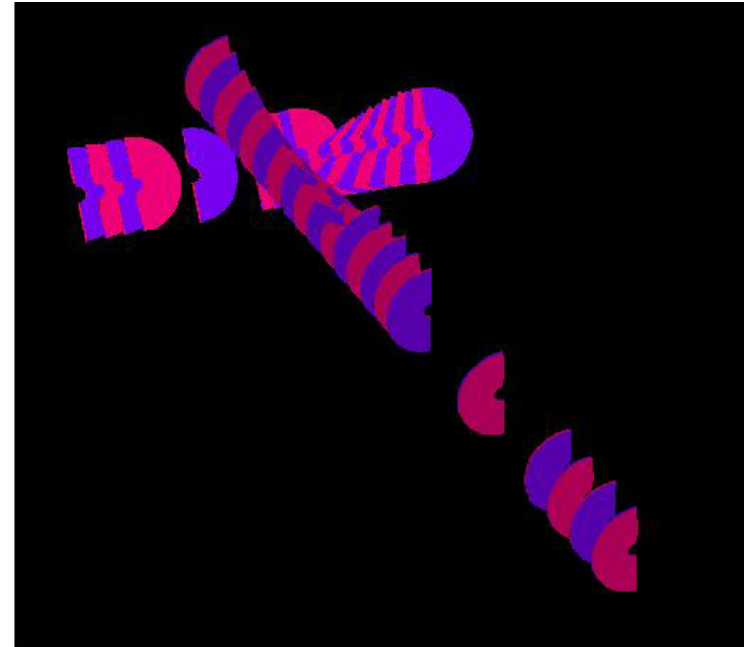




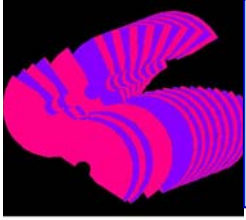
## Using misalignments in LHCb SW

### - General remarks

- All information is available, through the detector element, via the new GeometryInfo
- As long as daughter detector elements are accessed, and NOT daughter physical volumes, all should be automatic



Open VELO with Z-dependent  $\phi$  misalignemnt and rotation about X axis.  
From test XML conditions file

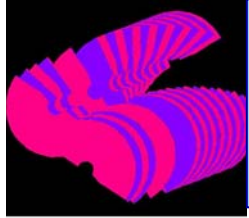


# Using misalignments in LHCb SW

## - More general remarks

- **Must bear in mind the misalignments are *conditions* and could change in a job**
  - Caching of geometrical information should be done in a controllable manner
  - Update mechanisms will exist, but depend on users associating methods to conditions
  - Up to users to ensure that re-running a condition-dependent method really does what it needs to
- **Remember many volumes are in close proximity**
  - What about possible overlaps?
  - How to avoid them?

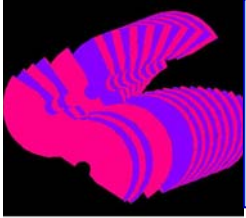
**This framework gives total freedom to move things... up to users to move them intelligently!**



# Using CondDB and UpdateManager

## - Accessing CondDB

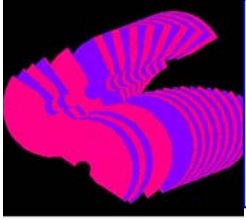
- **Reading and writing to CondDB**
  - Would like to say “it is easy”
  - Can’t say yet
    - I haven’t had the time to do it (although have pretty good examples from **Marco Clemencic**... thanks!)
  - But it looks easy!
- **Important points:**
  - Data store interface: just requires appropriate paths to conditions in XML
  - Store XML strings: can keep the same format as have now in test XML file



# Using CondDB and UpdateManager

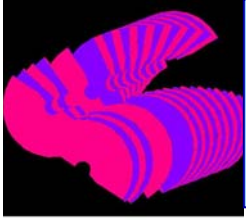
## - Updating information

- **How to update all necessary information when a condition changes**
  - Again, nice framework from Marco Ci. but I haven't had the time to use it yet
- **Basically, register function/condition pairs to UpdateManager**
  - It takes care of re-running function whenever condition changes
- **Important: authors of code must ensure changes get propagated wherever needed**
  - I take care of AlignmentCondition, GeometryInfo, default DetectorElement and any other "framework" code
  - For the rest, up to you!



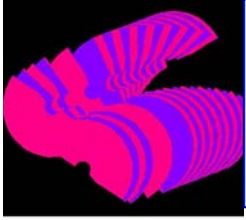
# Requirements from sub-detector SW

- **One detector element per alignable object**
  - Can be default (no code to write)
  - Must have path to an alignment condition
  - In current scheme this means a condition in the XML `<geometryinfo />` definition of the detector element. This contains the correct path in the data store
- **A catalogue of conditions with one-to-one mapping to detector elements**
  - In absence of condition identity matrix is created so current XML descriptions still work for ideal
- **A geometry description where the alignable objects can be associated to an LVolume**
- **A controlled and clear caching of condition-dependent geometry information**
  - Control all caching from a minimal number of methods as they will have to be registered somewhere
  - Ensure that if these methods are run everything is updated in consistent manner



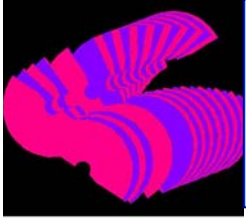
# Requirements from sub-detector SW /2

- **You are required to tell me what you require!**
    - Need to know about current and future uses of geometry information
  - **AlignmentCondition**
    - Currently a matrix wrapper with time validity
    - Could make it more interesting: errors on alignment parameters?
    - More ways to construct a transformation?
      - At the moment, just six numbers...
  - **GeometryInfo**
    - Any other information/functionality required?
  - **DetectorElement**
    - Anything more clever needed (besides returning pointer to GeometryInfo)?
  - **Likely uses**
    - Simulation
    - Tracking
    - Alignment
    - Others?
- Not all require full det. Description...**



## Next steps

- 1. Test algorithm to populate CondDB with snail matrices**
- 2. Retrieving snail matrices from CondDB**
  1. Simple test algorithm
  2. DetDesc (via appropriate paths in XML)
- 3. Test algorithm to manually change matrices in DetDesc**
- 4. Incorporate UpdateManager into GeometryInfo, DetectorElement**
- 5. Write dedicated GiGaStream to use misalignments in simulation**



## Recap: (some) open questions

- **More information in AlignmentCondition?**
- **More ways of defining transformations?**
- **IGeometryInfo interface?**
- **DetectorElement interface?**
- **Simulation: overlaps?**
- **Transport service: overlaps?**
- **Questions?**