Volumes

Vanya Belyaev[*]
ITEP, Moscow

February 27, 2000

Contents

1 General features of geometry tree

The construction of geometry tree withon $\mathcal{GAUDI}$ framework is based on the following postulates:

- The geometry tree is constructed from *Logical Volumes* and *Physical Volumes.*
- There is no "up-links" in the geometry tree. It means that each node have no information about the "up" (or "parent", "mother") node.
- Each *Logical Volume* has an information about its "down" ("children") nodes, represented by *Physical Volumes*
- Each *Logical Volume* has information about its shape and dimensions ("solid").
- Each *Logical Volume* has an access to the information about the material content.
- Neither *Logical Volumes* nor *Physical Volumes* have no any information about its absolute position in the space.
- *Logical Volumes* have no any information about its own position relative to other *Logical Volumes.*
- Each *Physical Volume* has an information about the its position inside the mother ("parent") *Logical Volume.* It is the only available geometry information in the whole tree.

[*]E-mail:`Ivan.Belyaev@itep.ru`

- All boolean operations on the level of *Logical Volumes* and *Physical Volumes* are strickly forbidden[1]. If one need to perform boolean operations, one should rely on boolean operations on the level of *Solids*. It is one of the most essential postulates of adopted geometry structure.

The geometry tree which fulfills all these postulates represent a very effective, simple and convinient tool for description of the geometry. Such tree is easily formalized. This tree have many features which are similar to the features of the geometry tree used within *Geant4* toolkit and it could be easily transformed to the *Geant4* geometry description.

There exist several general sequences derived from these base postulates:

- According to this schema the top-level *Logical Volume* (presumably the experimetal hall, or cave, or the whole *LHCb* detector) defines the absolute coordinate reference system. Frankly speaking the null-point (0,0,0) in the so called "Global Reference System" is just the center of the top "Logical Volume".
- All geometry calculations, computations, inputs and outputs, performed with the usage of the *Logical Volume* are in the local reference system of this *Logical Volume*.
- All geometry calculations, computations, inputs and outputs, performed with the usage of the *Physical Volume* are in the local reference system of its parent *Logical Volume*.

Sometimes one needs to get more effective way of extraction information from the geometry tree or to perform unique location of the point in the geometry tree. For these purposes a simplified detector description tree is introduced into the *GAUDI* framework[2].

2 Logical Volumes

2.1 *ILVolume* interface

This abstract interface is designed to fulfill the postulates of the geometry tree.

Here the main features and methods of logical volume interface *ILVolume* are described:

- `const   ISolid*       solid       () const ;`
  return the solid, associated with the Logical Volume
- `const   std::string&  materialName() const ;`
  return the material(by name) , associated with the Logical Volume
- `const   Material*     material    () ;`
  return the C++ pointer to the material, associated with the Logical Volume
- `ILVolume::ReplicaType noPVolumes  () const ;`
  return number of Physical(positioned) Volumes inside given Logical Volume
- Access to the contained physical volumes

[1]It is equivalent to the absence of `'MANY'` flag in *GEANT3* toolkit

[2]Within *Geant4* toolkit there exists 2 approaches for solving the same problem: *Read-Out-Geometry Tree* and *Navigator*. Our approach is quite close to the combined usage of both.

  - `IPVolume* operator[]( const ILVolume::ReplicaType& index );`
    return C++ pointer to the daughter *Physical Volume* by index (replica number)
  - `IPVolume* operator[]( const std::string& name ) const;`
    return C++ pointer to the daughter *Physical Volume* by it's name

- More conventional access to the daughter *Physical Volumes*:
  - `IPVolume* pvolume( const ILVolume::ReplicaType& index ) ;`
    return C++ pointer to the daughter *Physical Volume* by index (replica number)
  - `IPVolume* pvolume( const std::string& name ) const;`
    return C++ pointer to the daughter *Physical Volume* by it's name

- Access via iterators (very useful and practical in conjunction with STL algorithms):
  - `ILVolume::PVolumes::iterator       pvBegin()       ;`
  - `ILVolume::PVolumes::const_iterator pvBegin() const ;`
  - `ILVolume::PVolumes::iterator       pvEnd  ()       ;`
  - `ILVolume::PVolumes::const_iterator pvEnd  () const ;`

- Traverse the down-links and transfrom the sequence of replica numbers to the sequence of C++ pointers to *Physical Volumes*. These methods are indeed primary methods for unique locations of points withis geometry tree.

  -
    ```
    StatusCode traverse( ReplicaPath::const_iterator pathBegin,
                         ReplicaPath::const_iterator pathEnd  ,
         PVolumePath&                volumePath );
    ```

  -
    ```
    StatusCode traverse( const ReplicaPath&  replicaPath,
         PVolumePath&        volumePath ) ;
    ```

- `bool isInside ( const HepPoint3D& LocalPoint ) const;`
  return *true* if the *LocalPoint* in the local reference system of the *Logical volume* is inside the *Logical Volume*.

- Try to localize the *LocalPoint* in the local reference system inside the daughter volumes. Methods returns either the sequence of C++ pointers to the daughter *Physical Volumes* or the sequence of the *replica numbers* . Both methods are recursive and therefore could be *very* slow for complicated multi-level geometry. Sometimes the such very detailed information is not nesessary and therefore to speedup the method one should choose the appropriate level of deepth of the tree to be traversed.

  -
    ```
    StatusCode belongsTo( const HepPoint3D& localPoint ,
           const int         level      ,
           PVolumePath&      volumePath );
    ```

  -
    ```
    StatusCode belongsTo( const HepPoint3D& localPoint ,
           const int         level      ,
           ReplicaPath&      replicaPath );
    ```

- Overloaded print function to *std::ostream*

- `const ILVolume* reset() const ;`
  This methos performs the full reset to the initial state of the *Logical Volume*. All temporary values are cleared. It also triggers the *reset()* method for all daughter *Physical Volumes* and for its own *Solid*.

- Intersection of the *Logical Volume* with the line. Line is to be parametrized in the local reference system of the *Logical Volume* by initial point on the line and direction vector: $\vec{\mathbf{x}}(t) = \vec{\mathbf{p}} + \vec{\mathbf{v}} \times t$, where t is a parameter. Both methods fill the output container with intervals of parameter values associated with the material. Both methods return the size of this outpout container. Both methods are recursive and therefore could be *very* slow for multilevel complicated geometry. To speed-up the methods one could use the appropriate value of threshold parameter *Threshold.* If the estimated contribution of the some volume from the chain to the total radiation thickness (in the units of radiation length) is less then value of parameter *Threshold* this volume do not contribute to the output container. For the second method only values of line parametrization parameter t from the region *tickMin* $\leq t \leq$ *tickMax* are considered.

  -
    ```
                 /* initial point at the line    */
    unsigned int intersectLine( const HepPoint3D& Point ,
                 /* direction vector of the line */
                const Hep3Vector& Vector        ,
                 /* output container             */
                Intersections   & intersections ,
                 /* threshold value              */
                const double      threshold     ) ;
    ```

  -
    ```
                 /* initial point at the line    */
    unsigned int intersectLine( const HepPoint3D& Point ,
                 /* direction vector of the line */
                const Hep3Vector& Vector        ,
                 /* output container             */
                Intersections   & intersections ,
                 /* minimum value of Tick        */
                ISolid::Tick      tickMin       ,
                 /* maximal value of Tick        *.
                ISolid::Tick      tickMax       ,
                 /* threshold value              */
                const double      Threshold     ) ;
    ```

  These methods are essential for estimation of the distance in the units of the radiation length between 2 points.

2.2 *class LVolume*

The notion of the *Logical Volume* is implemented within $\mathcal{GAUDI}$ framework via the *class LVolume.* The essential features of this object are:

- *LVolume* represents an *identifiable object.* It inherits from *class DataObject*, and therefore it is identified within $\mathcal{GAUDI}$ Transiend Store by unique name ("path").
- It implements an abstract interface *ILVolume.*
- It also implements an abstract interface *IValidity* (not mentioned above).
- Some methods of *LVolume* class could throw the exception via *LVolumeException*, *SolidException* and/or *PVolumeException* classes.
- *class LVolume* has three constructors:

  - The default constructor as a quite fragil is declared to be *private*. It could be invoked only from methods of appropriate *friend* class. Currently *class LVolume* has only one *friend* - *class XmlLVolumeCnv*.
  - Public constructors are safe. They require the C++ pointer to *ISolid* to ve valid, overwise they throw *LVolumeException*. *class LVolume* takes the fullresposibility for its own *Solid*. The deletion of *Solid* is in the destructor of *class LVolume*.

```
      LVolume( const std::string& name            ,
       ISolid*           Solid           ,
               const std::string& material        ,
               const ITime&       validSince      ,
               const ITime&       validTill       ,
               IDataProviderSvc*  dataService = 0 ,
               IMessageSvc*       messService = 0 );
    ///
      LVolume( const std::string& name            ,
       ISolid*           Solid           ,
               const std::string& material        ,
               IDataProviderSvc*  dataService = 0 ,
               IMessageSvc*       messService = 0 );
```

- *class LVolume* take a full responsibility for creation and deletion of the *Physical Volume*.

```
  IPVolume* createPVolume( const std::string&    PVname ,
const std::string&    LVnameForPV    );
  IPVolume* createPVolume( const std::string&    PVname ,
const std::string&    LVnameForPV    ,
      /* position of PVolume inside LVolume */
        const HepPoint3D&     position );
  IPVolume* createPVolume( const std::string&    PVname ,
const std::string&    LVnameForPV    ,
     /* position of PVolume inside LVolume  */
        const HepPoint3D&     position,
     /* rotation to be applied              */
        const HepRotation&    rotation );
```

3 Physical Volumes

The notion of the *Physical Volume* within the adopted geometry schema is extremply primitive - it is just *Logical Volume* which is *positioned inside its mother Logical Volume*. Frankly speaking it is just a pair of name of the *Logical Volume* to be positioned inside the mother *Logical Volume* and the corresponding transformation matrix from the local reference system of mother *Logical Volume* to the local reference system *Logical Volume* to be positioned.

3.1 *IPVolume* interface

This abstract interface is designed to fulfill the postulates of the geometry tree. The essential methods provided via this interface are:

- `const std::string&     name() const ;`
  return the name of the *Physical Volume*. This name should be unique inside

the given mother *Logical Volume*. The same name can be used inside different mother *Logical Volumes*. This name is used in navigation of *DetectorElement* ("name-path").

- `const std::string&     lvolumeName() const ;`
  return the name of associated *Logical Volume*.
- `ILVolume*        lvolume    () const;`
  return C++ pointer to the associated *Logical Volume*.
- `const HepTransform3D&  matrix() const ;`
  return the transformation matrix from local reference system of mother *Logical Volume* to the local reference system of the *Logical Volume*, associated with given *Physical Volume*.
- `const HepTransform3D&  matrixInv() const ;`
  return the transformation matrix from local reference system of the *Logical Volume* associated with given *Physical Volume* to the local reference system of the mother *Logical Volume*.
- `HepPoint3D toLocal ( const HepPoint3D& PointInMother ) const ;`
  transform point from reference system of the mother *Logical Volume* to the reference system of the *Logical Volume*, associated with given *Physical Volume*
- `HepPoint3D toMother ( const HepPoint3D& PointInLocal ) const ;`
  transform point from local reference system of the *Logical Volume* associated with given *Physical Volume* to the reference system of the mother *Logical Volume*,
- `bool isInside( const HepPoint3D& PointInMother ) const ;`
  return *true* if the point in the reference system of mother *Logical Volume* is inside the *Logical Volume*, associated with given *Physical Volume*.
- Overloaded print-functions and stream operators to *std::ostream*.
- `const IPVolume* reset() const ;`
  this methos performs the reset of the *Physical Volume* to the initial state. All temporaries are cleared. It triggers the *reset()* method for the associated *Logical Volume*.
- Intersection with the line in the space. There exist two methods, which are identical to the methods in the *ILVolume* interface, and they just provide the delegation to the associated *Logical Volume*.

3.2 *class PVolume*

The notion of the *Physical Volume* is implemented within *GAUDI* framework via the *class PVolume*. The essential features of this object are:

- *PVolume* is not *identifiable object*.
- It implements an abstract interface *IPVolume*.
- Some methods of *PVolume* class could throw the exception via *PVolumeException*, *LVolumeException* and/or *SolidException* classes.
- *class PVolume* has no public constructors. Creation and destroy of objects are under the constrol of the *friend class LVolume*.

```
  PVolume( const std::string& PhysVol_name ,
     const std::string& LogVol_name ,
/* position in Mother Reference Frame!*/
     const HepPoint3D&  Position ,
/* rotation with respect to Mother Reference Frame*/
     const HepRotation& Rotation       = HepRotation() ,
     IDataProviderSvc*  dataService    = 0 ,
     IMessageSvc*       messageService = 0 );
```