THE VIRTUAL GEOMETRY MODEL

I. Hřivnáčová, IPN, Orsay, France

Abstract

In order for physicists to easily benefit from the different existing geometry tools used within the community, the Virtual Geometry Model (VGM) has been designed. In the VGM we introduce the abstract interfaces to geometry objects and an abstract factory for geometry construction, import and export. The interfaces to geometry objects were defined to be suitable to describe "geant-like" geometries with a hierarchical volume structure.

The implementation of the VGM for a concrete geometry model represents a small layer between the VGM and the particular native geometry. At the present time this implementation is provided for the Geant4 and the Root TGeo geometry models.

Using the VGM factory, geometry can first be defined independently from a concrete geometry model, and then built by choosing a concrete instantiation of it. Alternatively, the import function of the VGM factory makes it possible to use the VGM directly with native geometries (Geant4, TGeo). The export functions provide conversion into other native geometries or the XML format.

In this way, the VGM surpasses one-directional geometry converters within Geant4 VMC (Virtual Monte Carlo): roottog4 and g4toxml, and automatically provides missing directions: g4toroot, roottoxml. To port a third geometry model, then providing the VGM layer for it is sufficient to obtain all the converters between this third geometry and already ported geometries (Geant4, Root).

The design and implementation of the VGM classes, the status of existing implementations for Geant4 and TGeo, and simple examples of usage will be discussed.

ARCHITECTURE

The VGM concept

The Virtual Geometry Model (VGM) has been developed as a generalization of the existing converters roottog4, g4toxml provided within Geant4 VMC [1], when new directions: g4toroot, roottoxml were asked for by users.

Instead of adding other one-way converters and multiplying the implementations, the abstract layer to geometry has been defined and the geometry models have been "mapped" to this generalized scheme. Once this is done, the geometry objects in these integrated models can be handled in the same way.

This new concept is demonstrated in Fig. 1

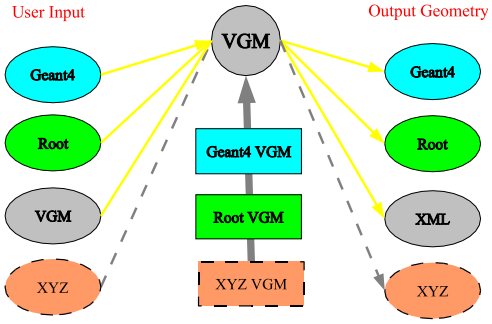

Figure 1: The VGM concept

Components

The main VGM components are shown in Fig. 2 together with their dependencies on the external software.

In the VGM, abstract interfaces to geometry objects and abstract factories for geometry construction, as well as import and export, are introduced. These interfaces are included in the core VGM package. Since CLHEP [2] has been chosen for representation of the 3D transformations, the VGM package requires this library.

The implementation of the VGM for a concrete geometry model represents a layer between the VGM and the particular native geometry. At present, this implementation is provided for the Geant4 [3] and the Root TGeo [4] geometry models. The dependence on the particular geometry models is hence restricted only to its VGM layer.

Besides the geometry model specific packages, the VGM provides also the XML exporter, which is included in its own package and does not bring dependency on any external software.

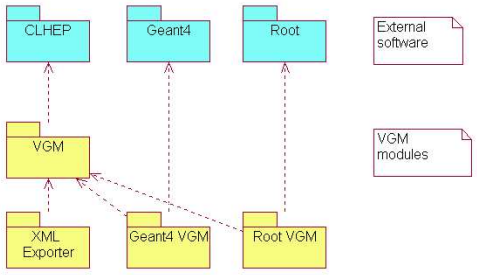

Figure 2: The VGM modules

Interfaces to geometry objects

The interfaces to geometry objects were defined with the intention to be suitable for a desription of “geant-like” geometries with a hierarchical volume structure. The basic entities sufficient for such geometry description have been identified: solid, volume and placement to describe volumes hierarchies; and element, material and medium to describe material properties.

The solid and placement entities can have more specifications. In case of the solid it is reflected by introducing a specific interface for each solid type (box, tubs, cons, ...), while in the placement case, both types introduced (simple placement and multiple placement) are described with a single interface.

The class diagram for the VGM implementation of the box object, see Fig. 3, demonstrates the mapping between the objects in the native geometry and the interfaces introduced in the VGM. The same approach has been applied to the other geometry entities. In case a common implementation of some functions (imposed by the interface) is wanted, an abstract base class providing this implementation is defined first and the class in the VGM layer for a particular geometry model makes a specification of this abstract base class only.

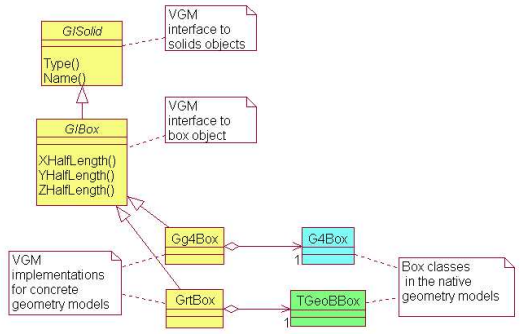

Figure 3: The class diagram of the box solid in the VGM

Abstract factories

The VGM abstract factories, GIFactory and GIMaterialFactory, provide functions for geometry construction, import and export. While the functions for geometry construction and import are specific to a geometry and have to be provided by the geometry specific layers, the export function could be implemented in a common way and is provided in the abstract base classes (GVFactory, GVMaterialFactory) in the VGM package.

The class diagram for the VGM factories is shown in Fig. 4

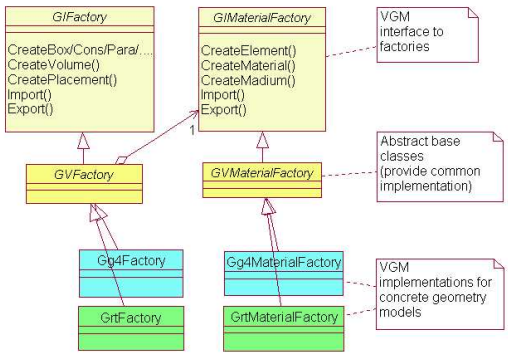

Figure 4: The class diagram of the VGM factories

USE OF VGM

Geometry conversions

To convert native geometry from one geometry model to another, the geometry has to be first imported in the VGM (the native geometry objects are mapped to the VGM interfaces) using the concrete VGM factory for this geometry model, and then exported using the VGM factory for the other geometry model. In Example 1 the conversion of geometry from Geant4 to Root is demonstrated.

Example 1: Converting geometry from Geant4 to Root via the VGM

```
#include ”Gg4Factory.h”
#include ”GrtFactory.h”
#include ”TGeoManager.h”

// Import Geant4 geometry to VGM
Gg4Factory g4Factory;
g4Factory.Import(physiWorld);
    // where physiWorld is of G4VPhysicalVolume* type

// Export VGM geometry to Root
GrtFactory rtFactory;
g4Factory.Export(&rtFactory);
gGeoManager->CloseGeometry();
return rtFactory.World();
    // returns Root top volume of TGeoVolume* type
```

Geometry construction via the VGM

The VGM interfaces can be used to define geometry independently from a concrete geometry model. The code in Example 2 builds a world box volume using the abstract VGM factory, choosing the concrete factory (Geant4 or Root VGM factory) will then build the geometry of the chosen model (Geant4 or Root).

Example 2: Geometry definition via the VGM

```
MyDetConstruction::Construct(GIFactory* factory)
{
  double ws = 10*m;
  GISolid* worldS
    = factory->CreateBox("worldS", ws, ws, ws);
        // create the world solid

  GIVolume* worldV
    = factory->CreateVolume("worldV", worldS, "Air");
        // create the world volume

  factory->CreatePlacement("world", 0, worldV, 0,
                                   0, Hep3Vector());
        // place the world volume
}
```

```
#include "Gg4Factory.h"
MyDetConstruction myDetConstruction;
Gg4Factory theFactory;
myDetConstruction->Construct(&theFactory);
        // Geant4 geometry will be built

#include "GrtFactory.h"
MyDetConstruction myDetConstruction;
GrtFactory theFactory;
myDetConstruction->Construct(&theFactory);
        // Root geometry will be built
```

Export to XML

The VGM geometry can be exported to XML in the AGDD [5] or GDML [6] format. Complying with the XML schema is embedded in the VGM XML exporter code itself, no external XML parser is then needed.

This is demonstrated in Example 3.

Example 3: Exporting geometry from the VGM factory in XML (AGDD, GDML)

```
#include "GAGDDExporter.h"
GAGDDExporter xmlExporter1(&theFactory);
xmlExporter1.GenerateXMLGeometry();
      // Export geometry to AGDD

#include "GGDMLExporter.h"
GGDMLExporter xmlExporter2(&theFactory);
xmlExporter2.GenerateXMLGeometry();
      // Export geometry to GDML
```

TESTING

The same simple geometry setups were defined via Geant4, Root and VGM to test different aspects of the VGM: Solids, Placements, Boolean Solids and Reflections (see Fig. 5, Fig. 6, Fig. 7 and Fig. 8). The test program can be then configured by a list of arguments to select the input geometry type, the selected geometry setup and the destination geometry model or XML output.

The testing procedure has been automated by a test suite shell script, which executes the test program with all possible combinations and generates the output that can be compared with the reference output.

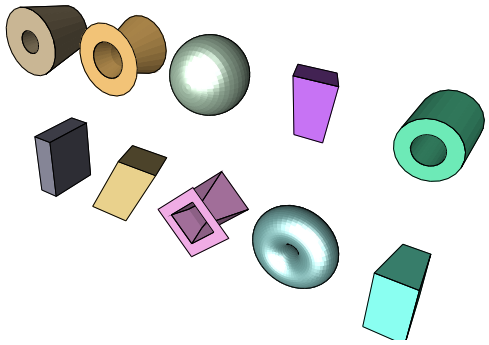

Figure 5: The geometry setup to test solids

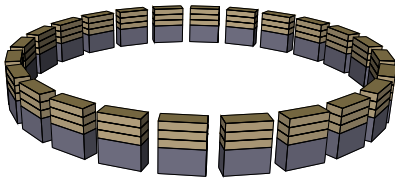

Figure 6: The geometry setup to test placements

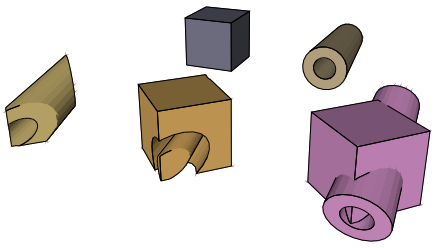

Figure 7: The geometry setup to test Boolean solids

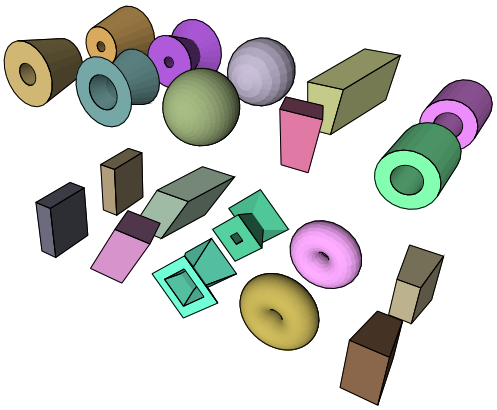

Figure 8: The geometry setup to test reflections

EXAMPLES

As the test program, being written for the purpose of testing all aspects, is rather complex, four simple examples demonstrating use of the VGM for converting native geometries are provided. The summary of the use cases demonstrated in these examples is given in Table 4.

Table 4: Summary of examples

	Use case	**Geometry source**
E1	G4 -> Root	Geant4 novice example N03
E2	Root -> G4	Root file with geometry generated in E1
E3	G4 -> XML	Geant4 novice example N03
E4	Root -> XML	Root tutorial macro rootgeom.C

PRESENT STATUS

Both VGM implementations for Geant4 and Root geometry models are well advanced, below we give the lists of supported and unsupported features for both models.

Supported features:

- Most of the solids: all CSG solids and some specific solids - G4Polycone, G4Polyhedra in Geant4 and their counterparts in Root
- Boolean solids (Geant4) and composite shapes (Root)
- Reflected solids (Geant4) and positioning with reflection (Root)
- Replicas (Geant4) and divisions (both Geant4, Root)

Unsupported:

- "Exotic" solids - solids that have no counterpart in the other geometry model
- Parameterised volumes (Geant4)
- Positions with the "MANY" option (Root)
- Boolean solids (in the XML exporter)

The first version of the tool is available at [8].

CONCLUSIONS

The VGM introduces a general approach for conversion of geometries between specific models. At present, it can be used for conversion between Geant4 and Root TGeo geometry models in both directions and conversion from both these models in XML (AGDD and GDML) formats. This gives the possibility for a user of one specific package to use the tools supported by other packages: the Virtual Monte Carlo [1] via the Root geometry package and GraXML [7] via XML.

The VGM also allows the user first to define the geometry independently from a concrete geometry model, and then to choose the concrete geometry model at run time, though this use case was not the primary goal of this tool.

REFERENCES

[1] By the ALICE Collaboration (I. Hrivnacova et al.), "The Virtual Monte Carlo", CHEP'03, La Jolla, California, March 2003, eConf C0303241:THJT006,2003. http://root.cern.ch/root/vmc/VirtualMC.html

[2] http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/

[3] S. Agostinelli, et al., "Geant4 - A simulation toolkit", NIM A 506 (2003), 250. http://wwwasd.web.cern.ch/wwwasd/geant4/geant4.html

[4] By ALICE off-line Collaboration (R. Brun et al.), "A Geometrical Modeller for HEP", CHEP'03, La Jolla, California, March 2003, eConf C0303241:THMT001,2003.

[5] Ch. Arnault et al, "Atlas Generic Detector Description in XML", CHEP'01, Beijing, September 2001, 8-001.

[6] R. Chytracek, "The Geometry Description Markup Language", CHEP'01, Beijing, September 2001, 8-009. http://gdml.web.cern.ch/gdml/

[7] J. Hrivnac, "GraXML - Modular Geometric Modeller", CHEP'03, La Jolla, California, March 2003, eConf C0303241:THJT009,2003. http://hrivnac.home.cern.ch/hrivnac/Activities/Packages/GraXML

[8] http://ivana.home.cern.ch/ivana/VGM.html