http://www.geant4.org

Update on common interfaces

Andrea Dotti (adotti@slac.stanford.edu) ; SD/EPP/Computing

U.S. DEPARTMENT OF ENERGY
Office of Science

SLAC NATIONAL ACCELERATOR LABORATORY

Status

Discussed in several meetings about geometry interface: critical step is SD.

- GDML as baseline, keep it as simple as possible

Discussed @ SLAC: first discussion about hits collection that are detector and framework independent

- Add a small G4 library in parasitic mode to existing applications

Reminder

Not discussing about a common framework/application to be used by community

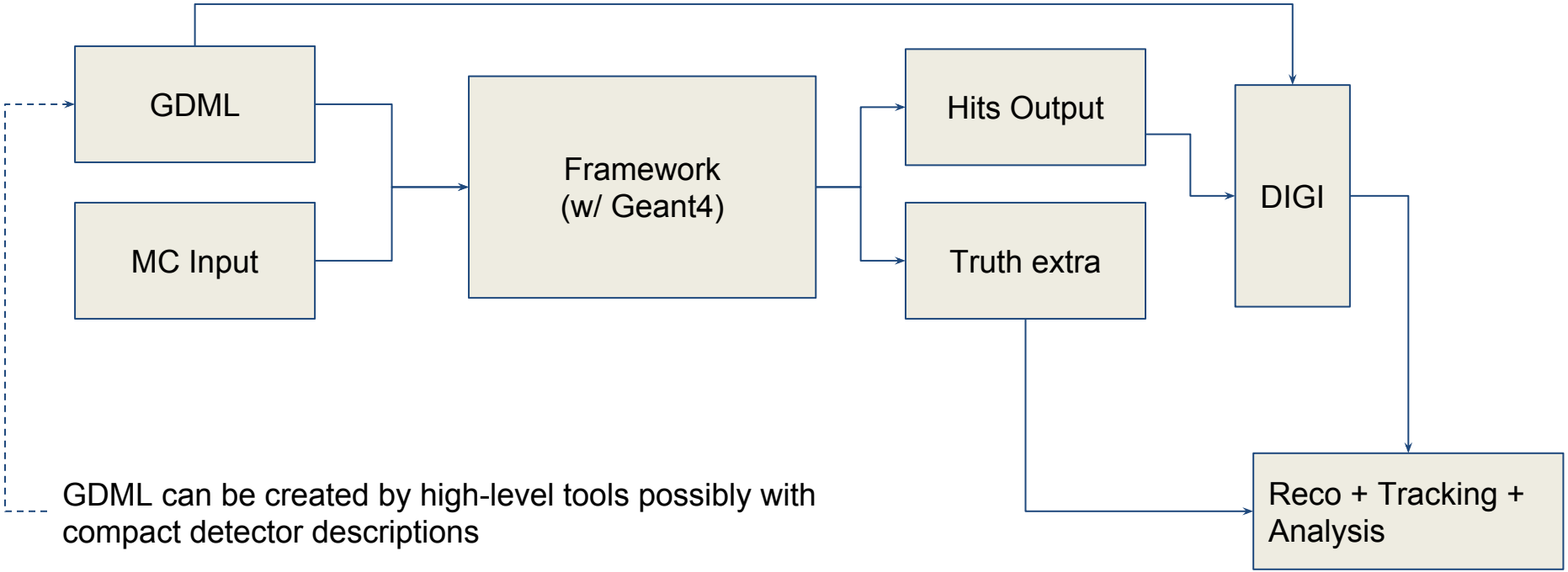
Instead focusing of **simple** interfaces formats to exchange data

Fact: if we want all current existing frameworks/applications to be player in this we have two options, ROOT or GDML

Since GDML is considered the “universal” format understood by everybody, we start from there. If you use G4 or ROOT you can at least read GDML

Introduction

Vision as discussed at SLAC



We re-discovered unix pipes and philosophy ca. 1973 :-)

Some definitions first

What is a G4Hit and what is a Sensitive Detector?

Geant4 performs simulation in steps: the atomistic-unit of the simulation: two (ordered) points in space, with associated energy deposit. One and only one track is associated to any given step. At any given moment during the simulation only one step exists in a Geant4 application (the current step)

A (Geant4) hit is a way for the user to make this information persist for the duration of an event, see: Geant4 Application Developers Manual chapter 4.4 Hits are organized in collections of the same type (e.g. c++ class), and SDs are responsible of creating and filling one or more collections

Consequences

A G4Hit is **not** the response of the detector (e.g. pulse shape, digital output, response of a PMT)

In Geant4 the concept of G4Digit is used for this (limited use)

However often some low-level detector effects (Birks' saturation, some zero suppression) are included in hits for convenience

A SD a relatively simple **algorithm** that transforms the G4Step in a G4Hit

With some detectors (e.g. calorimeters) it is impractical to transform each single G4Step to a hit (too many), thus accumulation is preferred (e.g. one hit per calorimeter cell, that accumulates all energy deposits)

Geometry

The role of GDML

As we decided this should be the interchange format

Requirement: your preferred framework, application, tool can (should) use the most appropriate way of defining geometry (extension of GDML, a-la SLIC; pure constructive; database). You should only care to be able to export/import GDML. G4 has utilities for that.

Status: Geant4, ROOT (via script?), VecGeom (?) all support GDML

Recap of discussion so far

Describe geometry in terms of shapes, materials and (hierarchical) structure is not enough

Main issue is how to associate Sensitivity and **exchange this information between applications/detectors**

Start with GDML

Already available: examples/extended/persistency/gdml/G04 :
"Simple example showing how to associate detector sensitivity to a logical-volume, making use of the auxiliary-information."

Hits

Hits

Todo: Define three types of hits:

- Tracker hit: the persistent representation of a single G4Step
- Calorimeter hit: the summation of all energy deposits in a given space
- TPC hit: I understand this is somehow in the middle of the two

A hit is identified by a key

The key is unique given an event and a collection
Does not have to be a number (e.g. a tuple, an hash, a name)

Managing Hits

Once the hit structure is defined, SDs make the mapping

Todo: Develop simple and generic SDs that can create these hits. This could be used as a starting point for a new detector-specific development or used directly without modifications in existing frameworks

Geant4 integration 1

Follow this approach, SDs and Hits become detector independent.
Todo: create a stand-alone library
Based on example G04: show how to integrate w/ GDML

The library should also provide a way to write out hits in files (options: **ROOT**, gprotobuf, HDF5, ASCII, …)

Note: hits file are not self-describing, one needs the GDML to be able to do reconstruction

Geant4 integration 2

Starting from Geant4 10.3 (2016), it is possible to have multiple SD associated to a single (logical) volume

Add our SDs to an existing detector preserving existing code

The proposed interchange format can run in parasitic mode

Hits Collections

A SensitiveDetector creates one or more **named** collections (e.g std::vector<hit>).

Names:

1. SD Name (can be different to LogicalVolume name)
2. Collection Name

For EIC we can add more names if needed

What should be in a hit: “tracker”

ID	Int
Volume ID(s)	[]*sizeof(int) **OR []*string**
PreStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
PostStepPoint {x,y,z,t} (both global/local)	8*sizeof(double)
ΔE	1*sizeof(double)
PDG code	sizeof(long) //needed for ions
G4Track ID	1*sizeof(int)

What should be in a hit: “calorimeter”

ID	Int
Volume ID(s)	[]*sizeof(int) **OR** []*string
Characteristic space Point {x,y,z} (both global/local)	6*sizeof(double)
Min t and Max t (both global/local)	4*sizeof(double)
At least one ΔE (for time info needs to distinguish late hits)	>1*sizeof(double)
PDG codes? All, only for ΔE>min{threshold,0}	>1*sizeof(long) //needed for ions
G4Track ID. All, only for ΔE>min{threshold,0}	>4*sizeof(double)

The idea is to have hits independent of detector effects (e.g. no cut on t), to be able to reprocess hits with different “detector electronics setups”. But we could allow to have simplified hits with less information (e.g. fixed time cut)

Relation with MC Truth

Need to keep track of MC particle ID And G4TrackID

Separate data-structure to keep G4Track ID and their relation (mother-daughter) to allow for shower reconstruction

Same structure could keep relation between G4TrackID of primaries in G4 and corresponding IDs in MC input

What we propose

Proposal is to create shareable classes (lib in gitlab eic repo) that can write out hits collections for track or calo style:

```
class G4EicTrackHit : public G4VHit { /.../ }

class MyHit : public G4EicTrackHit {
    //… the data formats we discussed
    virtual G4bool WriteHitToRootFile() override {
        //Here there is the dependency on ROOT.
        //Write out a object that can be read on ROOT
        //and does not depend on G4
    }
    virtual G4bool WriteHitToAnotherFormat() override {..}
}
```

Instead of conclusions

Output can be made framework independent

Very simple initial proposal: not adequate for real detector, but a starting point for sharing data
New features of Geant4 (multi-SD) allow for this to be run in "parasitic" mode in parallel to existing code

For visualization: from yesterday discussion, we should probably create a simple converter that reads hits file and transform that in the web-optimal file