LArHits Documentation

LArHits is an implementation of the ATLAS liquid−argon barrel calorimeter in Geant4. It is a work in progress; this is the documentation for version 0.3 (10−Apr−2001), available at <ftp://ftp.nevis.columbia.edu/pub/atlas/seligman/LArHits−0.3.tar.gz>.

This note focuses on the LArHits version developed by William Seligman (Nevis Labs) and tested until now with the tailored version of the EM barrel geometry description.

In its current form, LArHits illustrates three concepts:

- implementing the LAr barrel calorimeter geometry efficiently within the Geant4 simulation;
- recording hits in the LAr barrel cells using the Geant4 hit mechanism;
- saving simulated events to disk using an efficient ROOT−based persistency scheme.

The routines used to implement these concepts are described below. This document does not attempt to describe every coding trick used within the routines; the programs' comments already do this. The documentation for Geant4 and ROOT can be found at <http://wwwinfo.cern.ch/asd/geant4/G4UsersDocuments/Overview/html/index.html> and <http://root.cern.ch/root/RootDoc.html>, respectively.

LAr Barrel Calorimeter Geometry

Implementation notes

The main routine for implementing the LAr barrel geometry is `LArDetectorConstruction`.

The geometry of the electrodes and absorbers in the LAr barrel is quite complex. While it is possible to specify the geometry of the detector using the volume primitives of Geant4, the resulting memory use is high (see <http://chep2000.pd.infn.it/abs/abs_a354.htm>). It turns out to be more efficient to create a custom"accordion−shaped volume" in a routine that inherits from `G4VSolid`.

This custom solid is implemented in routine `LArBarrelSolid` (in an earlier version, the name of this routine created by Stefan Simion was `G4Accordion`).

During the development of the Geant4 hit code (discussed below), it was observed that calculations contained in `G4Accordion` were also useful in determining the barrel cell in which energy was deposited. Therefore, the common routines required for both geometric projection and cell identification were put into a new class, `LArBarrelCalculator`.

`LArBarrelCalculator` uses the singleton pattern, that is, there can be only one object of this class. The class is used by passing a `G4ThreeVector` "point" to the calculation methods `WhereIsHit` or `FindCell`. The user then "reads the results" of the calculator with accessor methods.

Notes on future revisions

If a common "Materials Manager" is ever adopted by Geant4 or by the ATLAS development team, it could be used within `LArDetectorConstruction`.

The `LArDetectorConstruction` class will eventually have to be integrated into a global detector geometry definition for the entire ATLAS detector.

The constants and code within `LArBarrelSolid` and `LArBarrelCalculator` should eventually be replaced by a detector database.

The simulation should be extended to include the LAr calorimeter endcaps.

LAr Barrel Hits

Implementation notes

In order to take advantage of the Geant4 hit mechanism, the following three steps are required:

1. A hit class that inherits from `G4VHit` must be defined by the user. This class defines what a "hit" means to the Geant4 simulation.
2. A sensitive–detector class that inherits from `G4VSensitiveDetector` must be defined by the user. This class describes how to take energy deposited by particle in the simulation and convert it into hit information.
3. Geant4 must be told which logical volume(s) are to be associated with the sensitive–detector class defined in step (2). This is normally done in the detector geometry definition routine.

Note: Geant4 also includes a digitization mechanism, but this is not used in LArHits.

Hit class

The hit class in LArHits is `LArBarrelHit`. The essential items of information associated with a `LArBarrelHit` are the calorimeter cell in which the energy is deposited, the hit energy, and the time.

Rather than storing the time, the quantity stored in the `LArBarrelHit` class is energy*time. The reason is that, for in–time hits in a cell, the total energy is the sum of all energy deposited, and the hit time is the energy–weighted average of the time of the energy depositions. It's easier to compute the time average if energy*time is stored and re–summed for each energy deposit in a cell.

Sensitive–detector class

In Geant4, the collections of hits associated with a given `G4Event` object are stored in a `G4HCofThisEvent` object. Since an object can have many kinds of hits (calorimeter hits, tracker hits, etc.), the hits are stored in a templated collection class; `G4THitsCollection<LArBarrelHit>` in this case (this is defined in the header file `LArBarrelHit.hh`).

The sensitive–detector class in LArHits is `LArBarrelSD`. This class is responsible for creating hits, and for adding each hit to a hit collection. Since the `ProcessHits` method of `LArBarrelSD` is called each time a particle deposits energy in the sensitive detector, it is important that this routine be as efficient as possible. `LArBarrelSD` also distinguishes between "in–time" and "out–of–time" hits; a hit is considered "out–of–time" if its energy is depostod at a time $(t-t_0) > 2.5$ *ns,* where t_0 is the time at the event vertex plus the time of travel to the point of energy deposition at the speed of light.

These requirements are implemented using the following algorithm:

- If an energy deposition is out–of–time, create a separate hit and add it to the hit collection.
- If an energy deposition is in–time, and in a cell in which no energy has yet been deposited, create a new hit and add it to the hit collection. Set a pointer in a array of "bookmarks" to indicate this cell has had energy deposited.
- If an energy deposition is in–time, and in a cell in which there has been an energy deposit, add the energy to the existing hit in the hit collection.

Logical volume

The logical volume is associated with the sensitive detector near the end of `LArDetectorConstruction` with the following statement:

```
Stac_log->SetSensitiveDetector(barrelSD);
```

The logical volume `Stac_log` is defined to be the liquid–argon toroidal cylinder that contains the calorimeter. The volumes that define the lead absorbers and the electrodes are positioned within this cylinder.

If a particle deposits energy within a absorber or an electrode, the simulation treats the deposition as if it occurred in the associated absorber or electrode logical volume, not within the liquid–argon volume of `Stac_log`. The result is that the `ProcessHits` method of `LArBarrelSD` is only called if the energy deposition occurs inside the liquid–argon portion of the calorimeter.

Notes on future revisions

The Geant4 hit mechanism is complex, but the benefit is flexibility in organizing and processing hits. If it were needed, the out–of–time calorimeter hits could be stored in a separate hit collection. When the LAr endcaps are added to the simulation, the endcap hits could be stored in a separate collection, or stored in the same collection as the barrel hits.

ROOT–based Event Persistency

Implementation notes

The event persistency in LArHits makes use of the abstract base class `RecorderBase` described at <http://www.nevis.columbia.edu/~atlas/lar/>. Since LArHits uses the ROOT I/O scheme for its persistency, the name of the class that implements `RecorderBase` is

`LArRootPersistency`.

The actual persistent classes written into a ROOT Tree are `LArRootEvent`, `LArRootPrimary`, `LArRootHit`, and `LArCellID` which contain event, primary particle, hit, and cell identifier data respectively. These classes are completely ROOT–based and contain no reference to any Geant4 data types. They correspond (roughly) to the Geant4–based classes `G4Event`, `G4PrimaryParticle`, `G4PrimaryVertex`, and `LArBarrelHit`.

The procedures for converting the Geant4 classes into the ROOT classes are contained in the conversion classes: `LArPersistentRootEvent`, `LArPersistentRootPrimary`, and `LArPersistentRootHit`. These are the "bridge" classes that connect Geant4 to ROOT.

There are several forms of I/O available in the ROOT Toolkit. LArHits uses the most efficient scheme identified in the study available at <http://www.nevis.columbia.edu/~atlas/lar/iostudy–20010404.html>: ROOT `TTrees` with all containers implemented as `TClonesArrays`, written with splitlevel=1 and compression=1.

Notes on future revisions

The ROOT–based persistency mechanism should be integrated into the Athena framework.

The `LArRootCellID` class should be upgraded to the Athena `Identifier` class.

Some routine names should be changed to better fit with the Athena scheme. For example, `LArPersistentRootHit` should be changed to `LArRootCnvHit`.

When `TClonesArrays` are used within a class written to a ROOT `TTree`, they must be static members of that class. `LArRootEvent` should be implemented as a singleton class to make it clear that a program can only contain one distinct `LArRootEvent` object.

Acknowledgments

The author wishes to thanks all those people working on the ATLAS LAr barrel in Geant4.