2005 International Linear Collider Workshop – Stanford, U.S.A.

Full Detector Simulation using SLIC and LCDD

J. McCormick
SLAC, Stanford, CA 94025, USA

Simulator for the Linear Collider (SLIC) and Linear Collider Detector Description (LCDD) provide a flexible and powerful package for full detector simulations. This paper outlines the main features of SLIC and LCDD and explains the structure of an LCDD document used for detector description input.

1. OVERVIEW

The Simulator for the Linear Collider (SLIC) [1] is a full simulations package for Linear Collider Detector studies developed by Jeremy McCormick (SLAC), with contributions from Ron Cassell (SLAC). The software is being used to study the physics response of ILC full detector designs, including the Silicon Detector (SiD) [2] and a number of its variants.

SLIC is a "hub" package that integrates nine, different physics software libraries into a single program. It reads StdHep [3] physics events, simulates them in a detector, and produces Linear Collider IO (LCIO) [4] output files. These can be analyzed using the Java-based org.lcsim [5] analysis and reconstruction framework.

The detector description, along with related information such as readouts, is described completely at runtime using an Extensible Markup Language (XML) [6] document conforming to the Linear Collider Detector Description (LCDD) [7] format. (The term LCDD refers both to the XML data format representing the detector, and the package that parses this format for input into the simulation runtime.) The LCDD subsystem is designed for maximum flexibility, while removing any requirement on the end user to author or modify C++ code when customizing the detector parameters.

2. LCDD

Loading the detector description into the runtime environment is typically the most complex part of a Geant4 [8] simulator package. The LCDD package deals with the specification of detector attributes, while SLIC contains the "command and control" engine. The LCDD data format covers the areas of geometry, materials, constants, readout, visualization, magnetic fields, tracking and stepping limits, and identifier definition. Various types of detectors, including test beams and full detectors, are fully describable using the LCDD XML format.

LCDD is built upon the Geometry Description Markup Language (GDML) [9] toolkit, whose primary author is Radovan Chytracek (CERN/LCG). GDML covers the core geometry of solids within a nested volume hierarchy. It also has facilities for defining materials and constants. GDML's SAX [10] parser can process a 15,000-line LCDD document in approximately 1.5 seconds on a Pentium III. LCDD uses this engine and benefits from its high performance.

The input to the LCDD system is an XML document representing the detector and its readouts, plus associated metadata. This document must conform to the XML Schema (XSD) [11] that defines the LCDD data format. The XML is read by SLIC, using the LCDD sub-package, to construct the complete Geant4 detector geometry, along with

the sub-detector readouts (called “sensitive detectors” in Geant4 terminology). LCDD also sets-up related objects that represent regions, sets of physics limits, visualization attributes, and magnetic fields.

The LCDD XML schema contains the GDML format as a subschema, importing the complete GDML geometry model into LCDD. This approach allows for extension and alteration of GDML, while keeping the core model of constants, materials, solids and volumes. LCDD only extends GDML by adding elements to the GDML that reference the LCDD. Thus, any GDML parser should be able to read the GDML data from an LCDD document simply by ignoring these extra tags. The extracted GDML could also be altered to conform to the GDML schema simply by removing these additional tags.

2.1. LCDD Data Format

An LCDD document is contained within the *lcdd* tag, the root element of the document. Other top-level elements have their own subschema encapsulating the data model for that concept. For instance, the *header* element contains metadata about the detector, like the name of the author and the detector’s unique tag (e.g. “sdjan03”).

The structure of a document is outlined by this fragment of pseudo-XML.

```
<lcdd>
    <header>
    <iddict>
    <sensitive_detectors>
    <limits>
    <regions>
    <display>
    <gdml>
    <fields>
</lcdd>
```

The *header* contains metadata about this detector, such as its unique tag and author. This element is used for writing out the detector’s unique tag into the LCIO output file.

The *iddict* specifies the contents of the 64-bit IDs written into the LCIO output file. A sensitive detector references an *idspec* that describes the fields to be included in the ID. These may include Geant4 volume numbers, physical volume identifiers, or bin numbers from a segmentation scheme.

The *sensitive_detectors* element contains a *tracker* or *calorimeter* element for each detector subsystem, such as a calorimeter barrel or vertex detector. (Calorimeters and trackers are the two basic types of subdetectors.) Both types of sensitive detectors can be customized using various attributes. The calorimeter sensitive detector needs to have a *segmentation* tag, such as a projective cylinder or Cartesian grid, which defines the virtual segmentation for that volume.

Limits on steps and tracks can be assigned using the *limits* block. A volume within the GDML element can reference a *limitset* in order to customize the physics response in that area. For instance, the production of secondary gammas could be suppressed by assigning a very large minimum to the tracking range cut for gamma particles.

The *regions* element can be used to specify groups of volumes with similar characteristics, such as whether or not secondary particles within the volume should be written into the LCIO output file.

Visualization attributes can be assigned using the *display* element. All options in the G4VisAttributes class are available using the *vis* element. These include toggling volume visibility and daughter volume visibility, line style, wireframe or solid drawing style, and the color.

The *fields* area defines one or more magnetic fields. A global solenoid is currently the only option.

The *gdml* tag contains an embedded GDML document with LCDD extension tags on the *volume* element. The extension tags are like references or pointers that assign an LCDD element to the volume.

This is an example of the extended GDML volume element.

```
<volume name="ecal_barr">
  <materialref ref="Air" />
  <solidref ref="ecal_barr_tube" />
  <physvol>
    <volumeref ref="ecal_barr_lay0" />
    <positionref ref="identity_pos" />
    <rotationref ref="identity_rot" />
  </physvol>
  <sdref          ref=“EcalSD” />
  <regionref   ref="EcalRegion" />
  <visref         ref="EcalVis"/>
  <limitsetref ref=”EcalLimits”/>
</volume>
```

The items in bold are LCDD extensions to the GDML format. The aforementioned LCDD tags are referenced from the volume, in order to setup a G4LogicalVolume with supplementary Geant4 objects: G4SensitiveDetector, G4Region, G4VisAttributes and G4UserLimits. In this way, LCDD is able to interface with GDML while preserving that format’s structure.

3. MONTE CARLO PARTICLE HANDLING

The handling of secondary particles is typically a key portion of a Geant4 full simulator. All of the tracking objects defined by Geant4 are transient with the event, so a simulator must create the particle tree and write it into the output file if this information is to be utilized in reconstruction and analysis. SLIC’s Monte Carlo particle handling is based on that of the Linear Collider Simulator [12] package by Ron Cassell. It uses custom implementations of G4VUserTrackInformation and G4VTrajectory to store the tracking information until the end of the event. The LCIO Monte Carlo particle tree is constructed after the event is simulated in the EndEventAction using information from these two objects, along with the G4PrimaryParticles and the initial particle tree read from the StdHep event file. Throughout this process, track ID is used as the primary unique identifier. SLIC also has the (unique) capability of reading-in and simulating events from the MCParticle collection of an LCIO file. This allows events written-out from other simulators to be simulated using SLIC.

4. COMMANDS

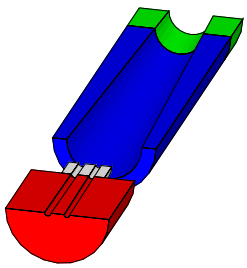
Each of SLIC's command-line switches activates a single Geant4 command. Most of these are commands that have been added specifically for SLIC. For instance, the "-i" switch executes the "/generator/filename" command. In this way, the interface is kept consistent between the command-line and macros or an interactive session. The command set has been designed with flexibility and convenience in mind. For instance, SLIC can automatically name the LCIO output file according to the geometry and input file parameters with the "/lcio/autoname" command.

5. PLANS

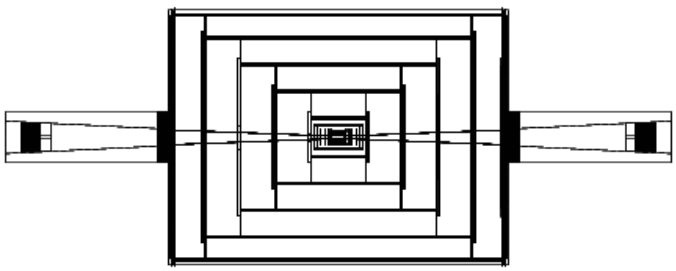
For the Snowmass 2005 Workshop [13], SLIC will be used to produce approximately 1 million physics events on a variety of different detector configurations. Each of ten detectors will have about 100,000 simulated events. Using these datasets, researchers will be able to compare the simulated response of different readout technologies and benchmark their analysis and reconstruction code, including Particle Flow Algorithms.

6. EXAMPLE GEOMETRIES

6.1. MDI-BDS



6.2. sidaug05



References

[1] http://www.lcsim.org/software/slic

[2] http://www-sid.slac.stanford.edu/
[3] http://www-cpd.fnal.gov/psm/stdhep/
[4] http://lcio.desy.de
[5] http://www.lcsim.org/software/lcsim
[6] http://www.w3.org/XML/
[7] http://www.lcsim.org/software/lcdd
[8] http://wwwasd.web.cern.ch/wwwasd/geant4/geant4.html
[9] http://gdml.web.cern.ch/GDML/
[10] http://www.saxproject.org/
[11] http://www.w3.org/XML/Schema
[12] http://www.lcsim.org/software/lcs
[13] http://alcpg2005.colorado.edu/