Recent Developments in the FullSimLight Simulation Tool from ATLAS

Raees Khan[1,*], *Marilena* Bandieramonte[1], *Joseph* Boudreau[1], *Riccardo Maria* Bianchi[1], *Andrea* Dell'Acqua[2], *Denys* Kleklots[2], *Vakhtang* Tsulaia[3] on behalf of the ATLAS Computing Activity.

[1]University of Pittsburgh, Pittsburgh, PA 15260, USA
[2]CERN, EP Department, Meyrin, 1211, Switzerland
[3]Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA

Abstract. FullSimLight is a lightweight, Geant4-based command line simulation utility intended for studies of simulation performance. It is part of the GeoModel toolkit (geomodel.web.cern.ch) which has been stable for more than one year. The FullSimLight component has recently undergone renewed development aimed at extending its functionality. It has been endowed with a GUI for fast, transparent, and foolproof configuration and with a plugin mechanism allowing users and developers with diverse goals to extend and customize the simulation. Geometry and event input can be easily specified on the fly, allowing rapid evaluation of different geometry options and their effect on simulation performance. User actions and sensitive detectors can also be loaded through the new plugin mechanism, allowing for customization of Geant4 processing and hit production. The geometry explorer (gmex), in a parallel development, has been enhanced with the capability of visualizing FullSimLight track and hit output. FullSimLight, brought to you by the ATLAS collaboration at the LHC, is an experiment independent software tool.

1 Introduction

With access to a standalone ATLAS detector [1] geometry and extraction of the `GeoModel` toolkit [2] from the Athena framework, lightweight full simulation through `FullSimLight` became possible in ATLAS at the LHC. The basic working of `FullSimLight` is shown in Figure 1. Originally created and used for geometry debugging and `Geant4` optimization studies [3], `FullSimLight` has recently gone through a period of development to allow users and developers with diverse goals to extend and customize the simulation. The motivations behind these developments were

- Make simulation using `FullSimLight` more accessible and foolproof.

*e-mail: raees.ahmad.khan@cern.ch

- Extend the functionality of FullSimLight by allowing the user to add Geant4 User Actions, Sensitive Detectors, Physics Lists, etc.
- Create built-in visualization for FullSimLight output in gmex (an interactive 3D geometry visualization tool which is part of GeoModel).
- Make FullSimLight experiment agnostic to extend its suitability beyond the ATLAS collaboration.

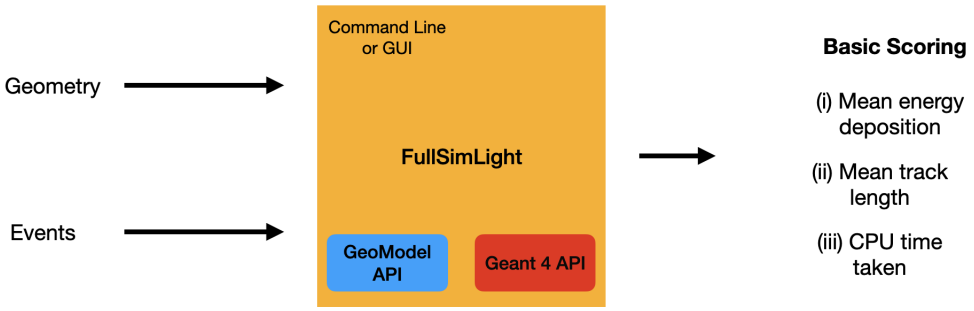

Figure 1. Basic working of FullSimLight

2 The FSL graphical user interface

fsl is the GUI to FullSimLight introduced in release 4.3.0 of GeoModel. Figure 2 showcases the main tab of the interface. fsl lets one select the desired configuration through its various tabs and ensures that the options selected are compatible with each other. Once the selections have been made, the user can run the simulation as well as other GeoModel tools such as gmclash for clash detection and gmex for geometry visualization right within the fsl interface. Alternatively, the configuration can be saved into a file in the standard JSON format which can be loaded back into fsl or run with FullSimLight on the command line through the -c flag. In this way, there is no longer any need to overcrowd the command line with possibly conflicting configuration flags when running FullSimLight. fsl also provides the user access to a limited amount of Geant4 user interface commands for verbosity on its main tab. More experienced users who are familiar with the Geant4 user interface can open the configuration file in an external editor for further customizations. In this sense fsl can be viewed as a configuration editor, providing a FullSimLight configuration file that can be used as a starting point for other customizations, or shipped to other platforms.

2.1 Event Generation

FullSimLight provides the user with a couple different ways to configure an event generator which can all be found on the generator tab of fsl. These are,

- Particle Gun

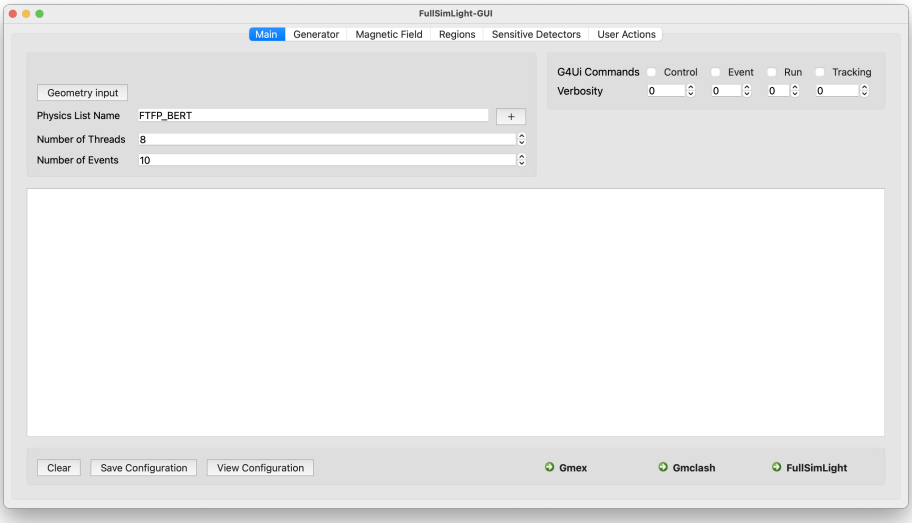

Figure 2. FSL Main Tab

- Pythia
- `HepMC3` File (New)
- Generator Plugin (New)

  Previously FullSimLight only supported `Geant4` Particle Gun and `Pythia` generators that were set through macro files. These generators can now be configured through the `fsl` interface. Files in the standard `HepMC3` (as well as the deprecated `HepMC2`) format can now also be used to specify events. Additionally `FullSimLight` now comes with an abstract class that interfaces `Geant4` to let the user write their own custom event generator plugins which is explained in Section 3.

2.2 Geant4 Regions Configuration

As part of the bid to extend the functionality of `FullSimLight` and make it experiment agnostic, hard coded ATLAS G4 regions were removed. The mechanism to configure G4 regions can now be found on the regions tab in `fsl` as shown in Figure 3. The user can specify root logical volumes and cuts as required.

3 Plugins

Plugins which come in the form of shared libraries (`.dylib` or `.so`) containing custom code were the mechanism chosen to extend `FullSimLight`. The basic idea can be seen in Figure 4. The user develops the plugin according to the functionality they need which can then be plugged into `FullSimLight` to produce the desired output. The benefit of this approach is that it gives the user the ability to develop what they need without needing the core team's intervention. Plugins also provide a nice structure to make `FullSimLight` experiment agnostic since ATLAS specific functionality can live in a set of ATLAS specific plugins [4].

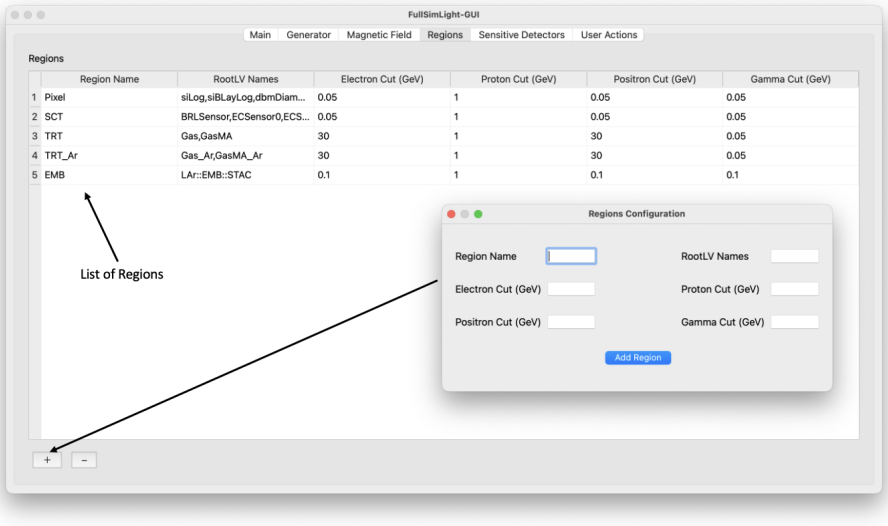

Figure 3. FSL Regions Tab

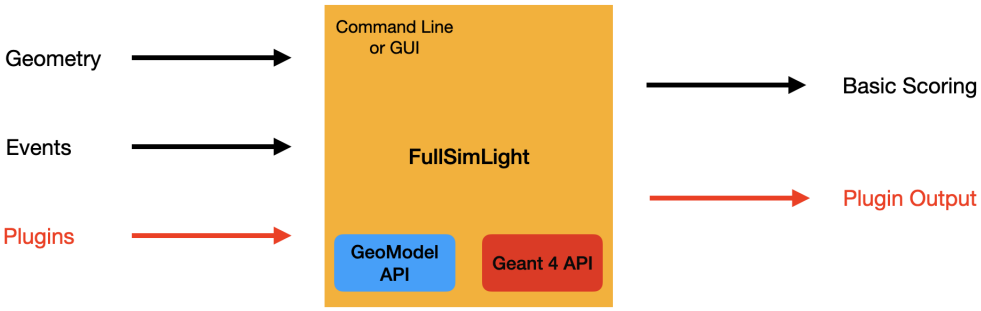

Figure 4. Customized FullSimLight

3.1 Plugin Architecture

Plugins can be used to interface the `Geant4` functionality (G4 objects) of

- User Actions
- Sensitive Detectors
- Magnetic Field
- Physics Lists
- Event Generators

Once the plugin has been built into a shared library, `fsl` provides a simple interface to add the plugin to the simulation through menus on its various tabs. For example, the User Action Plugin menu found on the User Actions tab can be seen in Figure 5. `FullSimLight` comes already included with a number of custom plugins to do various things such as record hits, generate the ATLAS magnetic field, etc as well as dummy plugins for example [5].

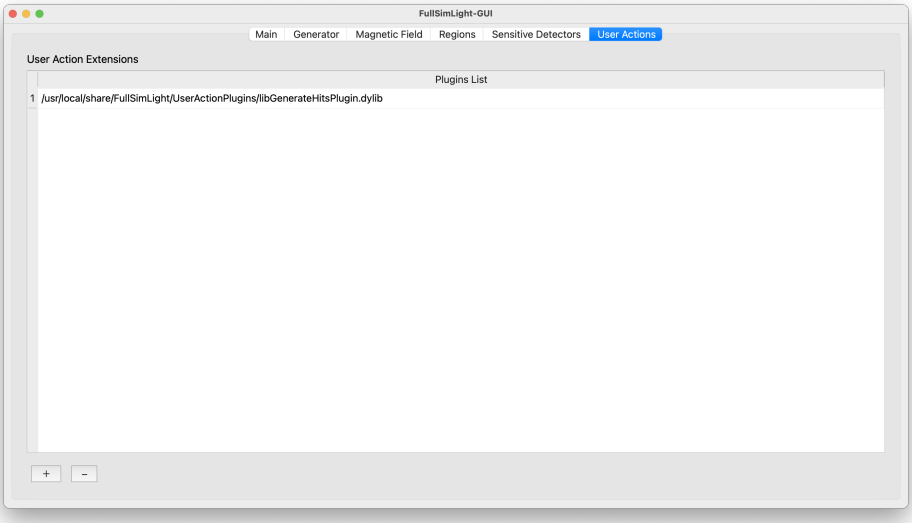

Figure 5. FSL Menu to add User Action Plugins

3.2 Writing Plugins

With the introduction of the Plugin Architecture in `FullSimLight` the relevant question is, How does one actually write a plugin? The basic idea is that `FullSimLight` provides abstract classes that can be overridden to allow the user to interface with `Geant4` functionality. As a concrete example consider, we want to write a simple "Hits" plugin which produces a record of `Geant4` stepping points. To do this we need access to a `G4UserSteppingAction` to get the stepping points and a `G4UserEventAction` to get the corresponding event ID. Listing out the steps to writing this plugin we have

- Define implementation classes `GenerateHitsStep` and `GenerateHitsEvent` which inherit from the `Geant4` classes `G4UserSteppingAction` and `G4UserEventAction` respectively, giving access to the required functions to get the stepping points and event ID.
- Define the plugin class, `GenerateHitsPlugin` which inherits from the abstract class `FSLUserActionPlugin` provided by `FullSimLight`.
- Override the virtual methods `getSteppingAction` and `getEventAction` contained in the abstract class since they correspond to the user actions we have used in the plugin.
- Have the overridden methods return an instance of the their corresponding implementation class.

A sample code snippet of the above described `GenerateHitsPlugin` class is shown in Figure 6. A more in-depth explanation of writing plugins, going over all technicalities as well as abstract classes available to the user can be found in Refs. [6] [7].

```
class GenerateHitsPlugin:public FSLUserActionPlugin {

public:

    GenerateHitsPlugin();
    virtual G4UserSteppingAction *getSteppingAction() const final override;
    virtual G4UserEventAction *getEventAction() const final override;

    GenerateHitsEvent* eventaction = new GenerateHitsEvent();
    GenerateHitsStep* stepaction = new GenerateHitsStep();

};

GenerateHitsPlugin::GenerateHitsPlugin()
{
//Passing the stepping action into the event action
    //as required by our implementation.
eventaction->SetSteppingAction(stepaction);
}

G4UserSteppingAction *GenerateHitsPlugin::getSteppingAction() const {
  return stepaction;
}

G4UserEventAction *GenerateHitsPlugin::getEventAction() const {
    return eventaction;

}

extern "C" GenerateHitsPlugin *createGenerateHitsPlugin() {
    return new GenerateHitsPlugin();
}
```


Figure 6. `GenerateHitsPlugin` class code

4 Visualization

`FullSimLight` now comes included with a more sophisticated version of the Hits plugin described briefly in the previous section which produced a record of `Geant4` stepping points as well as a Tracks plugin which produces a record of tracks when plugged into the simulation. The record is stored in the standard `HDF5` file format and is a part of the output of the simulation. Figure 7 depicts this process.

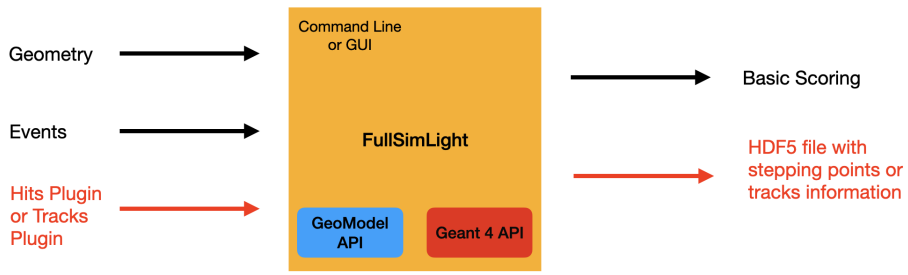

Figure 7. FullSimLight with Hits or Tracks Plugin

In parallel to the development of these plugins, `gmex` (the geometry visualization tool of `GeoModel`) was modified to provide a display for the simulation output. The `HDF5` file generated by the Hits or Tracks Plugin can be loaded into `gmex` to be co-displayed with the geometry. Examples of this can be seen in Figures 8, 9.

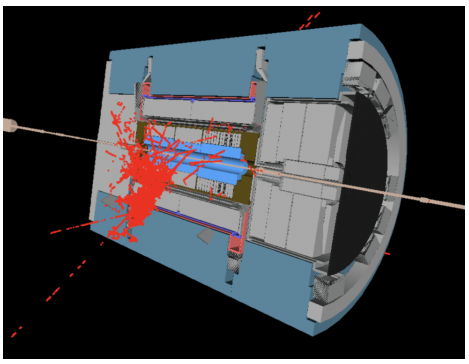

Figure 8. Steps Display in ATLAS detector

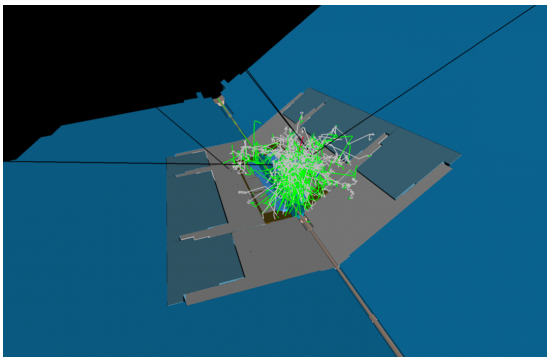

Figure 9. Tracks Display in ATLAS detector

5 Example

As explained in Section 1, a primary impetus behind the recent advancements was to render `FullSimLight` experiment agnostic. In this context, an illustrative instance of its application beyond the ATLAS framework is pertinent. It is commonly believed that grand unified theories (GUTs) predict proton decay. Muscovite Mica has emerged as a promising substrate for detecting evidence of such decay through the analysis of positron tracks through it [8]. Simulation of this process can be done using `FullSimLight`. To start one can create a geometry description of a cube of Mica using GeoModel , as depicted in Figure 10.

```
void MicaPlugin::create(GeoPhysVol *world, bool /*publish*/)
{
    GeoElement  *potassium = new GeoElement("Potassium",    "K",   19,  39*gram/mole);
    GeoElement  *oxygen    = new GeoElement("Oxygen",     "O",    8,  16*gram/mole);
    GeoElement  *aluminium = new GeoElement("Aluminium", "Al", 13,  26*gram/mole);
    GeoElement  *silicon   = new GeoElement("Silicon", "Si", 14,  28*gram/mole);
    GeoElement  *hydrogen  = new GeoElement("Hydrogen", "H", 1,  1*gram/mole);
    GeoElement  *fluorine  = new GeoElement("Fluorine", "F", 9,  19*gram/mole);

    //Defining Mica
    double densityOfMica = 2.82*gram/cm3;
    GeoMaterial *Mica = new GeoMaterial("Mica",densityOfMica);
    Mica->add(potassium,1);
    Mica->add(oxygen,11.8);
    Mica->add(aluminium,3);
    Mica->add(silicon,3);
    Mica->add(hydrogen,1.8);
    Mica->add(fluorine,0.2);
    Mica->lock();

    const GeoBox       *MicaBox  = new GeoBox(100*cm, 100*cm,100*cm);
    const GeoLogVol    *MicaLog  = new  GeoLogVol("MicaLog", MicaBox, Mica);
    GeoPhysVol         *MicaPhys = new GeoPhysVol(MicaLog);
    world->add(MicaPhys);
}
```

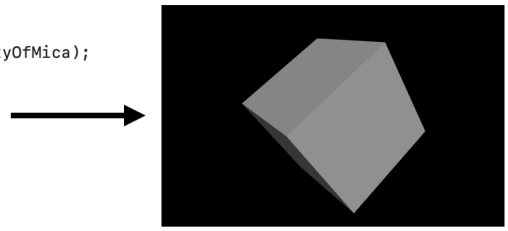

Figure 10. Mica geometry plugin code (left) and resulting geometry visualized in gmex (right).

The Mica geometry plugin can then directly be plugged into `FullSimLight` and positrons can be shot through it. Figure 11 shows the particle shower in the Mica cube

obtained by adding the Tracks plugin to the simulation and visualizing in `gmex`. One can obtain further desired information by writing and adding custom plugins to the simulation.

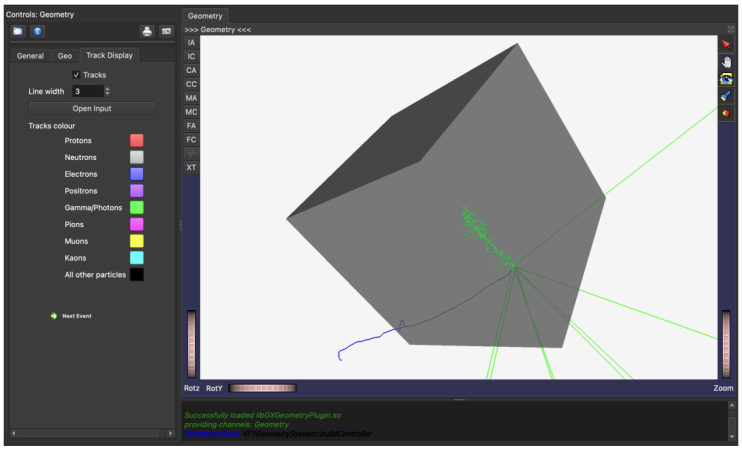

Figure 11. Particle shower inside Mica Cube

6 Conclusion

`FullSimLight` offers a powerful and versatile tool for simulation studies. Since its initial introduction as a basic tool a few years back, it has proven very useful in ATLAS for debugging and optimizations. Now with enhanced accessibility through `fsl`, increased extensibility through the plugin architecture, and newly built-in visualization capabilities, `FullSimLight` has grown considerably, with its lightweight nature and experiment independence making it suitable for a wide range of applications within and beyond the ATLAS collaboration at the LHC. Future work will be geared towards ensuring a stable and compatible product.

References

[1] ATLAS Collaboration, JINST 3 S08003 (2008)

[2] J. Boudreau, V. Tsulaia, *The GeoModel toolkit for detector description, in Computing in high energy physics and nuclear physics. Proceedings, Conference, CHEP'04, Interlaken, Switzerland, September 27-October 1, 2004* (2005), pp. 353–356, https://cds.cern.ch/record/865601

[3] M. Bandieramonte, R. M. Bianchi, J. Boudreau, EPJ Web of Conf. 245, 02029 (2020)

[4] https://gitlab.cern.ch/atlas/geomodelatlas/ATLASExtensions

[5] https://gitlab.cern.ch/GeoModelDev/GeoModel/-/tree/master/FullSimLight/Plugins

[6] https://geomodel.web.cern.ch/home/fullsimlight/plugins/

[7] https://geomodel.web.cern.ch/home/fullsimlight/plugin-support/

[8] F.M. Russell. In Quodons in Mica, J.F. R. Archilla et al, eds., Springer (2015) pp. 474–559