January 22, 2003

TileCal Beam Test Simulation Application in the FADS/Goofy Framework (GEANT4)

A. Solodkov
Institute for High Energy Physics, Protvino, Russia

V. Tsulaia*
Joint Institute for Nuclear Research, Dubna, Russia

ATL-TILECAL-2003-002
22 January 2003

Abstract

A new application for the Tile Calorimeter (TileCal) beam test simulation has been developed in GEANT4 within the FADS/Goofy framework. The geometry and readout systems for all the different TileCal modules have been implemented in a quite detailed way. This application allows to simulate all the TileCal beam test setup configurations existing so far. Details of the development as well as instructions to install and run the program are presented. The first tests have been performed for a beam test setup consisting of five prototype modules using negative pions with different energies and results of comparison to the experimental data from TileCal TDR are presented as well.

* On a leave of absence from Institute of Physics, GAS, Tbilisi, Georgia

1. Introduction

Up to the year 2000, GEANT3 was the main environment used by the TileCal collaboration to develop its simulation applications. This was true also for other subsystems of the ATLAS detector. In 1999 the ATLAS collaboration decided to start using GEANT4 [1] as a new tool for detector simulations, since GEANT4 should satisfy the new physics requirements at the LHC energy scale. Therefore, we started to develop a new simulation application using the GEANT4 toolkit.

In 2001 the ATLAS group produced a common framework for the ATLAS detector simulation in GEANT4, namely FADS/Goofy [2,3]. This framework extends the GEANT4 toolkit functionality and provides possibility to develop the simulation applications of the various ATLAS subsystems in a coherent way. The TileCal collaboration decided to develop a new code within the FADS/Goofy framework at the end of 2001. As a result we have now an application that can simulate all the TileCal beam test setup configurations realized so far:

- **Beam Test 1995** (5 prototypes)
- **Beam Test 1997** (5 prototypes + 2 extended barrel modules)
- **Beam Test 1998** (5 prototypes + module 0)
- **Beam Test 2000** (module 0 + central barrel module + 2 extended barrel modules)

Section 2 of this note describes some details of the new application, with particular emphasis on the geometry and readout system description, on the hits processing, on the user actions, on the event generator and physics lists. **Section 3** gives instructions to install and run the application. **Section 4** presents the first results obtained with our application for the 1995 beam test setup. Energy resolutions for negative pions are compared to beam test results from the TileCal TDR [4].

2. Application details

2.1 General structure

The FADS/Goofy environment provides users with a set of specific tools for different detector simulation domains: geometry management, sensitive detectors, user actions, physics lists etc. It includes also one simulation program (Goofy), which has no predefined detector geometry, no user actions and no physics lists. All the information needed to build a detector-specific application and to perform a simulation is interactively supplied by the user via commands typed in the Goofy prompt or via macro files in batch mode. Therefore, a typical FADS application should consist of one or more shared libraries used by Goofy at run time.

The TileCal beam test simulation application includes a few shared libraries:

- **`libTileTB.so`** – realization of the geometry, sensitive detectors and user actions;
- **`libTileGenerator.so`** – realization of specific event generator;
- **`libTileTBPhysics*.so`** – set of physics libraries developed using different hadronic physics lists provided by the GEANT4 experts [6];
- **`libTilePackaging.so`** - includes physics lists for General Physics, Electromagnetic Physics, Muon Physics and Ion Physics, developed by the GEANT4 experts [6].

2.2 Geometry

The XML language is used to describe the geometry of all the TileCal modules used in different beam test setup configurations: central barrel modules, extended barrel modules, module 0 and prototype modules. The XML code, which describes all the modules, is contained in just one file: **`modules.xml`**. Module geometry is described using XML elements of special types: *TileSection, TilePeriod, TileScintillator* etc. The primary numbers for the geometry description are identical to those used for the TileCal simulation in GEANT3 [5].

Additionally, for each particular beam test setup, we provide a dedicated XML file (for example: **`tb00.xml`** for the 2000 beam test). This file holds the information needed to build the setup geometry, namely the number and type of the modules presented and their relative position.

We use the FADS mechanisms to parse the XML code at run time and keep the geometry descriptions into the hierarchy of the FADS specific *DetectorDescription* objects (Fig.1). The following UI commands are used for this purpose:

```
/DetDescription/ReadFile modules.xml
/DetDescription/ReadFile tb00.xml
```

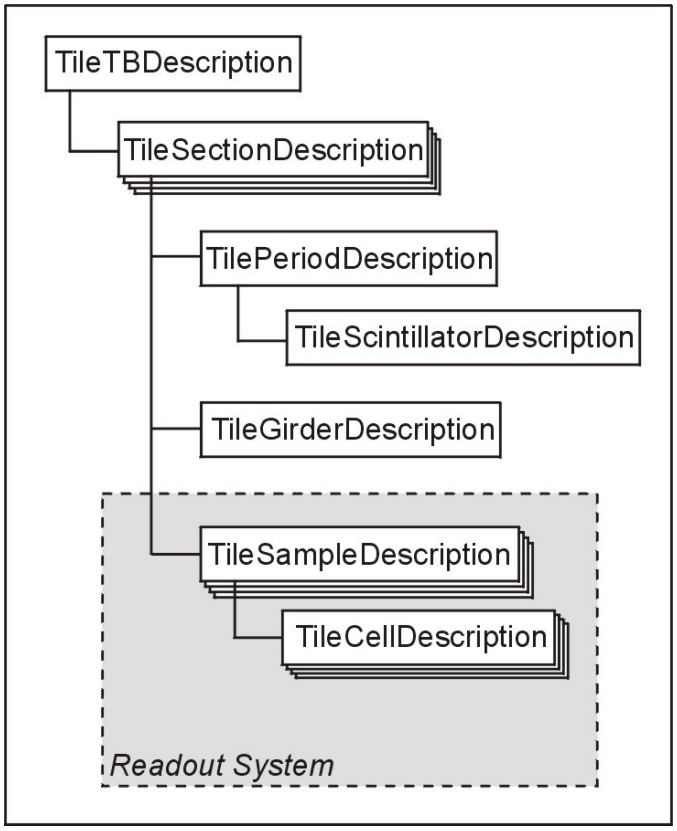

Figure 1. Hierarchy of the *DetectorDescription* objects that hold at run time the geometry and readout system description (gray area) of the TileCal beam test setup. The number of *TileSectionDescription* objects corresponds to the number of various module types used in that particular beam test setup.

FADS provides a mechanism to build the detector geometry at run time using specific *DetectorFacility* objects. The FADS application developers have to realize an abstract *DetectorFacility* interface in the number of concrete classes of their choice. The singleton objects of the user classes are instantiated and registered into a specific *DetectorFacilityCatalog*. FADS provides UI commands to choose the

necessary facility objects from the catalog, build the mother-child relationships between them, position and rotate child objects inside mothers, set one facility as World Volume (top level mother volume).

For the TileCal beam test simulation application we have developed two *DetectorFacility* objects:

- ***TileTB***. Object representing the whole beam test setup. The real geometry is built using the information from the *DetectorDescription* hierarchy.
- ***TBMother***. Large box that acts as World Volume in the TileCal beam test simulation.

The following commands get the facility objects from the catalog and position the ***TileTB*** inside the ***TBMother***:

```
/Geometry/GetFacility TBMother TBMother
/Geometry/GetFacility TileTB TileTB
/TBMother/AddDetector TileTB
```

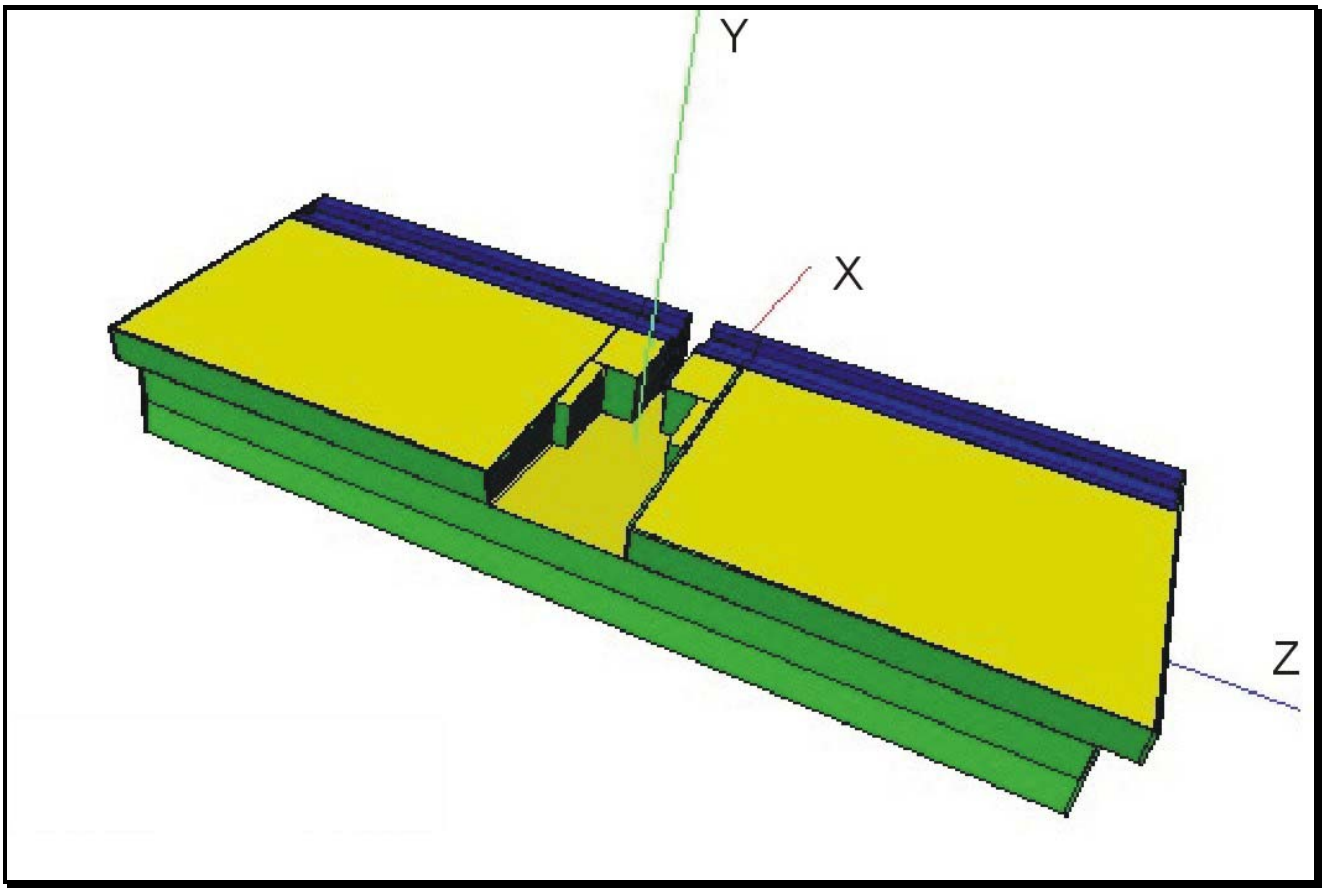

Figure 2. The TileCal 2000 beam test setup. There are: module0 at the bottom, a central barrel module in the middle and two extended barrel modules on top.

The center of the coordinate system corresponds to the geometrical center of the TBMother. We also consider the geometrical center of the central module as the center of the whole beam test setup (for example: in the 2000 beam test setup the barrel module is the central one). The execution of the above commands makes the beam test setup center coincident with the coordinate system center. The position of the ***TileTB*** inside the ***TBMother*** can be changed later using standard FADS UI commands: **`RotateX`**, **`RotateY`**, **`RotateZ`** and **`MoveTo`**.

Figure 2 shows the initial position of the 2000 beam test setup visualized using the VRML viewer. The coordinate axes are shown as well.

2.3 Readout system

We also use the XML language to describe the readout system of the TileCal modules (definition of Cells and Samples). The readout system of each particular module type is described in a dedicated XML file (for example: **`barrel_readout.xml`** includes the readout geometry description for a central barrel module). We provide specific types of XML elements: *TileSample* and *TileCell*. Each XML element has an identifier, which consists of 6 fields separated by a colon. The field values have the following meaning:

TileID/SampleID – "**`subsystem`**:**`detector`**:**`side`**:**`module`**:**`tower`**:**`sample`**".

- **`subsystem: 5`** – TileCal;
- **`detector: 0`** – module 0, **`1`** – central barrel, **`2`** – extended barrel, **`3`** – ITC, **`4`** – prototype;
- **`side`**: **`1`** – positive, **`-1`** – negative;
- **`module: 0-63`** (granularity in $\varphi, \Delta\varphi = 0.1$);
- **`tower: 0-15`** (granularity in $\eta, \Delta\eta = 0.1$);
- **`sample: 0`** – A; **`1`** – B, C, BC; **`2`** – D.

Note: For each field an 'X' indicates *all* or *undefined*.

The TileCell elements also have special attributes indicating the number of the corresponding photo-multipliers (PMTs) and their positions (hole numbers).

The readout system description is kept at run time in special *DetectorDescription* objects that form an additional branch in the whole detector description tree (Fig.1).

2.4 Hits processing and User Actions

In the TileCal beam test simulation application we use the *TileTBHit* class derived from *G4Vhit*. The *TileTBHit* objects hold the energy deposited in one PMT during one event and also the identifier of the corresponding cell. The hit objects are created only for those cells, which contain some energy deposition. Since for every cell there are two corresponding PMTs (called "Up" and "Down" here), we have implemented a simple energy distribution into PMTs, which depends on the Y coordinate of the energy deposition. The Birk's saturation law for organic scintillators has also been implemented.

At the end of each event the information from hit objects is stored into an ntuple and hits are deleted. Therefore, no persistency mechanism has been implemented for hits so far.

The operations with ntuples are realized with the use of standard HBOOK routines in special User Action classes. We have developed one parent class, *TileTBUserAction,* which provides common methods to open, fill and save HBOOK files and also two child classes that realize different concepts for ntuple structures. All produced ntuples have a common **`GENERAL`** block and also for each module used in that particular beam test setup the dedicated block of ntuple variables is presented. The difference in ntuple production lies in the different structure of the module blocks.

- ***TileTBUASamples***. Provides a structure similar to the one existent for the 2000 beam test simulation in GEANT3 [5];
- ***TileTBUADrawers***. Provides a structure similar to the one existent for the Beam Test ntuples [7].

We present a more detailed description of the ntuple variables in **Section 3.4.**

The concrete User Action class for a particular simulation is chosen at run time using the following FADS commands (*TileTBUASamples* is taken as an example):

```
/Actions/RegisterForAction TileTBUASamples BeginOfRun
/Actions/RegisterForAction TileTBUASamples EndOfRun
/Actions/RegisterForAction TileTBUASamples BeginOfEvent
/Actions/RegisterForAction TileTBUASamples EndOfEvent
/Actions/RegisterForAction TileTBUASamples Step
```

2.5 Physics lists

We provide a set of different physics lists for TileCal beam test simulations, which are based on different versions of the hadronic physics lists provided by the GEANT4 experts [6]. We have performed several tests to choose the best physics list but a final decision has not been made yet. Tests are still undergoing and the following packages are provided:

- **`libTileTBPhysicsLHEP_1.so`**, **`libTileTBPhysicsQGSP_1.so`**. We have developed these libraries in August 2002 using the current LHEP and QGSP lists for hadronic physics. Ion and muon physics from novice example **N04** and electromagnetic physics from extended example **TestEm3** are used.
- **`libTileTBPhysicsLHEP_2.so`**, **`libTileTBPhysicsQGSP_2.so`**. We have developed these libraries in the beginning of October 2002. They have to be used together with the **`libTilePackaging.so`** library, which includes GEANT4 EM, general, muon and ion physics. The code for the packaging library has been developed by the GEANT4 experts [6].

The users can choose a particular physics list at run time using the following commands:

```
/load TileTBPhysicsLHEP_1
/Physics/GetPhysicsList TileTBPhysics
```

2.6 Event generator

We have slightly modified the FADS *SingleParticle* generator and developed a *TileTBGenerator* that has some specific features for the beam test simulations.

- The initial direction of the particle momentum is always (1,0,0). So the beam is 'parallel', particles do not have a η/φ distribution.
- The initial positions of the particles can be distributed either along the Z-axis ('flat' distribution) or in the YZ plane ('uniform' distribution).
- All generated particles have the same initial energy;

The following commands configure the *TileTBGenerator* to generate negative pions with initial energy of 20GeV, a flat beam distribution of 10mm size along the Z-axis, with center in the initial point (-1500,0,0):

```
/Generator/Select TileTBGenerator
/Generator/TileTB/ParticleName pi-
/Generator/TileTB/momentum 20
/Generator/TileTB/spotType flat
/Generator/TileTB/spotSize 10. mm
/Generator/TileTB/initialPos -1500. 0. 0. mm
```

3. Installation and running

The TileCal beam test simulation application has been built on the LXPLUS cluster, but it can run on any machine where GEANT4 and FADS 1.1.0 are installed. In the following we will assume that GEANT4 and FADS 1.1.0 are already installed and properly configured on your system. This means in particular that you have two scripts prepared to set up the environment for FADS: **`AtlasEnvironment.sh`**(**`csh`**) and **`FADSsetup.sh`**(**`csh`**).

The sources are available on LXPLUS under the directory **`~tilecal/FADS/TestBeam`**. Some additional information and news are ale permanently placed on the TileCal simulation web site: http://tilecal.web.cern.ch/tilecal/Simulation/TileSoftwareSim.html.

Note: Due to some backward incompatibility introduced in the latest release of FADS the presented code will not work with FADS 1.1.1 and later. The new code is currently under development and will be available in the near future.

3.1 Installation

It is necessary to perform the following steps to install the TileCal beam test simulation application:

- Get the file **`~tilecal/FADS/TestBeam/Download/tiletb.tar.gz`**;
- Extract the archive in any directory of your choice (we will refer to this directory as **`$TILETB`**). You will have in **`$TILETB`** the same directory structure as in **`~tilecal/FADS/TestBeam`**;
- Source **`AtlasEnvironment.sh`**(**`csh`**) and **`FADSsetup.sh`**(**`csh`**) scripts to set up the environment for FADS;
- Go to **`$TILETB/TileTB`** and type: **`gmake`**. That builds the library **`libTileTB.so`** and places it under **`${FADS_WORKLIB}`**;
- Doing **`gmake`** in other subdirectories of **`$TILETB (./Generator`**, **`./Packaging`**, **`./TBPhysics*`**) also results in the creation of the corresponding shared libraries under **`${FADS_WORKLIB}`**.

Note: At the moment of writing, the application in **~tilecal/FADS/TestBeam** is built for RedHat 6.1 with egcs-1.1.2. If you have the same system, you can avoid the steps needed to build the libraries and simply copy the contents of **$TILETB/lib** to **${FADS_WORKLIB}**.

3.2 Running in interactive mode

There are two ways to run the simulations interactively:

- By using macro files;
- By typing command by command in the Goofy prompt.

We provide several macro files for the simulation of different beam test setups. There are four subdirectories of **$TILETB/TileTB/macros,** each one corresponding to one beam test setup. For example, in subdirectory **tb00,** which corresponds to the 2000 beam test, you will find the following macro files:

- **tb00_geo.mac** builds the geometry of the setup and visualizes it using the VRML file driver;
- **tb00_visevent.mac** allows the event visualization using the VRML file driver. The number of visualized events and the type of initial particles can be changed inside the macro;
- **tb00_samples.mac** performs sample simulation with negative pions using *TileTBUASamples* as user action;
- **tb00_nosamples.mac** performs sample simulations with negative pions but uses *TileTBUADrawers*.

In order to use any of the macro files listed above, you should follow these steps:

- Source **AtlasEnvironment.sh**(**csh**) and **FADSsetup.sh**(**csh**) scripts to set up the environment for FADS;
- Go to **$TILETB/TileTB/macros/tb00** and type **goofy**. You will get the Goofy logo and prompt;
- In the Goofy prompt type **/control/execute <macro_file_name>**.
- After the macro finishes and you get back the prompt, type **exit**.

You can also run the simulation interactively by typing the macro file commands one by one. The commands in macro files are accompanied by comments, helping to understand the command meaning.

3.3 Running in batch mode

Batch scripts can be found under the directory **$TILETB/run/batch**. For example, for the 2000 beam test setup, there is a subdirectory **tb00** with a file **tb00_samples.bat**. This batch file allows performing several simulations using the macro file **tb00_samples.mac**, just by changing the initial energy of the generated pions.

To submit a batch on LXPLUS, go to: $**TILETB/run/batch/tb00** and type, for example, the following command:

```
bsub -q 1nd tb00_samples.bat
```

When the batch finishes, the produced ntuples will be in the directory **$TILETB/run/output/tb00**. The names of the ntuple files indicate the energy, in GeV, of the generated pions.

3.4 Ntuple variables

There is a block of ntuple variables called '**GENERAL**' that is common for all the ntuples produced by the TileCal beam test simulation application. This block consists of the following variables:

- **EventNo**. Event number;
- **Vertex(3)**. Particle initial vertex;
- **Momentum(3)**. Particle initial momentum;
- **ESci**. Total energy deposited in all the setup scintillators;
- **LSci**. Total track length in all the setup scintillators;
- **EInner**. Total energy deposited in the inner iron (masters, spacers, front/end plates);
- **LInner**. Total track length in the inner iron;
- **EOuter**. Total energy deposited in the girders;
- **LOuter**. Total track length in the girders.

The remaining part of the ntuple depends on the beam test setup configuration and on the particular User Action chosen for that particular simulation.

3.4.1 Ntuples produced using `TileTBUASamples`

For each module used in a particular beam test setup, we provide a separate block of ntuple variables. The names of these blocks are made of two parts: the first one specifies the module type, the second one is an integer uniquely identifying a certain module within its type. The following names are used to distinguish different types of modules:

- **BARREL** – central barrel;
- **EXT** – positive extended barrel;
- **EXTN** – negative extended barrel;
- **MOD0** – module0;
- **PROTO** – prototype module;

For example, the ntuple produced for the 2000 beam test has the following blocks: **BARREL1**, **EXT1**, **EXTN1** and **MOD01**. For the 1998 beam test setup, which consisted of 5 prototypes and the module 0, the following blocks are present: **MOD01**, **PROTO1**, **PROTO2**, **PROTO3**, **PROTO4** and **PROTO5**.

In order to explain the variable names inside the blocks, let us consider the **BARREL1** block. The following variables are included:

- **EneB1** – total energy deposited in all the module scintillators;
- **EneB1S1T** – total energy deposited in all the scintillators of the first sample (A) of the module;
- **EneB1S1U(20)** – energies in the Up PMTs of the cells of sample A;
- **EneB1S1D(20)** – energies in the Down PMTs of the cells of sample A;

- **EneB1S2T** – total energy deposited in all the cintillators of the second sample (BC) of the module;
- **EneB1S2U(18)** – energies in the Up PMTs of the cells of sample BC;
- **EneB1S2D(18)** – energies in the Down PMTs of the cells of sample BC;
- **EneB1S3T** – total energy deposited in all the scintillators of the third sample (D) of the module;
- **EneB1S3U(7)** – energies in the Up PMTs of the cells of sample D;
- **EneB1S3D(7)** – energies in the Down PMTs of the cells of sample D;

3.4.2 Ntuples produced using `TileTBUADrawers`

The ntuples of this structure have blocks of variables similar to the ones described in S**ection 3.4.1**. The only exception is that all the variables corresponding to positive and negative extended barrel modules are grouped in one block called **EXT**. Each block consists of one or two variables that represent arrays of energies deposited in photomultipliers of the positive and negative side of the module. The PMTs are indexed by the corresponding hole number in the drawer. Table 1 lists all possible variables (1 is taken as module number for all the blocks)

Block	Variable	Comment
BARREL1	EneB1P(48)	Energies in PMTs of positive side in central barrel module 1
	EneB1N(48)	Energies in PMTs of negative side in central barrel module 1
EXT1	EneEB1P(48)	Energies in PMTs of positive side in extended barrel module 1
	EneEB1N(48)	Energies in PMTs of negative side in extended barrel module 1
MOD01	EneM01P(48)	Energies in PMTs of positive side in module 0
	EneM01N(48)	Energies in PMTs of negative side in module 0
PROTO1	EnePr1P(40)	Energies in PMTs of prototype module 1

Table 1. Variables in ntuples produced using **`TileTBUADrawers`**

Note: We put –999.9 for unused holes.

4. Test with negative pions

We have tested the application using a beam test setup consisting of 5 prototype modules. For the tests we have used negative pions with different energies in the range 20-300 GeV, entering the central module at angles: $\vartheta = 20^{o}$, $\varphi = 0$ (Fig.3).

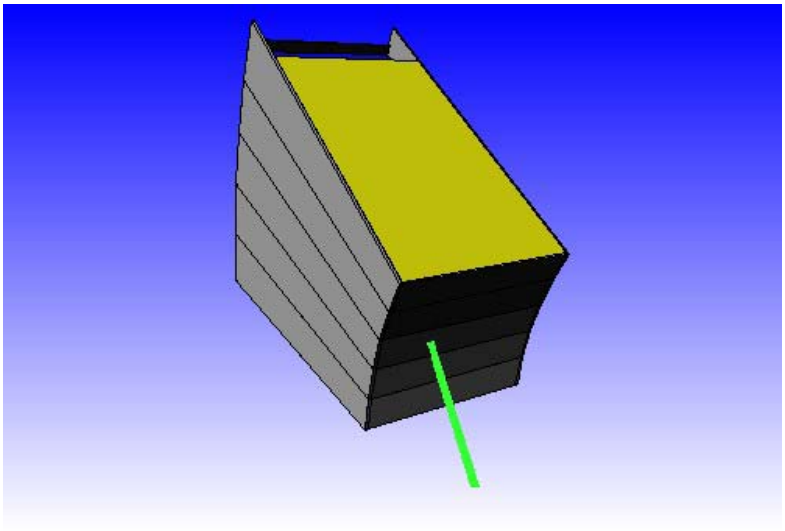

Figure 3. Beam test setup consisting of 5 prototype modules used for the tests with negative pions.

Figure 4 presents the resulting σ/E as a function of $1/\sqrt{E_{beam}}$ for the QGSP_1 and QGSP_2 physics lists. We have obtained the values for E and σ for the total energy deposited in all the scintillators by performing a Gaussian fits over a $\pm 2\sigma$ region from the mean. The points are fitted with the following function:

$$\frac{\sigma}{E} = \frac{a}{\sqrt{E_{beam}}} + b \tag{1}$$

The results from this simulation have been compared to the energy resolution plot for the same beam test setup as presented in the TileCal TDR [4]. In Figure 4 one can see that the simulation results obtained using two different hadronic physics lists are considerably different. The comparison to experimental results shows also that the QGSP_1 physics list approaches better the real data.

Simulations using the application presented in this note just started. We continue to participate in the GEANT4 physics validation process performing several tests with different beam test setups and different hadronic physics packages.

5. Acknowledgements

The authors would like to thank Andrea Dell'Acqua and Adele Rimoldi for the useful consultations, I. Ueda for the help in development of the first prototype TileCal simulation application in the FADS/Goofy and Anna Lupi for testing and bug reports.

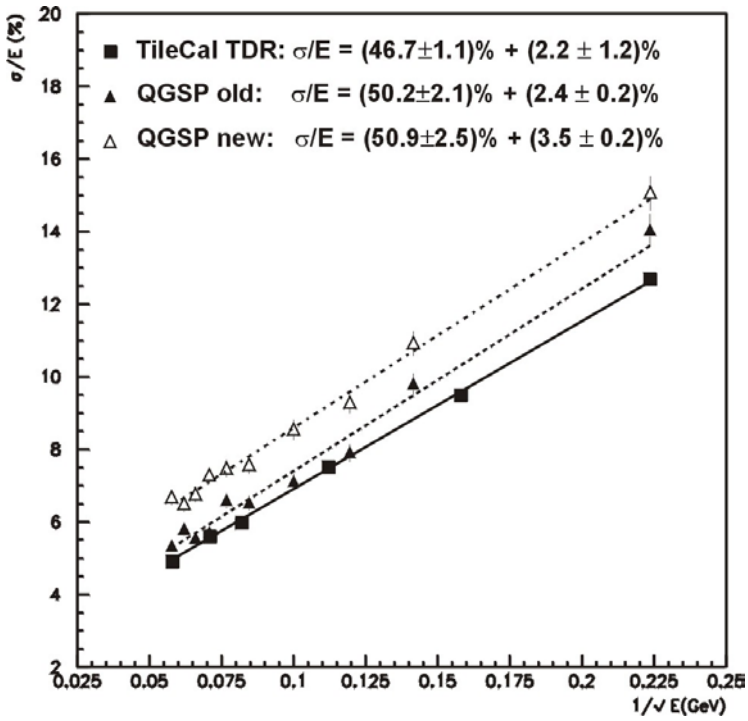

Figure 4. Comparison of the pion energy resolution obtained using the QGSP_1 and QGSP_2 physics lists, with the experimental results described in the TileCal TDR.

References

1. GEANT4 home page:
   http://geant4.web.cern.ch/geant4

2. FADS/Goofy web site:
   http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/domains/simulation

3. A. Dell'Acqua, K. Amako, A. Rimoldi et al.; Development of the ATLAS Simulation Framework; 8-037 CHEP01 paper

4. ATLAS Collaboration, Tile Calorimeter Technical Design Report, CERN/LHCC/**96-42** (1996)

5. T. Davidek; New Tilecal Beam Test Simulation Within Dice 95/Geant 3 Framework; ATLAS Internal Note ATL-TILECAL-2000-017

6. The home page of the GEANT4 Hadronic Physics working group:
   http://cmsdoc.cern.ch/~hpw/GHAD/HomePage

7. Description of testbeam ntuples
   http://atlas.web.cern.ch/Atlas/SUB_DETECTORS/TILE/testbeam/tb2002/analysis/standard_ntuples.html