Application of Geant4 Python Interface

Koichi Murakami
KEK / CRC

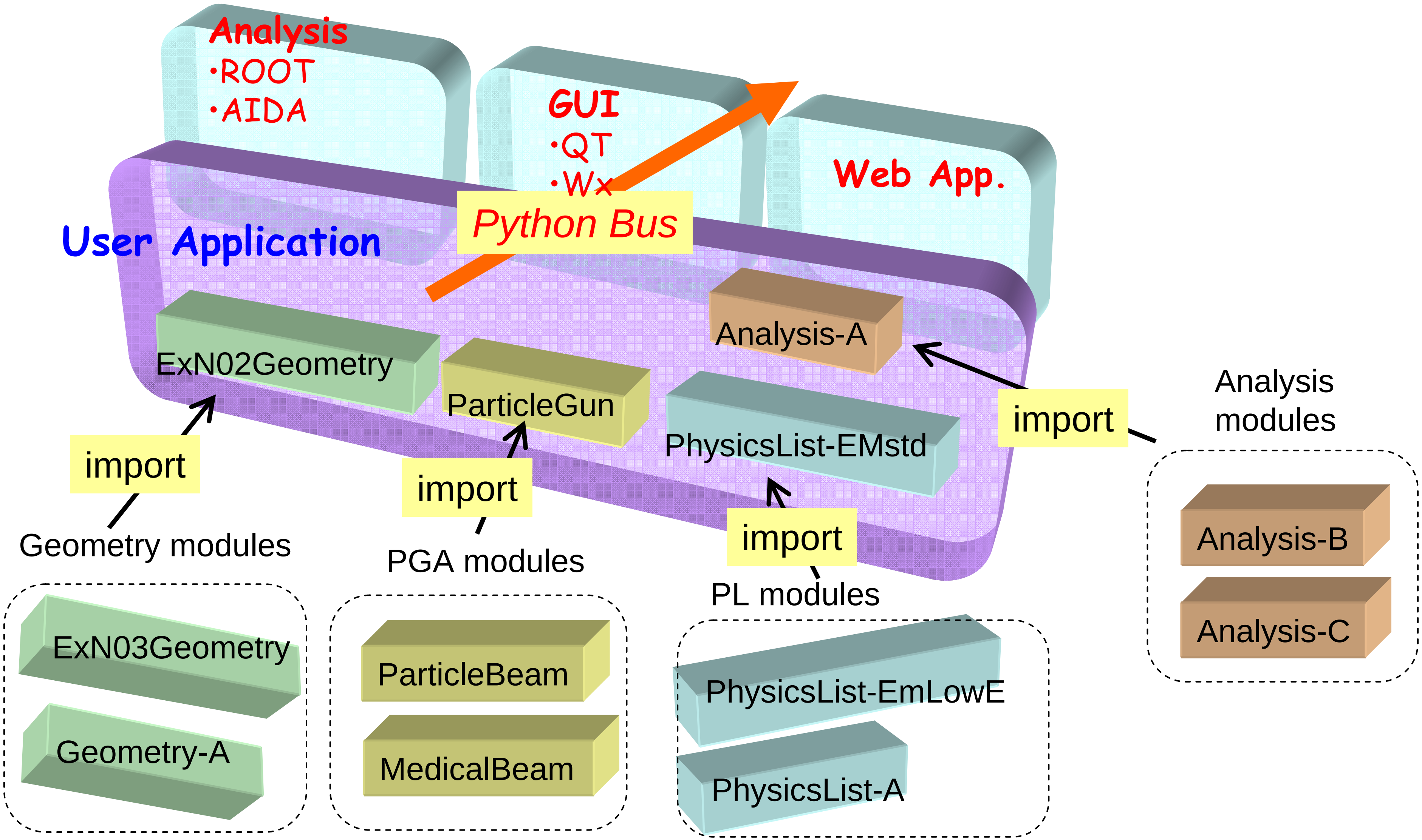
Python

- Shell Environment
  - ✓ front end shell
  - ✓ script language
- Programming Language
  - ✓ much easier than C++
  - ✓ supporting Object-Oriented programming
  - ✓ providing multi-language binding (C-API)
  - ✓ dynamic binding
    - » modularization of software components
    - » many third-party modules (*just plug-in*)
    - » software component bus
- Runtime Performance
  - ✓ slower than compiled codes, but not so slow.
  - ✓ Performance can be tunable between speed and interactivity.

Motivation of Geant4-Python Bridge

- Improving functionalities of current Geant4 UI
  - ✓ more powerful scripting environment
    - » driving Geant4 on a Python front end
    - » flow control, variables, arithmetic operation
- flexibility in the configuration of user applications
  - ✓ Modularization of user classes with dynamic loading scheme
    - » DetectorConstruction, PhysicsList, PrimaryGeneratorAction, UserAction-s
    - » It helps avoid code duplication.
  - ✓ quick prototyping and testing
- Software component bus
  - ✓ interconnectivity with many Python external modules,
    - » analysis tools (ROOT/AIDA), plotting tools (SciPy/matplotlib)
  - ✓ middleware for application developers
    - » GUI applications/web applications
    - » much quicker development cycle

Modular Approach and Software Component Bus

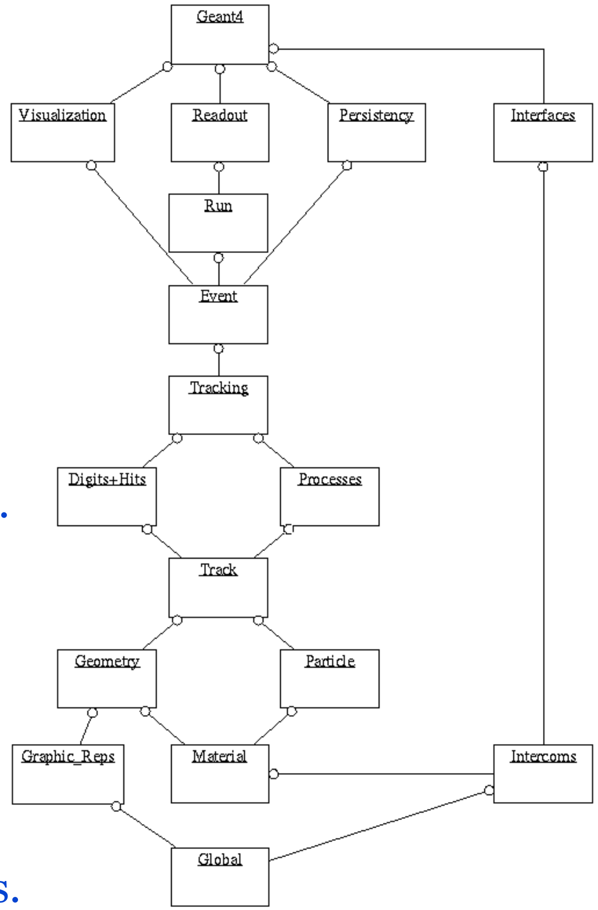


- "*Geant4Py*" is included in the Geant4 distribution since the 8.1 release.
  - ✓ please check the directory "`environments/g4py/`"
  - ✓ Linux and MacOSX(10.4+XCode 2.3/4) are currently supported.

- A G4-Python bridge as "*Natural Pythonization*" of Geant4
  - ✓ start with just importing the module;
    - » **`>>> import Geant4`**
  - ✓ not specific to particular applications
  - ✓ same class names and their methods
  - ✓ keeping compatibility with the current UI scheme
  - ✓ minimal dependencies of external packages
    - » only depending on *Boost-Python C++ Library*, which is a common, well-established and freely available library.

What is Exposed to Python

- Currently, over 100 classes over different categories are exposed to Python.
  - ✓ Classes for Geant4 managers
    - » G4RunManager, G4EventManager, ...
  - ✓ UI classes
    - » G4UImanager, G4UIterminal, G4UIcommand, ...
  - ✓ Utility classes
    - » G4String, G4ThreeVector, G4RotationMatrix, ...
  - ✓ Classes of base classes of user actions
    - » G4UserDetetorConstruction, G4UserPhysicsList,
    - » G4UserXXXAction
      - – PrimaryGenerator, Run, Event, Stepping,...
    - » can be inherited in Python side
  - ✓ Classes having information to be analyzed
    - » G4Step, G4Track, G4StepPoint, G4ParticleDefinition, ...
  - ✓ Classes for construction user inputs
    - » G4ParticleGun, G4Box, G4PVPlacement, ...



- NOT all methods are exposed.
  - ✓ Only safe methods are exposed.
    - » Getting internal information are exposed.
    - » Some setter methods can easily break simulation results.

Extensibility

- Your own classes can be exposed, and create your own modules in the Boost-Python manner.

```
BOOST_PYTHON_MODULE(mymodule){
    class_<MyApplication>("MyApplication", "my application")
        .def("Configure", &MyApplication::Configure)
      ;
```

- Once an abstract class is exposed to Python, you can implement/override its derived class in the Python side.

```
class MyRunAction(G4UserRunAction):
    """My Run Action"""
    def BeginOfRunAction(self, run):
        print "*** #event to be processed (BRA)=", \
        run.GetNumberOfEventToBeProcessed()
    def EndOfRunAction(self, run):
     print "*** run end run(ERA)=", run.GetRunID()
```

Compatibility with G4UImanager

- Geant4Py provides a bridge to G4UImanager.
  - ✓ Keeping compatibility with current usability
- UI Commands
  - ✓ **`gApplyUICommand("/xxx/xxx")`** allows to execute any G4UI commands.
  - ✓ Current values can be obtained by **`gGetCurrentValues("/xxx/xxx").`**
- Existing G4 macro files can be reused.
  - ✓ **`gControlExecute("macro_file_name")`**
- Front end shell can be activated from Python
  - ✓ **`gStartUISession()`** starts G4UIsession.
    - » `g4py(Idle): //` invoke a G4UI session
    - » when exit the session, go back to the Python front end
- *Python variables/methods starting "g" are global.*

Predefined Modules

- We will also provide site-module package as predefined components for easy-to-use as well as good examples.
  - ✓ Material
    - » NIST materials via G4NistManager
  - ✓ Geometry
    - » "exN03" geometry as pre-defined geometry
    - » "*EZgeometry*"
      - – provides functionalities for easy geometry setup
  - ✓ Physics List
    - » pre-defined physics lists
    - » easy access to cross sections, stopping powers, ... via *G4EmCalculator*
  - ✓ Primary Generator Action
    - » particle gun / particle beam
  - ✓ Sensitive Detector
    - » calorimeter type / tracker type
  - ✓ Scorer
    - » MC particle/vertex
- They can be used just by importing modules.

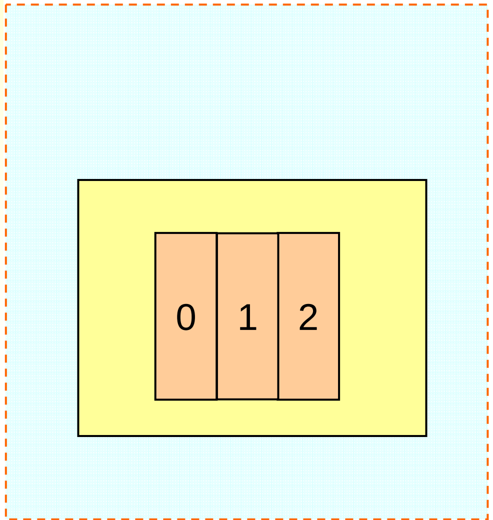
Yet Another Way to Create Geometry

- "EZgeom" module provides an easy way to create simple users geometries;
  - ✓ structure of geometry construction is hidden;
    - » Solid/Logical Volume/World Volume
    - » "EZvolume" is the only gateway to a physical volume from users side.
  - ✓ automatic creation of the world volume
    - » *volume size should be cared.*
  - ✓ creating CSG-solid volumes (Box, Tube, Sphere, ...)
  - ✓ changing volume materials
  - ✓ creating nested volumes
    - » placing a volume in the world by default
  - ✓ creating replicas / voxelizing BOX volumes
  - ✓ setting detector sensitivities
  - ✓ setting visualization attributes

Example of using EZgeom package

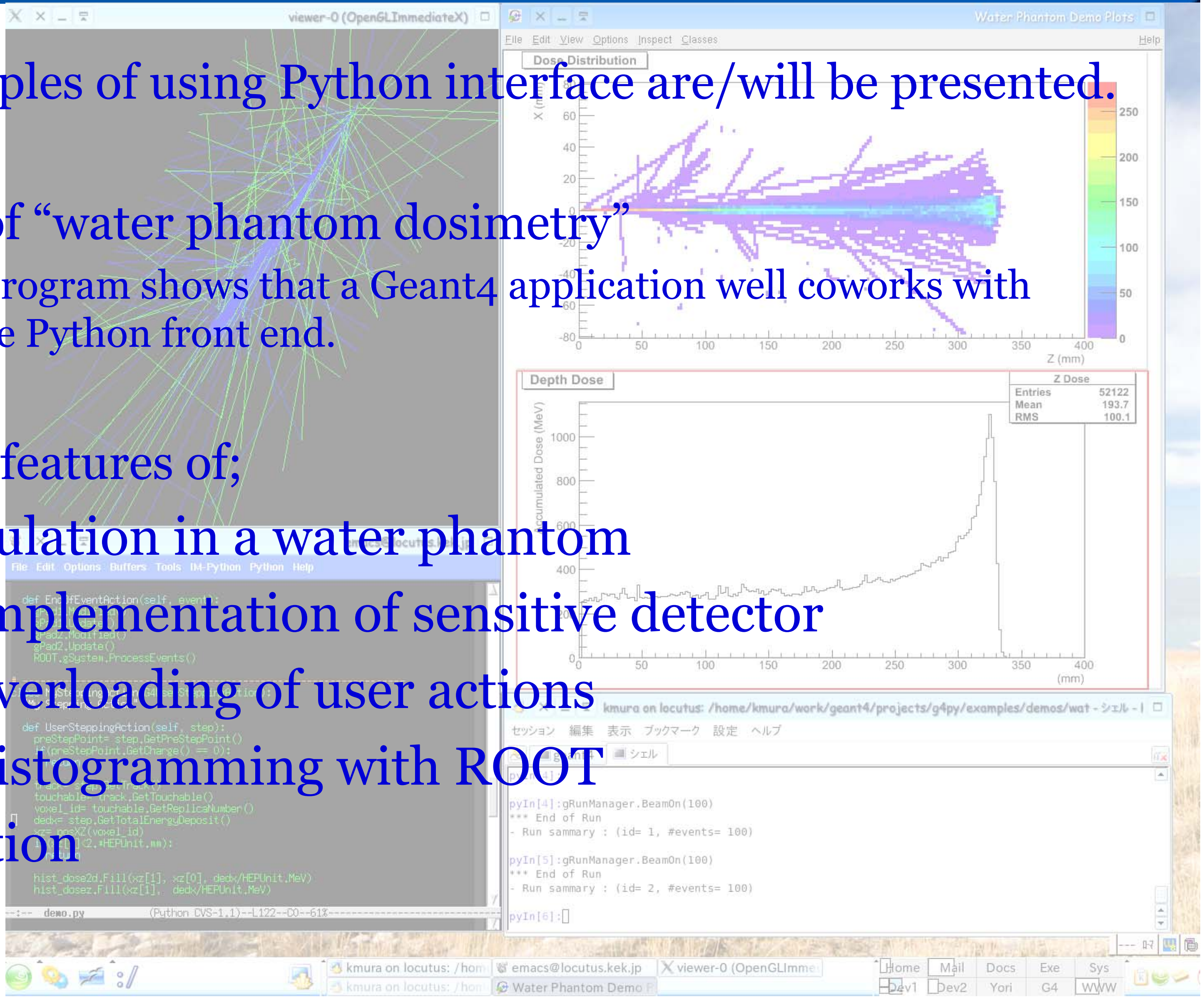
```
import NISTmaterials
from  EZsim import EZgeom
from EZsim.EZgeom import G4EzVolume

NISTmaterials.Construct()
# set DetectorConstruction to the RunManager
EZgeom.Construct()

# reset world material
air= gNistManager.FindOrBuildMaterial("G4_AIR")
EZgeom.SetWorldMaterial(air)

# dummy box
detector_box=G4EzVolume("DetectorBox")
detector_box.CreateBoxVolume(air, 20.*cm, 20.*cm, 40.*cm)
detector_box_pv=
detector_box.PlaceIt(G4ThreeVector(0.,0.,20.*cm))

# calorimeter placed inside the box
cal= G4EzVolume("Calorimeter")
nai= gNistManager.FindOrBuildMaterial("G4_SODIUM_IODIDE")
cal.CreateBoxVolume(nai, 5.*cm, 5.*cm, 30.*cm)
dd= 5.*cm
for ical in range(-1, 2):
  calPos= G4ThreeVector(dd*ical, 0., 0.)
  cal.PlaceIt(calPos, ical+1, detector_box)
```

less than 20 lines!!



A Medical Application Example

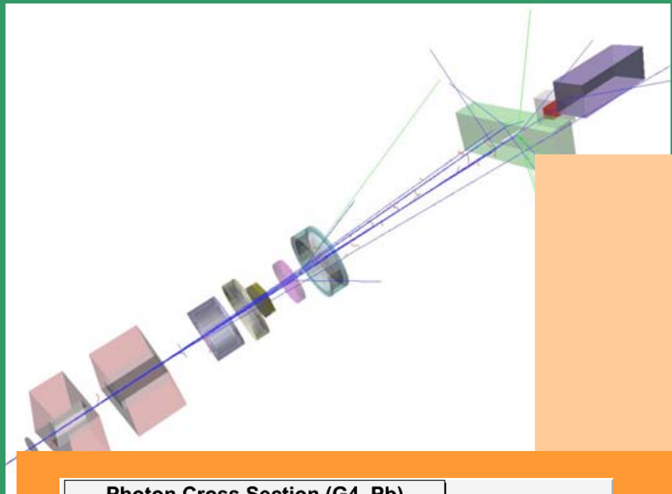
- Several examples of using Python interface are/will be presented.
- An example of “water phantom dosimetry”
  - ✓ This demo program shows that a Geant4 application well coworks with ROOT on the Python front end.
- You can look features of;
  - ✓ dose calculation in a water phantom
  - ✓ Python implementation of sensitive detector
  - ✓ Python overloading of user actions
  - ✓ on-line histogramming with ROOT
  - ✓ visualization

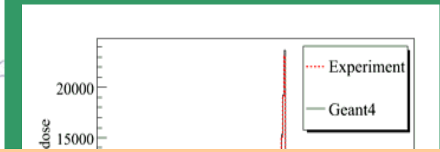


Pythonization Level

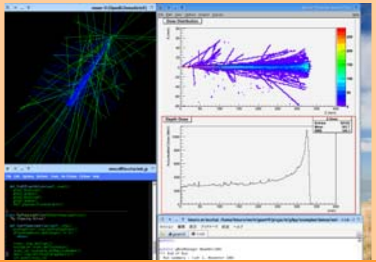
- Various level of pythonized application can be realized.
  - ✓ It is completely up to users!
  - ✓ Optimized point depends on what you want to do in Python.
- There are two metrics;
  - ✓ Execution Speed
    - » wrap out current existing C++ components, and configure them
    - » no performance loss in case of object controller
  - ✓ Interactivity
    - » more scripting in interactive analysis/rapid prototyping
    - » pay performance penalty to interpretation in stepping actions.

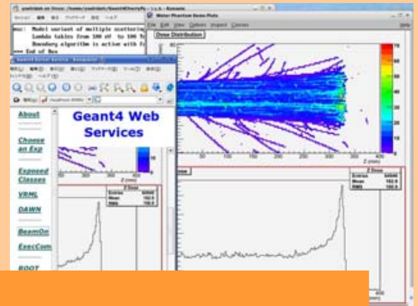
Applications of Pythonization



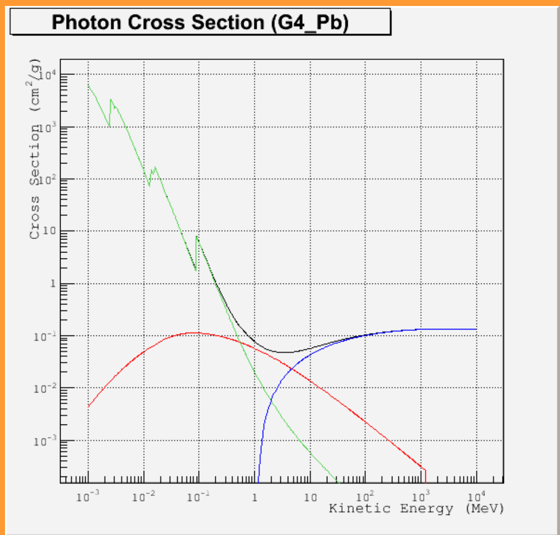


Execution Speed

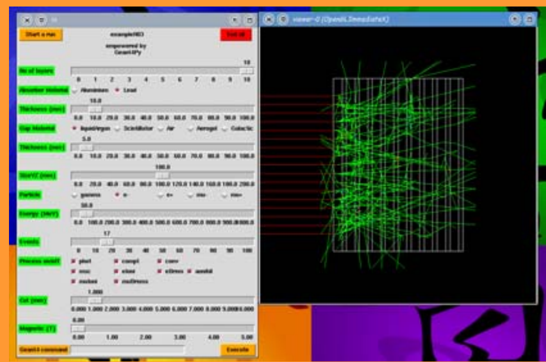




rver



"plotting photon cross sections"



"GUI control panel for educational uses:
Various parameters (detector parameters, initial particles, processes, etc) can be changed on GUI"

Inte
Pyth

tivity.

Summary

- A Python interface of Geant4 (Geant4Py) has been well designed and Geant4Py is now included in the latest release, 8.1.
  - ✓ check the “environments/g4py/” directory
- Python as a powerful scripting language
  - ✓ much better interactivity
    - » easy and flex configuration
    - » rapid prototyping
- Python as “Software Component Bus”
  - ✓ modularization of Geant4 application
  - ✓ natural support for dynamic loading scheme
  - ✓ interconnectivity with various kind of software components.
    - » histogramming with ROOT
- These applications show the flexibility and usefulness of dynamic configuration of user applications using Python.