Bruno: an introduction for beginners

A. Di Simone
INFN Tor Vergata

Outline

- Introduction
- Geometry
- User interface
- Physics lists
  - Physics recipes
- Particle generator
  - ROOT input
- Hits/Digits
- MCTruth
  - Detector boundaries
- Staged simulation
- Detector Survey

Introduction

- Bruno is presently the main full simulation tool used in SuperB
  - In parallel to a few standalone simulation programs for very specific use cases
- The result of a few years of frenetic activity, when main focus was to provide
  - As much information as possible
  - As reliable information as possible
  - As fast as possible
  - Using as few human resources as possible
- Result is a code with a lot of functionality, but whose general design may be probably optimized
  - Let us know if you are willing to help!
- One thing we did pretty well, however, was documenting the code:
- A quite detailed wiki is available
  - http://mailman.fe.infn.it/superbwiki/index.php/Geant4_SuperB_simulation_main_portal
- Please refer to that page for all the gory details
- Will give here just an overview of the main functionality

Geometry handling

- The geometrical description of the SuperB detector is presently defined using GDML
  - Geometry Description Markup Language
  - Application-indepedent geometry description format based on XML
- Provides text-based, human-readable definition of volumes
- Easy to modify without need of coding/compiling
- Being G4-independent, it allows interchange between different applications (i.e. G4-ROOT)
- Easily modularizable:
  - One xml file defining subdetector envelopes
  - One file for each subdetector, specifying the detailed geometry to plug inside the envelope
- Choice of “top” gdml file to use for geometry is done via the command line
  - ./Bruno -g SuperB.gdml

User interface

- We are presently using the “classical” G4 user interface
  - i.e. macro files
- We are well aware of its limitations
- Finding alternatives have been on our to-do lists for quite some time
  - Some of them, like python, have actually been prototyped, but never turned into a production code.
- If you are willing to help, let us know
  - In the meantime, please be patient a use .mac files...

Physics lists

- A *Physics List* is the definition of the physics processes to be simulated for each particle
- User can choose from the command line between some predefined physics lists
  - QGSP
  - QGSP_BERT (better for hadronic showers, but slower)
  - QGSP_EMV (worse msc treatment than QGSP, but faster)
- As long as CPU time is not an issue, QGSP_BERT is probably the best choice
  - If you are not interested in hadronic showers, you may gain some time by using QGSP
- ./Bruno -g SuperB.gdml -p QGSP

Physics recipes

- Use case presented by DCH at last meeting
  - Switch on/off individual physics processes on a per-volume basis
- Similar functionality is now implemented in Bruno, by means of “Physics Recipes”
  - Allow to tune the physics list with needed granularity
  - Caveat: this is potentially very dangerous. Do not use it unless you know what you are doing
    - you may severely harm the reliablity of the simulation
- Granularity is defined by Regions, not by Volumes
  - Makes more sense, since all physics-related quantities (such as production cuts) are related to the regions

Physics recipes

- In order to switch off or on a process, just create a region and associate a recipe to it:

```
/regions/create_region DIRCRegion DircWorld
/regions/physicsRecipe DIRCRegion myBadProcess remove
/regions/physicsRecipe DIRCRegion myNiceProcess add
```

- Every time a particle enters the DIRCRegion, myBadProcesses is suspended and myNiceProcess activated
- When the particle exits the region, the changes are undone, and the ones corresponding to the next region are applied (if any)
- This kind of manipulation requires that both processes are already present in the physics list.
  - Always keep in mind that by adding or removing with this procedure you are actually just suspending or resuming the processes.

Optical photons

- Optical processes deal with special particles called "opticalphoton"
  - think of them as photons in the optical range
  - it is important to keep in mind that they are not just "gammas" with lower energies
  - no matter how much you lower the energy of a "gamma" it will never become an "opticalphoton".
- The present implementation of Bruno masks all optical simulation behind a command line flag ("-O", capital o, not zero).

```
./bin/Linux-g++/Bruno -O myOpProp.mac -m singleparticle.mac
```

Particle generator

- A generator for background (RadBhabha) events is embedded in the full simulation
- In addition, the option to shoot single particles is available
  - Easy, fast check of simulation
  - May help detector experts in specific studies
- Example macro (singleparticle.mac) is provided:
  - ./Bruno.py -g SuperB.gdml -p QGSP -m singleparticle.mac
- Look at the wiki for detailed documentation

ROOT input

- Simulation input can be presently one of the following
  - Single particle: run in the same simulation job
  - Beam Strahlung events: run in the same simulation job
  - Ascii file: allows to use results from an external event generator (to be run beforehand as a different process)
- Now external generators can also use a ROOT file for data interchange
- A plain TClonesArray of TParticle, stored as branch in a tree
- BrunoROOTGenerator implemented and tested
- Configurable at runtime via macro file

```
/generator/ROOT/file /path/to/my/file.root
/generator/ROOT/tree NameOfTheTree
/generator/ROOT/branch NameOfTheBranch
```

Hits/digits

- Presently, hits are created for all subdetectors, and stored in a root file for further analysis
- Writing the code for creating the digits requires detailed knowledge of the detector readout
  - It should be done by detector experts
- We are providing a general infrastructure where detector-specific code can be plugged in
  - An example digitizer is provided to help developers
- In principle, all digitization code should be G4-independent
- This would allow to call digitization algorithms also without running the full G4 simulation, i.e. digitizing already existing hit files rather than newly created hits
  - For technical reasons this is not happening yet
  - However, this is a policy we would like to enforce

MCTruth

- Hits (digits) take into account detector response
  - They are the input for reconstruction
  - Their representation in memory could in principle be identical to the one used for real data
- Of course, when running simulation, you know many more things
  - Particle type, name of the process which originated it, exact position of the vertex where it was created, etc.
  - A HUGE amount of information, which needs to be somehow selected and stored on disk
    - Could include it in hits. Bad for many reasons. For example: you can have many hits from the same true particle and don't want to replicate info. Or, you can have a true particle not giving any hit and still want to record it
    - Better to use a separate class

MCTruth

- Some very basic MCTruth recording is implemented
  - Presently, one can save the status of any secondary particle at its creation
- Configuration is specified at runtime via a dedicated ascii file
  - Each line represents a policy
- Main parameter in a policy is the volume name:
  - The policy will affect only secondaries created in that volume (and its daughter volumes)
  - One can declare multiple policies for each volume

Detector boundaries

- The aim is to save a snapshot of particles exiting/entering a given volume (a subdetector)
  - Approach similar to the one used for MCTruth
  - Configuration done in a separate ascii file, but with less parameters:
    - SaveAllTracks
    - IgnoreAllTracks
    - trackPDG
- A set of policies for the main subdetectors is provided as default
- A very convenient way to have some “hit”-like information
  - Boundaries can be added to any volume, without writing any line of code
  - All is configured in .mac files at runtime
  - The desired information is automatically written in the output file

Truth Persistency

- Once the MCTruth/Boundary information is extracted from G4, one needs to write it to file
  - Presently, we are persistifying to ROOT file the following quantities
    - Trajectories (as chosen by some MCTruth policy)
    - MCTruth
    - Snapshots at volume boundaries
- In all cases, persistent objects are are plain ROOT objects (like Tparticle)
  - You can read them without Bruno

Staged simulation

- When combining the ROOTGenerator together with the detector boundaries, one gets *for free* the possibility to perform a *staged* simulation
- Simulate only up to a given point of the detector, e.g. the calorimeter
- In a second phase (i.e. a different simulation job), use boundary information and ROOTGenerator to resume the simulation job from where it was interrupted, e.g. completing simulation in the IFR
- This may result in huge savings of cpu time, in particular when testing different detector geometries
- Also, allows to quickly react to urgent requests:
  - e.g: SVT needs an urgent production.
  - We can simulate events only up to (excluding) the DCH
  - If one day DCH is interested in the same events, can resume simulation from where it was interrupted and add its own piece of code

Detector Survey

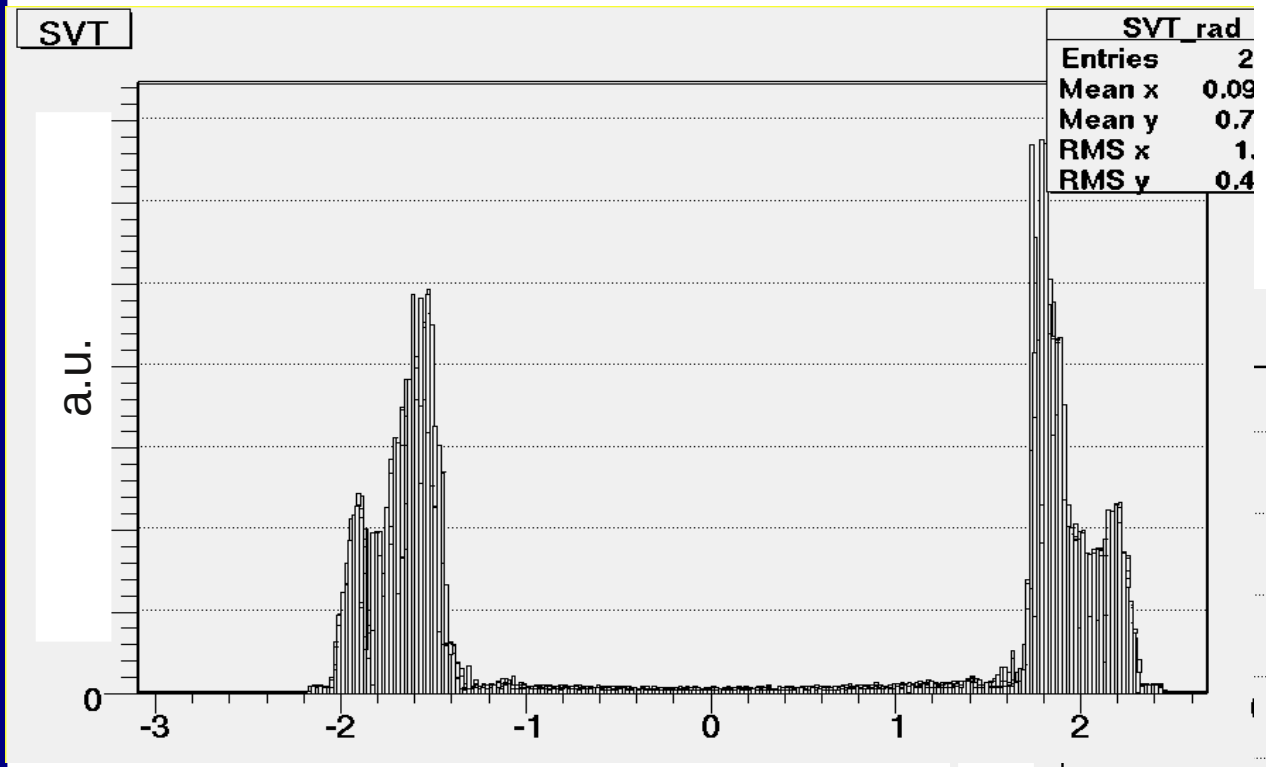
- Idea is to obtain eta/phi maps of relevant quantities concerning material distribution inside the detector
  - These can be easily calculated using geantinos as geometrical probe
- Shoot one geantino in a given direction
  - Record, step by step, the amount of material through which it is passing
    - Can be radiation length, nuclear interaction length
  - Fill 2D profiles
- One can choose to segment the material budget into several parts (i.e. subdetectors)

BrnDetSurvey

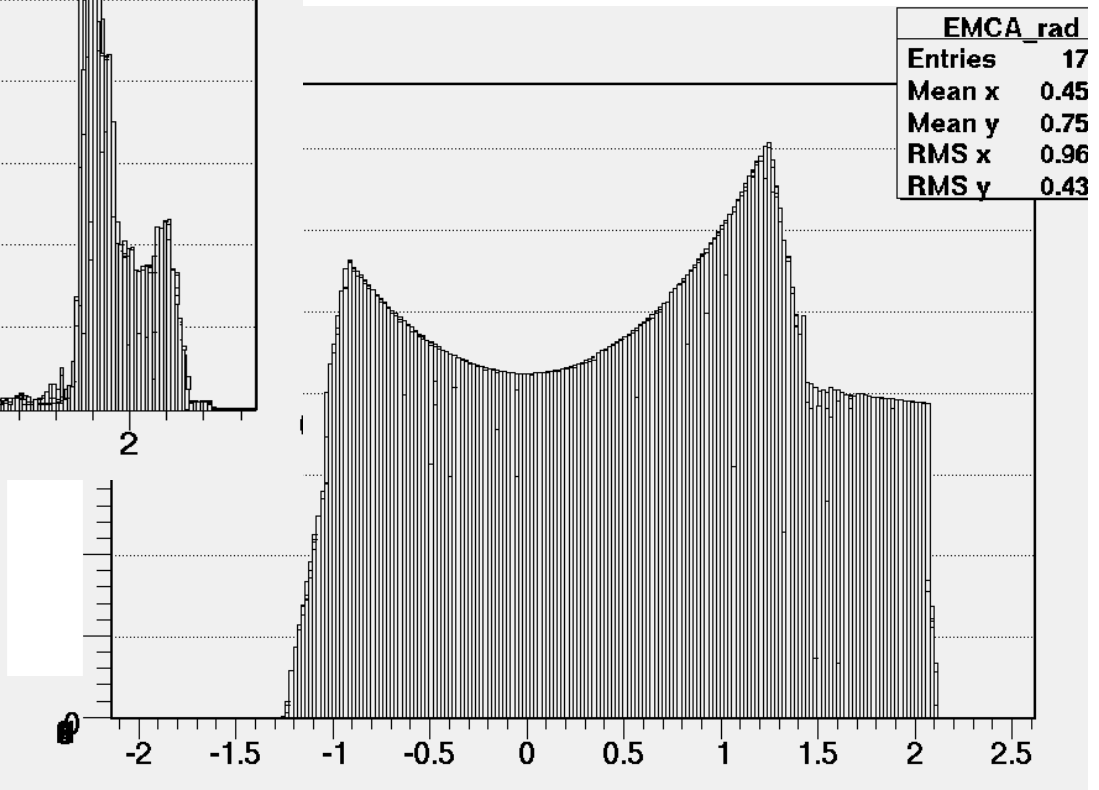
- A dedicated user action has been written to perform this kind of studies
- When activated, it will create a separate root file (DetSurvey.root) containing two folders (radLength and intLength), each one with 2D profiles for each subdetector
  - Note that it doesn't make any sense to activate this action when not using geantinos
  - In this case it's harmless, but its results have no physical meaning at all
- Configurability is still missing: one has to modify the code and recompile if behavior different from default is needed
- To be addressed (hopefully) in the near future, in the context of a global approach to the configurability of SuperB simulation

Example results

Of course plots are preliminary and unvalidated: axis units hidden on purpose



SVT rad. Length vs eta



EMCA rad. Length vs eta