SuperB Full Simulation

Andrea Di Simone
INFN Tor Vergata

Outline

- Introduction
- Simulation input:
  - Geometry
  - Physics list
  - Single particle generator
- Simulation output
  - Hits/digits
  - MCTruth
  - Detector boundaries
- Simulation monitoring
  - DetSurvey
- Open issues/plans

Introduction

- Full simulation can have a crucial role in this phase of the SuperB project
- While for high-statistics physics use-cases fast simulation is the preferred option, there are several detector-oriented studies for which one may need a complete, non-parameterized simulation of the particle interactions with the materials
  - This may also help in tuning the fast simulation itself
- The effort of the full-sim team is to provide to the community such a tool
  - GEANT4 as the underlying simulation engine
  - As detailed as possible geometry description
  - As complete as possible physics simulation

Geometry

- 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

Physics list

- 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

Particle generator

- A generator for background 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
- The key command is
  - /generator/gun/particle-properties 999 -5 5 0 1.5 100 100

PDG code (999 for geantinos)
Eta range
Phi range
Energy range

Hits/digits

- At each step, G4 checks whether the volume the particle is in is sensitive, and creates a hit
- A hit is a snapshot of a physical interaction of a track in a sensitive region of the detector
  - One can store various informations associated with a step, like:
    - Position and time of the step
    - momentum of the track
    - energy deposition of the step
    - geometrical information
- In general, *hits* represent the *physics* of the *detection* mechanism, while *readout electronics* simulation is taken into account when creating *digits*
- Example: muon crossing scintillator slab. Will do many steps, hence produce many hits. The PM however would produce only one signal of a given shape
  - Digitization should group together the hits and create the digit
    - This is where detector experts are really needed

Hits/digits (2)

- 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 data structure

MCTruth (2)

- Some very basic MCTruth recording is in place
- Presently, one can save the status of any secondary particle at its creation
- In addition, full trajectories (i.e. the “path” the particle follows inside the detector) can be saved as well
- 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
- Full syntax for policies is documented in the simulation wiki

Mtruth (3)

- Policies are designed to allow enough flexibility
  - Example:
    - Save all secondaries from my favorite subdetector
    - Save only photons above a given energy
    - Store trajectory of electrons above a given energy
    - Save all secondaries above threshold in some shielding volume and keep trajectory only for those which exit the original volume
- Possible improvements include
  - Possibility to save tracks based on the process which generated them (slot already present in policies)
  - Possibility to save interaction vertices in addition to interaction products (slot already present in policies)
  - Better configurability

Detector boundaries

- The aim is to save a snapshot of particles *exiting* a given volume (a subdetector)
  - Approach similar to the one used for MCTruth
  - Configuration done in a separate ascii file, but with less parameters
- A set of policies for the main subdetectors is provided as default
- Many uses for this kind of feature
  - Particle flux studies
  - detection/reconstruction efficiency measurement
  - Multi-stage simulation
    - Do simulation only up to detector A and save results to file
    - Use the snapshot at the exit of A as seed for a second simulation job (fast or full), if needed

Truth persistency

- Once the MCTruth/Boundary information is extracted from G4, one needs to write it to file
  - Presently, we are saving to ROOT file the following MCTruth-related quantities
    - Trajectories (as chosen by any MCTruth policy)
    - Snapshots at volume boundaries
      - One collection per subdetector, plus a default collection where snapshot coming from user-defined volumes will be stored
  - Still missing secondary particles for which no trajectory saving is requested
    - Will be done in the near future

Detector Survey

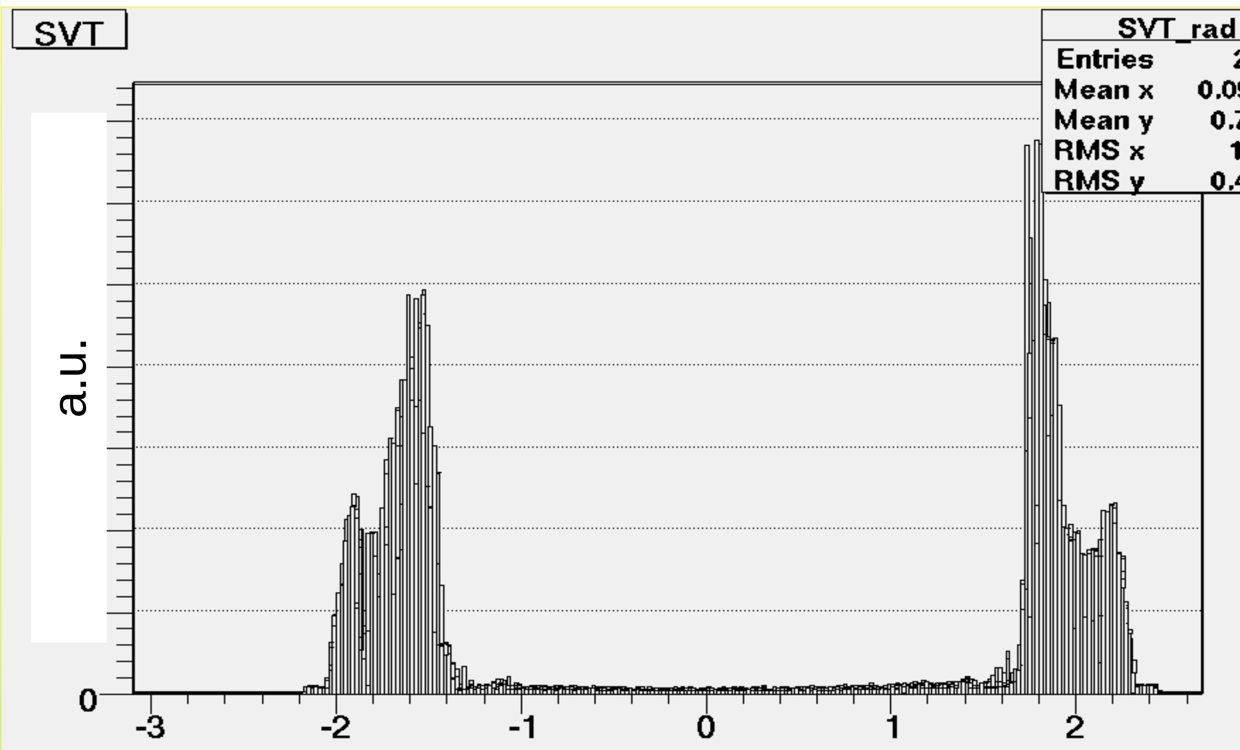
- Idea is to obtain eta/phi maps of relevant quantities concerning material distribution inside the detector
- These can be easily calculated using geantinos as geometrical probe
  - Shoot one geantino in a given direction
  - Record, step by step, the amount of material through which it is passing
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

BrunoDetSurvey

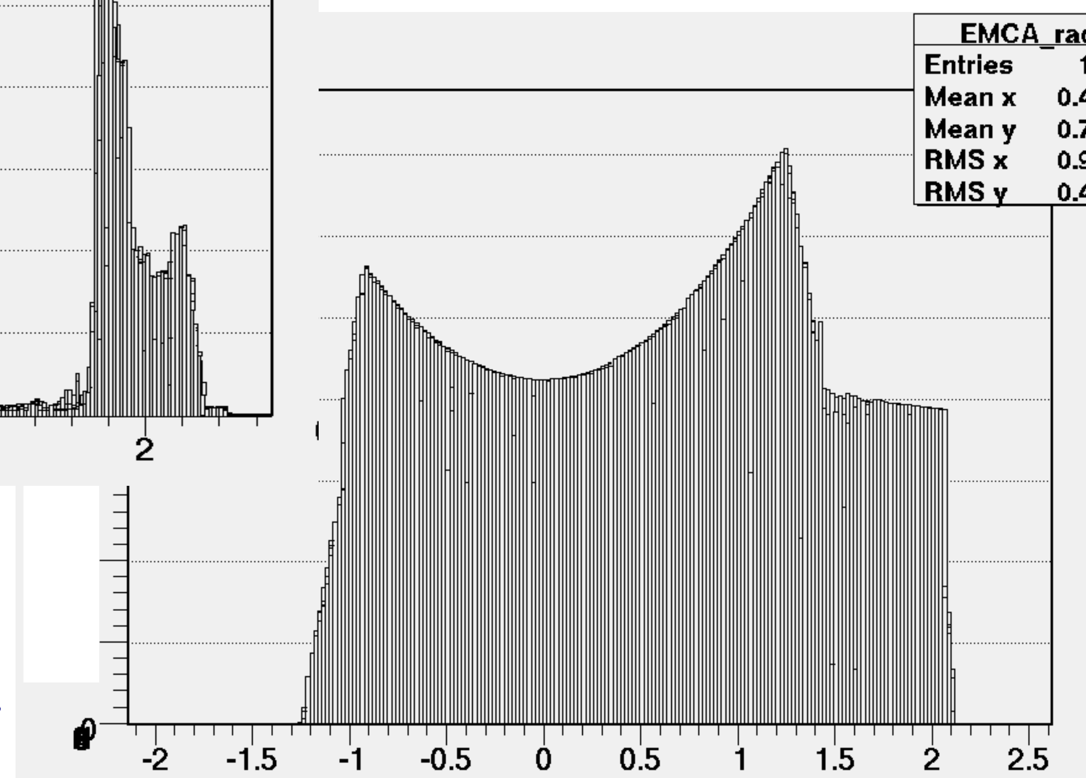
- Dedicated code has been written to perform this kind of studies
- When activated, it will create a separate root file (DetSurvey.root) containing two folders (radLength and intLength), each one with 2D profiles for each subdetector
- Note that it doesn't make any sense to activate this code when not using geantinos
  - In this case it's harmless, but its results have no physical meaning at all
- Configurability is still missing (sorry!): 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

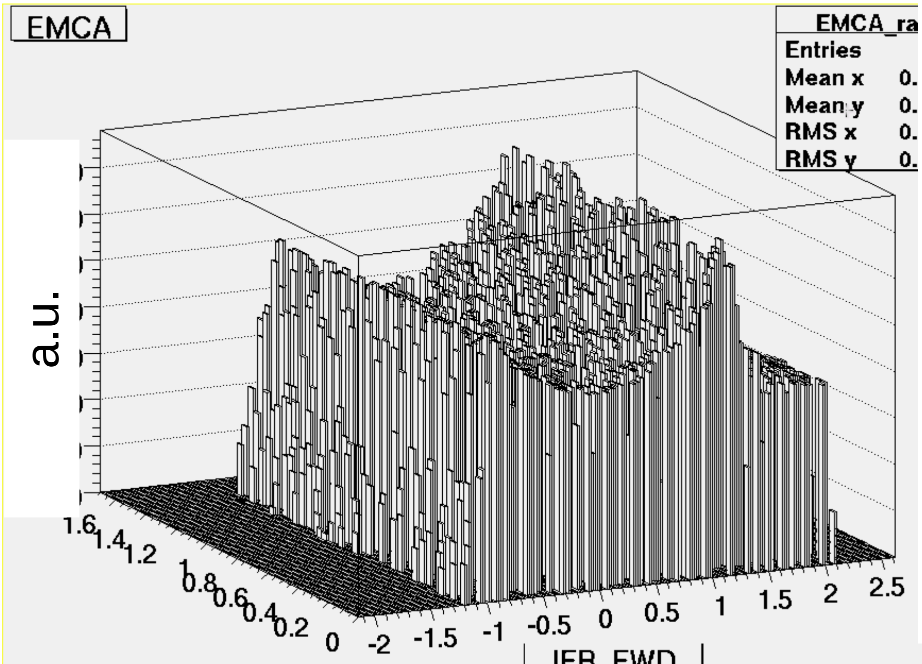


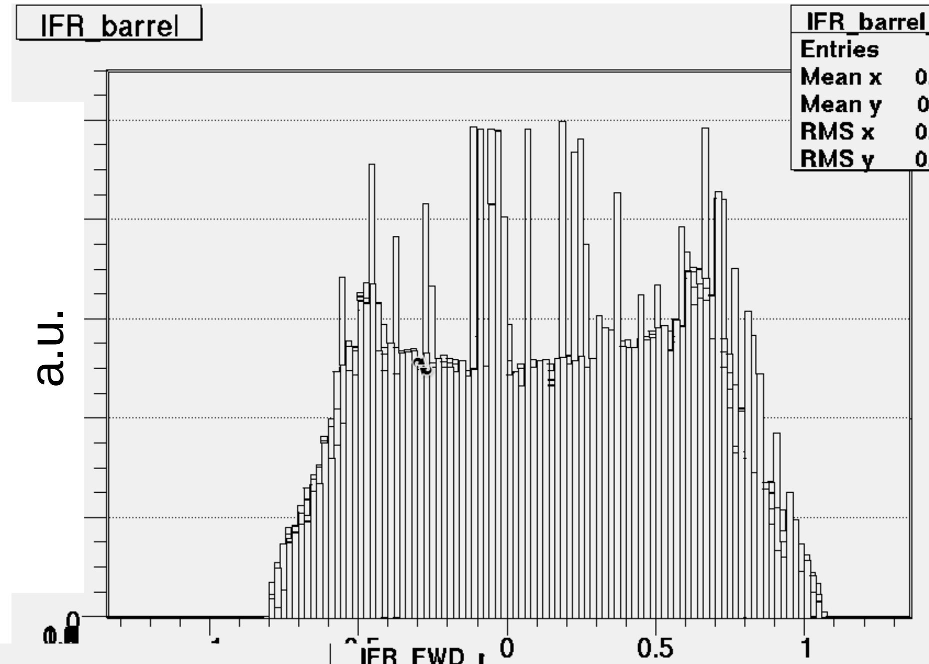
SVT rad. Length vs eta



EMCA rad. Length vs eta

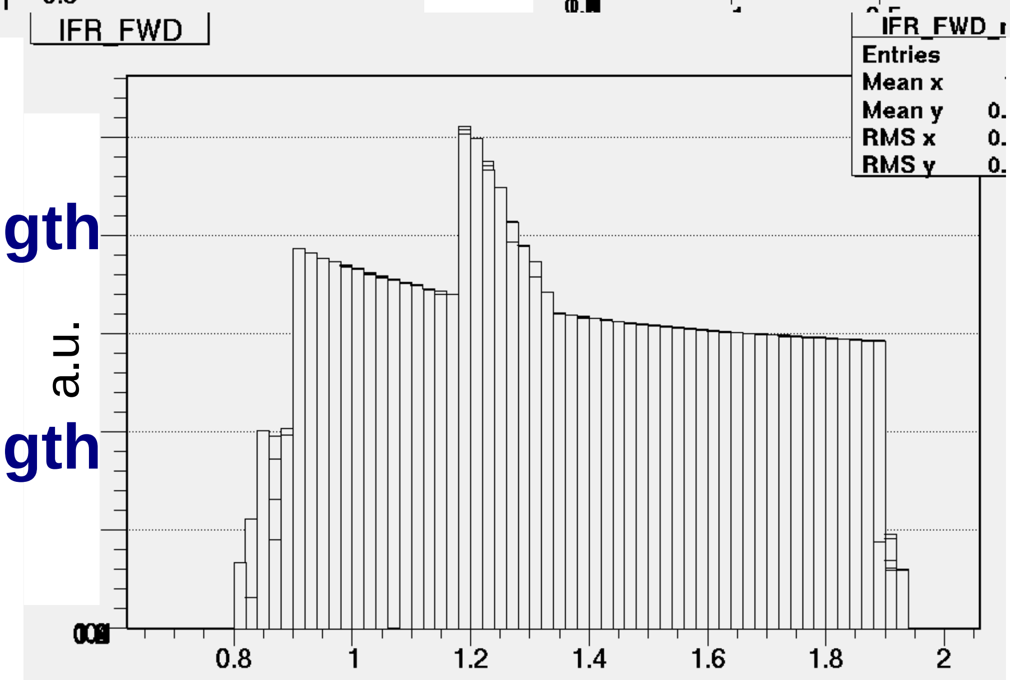
Example results





2D EMCA rad. length

IFR Barrel rad. length vs eta



IFR FWD rad. Length vs eta

Open issues

- Two main issues are arising with the increasing complexity of the simulation code
  - Configurability
    - I believe the macro/ascii-files approach is reaching its limits
    - Some better (more scalable, more flexible, less error-prone) method should be foreseen in the short-medium term
    - We tested in the last months a python-based configuration, with very encouraging results, but it may be worthwhile to discuss the configurability issue at a global (SuperB-wise) level, before implementing our custom solution
  - Persistency (structure of a simulated event when written to file)
    - We should discuss with subdetector experts and decide a structure for the simulated event (hits/digits/truth)
      - The sooner we freeze it, he easier the life will be for who is analyzing data
    - In parallel, it may be interesting to know whether some common framework for persistency is to be expected (SuperB-wise)

Plans (1)

- While basic functionality is in place, several improvements are still possible/needed
  - Modularization of some parts of the code
  - Persistency (where missing)
  - Automatic consistency tests
    - Check correctness of the geometry using both DetSurvey and G4 built-in tests
  - Code quality checks
    - finding bottlenecks (callgrind, perfmon)
    - Finding leaks (valgrind)
- Priority now will be migration to the latest G4 version
  - It will be transparent to the users
  - Better physics AND computing performance
  - Will bring in the latest GDML version, which would vastly ease the implementation of the IR geometry

Plans (2)

- Significant effort will be done in the context of code packaging/distribution
  - Presently one monolithic package contains core simulation code and subdetector contributions
    - This will be split, so that subdetectors will have their own place to develop and commit code, without interfering with core developments (and vice versa)
  - We did not have a release policy up to now
    - Will start providing releases on a regular basis
- Usability and configurability will be largely improved, making extensive use of python (or whatever alternative the SuperB offline SW foresees)
- From a general point of view, full simulation has reached a point where it ***needs*** *user feedback* and *subdetector contributions* in order to progress further
  - A document is being prepared and will be circulated to subdetector communities in order to collect specific requests

Conclusions

- SuperB full simulation is in a fairly good shape
- Hits are produced and saved to file, and slot for digitizer code is in place
- MCTruth information is extracted from G4 and saved to file too
- However, still lots of space for improvements
  - Coding improvements
  - Distribution/release improvements
  - Usability and configurability
- Closer interaction with fast sim is being investigated
- *User feedback will be most appreciated*
- DOCUMENTATION is provided in the simulation wiki