Enabling Ckov Monte Carlo Digitizer with C++ in MAUS

Ao Liu

Define an SD first

- Sensitive Detectors (SD) in MAUS are defined in the geometry/ .dat files (files/geometry/download);
  - For Ckov, they are in Ckov1.dat and Ckov2.dat;
  - 4 SD were previously defined at the PMTs, 1 is defined in the aerogel (Virtual);
- Defined SD are actually constructed in DetectorConstruction.cc (src/common_cpp/Simulation/DetectorConstruction.*);
  - Does not construct Virtual SDs. The one in the aerogel is NOT constructed

Current Status

- Sensitive Detectors (SD) in MAUS are defined in the geometry/ .dat files (files/geometry/download);
  - For Ckov, they are in Ckov1.dat and Ckov2.dat;
  - 4 SD were previously defined at the PMTs, 1 is defined in the aerogel (Virtual);

These have been done: aerogel has been defined as an SD (CKOV) and PMTs are no more SDs, but can be added back.

Define an SD first

- Sensitive Detectors (SD) in MAUS are defined in the geometry/ .dat files (files/geometry/download);
  - For Ckov, they are in Ckov1.dat and Ckov2.dat;
  - 4 SD were previously defined at the PMTs, 1 is defined in the aerogel (Virtual);
- Defined SD are actually constructed in *DetectorConstruction.cc* (src/common_cpp/Simulation/DetectorConstruction.*);
  - Does not construct Virtual SDs. The one in the aerogel was NOT constructed.

Define an SD first

- Sensitive Detectors (SD) in MAUS are defined in the geometry/ .dat files (files/geometry/download);
  - For Ckov, they are in Ckov1.dat and Ckov2.dat;
  - 4 SD were previously defined at the PMTs, 1 is defined in the aerogel (Virtual);

Added CKOV SD in the module construction.

- Defined SD are actually constructed in *DetectorConstruction.cc* (src/common_cpp/Simulation/DetectorConstruction.*);
  - Does not construct Virtual SDs. The one in the aerogel was NOT constructed.

Let the SD generate hits

- Hits are generated in SDs, *DetectorConstruction.cc* calls a module (*CKOVSD.** src/legacy/DetModel/Ckov/CKOVSD.hh) and a method (GetSDHits) to construct the hits at the SD.
  - CKOVSD.* were previously in an old format, written based on the old SD and Hit C++ modules. Now we use *MAUSSD.** and *Hit.** (src/common_cpp/DetModel/MAUSSD.*, src/common_cpp/DataStructure/Hit.*

Let the SD generate hits

- Hits are generated in SDs, *DetectorConstruction.cc* calls a module (*CKOVSD.** src/legacy/DetModel/Ckov/CKOVSD.hh) and a method (GetSDHits) to construct the hits at the SD.
  - CKOVSD.* were previously in an old format, written based on the old SD and Hit C++ modules. Now we use *MAUSSD.** and *Hit.** (src/common_cpp/DetModel/MAUSSD.*, src/common_cpp/DataStructure/Hit.*

These have been done: *CKOVSD.** remodeled, work with new Hit class, fills the channelId station, fills the hit structure (energy, time, etc.) and SD structure. They are moved to src/common_cpp/DetModel/Ckov/

Adding Hit types in Hit, increasing its capabilities

- The structure of the hits is defined in *Hit*.hh,* previously it did not have a CkovHit, and did not record mass of the hit particle (src/common_cpp/DataStructure/)
  - CkovHit needs CkovChannelId to record which station (detector) it is where the hit happens. Previously this file didn't exist.

franklin@

Adding Hit types in Hit, increasing its capabilities

- The structure of the hits is defined in *Hit*.hh*, previously it did not have a CkovHit, and did not record mass of the hit particle (src/common_cpp/DataStructure/)
  - CkovHit needs CkovChannelId to record which station (detector) it is where the hit happens. Previously this file didn't exist.

These have been done:
Hit.hh* has new methods: GetMass and SetMass, and a new attribute: _mass;
CkovHit is now registered as
typedef Hit<CkovChannelId> CkovHit;

franklin@

Adding *CkovChannelId.**

- In order to fill the template class, *CkovChannelId.** are needed (src/common_cpp/DataStructure/CkovChannelId.*)
  - CkovChannelId needs to record which station (detector) it is where the hit happens.

Adding *CkovChannelId.**

- In order to fill the template class, *CkovChannelId.** are needed (src/common_cpp/DataStructure/CkovChannelId.*)
  - CkovChannelId needs to record which station (detector) it is where the hit happens.

These have been done: Methods GetStation and SetStation have been defined. Records the station number now as _station.
They are called in the *CKOVSD.** to get the station number of the SD. Therefore, *Ckov*.dat* now have propertyInt: CkovStation – 0 for Ckov A and 1 for B.

Registering Ckov hits in an array and then in the mc tree

- Once individual hits are generated, , *MCEvent.** (src/common_cpp/DataStructure/) need to organize them into an array. Then, *MAUSEventAction.** (src/common_cpp/Simulation) use the above method at the end of the event action, and fill the hits.

Registering Ckov hits in an array and then in the mc tree

- Once individual hits are generated, , *MCEvent.** (src/common_cpp/DataStructure/) need to organize them into an array. Then, *MAUSEventAction.**(src/common_cpp/Simulation) use the above method at the end of the event action, and fill the hits.

These have been done: In *MCEvent.*,* CkovHitArray and GetCkovHits, SetCkovHits are defined.
_events->at(_primary)->SetCkovHits(new std::vector<CkovHit>());
added in *MAUSEventAction.cc,* now Ckov hits are in the mc tree.

Parsing the JSON structure so hits can be seen in ROOT

- Hits are ready, but can not be directly seen in ROOT, they must be processed from JSON to ROOT, done by *HitProcessor.**, and called by *MCEventProcessor.** (src/common_cpp/JsonCppProcessors/)

Parsing the JSON structure so hits can be seen in ROOT

- Hits are ready, but can not be directly seen in ROOT, they must be processed from JSON to ROOT, done by *HitProcessor.**, and called by *MCEventProcessor.** (src/common_cpp/JsonCppProcessors/)

These have been done: Added *CkovChannelIdProcessor.**, included them in *HitProcessor.** so that CkovChannelID are registered as branches.
Notice that *HitProcessor. also register a value branch called "mass", which was added in *Hit*.hh* and will be needed in the CkovMCDigitizer.**

*MCEventProcessor.** registers an array of Ckovhits by calling the GetCkovHits

Suggestion: Add details in *ObjectProcessor-inl.hh*

- When playing with the processors, I realized that when *ObjectProcessor-inl.hh.** (src/common_cpp/JsonCppProcessors/) attempts to process an object but found non-object, it produces an error:
  - throw(Exception(Exception::recoverable, "Attempt to pass a json "+tp+" type as an object","ObjectProcessor<ObjectType>::JsonToCpp"));
  - std::string tp = JsonWrapper::ValueTypeToString(json_object.type());
  - This will only show you the type of the thing being processed, but not where it belongs to, and what it's name is.

Suggestion: Add details in *ObjectProcessor-inl.hh*

- When playing with the processors, I realized that when *ObjectProcessor-inl.hh.** (src/common_cpp/JsonCppProcessors/) attempts to process an object but found non-object, it produces an error:
  - throw(Exception(Exception::recoverable, "Attempt to pass a json "+tp+" type as an object","ObjectProcessor<ObjectType>::JsonToCpp"));
  - std::string tp = JsonWrapper::ValueTypeToString(json_object.type());
  - This will only show you the type of the thing being processed, but not where it belongs to, and what it's name is.

I haven't done this in the codes, but I recommend: (makes the developer's life easier)
"Attempt to pass a json "+tp+" type as an object" + " whose name is "+JsonWrapper::JsonToString(json_object) + " in " + JsonWrapper::Path::GetPath(json_object)

Suggestion: Add details in MiceModule.cc

- When playing with the SDs, I realized that when *MiceModule.cc* (src/legacy/Config/) attempts to find a property (int, double, etc.) but fails will throw the "fullname()" of the module.
  - However, remember, that fullname() only finds the name of the current module. When propertyInt looks up in the parents, grandparents, … until the world geometry but fails to get what it wants, the fullname() is always the world's name!

Suggestion: Add details in MiceModule.cc

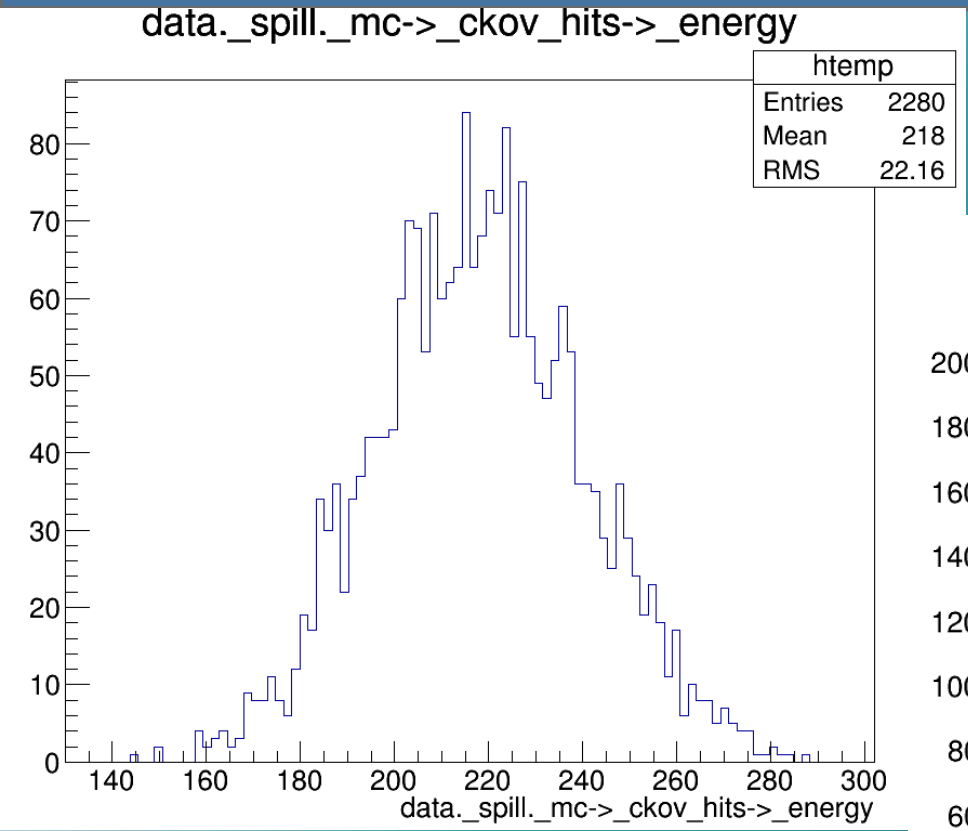
- When playing with the SDs, I realized that when *MiceModule.cc* (src/legacy/Config/) attempts to find a property (int, double, etc.) but fails will throw the "fullname()" of the module.
  - However, remember, that fullname() only finds the name of the current module. When propertyInt looks up in the parents, grandparents, … until the world geometry but fails to get what it wants, the fullname() is always the world's name!

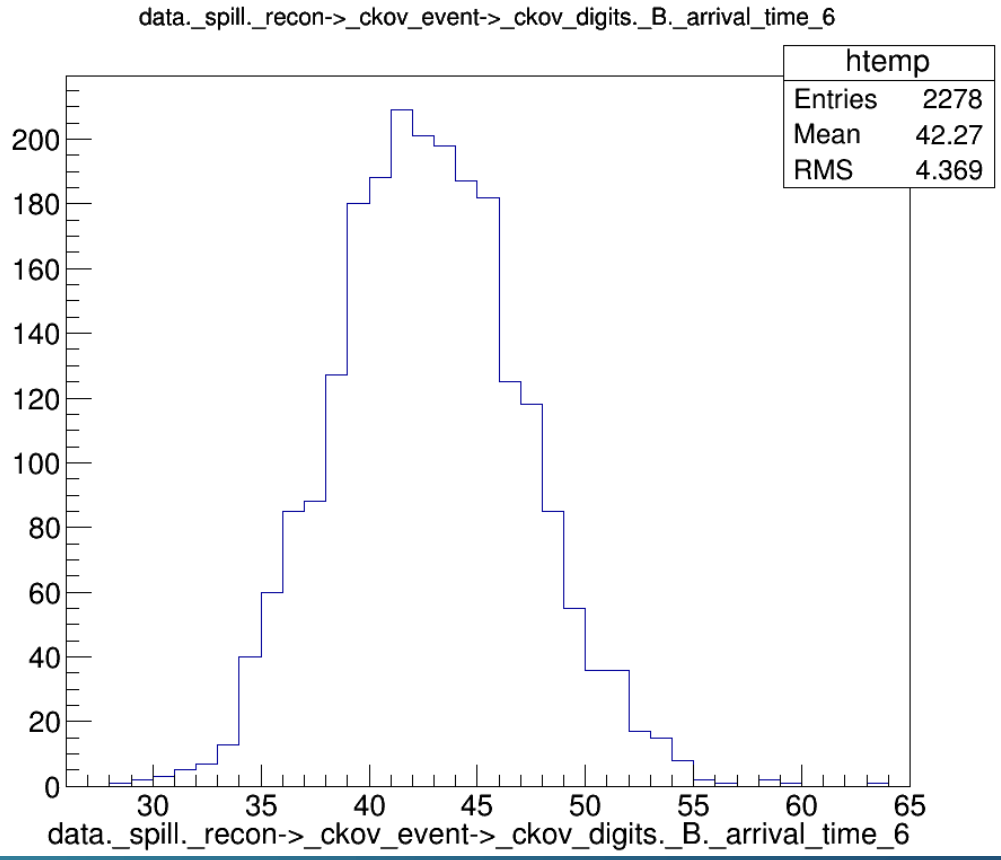
I have done this in only propertyInt:
I've set up a static string, called real_name;
Every time propertyInt is called, if it can't find the propertyInt in the current module, but the current module has a _mother, it does:
real_name += name()+" that is in ";
When throwing an Exception, it throws real_name+name()

Finally, Doing the MCDigitizer

- I have parsed Lucien's Python code to do the physics in the digitizer. I followed the digit structure (had to) Durga defined to make the structure work. Durga recently moved all the MCDigitizer from json to cpp, this new structure is also in CkovMCDigitizer.
- It goes through all the hits, defined in the previous slides, and calculate the npe based on the hit information. The npe are evenly distributed to the 4 pmts. Each arrival time is calculated accordingly.
- For a given event, there might be multiple hits;
  - If the hits belong to different stations, then they are filled into their corresponding digit structure at the same time;
  - Hits in the same station give one signal: total npe and an averaged arrival time at each of the PMTs.

The result is (just a working example)





ToDo in the future

- Look at the reconstructed data and compare with the MC data;
- Have a little plan: to move the MiceModule files from legacy to common_cpp and continue to maintain it there.
- Add functionalities like the real full name of the properties. etc.