Using multiple engines in the Virtual Monte Carlo package

Andreas Morsch\textsuperscript{a}, Benedikt Volkel\textsuperscript{a,b}, Ivana Hřivnáčová\textsuperscript{c}, Jan Fiete Grosse-Oetringhaus\textsuperscript{a}, Sandro Wenzel\textsuperscript{a}

\textsuperscript{a}CERN, \textsuperscript{b}Ruprecht-Karls-Universität Heidelberg, \textsuperscript{c}Institut de Physique Nucléaire (IPNO), Université Paris-Sud, CNRS-IN2P3

CHEP 2019, Adelaide, 05.11.2019
(Track 2 – Offline Computing)
CHAPTER I
Virtual Monte Carlo – how it used to be
VMC how it used to be

- abstract / unified interface to run detector simulation with different engines [such as GEANT3, GEANT4]
- one set of user hooks serves for any engine [e.g. stepping, begin / end of event, wrapped in one class derived from TVirtualMCApplication]
- one user stack implementation serves for any engine [class derived from TVirtualMCStack]
- 3 main interfaces, via
  1) TVirtualMC (e.g. via static TVirtualMC::Instance())
  2) any method of the MCAplication
  3) user stack

Schematic of dependencies and interplay between VMC, user framework and engine backend
VMC how it used to be

- abstract / unified interface to run detector simulation with different engines [such as GEANT3, GEANT4]
- one set of user hooks serves for any engine [e.g. stepping, begin / end of event, wrapped in one class derived from TVirtualMCApplication]
- one user stack implementation serves for any engine [class derived from TVirtualMCStack]
- 3 main interfaces, via
  1) TVirtualMC (e.g. via static TVirtualMC::Instance())
  2) any method of the MCApplication
  3) user stack

**limitation of running only a single engine**
Development goals

- overcome **limitation of running only one** simulation engine
- **allow partitioning** events among multiple different engines
  - e.g. use detailed GEANT4 simulation where necessary and use GEANT3 when less accuracy is already enough but full simulation is still desired
- **more freedom** for the user to inject his / her own VMC implementation
  - custom fast simulation to work with GEANT3 and GEANT4 on VMC level
  - complex / re-usable tasks neither suited for belonging to the stack nor to the application
- enable and test **interplay** of different engines
CHAPTER II
running multiple engines
Mixing multiple engines

vanilla sampling calorimeter to demonstrate mixing of engines

- sensitive layer
- passive layer

n particles of specific type and energy (here: electrons)
Mixing multiple engines

vanilla sampling calorimeter to demonstrate mixing of engines

- sensitive (GAPX)
- passive (ABSO)

<table>
<thead>
<tr>
<th>sensitive (GAPX)</th>
<th>passive (ABSO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEANT3</td>
<td>GEANT4</td>
</tr>
<tr>
<td>GEANT4</td>
<td>GEANT3</td>
</tr>
</tbody>
</table>

- in mixed scenario
  - keep detailed GEANT4 simulation of sensitive layers
  - use GEANT3 for passive layers

n particles of specific type and energy (here: electrons)
Mixing multiple engines

vanilla sampling calorimeter to demonstrate mixing of engines

simulation scenarios

<table>
<thead>
<tr>
<th>sensitive (GAPX)</th>
<th>passive (ABSO)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEANT3</td>
<td>GEANT4</td>
</tr>
<tr>
<td>GEANT4</td>
<td>GEANT3</td>
</tr>
</tbody>
</table>

- in mixed scenario
  - keep detailed GEANT4 simulation of sensitive layers
  - use GEANT3 for passive layers

TMCManager

```c
... void TransferTrack(Int_t targetEngineId) ...
...```
Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines

increase #layers

fix length

TMCManager

... void TransferTrack(Int_t targetEngineId) ...

...
Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines

- time elapsed relative to G3
- simulation more slowly using GEANT4 only
- speed-up is possible by mixing engines
- no scaling overhead with number of track transfers

TMCManger

```c
... void TransferTrack(Int_t targetEngineId) ...
...```
A custom VMC “fast simulation”

vanilla sampling calorimeter to demonstrate mixing of engines

- again a mixed scenario
A custom VMC “fast simulation”

vanilla sampling calorimeter to demonstrate mixing of engines

GEANT4   custom VMC “fast sim”

- again a mixed scenario
- “fast sim” draws total energy deposit from fitted distribution

disclaimer: proof-of-concept
A custom VMC “fast simulation”

- provide VMCFastSim class
  - only 2 methods to be implemented by the user
    1) VMCFastSim::Process()
    2) VMCFastSim::Stop()

- use VMCFastSim to implement a “FastShower” class

- code at
  - https://github.com/benedikt-voelkel/VMCFastSim
  - https://github.com/benedikt-voelkel/FastShower
A custom VMC “fast simulation”

• provide VMCFastSim class
  - only 2 methods to be implemented by the user
    1) VMCFastSim::Process()
    2) VMCFastSim::Stop()

• use VMCFastSim to implement a “FastShower” class

• code at
  - https://github.com/benedikt-voelkel/VMCFastSim
  - https://github.com/benedikt-voelkel/FastShower

actual fast simulation might be done in a few lines

```cpp
bool FastShower::Process() {
    if(GetCurrentParticle()->GetPDGCode() == 2212) {
        mStoreHit(mDistribution(mGenerator));
    }
    // ...
}
```
CHAPTER III

technical details – below the hood
Sketching the implementation

**partition simulation among multiple different engines**

1. **engine1** → **engine2**: track arrives at volume boundary
2. **engine1** → **engine2**: certain particle produced
3. **engine1** → **engine2**: track enters phase space
4. **engine1** → **engine2**: more complex condition

---

**Flowchart Diagram**:

- **engine manager**
- **engine 1**
- **engine 2**
- **user hooks**
- **user framework**
- **stack (exchange)**

1a. **engine 1** → engine manager
2. **engine manager** → **engine 2**
3. **engine manager** → **user hooks**
4a. **user hooks** → **engine manager**
4b. **user hooks** → **engine manager**
New classes and extensions

TMCManager

void ForwardTrack(Int_t toBeDone, Int_t trackId, Int_t parentId, TParticle* particle)
void TransferTrack(Int_t targetEngineId)
...

- singleton object
- needs to be explicitly requested by the user during construction of the UserApplication [keep runtime overhead as small as possible]

- VMCs are
  - owned by the manager
  - automatically registered when instantiated

- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines
New classes and extensions

**TMCManager**

void ForwardTrack(Int_t toBeDone, Int_t trackId, Int_t parentId, TParticle* particle)

void TransferTrack(Int_t targetEngineId)

... 

**TMCManagerStack**

A concrete implementation of TVirtualMCStack providing the interfaces accordingly for the usage and communication with the TMCManager.

**TVirtualMCApplication**

void RequestManager()

TMCManager* fMCManager

- singleton object
- needs to be explicitly requested by the user during construction of the UserApplication
  [keep runtime overhead as small as possible]

- VMCs are
  - owned by the manager
  - automatically registered when instantiated

- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines
Conclusion

- VMC package **enhanced** to allow usage of multiple engines and to **overcome previous limitations**
  - mix full simulation engines, e.g. GEANT3 and GEANT4
  - inject custom user VMC, e.g. some kind of fast simulation

- user is **free to decide how to partition** simulation between engines [geometry, particle type, phase space etc.]

- former run-mode (single engine) **fully preserved**

- **no runtime overhead** observed when moving tracks between engines

- **implementation details wrapped** into TMCManager and TMCManagerStack

- **example available in GEANT4_VMC package, E03c**
Thanks for your attention
Deployment overview (thanks to I. Hřivnáčová)

- crucial enhancements have been explained (more can be found in the BACKUP)
- example using multiple engines implemented along with GEANT4_VMC: E03c
  - a diff (e.g. to E03a) nicely shows that just a few modifications in the user code are necessary
- VMC now distributed via its own repository
- ROOT supports building with or without built-in VMC
  [ROOT version >= 6.18.00]
- releases
  - VMC, tag 1.0
    https://github.com/vmc-project/vmc
  - GEANT3_VMC, tag 3.0
    https://github.com/vmc-project/geant3
  - GEANT4_VMC, tag 5.0
    https://github.com/vmc-project/geant4_vmc
- new VMC documentation can be found at https://vmc-project.github.io
New classes and extensions  
(implementation examples)

<table>
<thead>
<tr>
<th>TMCManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SetUserStack(TVirtualMCStack* userStack)</td>
</tr>
<tr>
<td>void ForwardTrack(Int_t toBeDone, Int_t trackId, Int_t parentId, TParticle* particle)</td>
</tr>
<tr>
<td>void TransferTrack(Int_t targetEngineId)</td>
</tr>
<tr>
<td>template &lt;typename F&gt; Apply(F f)</td>
</tr>
<tr>
<td>template &lt;typename F&gt; Init(F f)</td>
</tr>
<tr>
<td>void Run(Int_t nEvents)</td>
</tr>
<tr>
<td>void ConnectEnginePointer(TVirtualMC*&amp; mc)</td>
</tr>
<tr>
<td>TVirtualMC* GetCurrentEngine()</td>
</tr>
</tbody>
</table>

- singleton object
- needs to be explicitly requested by the user during construction of the UserApplication
  [keep runtime overhead as small as possible]
- VMCs are
  - owned by the manager
  - automatically registered when instantiated
- handles
  - communication between engines
  - pausing and resuming engines
  - transferring particles / tracks between engines
# New classes and extensions

(implementation examples)

<table>
<thead>
<tr>
<th>TMCManager</th>
</tr>
</thead>
<tbody>
<tr>
<td>void SetUserStack(TVirtualMCStack* userStack)</td>
</tr>
<tr>
<td><strong>void</strong> ForwardTrack(Int_t toBeDone, Int_t trackId, Int_t parentId, TParticle* particle)</td>
</tr>
<tr>
<td>void TransferTrack(Int_t targetEngineId)</td>
</tr>
<tr>
<td>template&lt;typename F&gt; Apply(F f)</td>
</tr>
<tr>
<td>template&lt;typename F&gt; Init(F f)</td>
</tr>
<tr>
<td>void Run(Int_t nEvents)</td>
</tr>
<tr>
<td>void ConnectEnginePointer(TVirtualMC*&amp; mc)</td>
</tr>
<tr>
<td>TVirtualMC* GetCurrentEngine()</td>
</tr>
</tbody>
</table>

- user is still owner of constructed TParticle objects and numbering

- should be called in UserStack::PushTrack(...)

- additional last argument might be the target engine ID

```c++
void Ex03MCStack::PushTrack(Int_t toBeDone, Int_t parent, ..., Int_t& ntr, ...) {
    // TParticle construction yielding “particle”
    // define track ID
    ntr = GetNtrack() - 1;
    if(auto mgr = TMCManager::Instance()) {
        mgr->ForwardTrack(toBeDone, ntr, parent, particle);
    }
    // further implementation
}
```
New classes and extensions (implementation examples)

**TMCManager**

```cpp
void SetUserStack(TVirtualMCStack* userStack)
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                  Int_t parentId,
                  TParticle* particle)
void TransferTrack(Int_t targetEngineId)
template <typename F> Apply(F f)
template <typename F> Init(F f)
void Run(Int_t nEvents)
void ConnectEnginePointer(TVirtualMC*& mc)
TVirtualMC* GetCurrentEngine()
```

- call e.g. in `UserApplication::Stepping()`
- interrupts transport and transfers particle to target engine stack 
  [preserves momentum and geometry information]
- decide based on geometry, particle phase space / type etc.

```cpp
void Ex03MCApplication::Stepping() {
  // ...
  Int_t targetId = -1;
  if(fMC->GetId() == 0 && strcmp(fMC->GetCurrentVol(), "ABSO") == 0) {
    targetId = 1;
  } else if(fMC->GetId() == 1 && strcmp(fMC->GetCurrentVol(), "GAPX") == 0) {
    targetId = 0;
  }
  // ...
  fMCManager->TransferTrack(targetId);
}
```
New classes and extensions (implementation examples)

**TMCMManager**

```cpp
void SetUserStack(TVirtualMCStack* userStack)
void ForwardTrack(Int_t toBeDone, Int_t trackId,
                   Int_t parentId,
                   TParticle* particle)
void TransferTrack(Int_t targetEngineId)
template <typename F> Apply(F f)
template <typename F> Init(F f)
void Run(Int_t nEvents)
void ConnectEnginePointer(TVirtualMC*& mc)
TVirtualMC* GetCurrentEngine()
```

- the type F is assumed to implement () taking a TVirtualMC as an argument
- f is applied to all registered engines
- passed pointer will be kept up-to-date

```cpp
void Ex03MCApplication::InitMC(
    std::initializer_list<const char*> setupMacros) {
    // ...
    fMCManager->Init([this](TVirtualMC* mc) {
        mc->SetRootGeometry();
        mc->SetMagField(fMagField);
        mc->Init();
        mc->BuildPhysics();
    });
    // ...
}
```

```cpp
Ex03DetectorConstruction::Ex03DetectorConstruction() {
    // ...
    if(auto mgr = TMCMManager::Instance()) {
        mgr->ConnectEnginePointer(fMC);
    }
    // ...
}
```
Mixing multiple engines (continued)

vanilla sampling calorimeter to demonstrate mixing of engines

- track length in ABSO (top) relative to G3
- track length in GAPX (top) relative to G3
- no cut optimisation done per engine yet, however, simulated track lengths of same order of magnitude