Geant4e Track Extrapolation in the Belle II Experiment

Leo Piilonen, Virginia Tech
Thomas Kuhr, KIT
Takanori Hara, KEK

on behalf of the Belle II Collaboration

CHEP 2013, Amsterdam
geant4e, a part of geant4, is used during event reconstruction (*not simulation*!). It computes

- the average trajectory of a charged track, assuming a local helix in local magnetic field for each step
- the covariance matrix along this trajectory due to
  - multiple scattering
  - ionization
  - track curvature

using C++ port of the geane code in geant3 (developed by the European Muon Collaboration)
During event reconstruction, use Geant4 for track propagation outward from the drift chamber’s exit; needed for particle identification.

$e^-$ (7 GeV) to $e^+$ (4 GeV)

- Vertex detectors
- Drift chamber
- Particle identifiers
- EM Calorimeter
- $K_L$ and muon detector
**geant4** detailed model of the Belle II detector:
- non-uniform solenoidal magnetic field (~1.5 T)
- for **geant4** simulation and **geant4e** track propagation

---

**Beam-backgrounds:**
- neutrons
- photons
- electrons
- positrons
For particle identification, each track is extrapolated outward using 5 hypotheses \((e, \mu, \pi, K, p)\)

- **swim each track** from outer edge of drift chamber through the calorimeter (or until it stops)
- **store** time, position, momentum and covariance matrix at entrance/exit of selected volumes
For muon identification, each track is extrapolated outward using $\mu$ hypothesis (but could use the other 4)

- ✔️ swim each track through $K_L$-muon detector with Kalman fitting to matching hits and track adjustment
- ✔️ store time, position, momentum and covariance matrix at entrance/exit of each layer
Geant4e and Geant4:

Belle II has two usage modes of Geant4e:

- for real events: standalone

- for simulated events: coexisting with Geant4, since we do simulation and reconstruction in one pass
Geant4e and Geant4, cont’d:

geant4e, as distributed, cannot be used with geant4:
  × incompatible particle lists
  × incompatible physics processes
  × conflicting usage of sensitive-detector geometry
  × distinct states when calling RunManager
  × incompatible user actions (SteppingAction etc)

geant4e, as distributed, is limited:
  × propagates only electrons, positrons and photons

We have resolved these issues and extended geant4e. All mods are done outside the geant4 code base.
1) Particles and Physics Processes:

- PhysicsList is user’s concrete implementation of G4VUserPhysicsList, and must define:
  - ConstructParticle()
  - ConstructProcess()
  - SetCuts()

- geant4 and geant4e use distinct PhysicsLists.

- Significant overhead to change PhysicsList when switching between geant4 and geant4e so avoid this!

- Define a combined PhysicsList that incorporates geant4 and geant4e functionality.
1) Particles and Physics Processes, *cont’d*:

- **ConstructParticle()** defines

  - gamma, e+, e−, mu+, mu−, pi+, pi−, pi0, kaon+, kaon−, kaon0, kaon0L, kaon0S, proton, anti_proton, neutron, anti_neutron, geantino, chargedgeantino, opticalphoton, etc. (the standard particles)

  - g4e_gamma, g4e_e+, g4e_e−, g4e_mu+, g4e_mu−, g4e_proton, g4e_antiproton, g4e_pi+, g4e_pi−, g4e_kaon+, g4e_kaon−

  with PIDcode = 1000000000 + stdPIDcode

PhysicsList in the distributed geant4e defines only three particles (gamma, e+, e−) and these conflict with geant4 usage.
1) Particles and Physics Processes, cont’d:

- SetCuts() does

  SetCutsWithDefault() using default = 1.0*mm for the standard particles

- SetCutsWithDefault() using default = 1.0E9*cm for the new g4e_* particles
2) Common detector geometry:

- During simulation, G4SteppingManager calls user code to process steps through “sensitive” detector volumes and record the hits therein.

- During reconstruction, our custom version of StepLengthLimitProcess() disables this behaviour:

```c++
G4ParticleChange aParticleChange;

G4VParticleChange*
   ExtStepLengthLimitProcess::PostStepDoIt( const G4Track& track,
                                             const G4Step& )
{
    aParticleChange.Initialize( track );
    aParticleChange.ProposeSteppingControl( AvoidHitInvocation );
    return &aParticleChange;
}
```
3) **geant4e navigation and “target” geometry:**

- Do not use the special `G4ErrorPropagationNavigator` in `geant4e`. Instead, use `G4Navigator` defined in `geant4`.

- `geant4e` requires a “target” surface; its navigator [which we avoid] checks if the track crosses this surface after each step. **We do this check in our steering code.**

- The available surfaces are not adequate for our needs because they are not closed.

- Our custom version of `G4ErrorCylSurfaceTarget` is a closed surface that includes the cylinder endcaps.
4) Distinct run states and user actions:

- During our `geant4e` initialization, detect the presence of `geant4` by a non-empty `G4ParticleTable`.

- If `geant4e` is running standalone, there is no need to preserve the `geant4` state from one event to next.

- If `geant4e` co-exists with `geant4`, restore the `geant4 idle` state and save pointers to its `UserActions`:

  ```cpp
  InitGeant4e();
  G4StateManager::GetStateManager() -> SetNewState(G4State_Idle);
  m_savedTrackingAction = UserTrackingAction;
  m_savedSteppingAction = UserSteppingAction;
  ```
4) Distinct run states and user actions, cont’d:

During reconstruction of one event:

```c++
if ( geant4e co-exists with geant4 ) { // hide geant4 actions
    UserTrackingAction = NULL;
    UserSteppingAction = NULL;
}
```

extrapolate all tracks in the event using g4e_* particles;

```c++
if ( geant4e co-exists with geant4 ) { // restore geant4 actions
    UserTrackingAction = m_savedTrackingAction;
    UserSteppingAction = m_savedSteppingAction;
}
```
5) Other *geant4e* modifications:

- The distributed `G4ErrorPropagatorManager` replaces the standard `G4Navigator` with `G4ErrorPropagationNavigator`. Our custom version avoids this.

- The distributed `MagFieldLimitProcess` assumes that the magnetic field is along the \( z \) axis. Our custom version removes this assumption.

- The distributed `G4EnergyLossForExtrapolator` defines energy-loss processes for electrons and positrons. Our custom version extends these to muons, pions, kaons, and protons (both signs).
6) Muon identification:

- Extrapolate each reconstructed track from the CDC exit point into the KLM (barrel and endcap) using geant4e
  - default is muon hypothesis only *(but others are allowed)*
- Look for matching 2D hit upon crossing each KLM layer
- Kalman fitting: If there is a matching 2D hit in the layer, use its position and uncertainty to adjust the position and direction of the extrapolated track before continuing to the next layer
- Accumulate $\chi^2$ between in-plane hit and track positions
- Finish extrapolation when the track exits the KLM or stops
6) Muon identification, *cont’d*:

- Use two variables to distinguish muon from hadron in KLM:
  - $\chi^2$ per degree of freedom for in-plane position differences of all matching 2D hits and the extrapolated track ($\chi^2_{dof}$)
  - Difference in range between outermost matching 2D hit and the extrapolated track ($\Delta \ell$)

- Consult two-dimensional PDFs:
  - $P^\pm_\mu (\Delta \ell, \chi^2_{dof})$ for muons
  - $P^\pm_\pi (\Delta \ell, \chi^2_{dof})$ for pions
  - $P^\pm_K (\Delta \ell, \chi^2_{dof})$ for kaons

- Compute likelihood of the track being a muon vs hadron:
  - $L \equiv 0$ if no matching 2D hits; otherwise, ...
  - $L = \frac{P_\mu}{P_\mu + P_h}$ (where $h \in \{\pi, K\}$)
Conclusion

In the Belle II software library, we have implemented geant4e track propagation for particle identification (in the inner-PID detectors) and muon identification (in the KLM) during event reconstruction, either standalone or co-existing with geant4 event simulation:

✓ merged particle list including standard and custom g4e_* particles
✓ distinct physics processes for standard and custom g4e_* particles
✓ no hit invocation in sensitive volumes for geant4e
✓ distinct states and user actions during event processing
✓ Kalman fitting for muon extrapolation