SimStep and SimTrajectory

Hans Wenzel

16th December 2019

The current SimEnergyDeposit

```
// For time, it's possible for long-lived particles like neutrons
// to deposit energy after billions of ns. Chances are time cuts
// will take care of that, but let's make sure that any overlay studies
// won't suffer due to lack of precision.

int           numPhotons;   //< of scintillation photons
int           numElectrons; //< of ionization electrons
float         edep;         //< energy deposition (MeV)
geo::Point_t      startPos;     //< positions in (cm)
geo::Point_t      endPos;
double        startTime;    //< (ns)
double        endTime;      //< (ns)
int           trackID;      //< simulation track id
int           pdgCode;      //< pdg code of particle to avoid lookup by particle type later
```

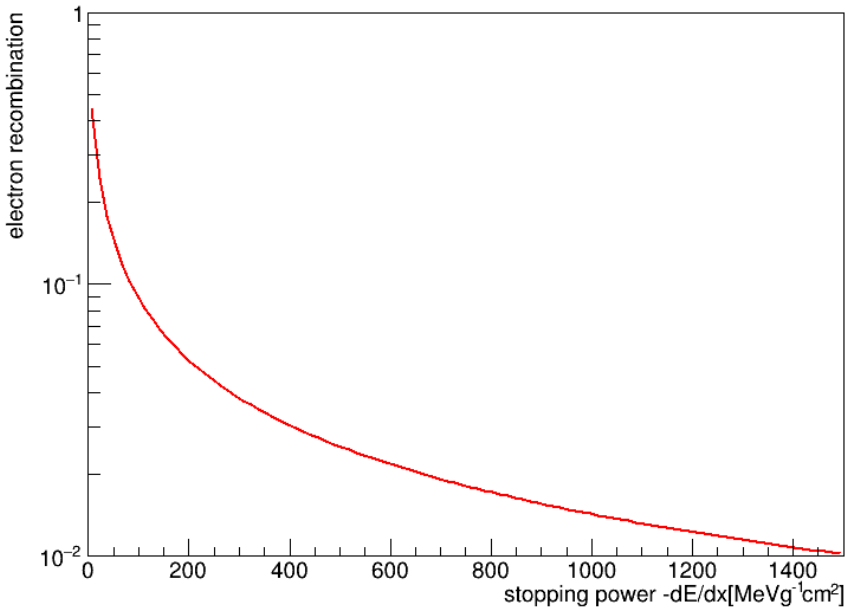
The current SimEnergy deposit:

- we are not really filling numPhotons and numElectrons in Geant4 anymore but in a separate module (ISCalculationSeparate). So this doesn't need to be stored here.
- trackID and pdgCode are repeated for every step on a trajectory
- two points for every step
- do we need double precison for time?
- do we need geo::Point_t for the persistent object (it's just x,y,z)?

Can we save space and time? Make back tracking more efficient? Be independent of external libraries?

Electron recombination from ISCalculationSeparate (Icarus)

```
  Double_t boxrecomfun(double x) {
    // Nucl.Instrum.Meth.A523:275-286,2004
    double fGeVToElectrons = 4.237e+07;
    double fRecombA        = 0.800;
    double fRecombk        = 0.0486;
    double fModBoxA        = 0.930;
    double fModBoxB        = 0.212;
    double EFieldStep      = 0.5;
    double dEdx=x;
    double Xi = fModBoxB * dEdx / EFieldStep;
    double recomb = log(fModBoxA + Xi) / Xi;
    return recomb;
  }
void recom()
{
  TCanvas *c = new TCanvas("c","electron recombination",800,600);
  //gPad->SetMaximum(1.);
  //gPad->SetMinimum(0.01);
  gPad->SetLogy();
  //gPad->SetLogx();
  TF1 *fa3 = new TF1("fa3","boxrecomfun(x)",0,1500);
  fa3->SetTitle(";stopping power -dE/dx[MeVg^{-1}cm^{2}]; electron recombination");
  fa3->SetMaximum(1.);
  fa3->SetMinimum(0.01);
  fa3->Draw();
}
```



Note: PhotonYield just proportional to dE/dx, not affected by recombination

SimTrajectory as vector of SimSteps

```
class SimStep {
private:
    float x;
    float y;
    float z;
    float len;
    float t;
    float edep;
public:
```

```
class SimStep;

class SimTrajectory {
private:
    G4int TrackID;
    std::vector<SimStep*>* trajectory;
public:
    SimTrajectory();
    SimTrajectory(G4int id);
    SimTrajectory(const SimTrajectory& orig);
    ~SimTrajectory();
    std::vector<SimStep*>* GetTrajectory();
    void SetTrackID(G4int TrackID);
    G4int GetTrackID() const;

};
```

Actually stored as:

```
std::map<int, SimTrajectory*>*t;
```

len probably unnecessary?
Add pdg Code?
Proper handling of start point (need at least 2 SimSteps for trajectory)

SimTrajectory as vector of SimSteps(cont.)

G4Trajectory exists in Geant4 but:

- Track only trajectories in the active liquid Ar (filled in Sensitive Detector)
- No need for visualization controls

Status

Implemented in lArTest (Geant4 stand alone) for testing and profiling

- new version of lArTest available in github.https://github.com/hanswenzel/lArTest
- added Root persistency
- the tracker writes out the trajectory of particles in the sensitive volume where the trajectory is a vector of SimSteps. It looks like compared to SimEnergyDeposit that we are now using in lArSoft we save about 30% in storage and CPU.
- there are example executables (readhits) to demonstrate how to access the new objects in the root file
- For 1000 muons I get might be different in lArSoft:

	SimEnergyDeposit	SimTrajectory
CPU (user)	16.183	12.742
Space	77Mb	60Mb