



Simulation Updates

Brian Rebel
August, 2011

Simulation Reorganization



- Running a job that does the Geant4 tracking and detector electronics simulation for neutrino interactions in μ BooNE can take over 4 GB of memory, which makes running on the grid impossible
- We are storing several objects that bloat our memory usage, `sim::LArVoxelData`, `sim::Electrons`, and `sim::Particles`
- The `sim::Electrons` were used to determine the signal on each channel in the detector, but we were making 1 object for every 600 ionization electrons
- The `sim::LArVoxelData` objects were not actually needed once we put the process of making `sim::Electrons` into `larg4::ParticleListAction` rather than having a separate module for them
- The number of saved `sim::Particle` objects gets out of hand for EM showers with all the secondaries, tertiaries, etc

Solution to sim::Electrons Storage



- sim::Electrons were used to keep track of which G4 track id produced which energy deposition and where
- Moved that information into sim::SimChannel, which previously just stored a collection of sim::Electrons
- sim::SimChannel now owns a collection of sim::IDE (=track id and energy info) objects
- Map TDC value for ionization electron arrival time at wire planes to vector of sim::IDE objects, one for each G4 track id
- xyz values are weighted means of ionization locations for each track id

```
class IDE{
public:

    IDE();
    virtual ~IDE();

    int   trackID;      ///< Geant4 supplied track ID
    double numElectrons; ///< total number of electrons for this track ID and time
    double x;          ///< x position of ionization
    double y;          ///< y position of ionization
    double z;          ///< z position of ionization
};
```

Solution to sim::Electrons Storage



```
class SimChannel
{
public:
    // Default constructor
    SimChannel();
    explicit SimChannel(unsigned short channel);

    // Destructor.
    virtual ~SimChannel();

    // method to add ionization electrons to this channel
    void AddIonizationElectrons(int trackID,
                                unsigned int tdc,
                                double numberElectrons,
                                double *xyz);

    unsigned short Channel() const { return fChannel; }

    // method to return a collection of IDE structs for all geant4
    // track ids represented between startTDC and endTDC
    std::vector<sim::IDE> TrackIDsAndEnergies(unsigned int startTDC,
                                              unsigned int endTDC) const;

    const std::map<unsigned short, std::vector<sim::IDE> >& TDCIDEMap() const { return fTDCIDEs; }

    // The number of ionization electrons associated with this channel for the
    // specified TDC.
    double Charge(unsigned int tdc) const;

    bool operator< (const SimChannel& other) const {return fChannel < other.Channel();}

private:
    unsigned short fChannel; ///< electronics channel associated with these sim::Electrons
    std::map< unsigned short, std::vector< sim::IDE > > fTDCIDEs; ///< vector of IDE structs for each TDC with signal
};
```

What about Voxels?



- Voxel objects still exist in LArSoft, but they are not stored in files
- Instead use `sim::SimListUtils::GetLArVoxelList()` method to get a list of voxels made by looping over the information in the `sim::SimChannels`
- If people are already using that method, the interface has not changed, so you should see no difference in your code
- If instead, people were getting an `art::Handle<std::vector<sim::LArVoxelData> >`, they need to change their code to using the method from `sim::SimListUtils`
- I have changed all instances of the latter that were checked into the repository

What about Back Tracking and MCCheater?



- I have changed the interface to the `cheat::BackTracker` to make use of the new structure of the code
- It has been fully tested and works the same as before
- All instances of using `cheat::BackTracker` in the code have been updated to use the new interfaces

sim::Particle Related Changes



- The object itself has not changed
- New flag in LArG4Parameters service to turn saving EM shower secondaries, tertiaries, etc called KeepEMShowerDaughters
- If flag is set to false, ParticleListAction does not make sim::Particles for those particles,
 - they are still tracked through the detector
 - the G4 track ID associated with their energy depositions is the Eve ID causing the EM activity
- The processes affected are: pair production, compton scattering, photoelectric effect, bremstrahlung, annihilation, ionization



How much does it Matter?

	Peak Memory Usage	Fractional Change from Baseline
Baseline	4 GB	NA
No Voxels or Electrons saved	3.3 GB	-20%
No EM daughter Particles saved either	2.2 GB	-45%

- Ran jobs to do LArG4 module and SimWire module on an already produced GENIE file for μ BooNE, 100 events simulated
- Set the seeds to be the same using the functionality of the `art::RandomNumberGenerator` service, so identical events were tracked in each case

What it Means to You



- Your jobs don't eat so much memory - better for everyone
- You have to remake any MC files that had stored `sim::Electrons` or `sim::SimChannels`
- You have to update any code that is in your test release but not in the repository