SLIC Filtering

Omar Moreno SLAC

January 18, 2022

Overview

Biasing will allow HPS to focus the available computational nodes on fully propagating events of interest

- ⇨ Originally, the idea was to build around `ldmx-sw's SimCore` and `Framework` packages that already have biasing enabled
  - Framework has changed significantly and is currently undergoing a major rewrite
  - Too volatile for HPS collaborators to help maintain going forward

Infrastructure needed for biasing was ported into slic a few years back (Thanks Jeremy!)

- ⇨ Easiest solution based on maintainability and integration into the HPS ecosystem

User Actions

Plugin framework allows for implementation of user actions (run, event, tracking etc.) that can be loaded dynamically at run time.

Enabling a call to a specific hook is done by setting the corresponding function to `true`

slic/include/UserActionPlugin.hh

```
class UserActionPlugin {

    public:

        /**
         * Class destructor.
         */
        virtual ~UserActionPlugin() {
        }

        /**
         * Get whether this plugin implements the run action.
         * @return True if the plugin implements the run action.
         */
        virtual bool hasRunAction() {
            return false;
        }

        /**
         * Get whether this plugin implements the stepping action.
         * @return True if the plugin implements the stepping action.
         */
        virtual bool hasSteppingAction() {
            return false;
        }

        /**
         * Get whether this plugin implements the tracking action.
         * @return True if the plugin implements the tracking action.
         */
        virtual bool hasTrackingAction() {
            return false;
        }

        /**
         * Get whether this plugin implements the event action.
         * @return True if the plugin implements the event action.
         */
        virtual bool hasEventAction() {
            return false;
        }

        /**
         * Get whether this plugin implements the stacking action.
         * @return True if the plugin implements the stacking action.
         */
        virtual bool hasStackingAction() {
            return false;
        }

        /**
         * Get whether this plugin implements the primary generator action.
         * @return True if the plugin implements the primary generator action.
         */
        virtual bool hasPrimaryGeneratorAction() {
            return false;
        }

        /**
         * Begin of run action.
         */
        virtual void beginRun(const G4Run*) {
        }

        /**
         * End of run action.
         */
        virtual void endRun(const G4Run*) {
        }

        /**
         * Stepping action.
         */
        virtual void stepping(const G4Step*) {
        }

        /**
         * Pre-tracking action.
         */
        virtual void preTracking(const G4Track*) {
        }

        /**
         * Post-tracking action.
         */
        virtual void postTracking(const G4Track*) {
        }

        /**
         * Begin of event action.
         */
        virtual void beginEvent(const G4Event*) {
        }

        /**
         * End of event action.
         */
        virtual void endEvent(const G4Event*) {
        }
```

User code is placed within the desired user action function

Pair Conv Filter

slic/plugins/*/PairConvFilter.*

```
class PairConvFilter : public UserActionPlugin {

public:
  /// Default constructor
  PairConvFilter() = default;

  /// Destructor
  ~PairConvFilter() = default;

  /// @return The name of this plugin
  std::string getName() { return "PairConvFilter"; }

  /// @return True if this plugin has a stepping action
  bool hasSteppingAction() final override { return true; }

  /// @return True if this plugin has an event action
  bool hasEventAction() final override { return true; }

  /// @return True if this plugin has a stacking action
  bool hasStackingAction() final override { return true; }

  /// @return True if this plugin has a run action
  bool hasRunAction() final override { return true; }

  /**
   * Implementmthe stepping action which performs the filtering.
   *
   * @param[in] step Geant4 step
   */
  void stepping(const G4Step *step) final override;

  /**
   * Method called at the end of the event.
   *
   * @param event Geant4 event object.
   */
  void endEvent(const G4Event *) final override;

  /**
   * Method called at the end of the run.  This will be used to print out
   * a summary of the total conversion statistics.
   *
   * @param run Geant4 run object.
   */
  void endRun(const G4Run* run) final override;
```

```
G4ClassificationOfNewTrack PairConvFilter::stackingClassifyNewTrack(
    const G4Track *track, const G4ClassificationOfNewTrack &currentTrackClass) {

  if (track == currentTrack_) {
    currentTrack_ = nullptr;
    //std::cout << "[ PairConvFilter ]: Pushing track to waiting stack."
    //          << std::endl;
    return fWaiting;
  }

  // Use current classification by default so values from other plugins are
  // not overridden.
  return currentTrackClass;
}

void PairConvFilter::stepping(const G4Step *step) {

  if (hasPairConv_) return;

  // Get the track associated with this step.
  auto track{step->GetTrack()};

  // Get the particle type.
  auto particleName{track->GetParticleDefinition()->GetParticleName()};

  // Get the kinetic energy of the particle.
  auto incidentParticleEnergy{step->GetPostStepPoint()->GetTotalEnergy()};

  auto pdgID{track->GetParticleDefinition()->GetPDGEncoding()};

  // Get the volume the particle is in.
  auto volume{track->GetVolume()};
  auto volumeName{volume->GetName()};

  // Get the PDG ID of the track and make sure it's a photon. If another
  // particle type is found, push it to the waiting stack until the photon has
  // been processed.
  if (pdgID != 22) {
    currentTrack_ = track;
    track->SetTrackStatus(fSuspend);
    return;
  }

  // Only conversions that happen in the target, and layers 1-3
  // of the tracker are of interest.  If the photon has propagated past
  // the second layer and didn't convert, kill the event.
  // TODO: OM: This really should be done with regions.
  if (volumeName.find("module_L4") != std::string::npos) {
    //std::cout << "[ PairConvFilter ]: Photon is beyond the sensitive"
    //          << " detectors of interest. Killing event." << std::endl;
    track->SetTrackStatus(fKillTrackAndSecondaries);
    G4RunManager::GetRunManager()->AbortEvent();
    return;
```

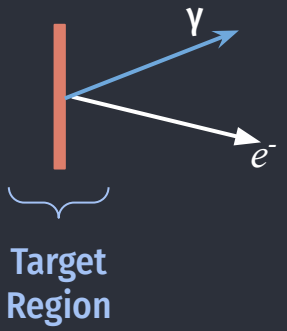
Loading the plugin

Use the load command followed by the plugin name

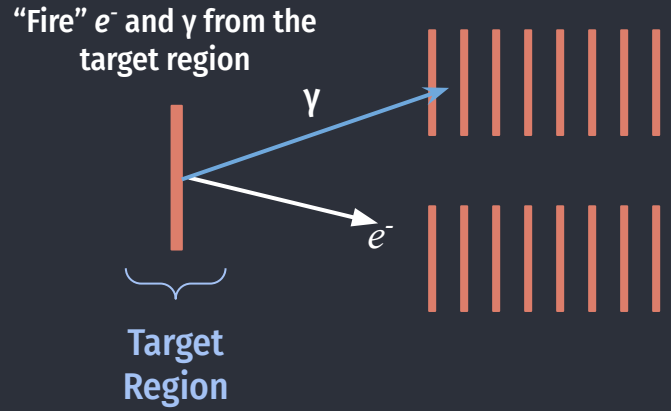
```
# Set the LCDD geometry file URL
/lcdd/url <detector_name>.lcdd

/run/initialize

# Set the parth to the file containing the events to filter
/generator/filename <input_file>.stdhep

# Load the pair conv plugin
/slic/plugins/load PairConvFilter

# Set the output file name
/lcio/filename <output_file>.slcio
/lcio/fileExists delete

# Specify the number of events to process.
/run/beamOn 1000
~
```

Example Biasing Scheme

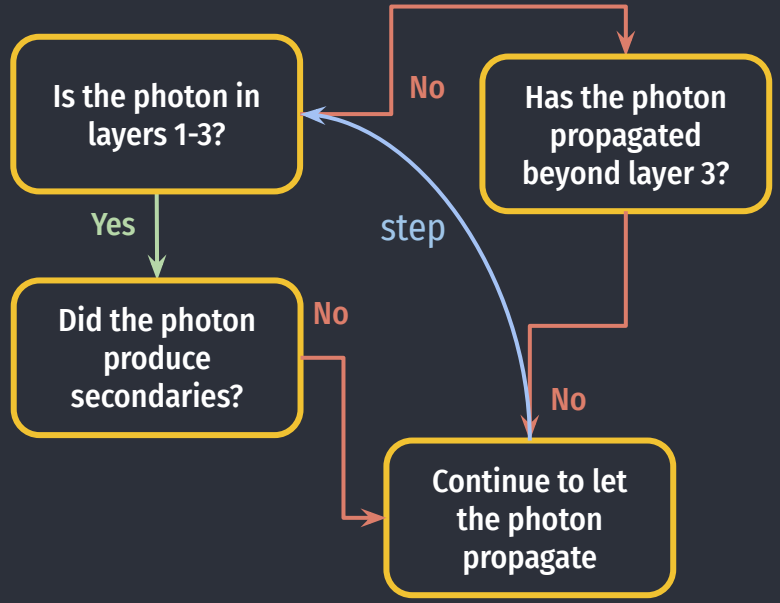
"Fire" e^- and γ from the target region



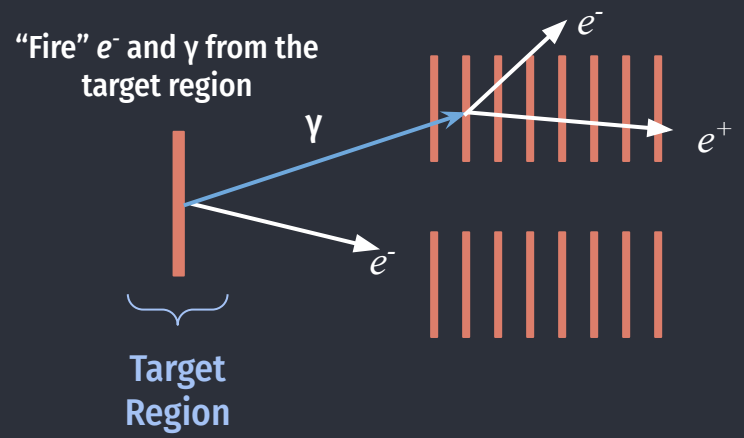
Example Biasing Scheme



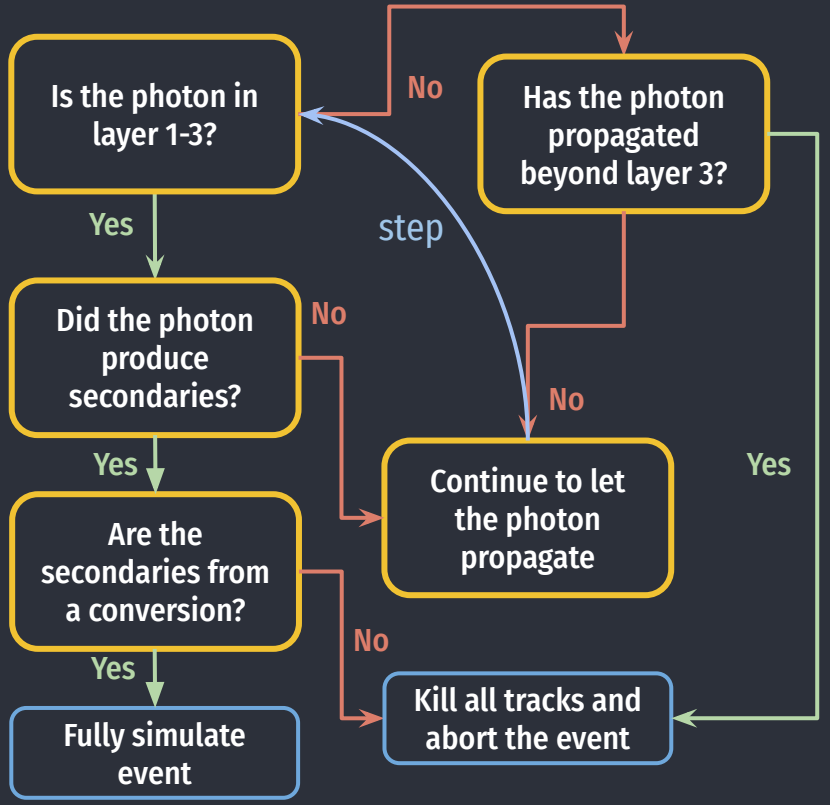
PairConvFilter in Biasing Module



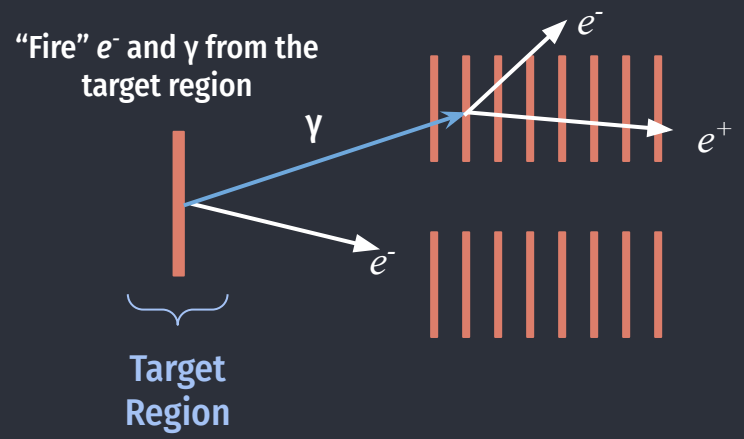
Example Biasing Scheme



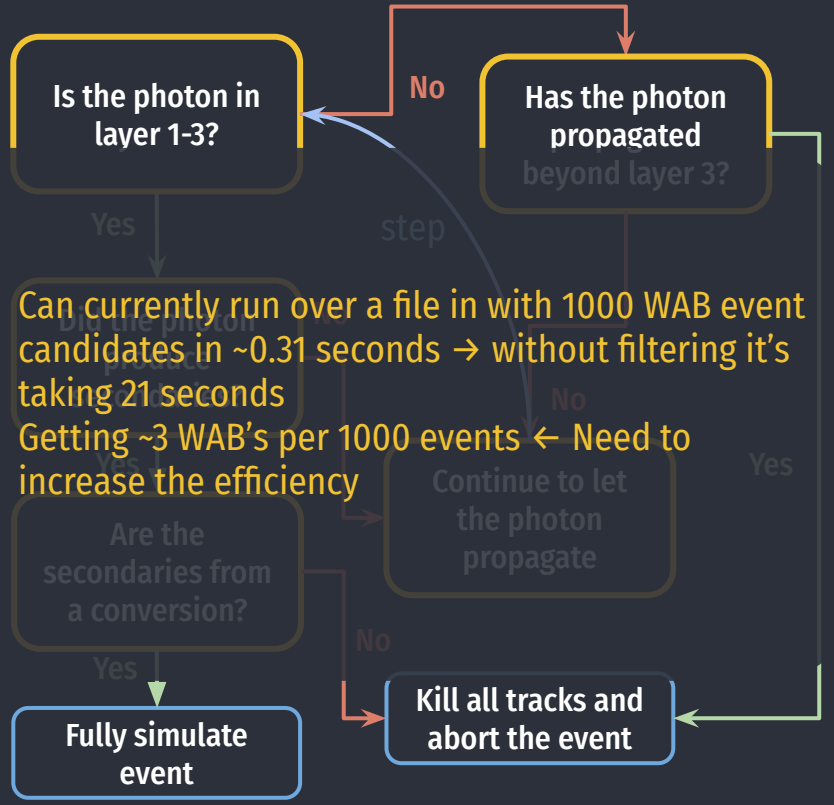
PairConvFilter in Biasing Module



Example Biasing Scheme



PairConvFilter in Biasing Module



⇨ Can currently run over a file in with 1000 WAB event candidates in ~0.31 seconds → without filtering it's taking 21 seconds

⇨ Getting ~3 WAB's per 1000 events ← Need to increase the efficiency

Next Steps

Update hps-mc to make use of the filter when generating WABs

Enable Geant4 biasing framework that will allow for increasing the pair-conversion cross-section within the layers of interest