Seeding-to-Fitting with Detray geometry

Beomki Yeo

PR Summary

- Add CKF examples with detray toy detector (#411)
  - Add **truth_track_finding** (CKF + KF) example for CPU and CUDA
- Seeding with detray - Version1 (#421)
  - Add detray geometry in **seeding_example**. Seeding doesn't need a geometry though
- Seeding and Finding with detray geometry(#430)
  - **seeding_examples** includes CKF and KF

Changes in Spacepoint reader

- In the current spacepoint reader, the measurement is calculated by applying global-to-local transformation to spacepoint
- Wrong approach in case the spacepoint is read from truth hit csv file which doesn't include measurement error
  - **point_to_local** or **point_to_global** also should not be used for non-cartesian measurements
- So I changed to read the measurement csv file directly

```
-         // Find the local<->global transformation for the spacepoint's detector
-         // module.
-         const transform3& placement = geom[iohit.geometry_id];
-
          // Construct the global 3D position of the spacepoint.
          const point3 pos{iohit.tx, iohit.ty, iohit.tz};

          // Construct the local 3D(2D) position of the measurement.
-         const point3 lpos = placement.point_to_local(pos);
+         // const point3 lpos = placement.point_to_local(pos);
+         alt_measurement meas;
+         for (auto const [meas_id, hit_id] : meas_hit_ids) {
+             if (hit_id == result_spacepoints.size()) {
+                 meas = meas_reader_out.measurements[meas_id];
+             }
+         }

          // Create the spacepoint object (with its member measurement) from all
          // this information.
-         const traccc::spacepoint sp{
-             pos, {point2{lpos[0], lpos[1]}, variance2{0., 0.}, link}};
+         const traccc::spacepoint sp{pos, meas};

          result_spacepoints.push_back(sp);
      }
```

Impact on Track Parameter Estimation

- After modifying the spacepoint reader, the local position of ACTS and traccc track parameter estimation is not the same
  - Traccc: local position from measurement
  - ACTS core: local position from global-to-local transformation

- Thus, I removed the track parameter estimation part of **compare_with_acts_seeding** test

Traccc

```
// The measured loc0 and loc1
const auto& meas_for_spB = spB.meas;
getter::element(params, e_bound_loc0, 0) = meas_for_spB.local[0];
getter::element(params, e_bound_loc1, 0) = meas_for_spB.local[1];
```

ACTS Core

```
// Transform the bottom space point to local coordinates of the provided
// surface
auto lpResult = surface.globalToLocal(gctx, spGlobalPositions[0], direction);
if (not lpResult.ok()) {
  ACTS_ERROR(
      "Global to local transformation did not succeed. Please make sure the "
      "bottom space point lies on the provided surface.");
  return std::nullopt;
}
Vector2 bottomLocalPos = lpResult.value();
// The estimated loc0 and loc1
params[eBoundLoc0] = bottomLocalPos.x();
params[eBoundLoc1] = bottomLocalPos.y();
```

Changes in Track Parameter Estimation

- The output of track parameter estimation is a bound track parameter which contains:
  - Surface ID
  - Bound vector
  - Bound covariance
- I had to fix the track parameter estimation to set the surface ID and bound covariance

```
45  -   output_type operator()(const spacepoint_collection_types::host& spacepoints,
46  -                           const seed_collection_types::host& seeds,
47  -                           const vector3& bfield) const override;
48  +   output_type operator()(
49  +       const spacepoint_collection_types::host& spacepoints,
50  +       const seed_collection_types::host& seeds,
51  +       const cell_module_collection_types::host& modules,
52  +       const vector3& bfield,
53  +       const std::array<traccc::scalar, traccc::e_bound_size>& stddev = {
54  +           0.03 * detray::unit<traccc::scalar>::mm,
55  +           0.03 * detray::unit<traccc::scalar>::mm, 0.017, 0.017,
56  +           0.001 / detray::unit<traccc::scalar>::GeV,
57  +           1 * detray::unit<traccc::scalar>::ns}) const override;
```

How to Run the Examples

- Generate simulation data

```
./bin/traccc_simulate_toy_detector --gen-vertex-xyz-mm=0:0:0 --gen-vertex-xyz-std-mm=0:0:0 --gen-mom-gev=100:100
--gen-phi-degree=0:360 --events=10 --gen-nparticles=2000 --output_directory=detray_simulation/toy_detector/n_particles_2000/
--gen-eta=-3:3 --constraint-step-size-mm=1
```

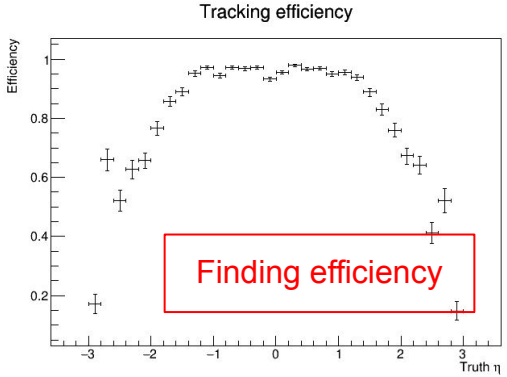
- Run Truth Finding (CKF + KF)
  - I just found that I have not generalized it with detray json input :p …

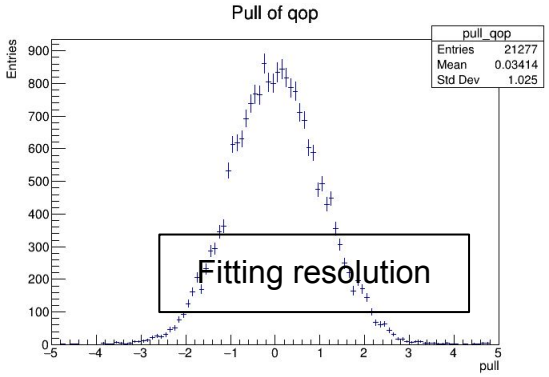
```
./bin/traccc_truth_finding_example_cuda --input_directory=detray_simulation/toy_detector/n_particles_2000/ --events=10
--track_candidates_range=3:10 --run_cpu=1 --check_performance=true --constraint-step-size-mm=5
```

- Run Seeding (Seeding + CKF + KF)

```
./bin/traccc_seeding_example_cuda --input_directory=detray_simulation/toy_detector/n_particles_2000/
--run_detray_geometry=true --check_performance=true --detector_file=detray_json/toy_detector_geometry.json --event=1
--track_candidates_range=3:10 --constraint-step-size-mm=5 --run_cpu=1
```

Truth Track Finding Example (CKF + KF)

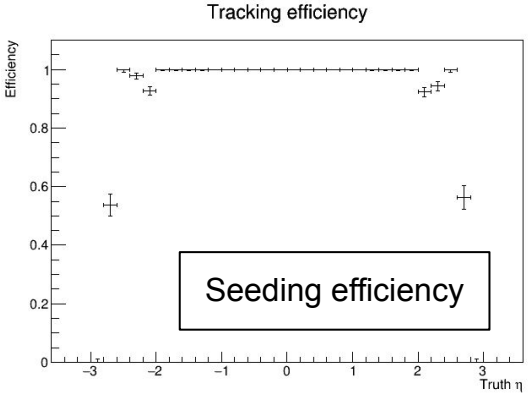


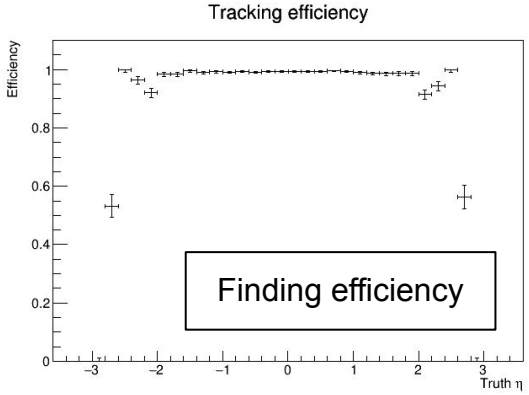



```
===>>> Event 0 <<<===
Track candidate matching Rate: 1
===>>> Event 1 <<<===
Track candidate matching Rate: 1
===>>> Event 2 <<<===
Track candidate matching Rate: 1
===>>> Event 3 <<<===
Track candidate matching Rate: 1
===>>> Event 4 <<<===
Track candidate matching Rate: 1
===>>> Event 5 <<<===
Track candidate matching Rate: 1
===>>> Event 6 <<<===
Track candidate matching Rate: 0.999502
===>>> Event 7 <<<===
Track candidate matching Rate: 1
===>>> Event 8 <<<===
Track candidate matching Rate: 1
===>>> Event 9 <<<===
Track candidate matching Rate: 1
==> Statistics ...
- created (cuda) 21277 found tracks
- created (cuda) 21277 fitted tracks
- created  (cpu) 21272 found tracks
- created  (cpu) 21272 fitted tracks
==>Elapsed times...
        Track finding  (cuda)  592 ms
        Track fitting  (cuda)  279 ms
         Track finding  (cpu)  2126 ms
         Track fitting  (cpu)  989 ms
```

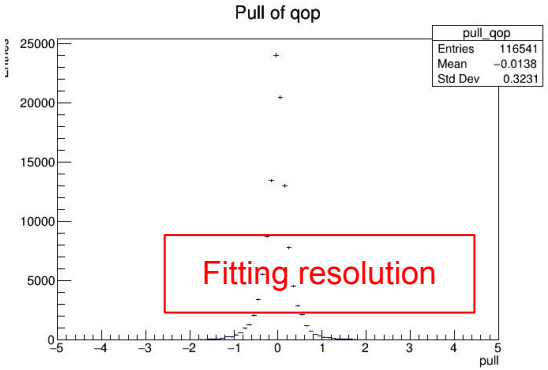
Tracking efficiency of finding is too low, especially at the forward regions

Seeding Example (Seeding + CKF + KF)





Pull distributions from KF are not great right now




```
===>>> Event 0 <<<===
Number of seeds: 11286 (host), 11285 (device)
  Matching rate(s):
    - 6.47705% at 0.01% uncertainty
    - 26.7322% at 0.1% uncertainty
    - 76.4487% at 1% uncertainty
    - 93.4698% at 5% uncertainty
Number of track parameters: 11286 (host), 11285 (device)
  Matching rate(s):
    - 53.6062% at 0.01% uncertainty
    - 84.8751% at 0.1% uncertainty
    - 98.8127% at 1% uncertainty
    - 99.8848% at 5% uncertainty
Track candidate matching Rate: 0.998918
```

```
==> Statistics ...
- read    106623 spacepoints from 26365 modules
- created  (cpu)  113708 seeds
- created (cuda)  113701 seeds
- created  (cpu) 116576 found tracks
- created (cuda) 116545 found tracks
- created  (cpu) 116576 fitted tracks
- created (cuda) 116545 fitted tracks
==>Elapsed times...
            Hit reading  (cpu)  2860 ms
                Seeding (cuda)  14 ms
                Seeding  (cpu)  592 ms
            Track params (cuda)  2 ms
            Track params  (cpu)  23 ms
 Track finding with CKF (cuda)  1062 ms
  Track finding with CKF (cpu)  9710 ms
  Track fitting with KF (cuda)  542 ms
   Track fitting with KF (cpu)  4862 ms
                      Wall time  19688 ms
```

Summary

- Integrated seeding, track finding and track fitting
- There are still some performance issues under investigation
- Integrating with clusterization still needs some works
  - Need to pass a digitization map for detray geometry
  - Spacepoint formation also requires the detray geometry for local to global transformation