Storing reconstructed 3D points with charge

Gianluca Petrillo

Fermi National Accelerator Laboratory

LArSoft architecture meeting, September 6th, 2017

Outline

Charge in space

We need a data product to save a charge together with its absolute position in detector.

Candidate users:

- WireCell (BNL)
- `SpacePointSolver` (Christopher Backhouse)
- detectors with pixel-based readout

These slides summarise the current status and advance a proposal.

Note that this has nothing to do with the space charge correction pioneered by Mike Mooney.

The available products

We have two data products related to position in the detector:

`recob::SpacePoint` effectively representing a reconstructed 3D position

- an array of coordinates
- an error matrix (triangular)
- a χ^2
- an ID

`recob::Vertex` effectively representing a 3D position

- an array of coordinates
- an ID

My feeling is that the roles are inverted...

The more proper candidate *by name* is `recob::SpacePoint`.

Where are `recob::SpacePoint` used

I have found 8 algorithms using `recob::SpacePoint` errors:

- 1 module uses them fully (`Track3DKalmanSPS`)
- a few other algorithms just copy or print or plot them (and some misinterpret it, too)

Out of 25 modules claiming to produce `recob::SpacePoint`:

- 6 produce the full fit information (5 use `SpacePointAlg`, 1 uses `SpacePointAlg_TimeSort`)
- 2 put sort-of-meaningful, fixed values for
- 1 puts quality-related information as χ^2 (not a real χ^2)
- 1 puts full σ^2_{ij} and dummy χ^2
- 2 just copy from existing `recob::SpacePoint`
- 1 chickens out and does not create any
- 12 (the rest) set dummy values for both σ^2_{ij} and χ^2

A proposal from LArSoft

1. discorporate the errors from `recob::SpacePoint`
2. have a stand-alone data product for charge
3. link them by *implicit association*

\+ `recob::SpacePoint` becomes a very lightweight object
(which is good, because there may be plenty)

\+ modules won't need to fill dummy values for fit-related quantities

\+ algorithms using `recob::SpacePoint` will work with the output of the "charged" modules (e.g. WireCell and `SpacePointSolver`)

\+ compared to adding a data member, modules won't need to fill a dummy value for charge

– need to juggle with two data products when charge is needed
(solvable with a "proxy")

Proposal details

- charge saved in its own class `recob::PointCharge`, single precision
- `recob::SpacePoint` will retain only the position information
  - position stored with the same 3D point type as `recob::Track`
  - should the ID stay?
  - interface for position is preserved and expanded
- `recob::SpacePointFit` will hold the fit information
  - error matrix as ROOT SMatrix double precision symmetric 3×3 matrix
  - χ^2 (still double precision)
  - expanded interface (with respect to old `recob::SpacePoint`)

Proposal: `recob::SpacePoint`

```
1    struct SpacePoint {
2      using ID_t = int;
3      using Point_t = tracking::Point_t;
4
5      SpacePoint(double x, double y, double z);
6
7      Point_t const& Position() const;
8      double X() const;
9      double Y() const;
10     double Z() const;
11     ID_t ID() const;
12   };
```

Position is defined in the “global” reference frame, stored in centrimetres.

Proposal: `recob::PointCharge`

```
1    struct PointCharge {
2      PointCharge(float charge);
3      float Charge() const;
4    };
```

How do we define the charge?

The association with `recob::SpacePoint` will be *by protocol* only, via what I call parallel data product requirement: the i^{th} point charge is associated to the i^{th} space point.

Proposal: `recob::SpacePointFit`

```
1    struct SpacePointFit {
2      using Covariance3D_t
3        = ROOT::Math::SMatrix<double,3U,3U,ROOT::Math::MatRepSym<double,3U>>;
4
5      double Chisq() const;
6      double VarX() const;
7      double VarY() const;
8      double VarZ() const;
9      double CovXY() const;
10     double CovYZ() const;
11     double CovZX() const;
12     double Cov(std::size_t i, std::size_t j) const;
13     Covariance3D_t const& Covariance() const;
14   };
```

The collection of `recob::SpacePointFit` will also fulfill the parallel data product requirement.

Proposal: proxy and `proxy::SpacePointWithCharge`

We may provide a “proxy” collection with the same (unpublished) infrastructure used for tracking. This is how it would look like:

```
 1    mf::LogInfo log("DumpSpacePoints");
 2
 3    proxy::SpacePointsWithCharge sps(event, spacePointTag);
 4    log << sps.size() << " space points:";
 5    for (auto&& point: sps) {
 6      log << "[" << point.index() << "] ID=" << point.point().ID()
 7        << " at " << lar::dump::vector3D(point.position())
 8        << " with charge " << point.charge();
 9    }
10    if (!sps.empty())
11      log << "\nMiddle point: ID=" << sps[sps.size() / 2].point().ID();
```

- the collection is iterable and random-accessible
- each element
  - remembers the original index, the space point and the charge
  - has a few methods for direct information accessible
  - is temporary (r-value) containing pointers and an index

Proposal from Brett Viren

From private communication, my interpretation of Brett's proposal:

```
1    struct SpacePoint {
2      using ID_t = int;
3      using Point_t = tracking::Point_t;
4
5      SpacePoint(double x, double y, double z, double value = 0.0);
6
7      Point_t const& Position() const;
8      double X() const;
9      double Y() const;
10     double Z() const;
11     float value() const;
12     ID_t ID() const;
13   };
```

- `recob::SpacePoint` and `recob::PointCharge` information all together
- dummy value if charge is not produced
- "value" instead of "charge" because somebody might store information that is not exactly charge

Onward...

- we need a way to store charge together with a position
- a single proposal has been drafted here
- ... now it's time for discussion and counterproposals