LArSoft architecture review: status report

Extended geometry interface

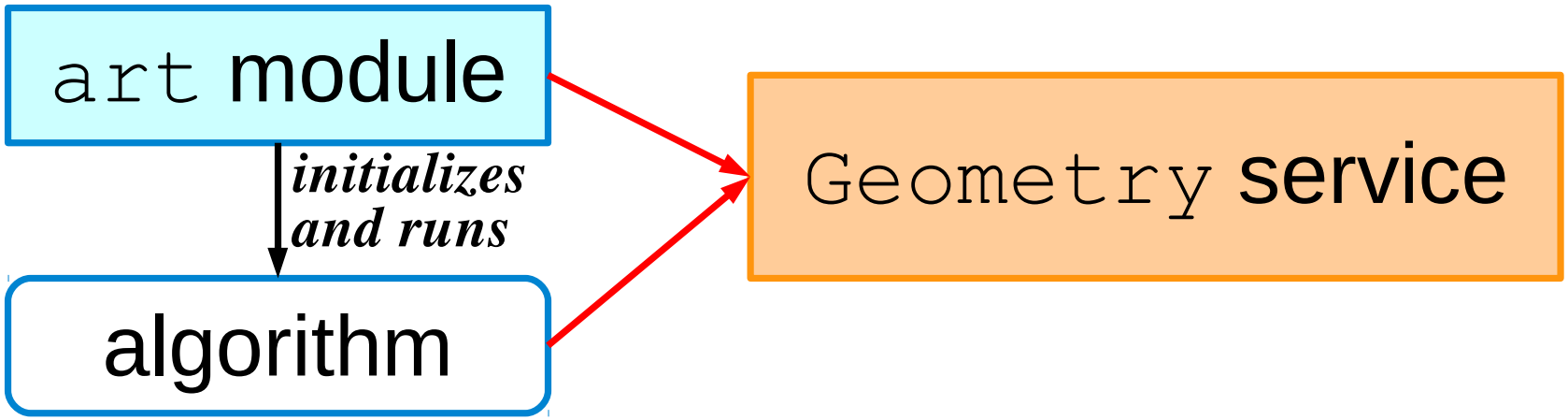
Gianluca Petrillo, Saba Sehrish, Erica Snider

University of Rochester/Fermilab

LArSoft Coordinators Meeting, May 19th , 2015

Service factorization — a reminder

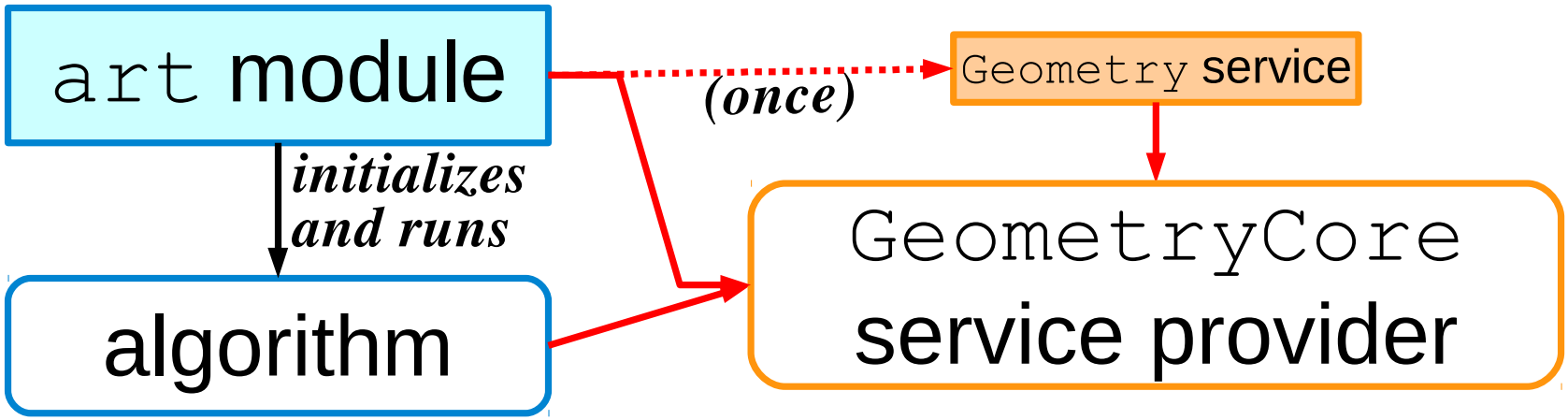
Factorization was discussed last week. In short, the old model:



- the geometry *art service* (Geometry) serves the geometry to the modules, that can use it and pass it to their algorithms, *and* provides the functionality

Service factorization — a reminder

Factorization was discussed last week. In short, the new model:



- the geometry *art service* (Geometry) serves the geometry to the modules, that can use it and pass it to their algorithms
- the geometry *service provider* (GeometryCore) provides the functionality and is framework-independent: algorithms can use it directly

Up to this point, user interface is *not* changed, and you can replace geo::Geometry with geo::GeometryCore in your algorithms.

Some `Geometry[Core]` interface issues

- a lot of code relies only on a single index (e.g. plane), making the code fragile when moving to a detector with multiple TPCs
- `Geometry` interface usually accepts only indices
  - very little support for using the IDs (e.g. `geom->TPC(tpcid)`)
  - the order of the indices is not exactly intuitive: e.g.
    `geom->PlaneWireToChannel(p, w, t, c)`
  - the TPC and cryostat specifications are often optional (and user code routinely omits it)
- converting IDs, e.g. from a `PlaneID` to a `TPCID`, is tedious (and slightly inefficient, since it copies the information)
- processing all TPCs involves two nested loops; all planes, three nested loops...

Geometry ID objects

- TPC detector elements are characterized by a set of indices
- ID objects are provided that carry around all of them; for example, `geo::WireID` contains a cryostat, a TPC, a plane and a wire index
- there are three *independent* IDs so far: `geo::TPCID`, `geo::PlaneID` and `geo::WireID`

Note: ID validity

IDs have also a `isValid` field:

- it only expresses that the ID structure is fully initialized
- does not necessarily convey information about whether a geometry element with this ID actually *exists*

I consider it unreliable, except when explicitly documented otherwise.

New geometry IDs

The new `Geometry` interface focuses on the IDs:

- each object has its ID: `geo::CryostatID`, `geo::TPCID`, `geo::PlaneID` and `geo::WireID`
- an more specific ID can be used as a less specific one, so that a `PlaneID` can directly express also the `TPC` the plane belongs to
- ID construction explicitly requires all the indices: no assumptions on the TPC
- an ID can be constructed adding an index to a less specific ID, e.g. a `PlaneID` can be constructed from a `TPCID` plus a plane index

Although the code is designed to allow different types, currently all the indices are `unsigned int` (tell DoE your software supports 2^{32} TPCs, and so they should).

Geometry interface extension

The goal

Users should not need single indices except in very specific cases.

The geometry interface (`geo::GeometryCore`, `geo::CryostatGeo`, `geo::TPCGeo` and `geo::PlaneGeo`) has been expanded to support IDs wherever is possible. For example:

- `geom->Plane(planeid)` is equivalent to `geom->Plane(p, t, c)`
- `geom->NearestWire(worldLoc, planeid)` is equivalent to `geom->NearestWire(worldLoc, p, t, c)`
- `geom->Plane(wireid)` fetches the plane the specified wire belongs to

To encourage the use of IDs, the legacy, index-based methods in the geometry now internally construct an ID and invoke the new interface.

New geometry iterators

Iterators go through the ID of *all the elements* in the geometry:

```
for (geo::PlaneID const& planeid: geom->IteratePlaneIDs()) {
  LOG_DEBUG("MyTest") << "Processing plane " << planeid;
  geo::PlaneGeo const& plane = geom->Plane(planeid); // get the plane
  geo::TPCGeo const& TPC = geom->TPC(planeid); // get its TPC
  // ...
} // for
```

Listing 1: range-for loop through all the plane IDs

or, more traditionally,

```
geo::GeometryCore::tpc_id_iterator iTPC, end_tpc = geom->end_TPCID();
for (iTPCID = geom->begin_TPCID(); iTPCID != end_tpc; ++iTPCID) {
  LOG_DEBUG("MyTest") << "Processing TPC " << *tpcid;
  geo::TPCGeo const& TPC = iTPCID.get(); // get the TPC
  // ...
} // for
```

Listing 2: iterator-based loop through all the TPC IDs

I might implement iterators directly through geometry objects (not IDs).

Any suggestion? how would you like to use the iterators?

Solution to `Geometry[Core]` interface issues

These changes are designed to help with the issues I described:

- `Geometry` interface now accepts IDs everywhere
  (and if you use the old interface you pay a small overhead)
- the construction of these IDs follows an intuitive top-down order (cryostat, TPC, plane, wire)
- all fields in the construction are mandatory
- IDs are "upcasted" transparently (e.g. from a `PlaneID` to a `TPCID`)... and with no overhead
- iterators allow flattening iterations on all TPCs, planes etc. into a single loop
- existing code relying a single index (e.g. plane) is still valid; but writing better code should now be easier

My hope is to deprecate and, in some non-near future, remove the single-index interface, so that the compiler itself will tell us where the problems arise.

Unit testing

For the algorithms who need geometry access, `art`-independent unit testing requires additional, experiment-specific setup. I have provided structures to make this setup easier, in:

generic	`larcore/test/Geometry/geometry_unit_test_base.h`
MicroBooNE	`uboonecode/test/Geometry/geometry_unit_test_uboone.h`
DUNE 35t	`lbnecode/test/Geometry/geometry_unit_test_dune.h`
DUNE 10kt	`lbnecode/test/Geometry/geometry_unit_test_dune.h`
SBND	`lar1ndcode/test/Geo/geometry_unit_test_sbnd.h`
ArgoNeuT	`argoneutcode/test/Geometry/geometry_unit_test_argoneut.h`
LArIAT (to do)	`lariatsoft/test/Geometry/geometry_unit_test_lariat.h`

Examples of the unit tests are usually found in the same directory, as `geometry_iterator_loop_*_test.cxx`, `geometry_iterator_*_test.cxx` and `geometry_*_test.cxx`.

Each such test hard-codes the use of a specific geometry!

Unit testing: example

```
#include "test/Geometry/geometry_unit_test_dune.h"
#include "test/Geometry/GeometryTestAlg.h"
#include "lbne/Geometry/ChannelMap35OptAlg.h"

using DUNE35tGeometryConfiguration
  = lbne::testing::DUNE35tGeometryFixtureConfigurer
    <geo::ChannelMap35OptAlg>;

int main() {
  // object describing geometry configuration (geometry_unit_test_dune.h)
  DUNE35tGeometryConfiguration config("geometry_test_DUNE35t");

  // object providing the test environment (geometry_unit_test_base.h)
  testing::GeometryTesterFixture<DUNE35tGeometryConfiguration>
    TestEnvironment(config);

  // construct and initialize the algorithm ("geotest" parameter set)
  geo::GeometryTestAlg Tester(TestEnvironment.TesterConfiguration());
  Tester.Setup(*(TestEnvironment.Geometry()));

  return Tester.Run(); // run the test, return the number of errors
} // main()
```

Listing 3: Excerpts from `geometry_dune35t_test.cxx`

Status of the code: done

New code is in `feature/gp_FactorizeGeometryService`:

larcore	the new geometry
lardata	updated `classes_def.xml` for new geometry IDs
larreco	updated iterator names (only case in LArSoft)
uboonecode	added unit tests (plus unspeakable CLHEP workaround)
lbnecode	new geometry interface, added unit tests
argoneutcode	added unit tests (currently failing, never tested before?)
lar1ndcode	updated from LArSoft 3.8.2, new geometry interface, added unit tests (currently failing, probably because of incomplete channel mapping implementation)
lariatsoft	in progress

- branches have not been published yet
- updated to LArSoft v4.8.0
- unit tests succeed except as noted above, plus unrelated failures in `uboonecode` and `lbnecode`

Status of the code: not done

Still to do:

- "physics" test: so far, I have run only the standard test suite
- convert the Geometry test to use the new interface, as example

Not included:

- 3D points are accepted in at least four ways through Geometry interface: C array, C pointer, three coordinates, `TVector3`, `std::vector<double>`... needs standardization
- compacting ID structure (now the largest spans 20 bytes)
  - replace `isValid` flag by a method checking the ID
- ID topology, e.g. "get the wire before this `wireid`"
- add a ID to each of the geometry objects (currently, a `TPCGeo` does not know its TPC ID)

Summary

- `Geometry` service has been factorized
- user code almost completely updated (LArIAT is the last one)

⇒ this is *not breaking change* EXCEPT: if you have used the geometry iterators... well, they are being renamed

- documentation is available in Doxygen format
  (I have to ask Brian Rebel if his configuration supports markup)

I would like to have this merges as soon as possible!

Please let me know which additional tests you want to be passed before this can happen.

Other architecture topics

Modules candidate for architecture work at June 3rd workshop:

`OpFlashFinder`	`Calorimetry`
`GausHitFinder`	`CosmicPFParticleTagger`
`RawHitFinder`	`Track3DKalmanHit`
`DisambigCheater`	`TrackStitcher`
`fuzzyCluster`	`ShowerReco3D`
`CosmicTracker`	`AnalysisTree`

We need to pick a handful (≈ 5) of them. Which ones do you like most?
(also: register the workshop! There will be snacks, too. Just saying.)

LArSoft Architecture meetings

We will resurrect the architecture meetings for discussion of the details of the ongoing work.
I will still report to the Coordinators, but more tersely.
If I can.

Backup

Code bits

How to check if the geometry ID represents an existing detector element:

```
// check if the TPC of the plane ID exists
if (!geom->HasTPC(planeid)) throw /* ... */;
// check if the TPC pointed by the TPC ID exists
if (!geom->HasElement(tpcid)) throw /* ... */;
```

Listing 4: Checking the "validity" of a geometry ID