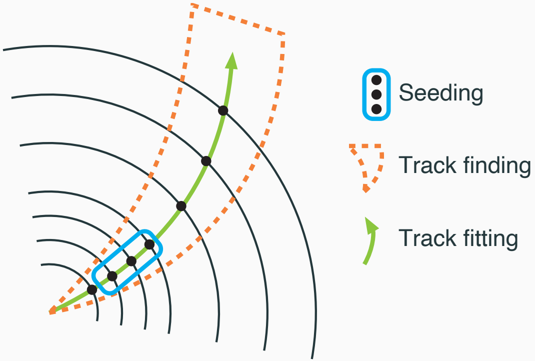
ACTS Status Update

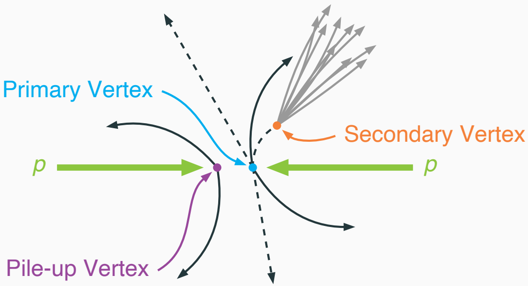
Paul Gessinger

CERN

2021-12-15 - EP R&D Software Working Group Meeting

Track reconstructionin a nutshell





What is acts?

What is ACTS?

Experiment-independent toolkit for track reconstruction applications

What is ACTS?

Modern architecture and code, unit tested, continuous integration

Experiment-independent toolkit for track reconstruction applications

What is ACTS?

Minimal external dependencies, easy to build

Modern architecture and code, unit tested, continuous integration

Experiment-independent toolkit for track reconstruction applications

What is ACTS?

- Robust concurrency through thread-safety by design
- Minimal external dependencies, easy to build
- Modern architecture and code, unit tested, continuous integration
- Experiment-independent toolkit for track reconstruction applications

What is ACTS?

Community platform for R&D across various experiment

Robust concurrency through thread-safety by design

Minimal external dependencies, easy to build

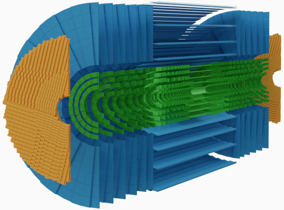
Modern architecture and code, unit tested, continuous integration

Experiment-independent toolkit for track reconstruction applications

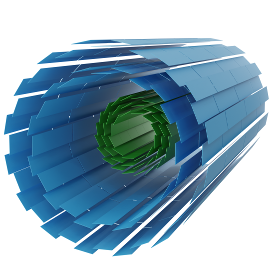
Goals:

- Provide established algorithms in a modern package
- Provide testbed for R&D activities including new algorithms, machine learning, heterogeneous computing

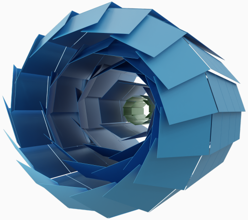
Evaluation and/or deployment by multiple experiments



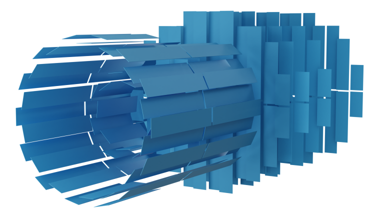
ATLAS Inner Tracker



sPHENIX

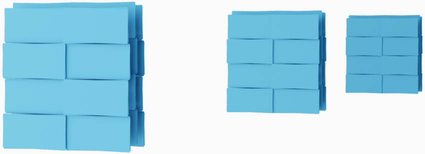


Belle II



Panda

+ ALICE,
EIC



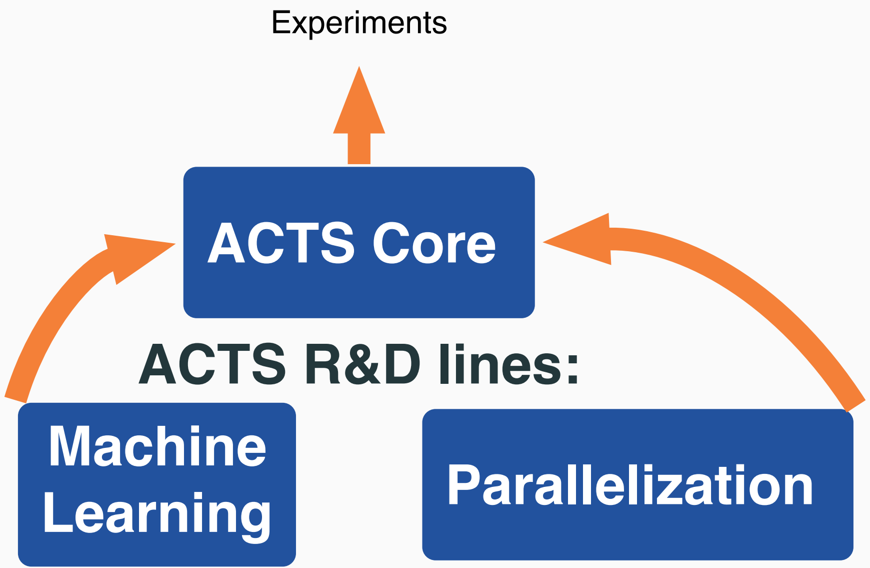
FASER

ACTS Core

ACTS R&D lines:

Machine Learning

Parallelization


Experiments
ACTS Core
ACTS R&D lines:
Machine Learning
Parallelization

ACTS Core

Python bindings for the examples

ACTS examples

- ACTS ships with a set of **examples** to show assembly of a track reconstruction chain
- Ships with a minimal event processing framework: not intended for production

- **Currently:** large number of executables for different purposes: controllable via command line arguments
- Drawback: large number of options for everything, expose almost all configuration via CLI arguments
- **Recent development:** at python bindings to example classes: allows writing simply python scripts to run example payloads
  - **Advantage:** can follow configuration flow, understand what is actually happening
- Provide shorter special use-case scripts + one *full-chain* script

Python script example

```
detector, trackingGeometry, decorators = acts.examples.GenericDetector.create()
field = acts.ConstantBField(acts.Vector3(0, 0, 2 * u.T))
rnd = acts.examples.RandomNumbers()

s = acts.examples.Sequencer(
    events=100, numThreads=-1, logLevel=acts.logging.INFO
)

s.addReader(someParticleInput) # e.g. particle gun, pythia8 ...

selector = acts.examples.ParticleSelector(level=acts.logging.INFO,
  inputParticles=inputParticles, outputParticles="particles_selected")
s.addAlgorithm(selector)

alg = acts.examples.FatrasSimulation(
    level=acts.logging.INFO, randomNumbers=rnd, trackingGeometry=trackingGeometry,
    magneticField=field, generateHitsOnSensitive=True, # + input/output collections
)
s.addAlgorithm(alg)

s.addWriter(someWriter) # e.g. CSV, ROOT, ...
```

Reproducibility tests at the python level

- *Old* C++ example executables: largely **untested**
- Used opportunity to add tests for all examples implemented in python
- Cover use cases: Magnetic field writing, digitization, HepMC3 recording, FATRAS, geometry construction, material recording/mapping/ validation, particle gun, propagation tests, Pythia8 input, seeding, truth tracking, CKF track finding
- Tests run in CI, check multi-threaded execution succeeds, asserts outputs in some cases
- Added **reproducibility tests**: ROOT outputs are *hashed*, current test results are compared against stored hash
- Hashes are ordering independent. Can test
  - **Multi-threaded** reproducibility
  - Functional regressions (same output as before)

Build-time memory consumption

- **Recall:** large parts of ACTS are written as templated code (+ Eigen is heavy)
- Allows zero-cost extension mechanisms, e.g. in the Propagator
- **But:** templates located in headers, pulled into many compilation units
  $\Rightarrow$ compilation becomes resource intensive
- **Example:** all compilation units using (C)KF clock in at $>$ 4GB of peak memory
- Overarching goal: rationalize + factorize to try to reduce this

Kalman Filter extension mechanism

- KF itself is implemented as an *actor* in the propagation
- KF can also be extended/customized:

```
template <typename propagator_t, typename updater_t = VoidKalmanUpdater,
          typename smoother_t = VoidKalmanSmoother>
class KalmanFitter;

template <typename source_link_t>
struct KalmanFitterResult;

template <typename calibrator_t, typename outlier_finder_t,
          typename reverse_filtering_logic_t>
struct KalmanFitterOptions;
```

- Consequence: KF template is instantiated often (different parameters) $\Rightarrow$ **heavy memory footprint**

New Kalman Filter extension mechanism

```
struct KalmanFitterExtensions {
  using Calibrator = Delegate<void(const GeometryContext&, TrackStateProxy)>;
  using Smoother = Delegate<Result<void>(
      const GeometryContext&, MultiTrajectory&, size_t, LoggerWrapper)>;
  using Updater =
      Delegate<Result<void>(const GeometryContext&, TrackStateProxy,
                            const NavigationDirection&, LoggerWrapper)>;

  Calibrator calibrator;
  Updater updater;
  Smoother smoother;
  // + additional components
};
```

```
KalmanFitterExtensions extensions;
extensions.calibrator.connect<&voidKalmanCalibrator>();
extensions.updater.connect<&voidKalmanUpdater>();
extensions.smoother.connect<&voidKalmanSmoother>();
//...
```

Kalman Filter math component factorization

What is `SourceLink`?

- Two types of measurements: **calibrated** and **uncalibrated**
- `SourceLink` is ACTS' proxy for **uncalibrated measurements**
- Are turned into **calibrated measurements** by a **calibrator** during track fitting

- **Previously:** concrete type, given as template parameter to everything
- **New:** inherits from `Acts::SourceLink` (minimal base class, no virtual methods)
- Allows splitting up definition/declaration, create smaller compilation units

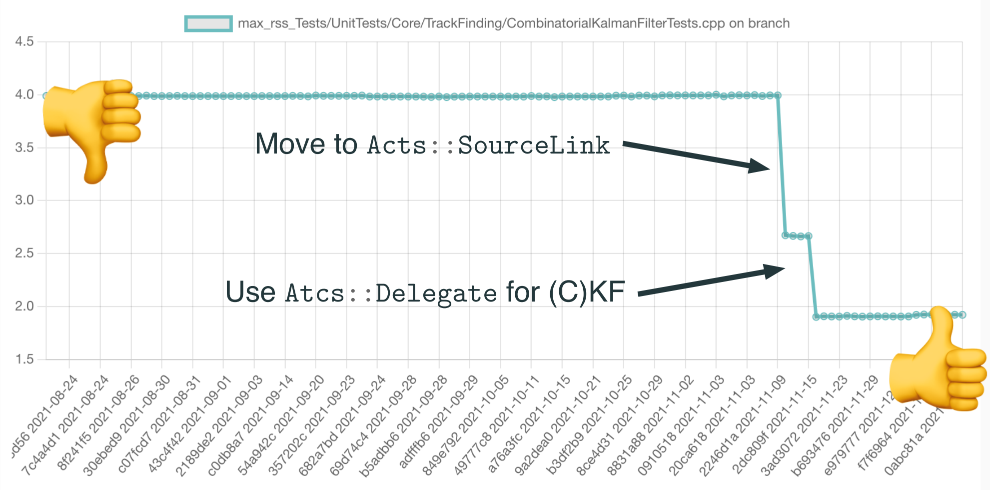
Kalman Filter math component factorization

```cpp
class GainMatrixUpdater {
 public:
  template <typename source_link_t,
    size_t kMeasurementSizeMax>
  Result<void> operator()(
      const GeometryContext& gctx,
      detail_lt::TrackStateProxy<
        source_link_t,
        kMeasurementSizeMax,
        false
      >& trackState,
      const NavigationDirection&
        direction = forward,
      LoggerWrapper logger
        = getDummyLogger()) const {
    /* CODE HERE */
  }
};
```

```cpp
class GainMatrixUpdater {
 public:
  Result<void> operator()(
    const GeometryContext& gctx,
    MultiTrajectory::TrackStateProxy
      trackState,
    NavigationDirection direction
      = forward,
    LoggerWrapper logger = getDummyLogger()
  ) const;
};
```

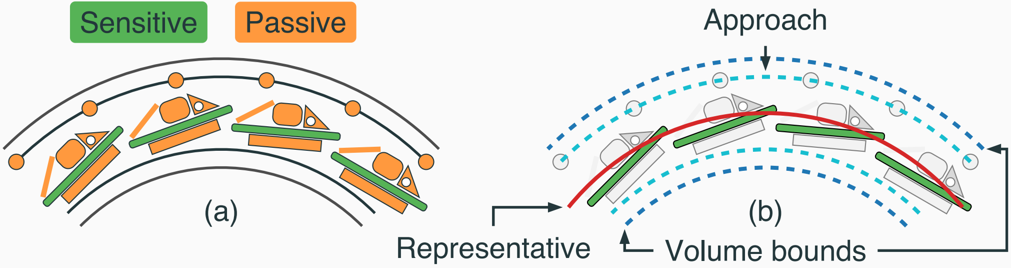
GainMatrixUpdater.hpp

Impact on build performance



Internal R&D: geometry model without layers

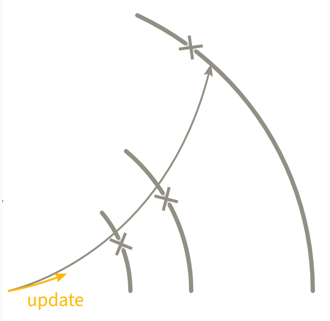
- *Conventional* navigation model used by ACTS: **sensors** on **layers** in **volumes**



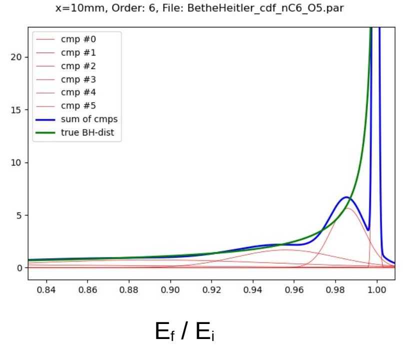
- However: layers with thickness are essentially volumes
- Recent work to try to remove the concept **layer** and only use hierarchy of volumes
- Promising reduction in complexity of the navigation code, fewer navigation states

Development of new fitters

- Integrated and tested so far: Kalman Filter (+ *Combinatorial* for track finding)
- Want to add alternatives:
  - **Global χ^2 fitter** for precision KF alternative (recent presentation)
  - **Gaussian Sum Filter** for treatment of non-gaussian noise (e.g. Bremsstrahlung) (recent presentation)
- Progress is being made, hope to integrate early next year

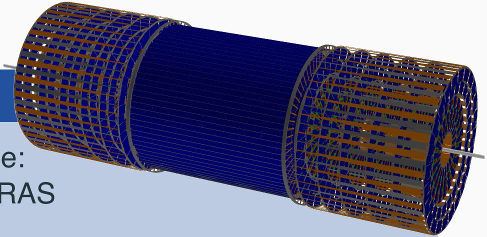


R. Farkas



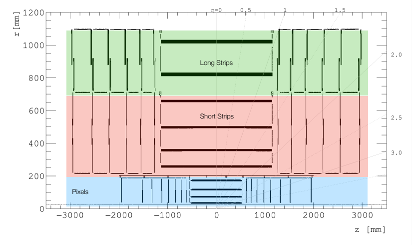
B. Huth

OpenDataDetector

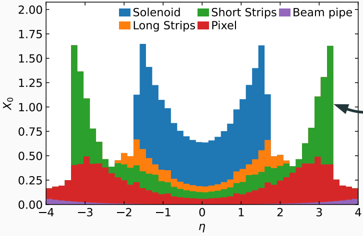


From TrackML to OpenDataDetector

- Dataset for Tracking Machine Learning challenge: generated with generic detector and ACTS FATRAS
- Caveat: no realistic passive material description
- Follow up: **OpenDataDetector** with passive material



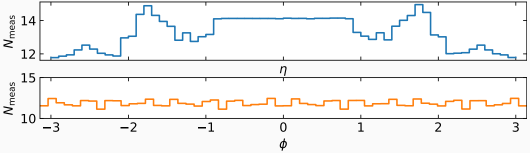
- Current focus: characterize detector, performance
- Tools for this added to core



material budget

layout

number of measurements



Parallelization

Paralellization R&D projects

vecmem

Ergonomic and consistent host+device memory management for CUDA, SYCL, HIP

detray

Geometry implementation with simplified polymorphism (no inheritance)

algebra-plugins

Generalized linear algebra for geometry needs (wraps Eigen, Vc, SMatrix, cmath + STL & vecmem storage)

traccc

Combination targeting tracking chain demonstrator

vecmem

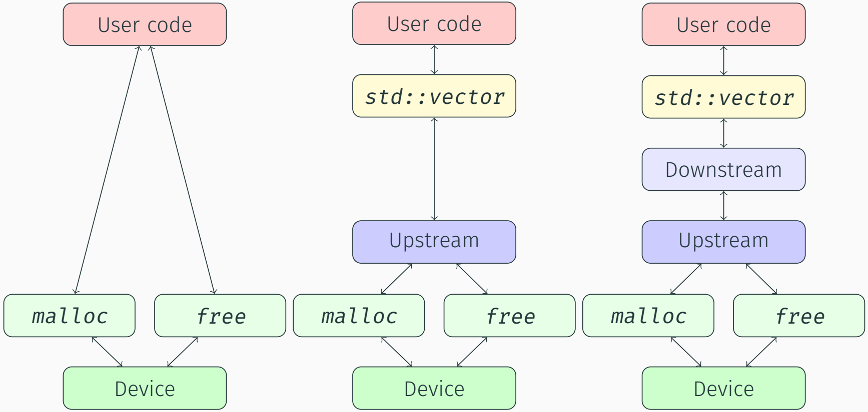
- *Modern* C++ heavily encourages use of STL containers like `std::vector`
- Idea: use `std::pmr` from C++17[1] to provide memory management

```
int main(void) {
    vecmem::cuda::managed_memory_resource mem;
    vecmem::vector<int> vec(&mem);
    // All data that we insert into this vector is transparently
    // accessible on the device!
    v.push_back(5);
    v.push_back(10);
    v.push_back(2);
    my_kernel<<<...>>>(vecmem::get_data(vec));
}
```

- See also Attila's recent talk in CAF and Stephen's talk at ACAT 2021

[1] `libc++` only has partial support, workaround exists

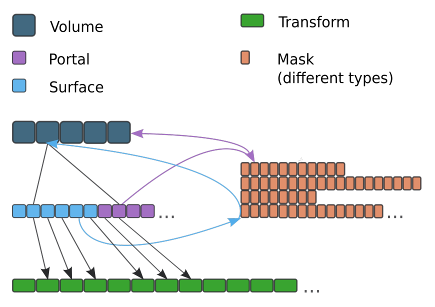
vecmem: memory resources

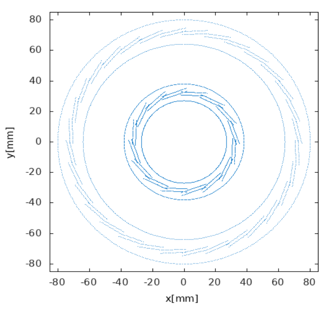


- Ship memory resources like arena, instrumenting + others
- Support implicit STL allocation, but also explicit copies

detray

- Geometry description using only flat containers and index ranges
- Recent development effort to **generalize** the containers, **bullet-proof** the interlinking, **automated testing** of navigation stream
- See Joana's talk at ACAT 21
- With basic navigation implemented: focus on GPU grid to enable geometry processing





traccc

- Overall since May: focus on cleanup, restructuring, deduplication
- Converged on our algebra-plugins for linear algebra math needs
- Overhauled algorithm semantics to test different reconstruction chain combination

Types	Algorithms	CPU	CUDA	SYCL
Hit clusterization	CCL			
	measurement creation			
	spacepoint formation			
Track finding	binning spacepoints			
	seed finding			
	track param estimation			
	Combinatorial KF			
Track fitting	KF			

Merged

Work in progress

Not yet started

Beomki Yeo

Machine learning

Machine learning

- Collaboration/exchange with Exa.TrkX: Graph Neural Networks for track finding
  - Promising speedups achieved, quite performance not quite there yet
  - Evaluating / comparing different approaches
- Hyper-parameter optimization of tunable parameters (e.g. seed finding) still under study

Conclusion

- ACTS development is progressing
- Strong developments in ACTS Core
  - Python bindings simplify example workflows and enable reproducibility tests that will help us a lot going forward
  - Improvements to build resource consumption
  - Simplification of the (C)KF extension mechanism (will likely expand to other components)
  - Developments of additional fitters progressing nicely!
  - OpenDataDetector validation drives addition of analysis/validation scripts
- R&D lines
  - Parallelization lines are converging towards a GPU KF implementation
  - Machine learning developments mostly driven through cooperation, very interesting results
- ACTS paper accepted for publication in CSBS