The Acts project: track reconstruction software for HL-LHC and beyond

Paul Gessinger[1,2,*], *Hadrien* Grasland[3], *Heather* Gray[4,8], *Moritz* Kiehn[5], *Fabian* Klimpel[1,6], *Robert* Langenberg[1], *Andreas* Salzburger[1], *Bastian* Schlag[1,2], *Jin* Zhang[7], and *Xiaocong* Ai[8]

[1]CERN
[2]Johannes Gutenberg-Universität Mainz
[3]Centre National de la Recherche Scientifique
[4]Lawrence Berkeley National Laboratory
[5]Universite de Genève
[6]Technische Universität München
[7]Chinese Academy of Sciences
[8]University of California, Berkeley

Abstract. The reconstruction of trajectories of the charged particles in the tracking detectors of high energy physics (HEP) experiments is one of the most difficult and complex tasks of event reconstruction at particle colliders. As pattern recognition algorithms exhibit combinatorial scaling to high track multiplicities, they become the largest contributor to the CPU consumption within event reconstruction, particularly at current and future hadron colliders such as the LHC, HL-LHC and FCC-hh. Current algorithms provide an extremely high standard of physics and computing performance and have been tested on billions of simulated and recorded data events. However, most algorithms date back to more than 20 years ago and maintaining them has become increasingly challenging. In addition, they are challenging to adapt to modern programming paradigms and parallel architectures.

Acts is based on the well-tested and highly functioning components of LHC track reconstruction algorithms, implemented with modern software concepts and inherently designed for parallel architectures. Multithreading becomes increasingly important to balance the memory usage per CPU core. However, a fully multithreaded event processing framework blurs the clear border between events, which has in the past often been used as a clearly defined validity boundary for event conditions. Acts is equipped with a full contextual conditions concept that allows to run concurrent track reconstruction even in case of multiple detector alignments, conditions or varying magnetic field being processed at the same time. It provides an experiment and, in particular, framework-independent software toolkit and light-weight, highly optimised event data model for track reconstruction. Particular care is given to thread safety and data locality. It is designed as a toolbox that allows to implement and extend widely known pattern recognition algorithms, and in addition suitable for algorithm templating and R&D. Acts has been used as the fast simulation engine for the Tracking Machine Learning (*TrackML*) Challenge, and will provide reference implementation of several submitted solution programs of the two phases of the challenge.

[*]e-mail: paul.gessinger@cern.ch

1 Track reconstruction

In high energy physics experiments the primary goal is to detect particles and measure their properties in order to use them to further our understanding of the physical processes involved. One aspect of particle detection and measurement is the reconstruction of charged particle trajectories. This is done using dedicated detection systems, often based on silicon sensors that record deposited energy from traversing charged particles.

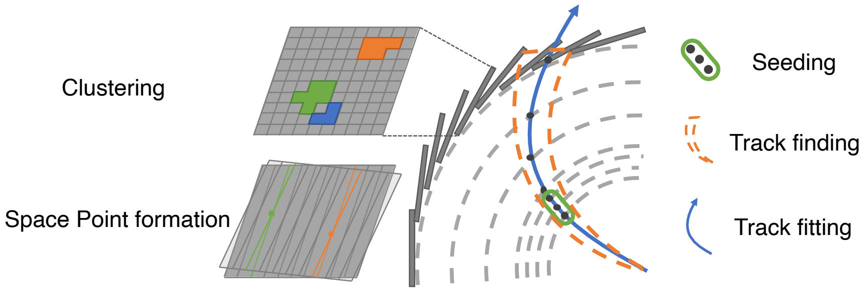

Figure 1. Overview of a possible tracking chain: Clustering, space point formation, seeding, track finding and track fitting.

The generic workflow employed in many track reconstruction implementations (see fig. 1) in HEP starts with building measurement space points from the raw information collected from the detectors. Three space points are then combined into triplets, and filtered for compatibility with them coming from the interaction point and kinematic requirements.

During track finding, triplets are extended along the direction of the assumed momentum of the particle, and additional measurements are added to the track candidate. Subsequently, a dedicated fit [1, 2] of the particle trajectory is performed to extract estimates for the particle's properties. Ambiguities in the assignment of measurements to tracks need to be resolved using dedicated algorithms.

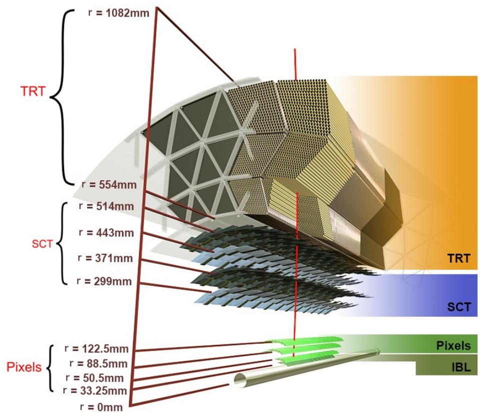

Figure 2. Schematic overview of the ATLAS Inner Detector tracking system. Shown are the barrel parts of the Pixel, SCT and TRT detectors in ascending order of radii. Figure taken from [3]

To enable the above procedures a complete description of the geometry of the detector in question is required.

In the ATLAS experiment [4], the tracking system (see fig. 2) in the center of the detector, called the *Inner Detector* consists of two silicon based components the *Pixel* detector and *Semi Conductor Tracker* (SCT), each featuring multiple barrel layers and endcap disks. In addition, ATLAS employs a *Transition Radiation Tracker* (TRT), containing gas filled straw tubes with a wire at the center.

Using the fitted tracks as an input, their respective origins, called vertices, can be determined. The vertex parameters can be estimated from the assigned tracks. High quality vertices can be an effective method to reduce tracks from background activity.

1.1 Challenges at the HL-LHC

In the past, tracking has been found to routinely be one of the largest consumers of CPU resources in event reconstruction [5]. Due to the nature of the problem, tracking complexity is very tightly related to the number of concurrent interactions occurring in the interaction region (pile-up), which will be increased by a factor of 3-4, for the *High-Luminosity LHC* (HL-LHC) [6]. This means reconstruction performance in general, and track reconstruction in particular will become even more critical.

To accommodate this new environment, experiments like ATLAS are adapting as well. New subdetectors are built, upgrades performed, algorithms can be rewritten and tuned, in order to optimize performance and remove aspects which are not required anymore. These optimizations (such as [7]) can improve the scaling of runtime with increasing luminosity significantly.

Apart from this, computing workloads also need to be able to run in the environments that are available for execution. Making sure to maximize utilization of available resources is therefore crucial. One such optimization is to make sure that the applications run within the memory limit of the execution environment. ATLAS, for instance, is migrating most of their software [8] towards a multi-threaded execution model [9]. This can enable new techniques to reduce the memory usage of the computations by sharing memory between threads. However, software needs to be explicitly written to be *thread-safe* in order to ensure error free and correct execution. To achieve this, all parts of the ATLAS software and the track reconstruction software [10] in particular, are undergoing development to remove any code that is not thread-safe, and build alternative implementations.

2 The Acts Project

2.1 Goals and project description

The Acts project [11] is an experiment independent tracking framework containing algorithms and tools. It is originally based on the ATLAS tracking software, and many of the concepts and methods are influenced by it. The goal is to enable building potentially experiment-specific *applications* using the components provided by Acts. These components are designed from scratch to be thread-safe by default. The components provide extensibility and flexibility, to allow injection of experiment specific configuration and workflows where necessary. One example is a generic geometry module, that can be adapted for a specific experiment. Compile-time calculations and optimizations are extensively used, which enable high performance at runtime. At the same time, virtual inheritance is avoided where possible.

Extraction of the code into an experiment independent environment has the benefit that changes can be made without having to adjust potential clients immediately. This facilitates shorter development cycles. Additionally, it enables the possibility to use the software in the context of other experiments. While initial developments in Acts focus on R&D, an eventual deployment as part of the production reconstruction software in ATLAS is foreseen. This contribution presents progress that has been made towards deployment, by connecting Acts to the ATLAS geometry, and providing thread-safe access to information required during processing.

2.2 Modeling of the ATLAS geometry

The long term goal of deployment of Acts components into the ATLAS software stack requires implementing the current and future ATLAS geometry using Acts' geometry libraries. To this end, Acts is built as part of the ATLAS externals project, which delivers libraries for use within the Athena code. Dedicated geometry building code reads the geometry from ATLAS' internal geometry representation, GeoModel [12]. This information is then processed and converted into the Acts geometry description. The geometry description is optimized for fast navigation, by preparing its structure during construction. To this end, elements are logically ordered and collected to efficiently retrieve potential navigation targets during propagation. Sensors are grouped into layers which, in turn, are grouped into volumes. Navigation occurs from sensor to sensor, then from layer to layer, and finally from volume to volume. Volume navigation works by testing which boundary surface of the current volume is intersected by the trajectory, and looking up what the corresponding volume to enter on the other side is.

Construction and modeling of the silicon detectors are very similar to the implementation in the ATLAS tracking software. For the TRT it differs in that the Acts version models every straw individually, rather than grouping them, since it is conceptually simpler. Another difference is the modeling of the calorimeter. While the calorimeter is not explicitly used for tracking, it is still required to be able to propagate trajectories through it. This propagation occurs through a simplified version of the calorimeter geometry (see figure 3), which does not correspond directly to the physical geometry of the hardware, but rather a representation of where the readout's location fits best in the laboratory frame, in order to assess which cells are hit. The ATLAS implementation uses a sophisticated procedure to interconnect cells, and uses passive layers to account for the significant material in the calorimeter. This method is tightly coupled to the specific layout of the ATLAS calorimeter, and cannot handle overlapping cells, which occur in the readout version of the calorimeter geometry.

The Acts version of the calorimeter geometry uses a different approach: All cells are wrapped into bounding boxes, whose defining edges are aligned with the axes of the laboratory frame. These bounding boxes are then put into a hierarchy of bounding boxes of the same type. Navigation through this hierarchy then works by intersecting rays with the hierarchy, finding all compatible cells, and then testing all possible entry and exit surfaces. This approach is insensitive to the specific layout of the geometry, and doesn't require interconnecting the cells. It can also handle overlapping cells. The ray intersection procedure was integrated into the navigation sequence and tested to work. However, the ray hypothesis only holds for high-momentum particles, where particle tracks are almost straight. For lower momentum particles, an implementation using a frustum intersection was implemented and tested, but requires further work to be integrated into the navigation sequence.

2.3 Concurrent conditions handling and alignment

To achieve the most precise parameter estimations, careful handling of potential misalignments of the sensors is required. This requires determining the location of sensors for each

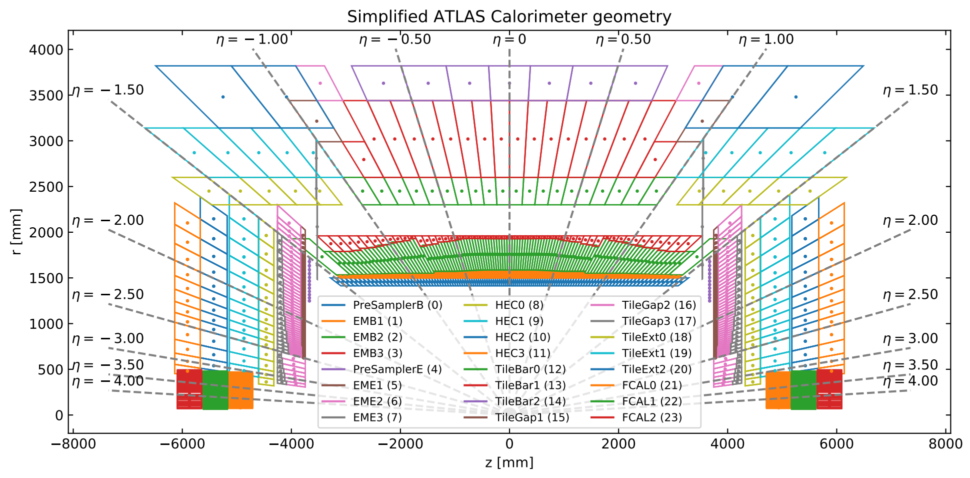

Figure 3. Visualization of the ATLAS calorimeter readout geometry. The three subsystems, Tile, Liquid Argon and the Forward calorimeters, are shown.

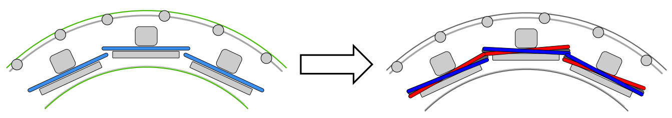

Figure 4. Illustration of the impact of misalignment on the sensitive elements of a tracking detector. On the left, the blue sensors are aligned nominally, while on the right the impact of differing alignment per IOV is illustrated by the variation of blue and red sensor rotations.

interval of validity (IOV), a time period while data taking during which the detector positions are assumed not to change. The multi-threaded execution model presents the challenge that the varying parameters need to be loaded and applied in a concurrency-safe manner. This is complicated by the fact that the call chains and object hierarchies in the ATLAS tracking software are relatively deep, which causes propagation of information to require more effort. In ATLAS, the strategy is to copy the nominal collection of detector elements in memory, which is always loaded, for every IOV. After copying, the internal caches of the detector elements are updated with the alignment data for the IOV. Finally, the tracking geometry is constructed from scratch with the correct alignment. In the Acts implementation of the ATLAS geometry, a different approach is chosen.

All Acts classes and functions that need to be aware of context dependent changes to the geometry accept an explicit context object as their first argument. They also pass the object to any function that they call which also requires this information. Acts is unaware of the concrete type of this object, an `std::any` object from the C++ standard library is used. At the end of many of the call chains, the `Acts::Surface` object receives the context object. This class represents all sensitive modules, and is written in an experiment agnostic way. In ATLAS, the surface object holds a reference to an underlying detector element class, which is aware of the experiment. This class can therefore unpack the context object into a known type (by construction), and use its information to calculate the correct alignment for the IOV.

A tracking geometry tool, which makes sure that the geometry context is available for every IOV, and provides a method for clients to retrieve it, was implemented. Clients using the Acts extrapolation request the current geometry context object, while providing the general event context. The geometry context is used to call the extrapolation routine. Detector elements receiving requests for aligned transforms unpack the context, and unpack the required transform. The detector element does not need to be copied per IOV, and also doesn't have to explicitly talk to the data source.

The setup mentioned above has been implemented for the Acts version of the ATLAS silicon trackers. Alignment constants are concurrently read from the upstream data-source, and the Acts propagation uses it to retrieve IOV dependent transforms. This was tested to run safely and concurrently.

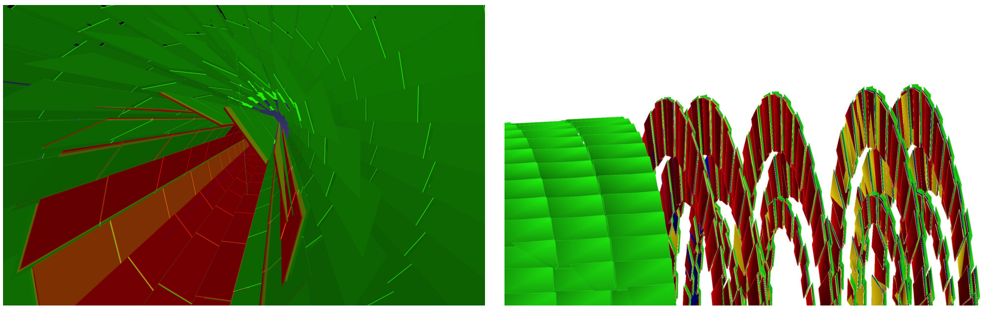

Figure 5. Exaggerated visualizations of the effect of alignment constants for the Pixel detector on the left and the SCT endcaps on the right. The colors indicate different IOVs. The varying detector positions indicate that the sensor positions depend on the IOV. In the left image, the IOVs are from a single data taking run, and the bowing of the innermost sensor layer increases with the IOV as expected. The information is retrieved and applied concurrently.

3 Summary and outlook

Since CHEP 2018 [13], Acts has received substantial development efforts on many fronts. The recent addition of time to the propagation, performance characterization of the Kalman Filter and first tools for vertex reconstruction are a few examples. With critical components for seeding, track fitting and vertexing now available, a full track reconstruction chain can soon be demonstrated. Remaining building blocks such as track finding are under active development.

Alongside this progress, the effort to deploy Acts components within ATLAS is advancing as well. In addition to the Inner Detector, an Acts version of the calorimeter geometry is now available, using a new approach to navigation. Concurrent handling of time dependent information, such as alignment of tracking detectors, was implemented in ATLAS. Acts code that makes use of the provided infrastructure was added, and shown to work correctly. Using this as a basis, Acts components will be provided as packaged tools to enable usage from a growing number of clients.

4 Acknowledgements

This work was supported by the National Science Foundation under Cooperative Agreement OAC-1836650. This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 05E15CHA)

References

[1] R. Frühwirth, Nucl. Instrum. Meth. **A262**, 444 (1987)

[2] T. Cornelissen (2007), `https://indico.cern.ch/event/3580/contributions/1768469/`

[3] M. Aaboud et al. (ATLAS), Eur. Phys. J. **C77**, 673 (2017), `1704.07983`

[4] M. Aaboud et al. (ATLAS), Journal of Instrumentation **3**, S08003 (2008)

[5] A. Salzburger (2015), `https://indico.cern.ch/event/304944/contributions/1672482/`

[6] G. Apollinari et al., CERN Yellow Rep. Monogr. **4**, 1 (2017)

[7] Tech. Rep. ATL-PHYS-PUB-2019-041, CERN, Geneva (2019), `https://cds.cern.ch/record/2693670`

[8] M. Aaboud et al., *Athena* (2019), `https://doi.org/10.5281/zenodo.2641996`

[9] S. Kama et al. (ATLAS Collaboration), Tech. Rep. ATL-SOFT-PROC-2018-038, CERN, Geneva (2018), `https://cds.cern.ch/record/2649080`

[10] A. Salzburger et al., *The ATLAS Tracking Geometry Description*, `http://cds.cern.ch/record/1038098`

[11] A. Salzburger et al., *Acts project* (2020), `https://doi.org/10.5281/zenodo.3606011`

[12] J. Boudreau et al. (2004), `https://indico.cern.ch/event/0/contributions/1294152/`

[13] A. Salzburger, `https://indico.cern.ch/event/587955/contributions/3012710/`