Tracking SW Developements

Julia Hrdinka (TU Wien)

On behalf of the ACTS developers and the FCCSW team

FCC Tracking

FCC should profit from already existing software – well tested and used in running experiments

=> using the ACTS - A(TLAS) Common Tracking Software package for tracking

https://gitlab.cern.ch/acts/a-common-tracking-sw

- Encapsulation of the ATLAS tracking software Code
  - updated to new coding standards
  - prepared for concurrent use (GaudiHive)

- Common tracking software toolkit useable in various applications
  - ATLAS Run3
  - FCC
  - Machine Learning Challenge 2016

ACTS – Dependencies & Build

ACTS is an **experiment independent** package build with Cmake

Dependencies

- Event processing framework: Gaudi

  http://proj-gaudi.web.cern.ch/proj-gaudi/

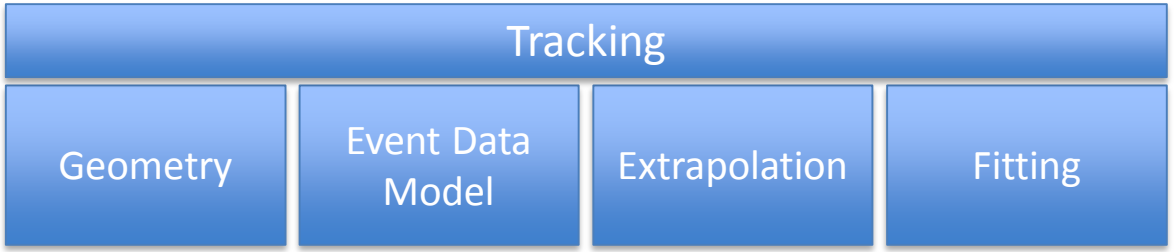
  - experiment independent
  - flexible structure: algorithms, services, tools
  - development of concurrent framework as GaudiHive

    http://concurrency.web.cern.ch/GaudiHive

    => further decoupling from Gaudi under investigation
- Math library: Eigen

  http://eigen.tuxfamily.org/
- Boost library

ACTS – package structure



- Core
  - Algebra (Math library)
  - Identifier
  - CoreInterfaces (AlgorithmBase, ServiceBase, ToolBase)
  - …
- Geometry
  - Detector
  - DetectorElementBase
  - Volumes
  - Surfaces
  - GeometryServices
  - …
- EventData
  - Measurements
  - TrackParameters
  - ...
- Examples
- Extrapolation
  - ExtrapolationEngine
  - RungeKuttaEngine
  - …
- Fitting
  - KalmanFitter
  - …
- Simulation
  - Fatras
  - Digitization
  - …
- Plugins
  - AtlasPlugins
  - GenericDetectorPlugins
  - FCCPlugins
  - ...

ACTS – Event Data Model

Two base classes: `TrackParametrization` & `Measurement`

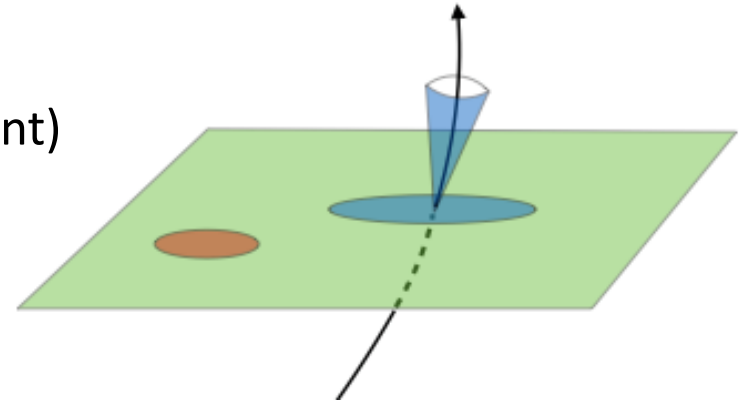
fixed size vector

dynamic sized vector

$$(l_0, l_1, \boxed{\varphi, \theta, \frac{q}{p}})$$

=> arbitrary (need to provide coordinate transform)

- in respect to coordinate system of specific surface (=detector element)
- using `ParameterSet` class (vector + covariance matrix) for flexible design and to customize parameters



ACTS – Extrapolation

Extrapolates the track parameters through the tracking geometry

- uses different engines to describe the sub processes
  - material effects
  - propagation
  - predictive navigation (used for Fast Track Simulation)
  - magnetic field integration (Runge Kutta Method)
- engines are stateless services for threat safety

ACTS – Fitting

Kalman Filter

- standard implementation
- reference based implementation
- possibility to recalibrate measurement

Extrapolation

- single track parameters and multi track parameters possible, supports Kalman Filter and Gaussian Sum Filter
- configurable for hole search

ACTS – Fast Simulation

Using the simplified reconstruction geometry as simulation geometry is common concept for fast simulation

The ACTS package provides

- Internal navigation through tracking geometry
- Extrapolation
  - transport through geometry & magnetic field
  - Monte Carlo material effects

  => Fast Track Simulation
- Simplified digitization

ACTS – Plugins

ACTS package is **experiment independent**

- Possibility to configure toolkit (Geometry Intput, EDM, Identifier, Magnetic Field)

  => Plugins

Provided Plugins

- ATLAS Plugin
- Generic Plugin (containing detector building from python script)
- Geometry Plugins: GeoModel , TGeo (Root) ,DD4hep

Examples - containing geometry building tests and fast simulation

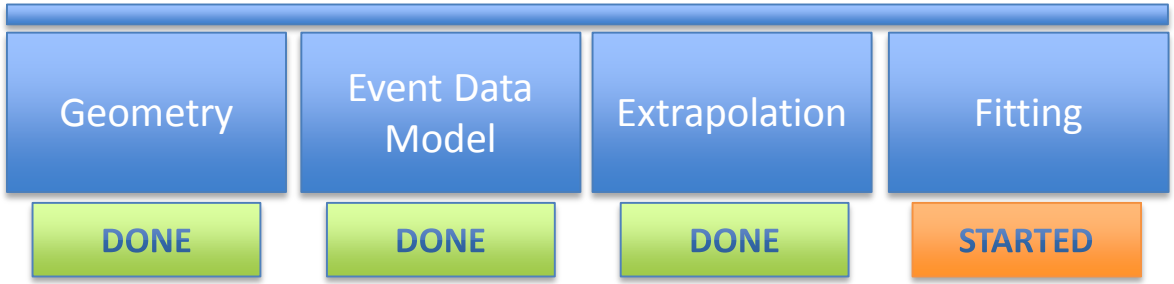
- Generic detector example

  https://gitlab.cern.ch/acts/a-common-tracking-sw/wikis/ATS-hello-detector
- FCC example

  https://gitlab.cern.ch/acts/a-common-tracking-sw/wikis/ATS-DD4hep-detector

ACTS – Status

0.1.0 (alpha) release – done



- compiling against Gaudi (Cmake) & ATLAS environment
- first dummy detector builds with python script
- limited fast simulation available

0.2.0 (beta) release – by the end of this month

- Kalman Filter prototype
- Fast simulation for π, e, p (no decay supported yet)
- Geometric digitization

Short test bed for pattern recognition - first components foreseen autumn 2016

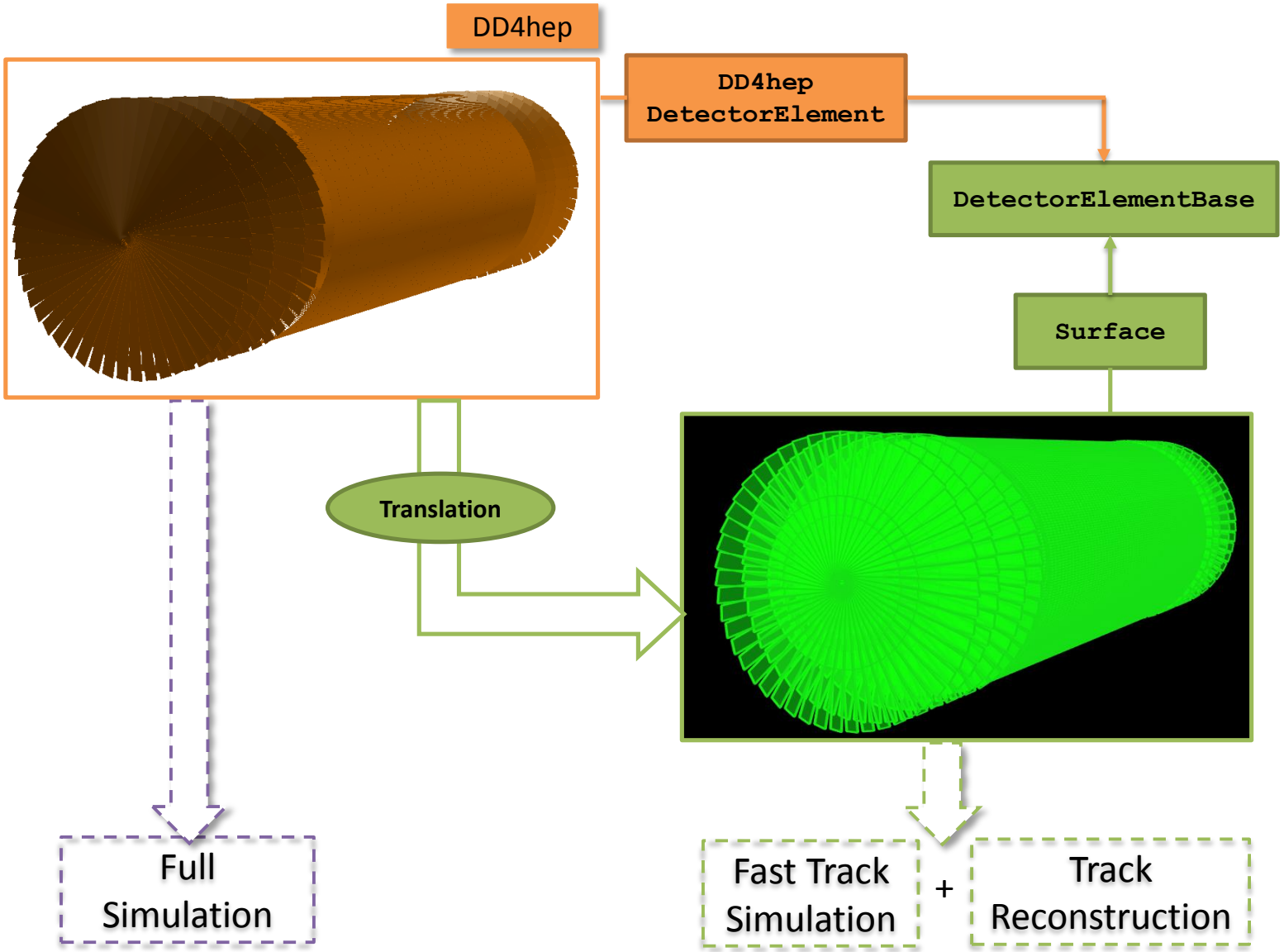
Usage in FCC

http://fccsw.web.cern.ch/fccsw/

ACTS is used as an external package

- Configure services and tools with python job option file
- Provide Identifier: DD4hep
- Provide MagneticFieldSvc (inheriting IMagneticFieldSvc)
  - ⇒ currently using const magnetic field service
- DD4hep (based on Root TGeo package) used for full detector description http://aidasoft.web.cern.ch/DD4hep
  - create `DD4hepDetElement` inheriting from `TGeoDetelement/DetectorElementBase`
  - create DD4hep translation tools – particularly simplified by geometry building tools provided by ACTS

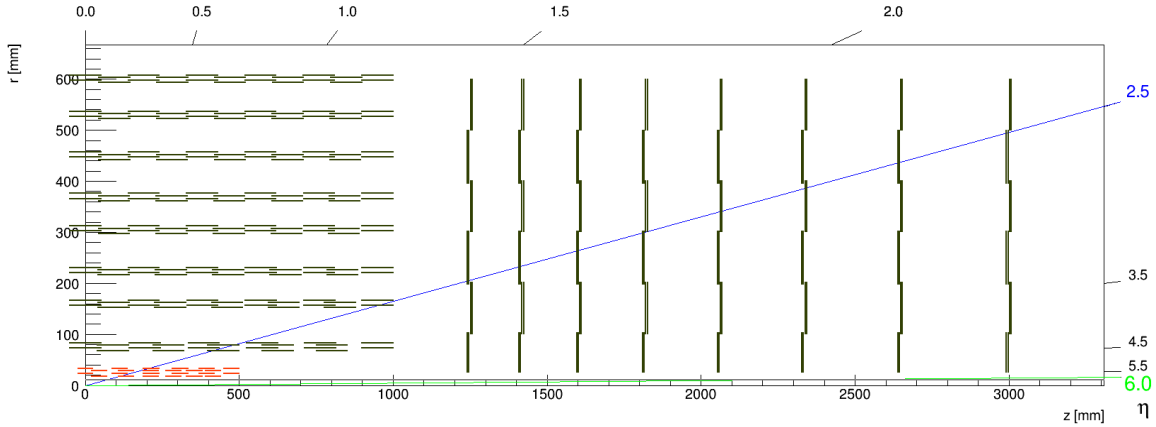
FCC geometry conversion



FCC(-hh) Test Tracker

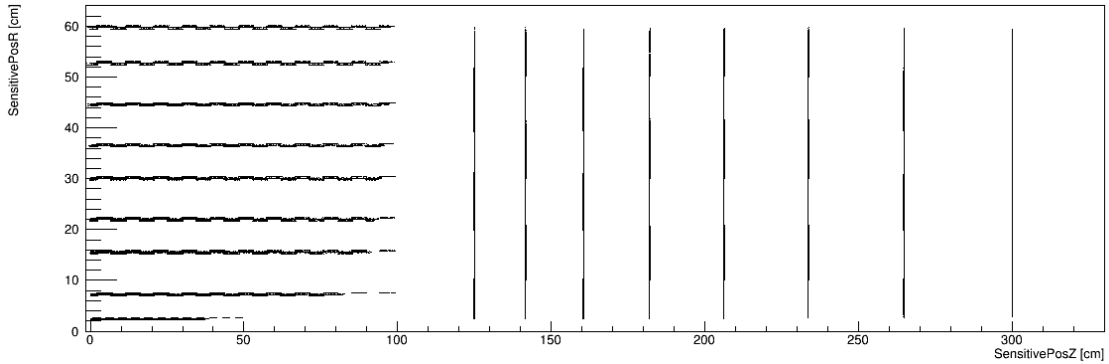
Geometry of FCC-hh tracker as created with tkLayout (a toolkit for ongoing tracker design studies of CMS phase 2 upgrade tracker, adapted to FCC-hh needs)

(-> Z.Drasal) https://indico.cern.ch/event/438866/contributions/1085165/



TKLayout

Translated into DD4hep and used in ACTS



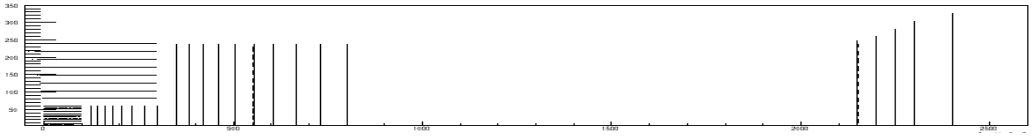
ACTS – Fast Simulation

FCC-hh Detector Design

with forward detectors

(->W.Riegler)

DD4hep



ACTS – Fast simulation

Conclusion & Outlook

- First ACTS demonstrator available for FCC
  - Geometry building from DD4hep
  - Fast simulation

- Track Fitting to be integrated until the end of April
- Start of pattern recognition integration into the ACTS package in summer 2016

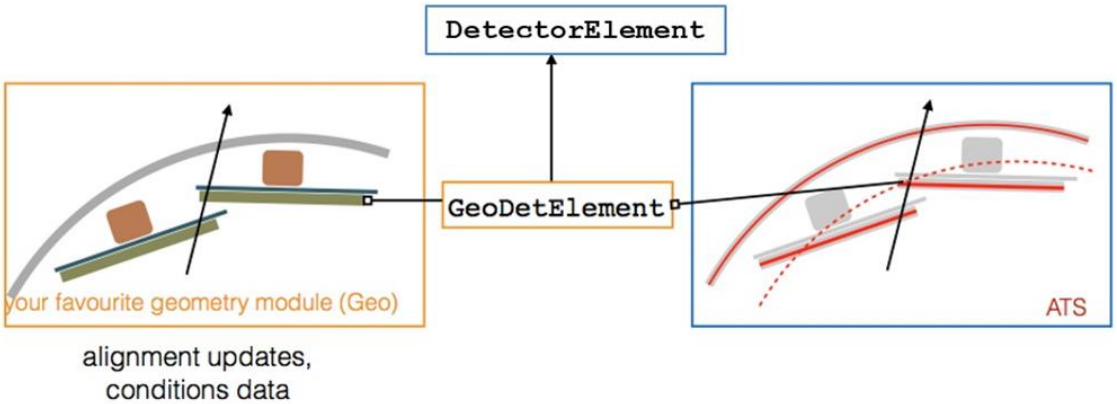
BACKUP SLIDES

Geometry - Basics

Tracking geometry = Simplified geometry describing sensitive material + approximated material setup

Core of the Geometry: `Surface` class

- represents detector element
  - connection to describing geometry via `DetectorElement`
  - base for measurement and parametrization

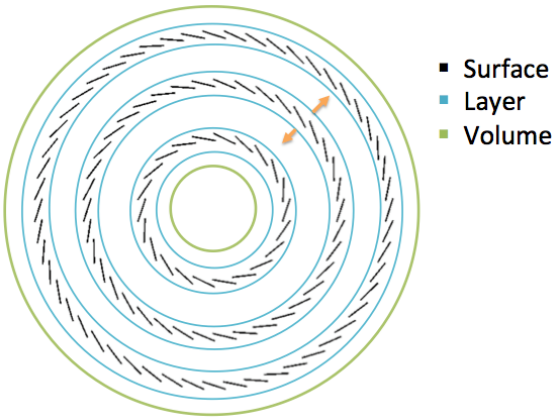


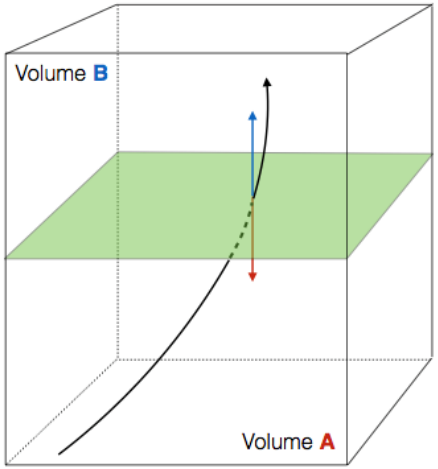
- layers extend surfaces
- volumes are enclosed by boundary surfaces

Geometry - Navigation

Internal Navigation

- layers contain binned array of surfaces
- volumes contain binned array of layers
- layers point to previous/next layer
- volume hierarchies staggered into each other – container volumes
- boundary surfaces of volumes point to previous/next volume





Geometry Plugins

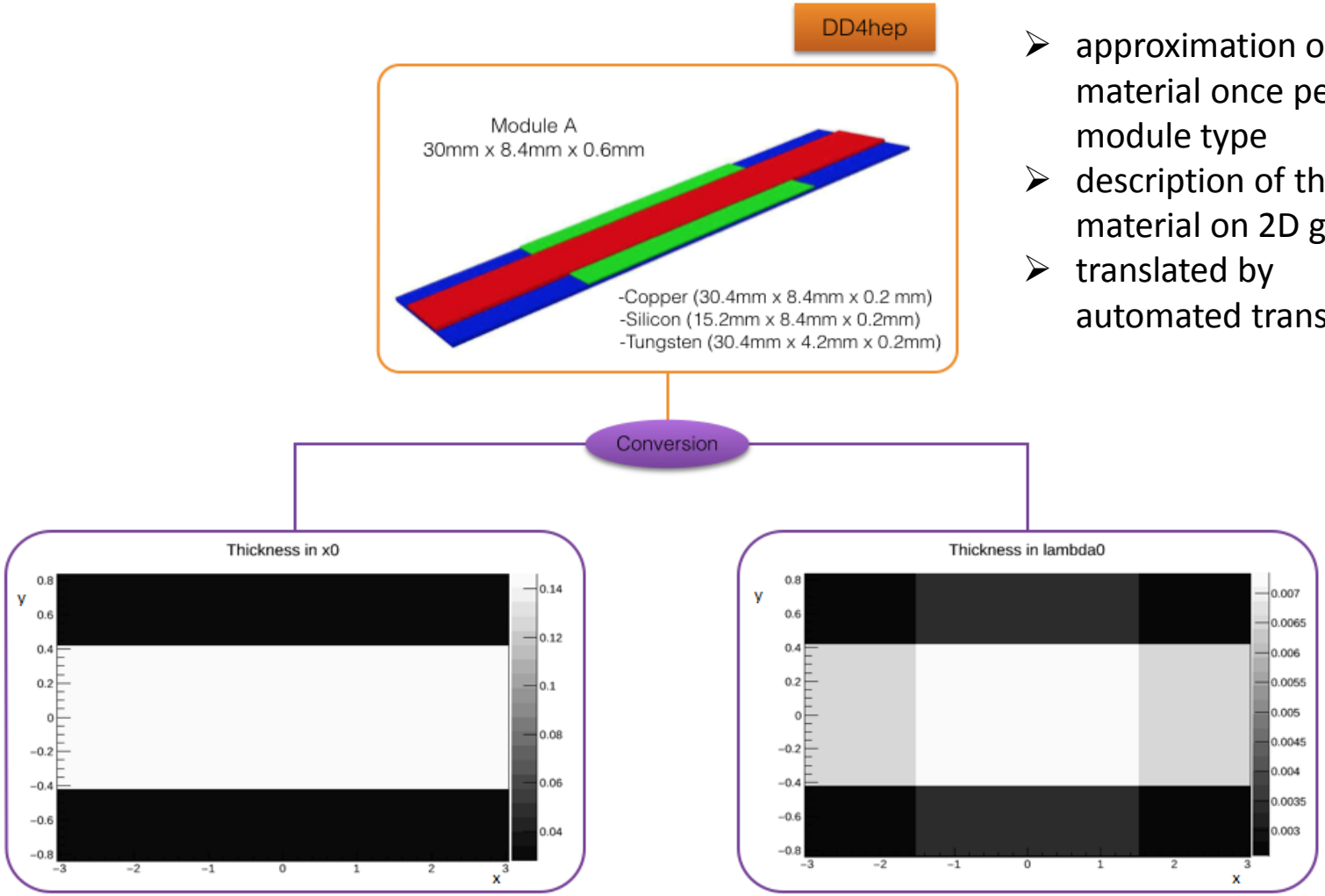
`DetectorElement` class

- implement `DetectorElement` inheriting from the `DetectorElementBase` class
- serves as an interface for the `Surface` class to access detector specific information from the building geometry
- direct link to tracking geometry via `Surface` class

Translation tools

- needed to translate the geometry of the detector which is not a detector element e.g. beam tube, supporting structure
- special building tools provided by ATS simplify the translation
    - building of container volumes
    - automatic interglueing of volumes for internal navigation
    - automatic binning of layers within volumes
- Provide layer builder which builds the layers with its containing surfaces

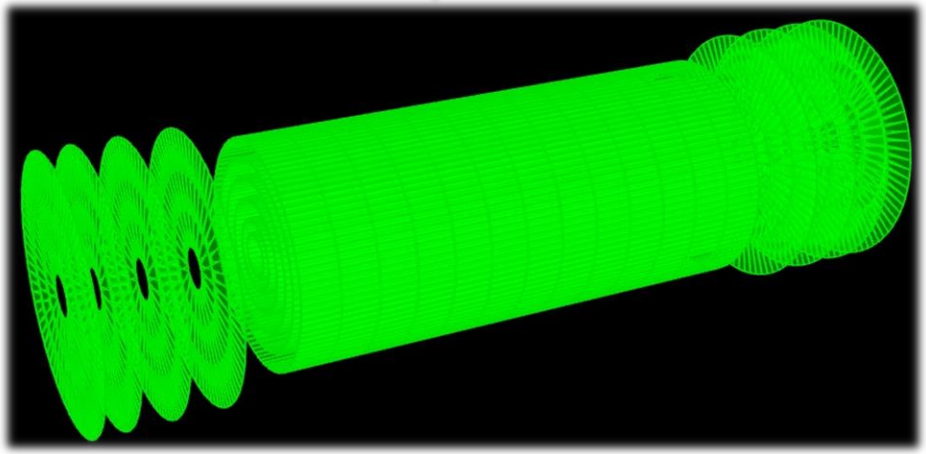
Material Mapping of a Detector Module



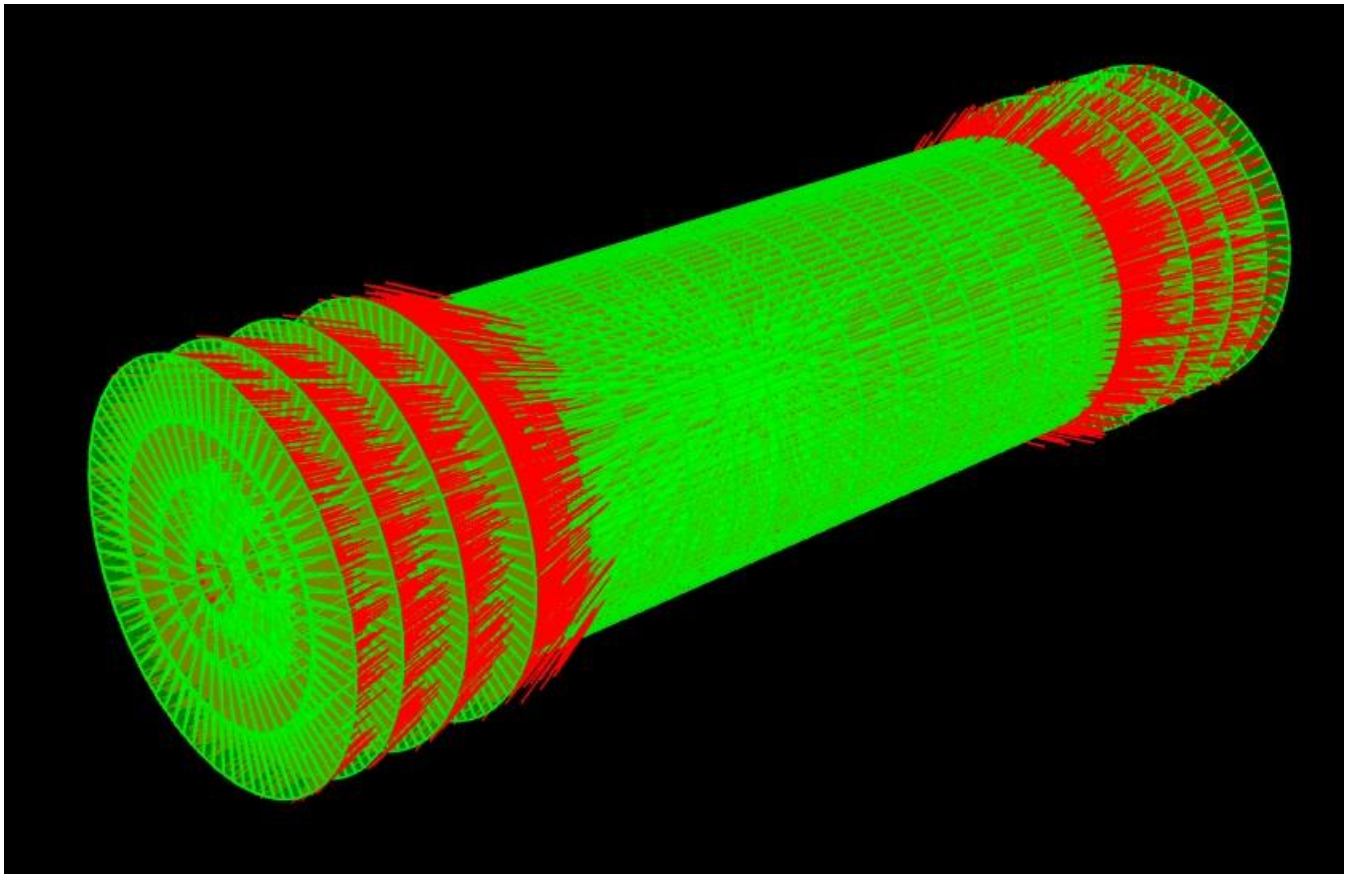
- approximation of the material once per module type
- description of the material on 2D grid
- translated by automated transcript

Example: Generic Detector

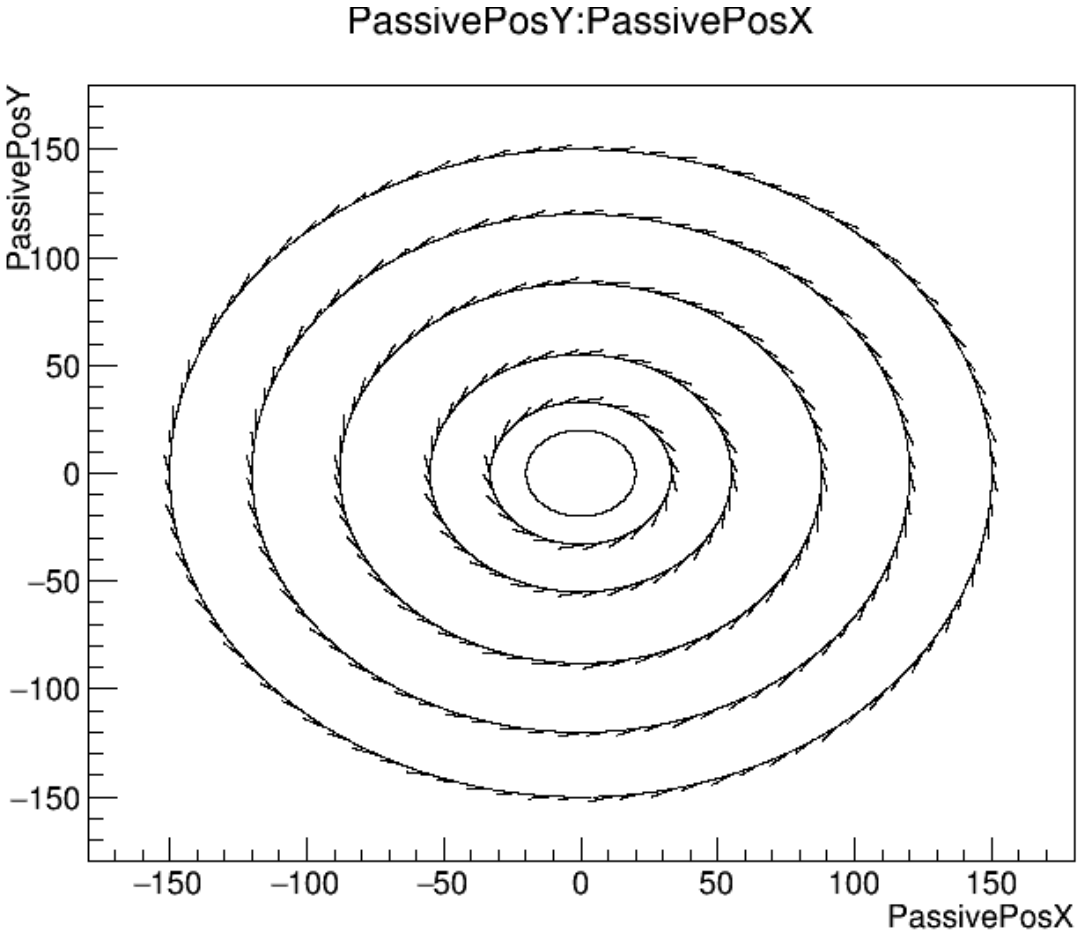
```
# the pixel module
PixelModule = DetectorModule(None,8.4,32.0,0.15)
# the first layer
PixelLayer0 = CylinderLayer(PixelModule, 33., 24, 13, 0.2, 2., 0.5, 5.)
PixelLayer1 = CylinderLayer(PixelModule, 55., 40, 13, 0.2, 2., 0.5, 5.)
PixelLayer2 = CylinderLayer(PixelModule, 88., 60, 13, 0.2, 2., 0.5, 5.)
PixelLayer3 = CylinderLayer(PixelModule, 120., 72, 13, 0.2, 2., 0.5, 5.)
PixelLayer4 = CylinderLayer(PixelModule, 150., 84, 13, 0.2, 2., 0.5, 5.)
# define the pixel barrel volume
PixelBarrel = BarrelVolume( [ PixelLayer0, PixelLayer1, PixelLayer2, PixelLayer3, PixelLayer4 ] )
```



Straight line tracks through dummy detector



First Tests



- picture done with fast track simulation
- shows a mockup detector using ATS and the GenericGeometryPlugin

Dynamic vs. fixed size matrices

- ATS is using Eigen as algebra library
- What is the performance penalty when using dynamic instead of fixed size matrices?

1) Eigen-3.2.7, g++ 4.9.2, 1M operations

	Dynamic sized	Fixed size	converted
M(2,5) x v(5)	1.7277s	0.797514s	1.45323s
M(5,2) x M(2,5)	3.53559s	2.67478s	3.48556s

2) Eigen-3.2.7, g++ 4.9.2 –O3, 100M operations

	Dynamic sized	Fixed size	converted
M(2,5) x v(5)	0.154546s	0.00747539s	0.0116945s
M(5,2) x M(2,5)	0.217145s	0.031977s	0.0326164s

=> optimization compiler flags give huge speedup, fixed size operations are a factor 8-20 faster

ParameterSet class

```
ParameterSet<loc1,phi> p1(0.5,1.5*M_PI);
cout << p1→get<loc1>() <<endl;  // prints 0.5
cout << p1→get<loc2>() <<endl;  // gives compiler error
cout << p1→get<phi>() <<endl;   // gives -0.5*M_PI
p1→getParameters();             // returns fixed size vector
p1→projector();                 // returns fixed size projection matrix
```

- perform consistency checks at compile time whenever possible
  => improved performance and code stability
- provide a mechanism for checking/transforming parameter values
  caveat: at the moment only for non-local parameters
- make use of fixed size matrices/vectors whenever possible
- **<u>fixed size</u>** projection matrix from full parameter space created at compile time

$$\begin{pmatrix} l_1 \\ \varphi \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}}_{projector} \begin{pmatrix} l_0 \\ l_1 \\ \varphi \\ \eta \\ \frac{q}{p} \end{pmatrix}$$

Measurement class

dynamic sized vector

```
class MeasurementBase
{
  virtual VectorX parameters() const = 0;
};
```

```
class Measurement1D: public MeasurementBase
{
  VectorX parameters() const override
  {
     return getParameters();
  }

  Vector1 getParameters() const;
};
```

```
class Measurement2D: public MeasurementBase
{
  VectorX parameters() const override
  {
     return getParameters();
  }

  Vector2 getParameters() const;
};
```

fixed size vectors

- concrete measurements are of different (C++) types
  => need common base class to store concrete measurements in a single vector, right?
- measurement base class must rely on dynamically sized vectors/matrices

ATS approach: `boost::variant`

- keep performance benefit from using fixed size matrix operations while allowing to treat different concrete measurements uniformly

```
enum class ParDefs {loc1, loc2, eta, phi, qop};

template<ParDefs… pars>
class Measurement
{
  typedef Vector<sizeof...(pars)> ParVector;
  ParVector parameters() const;
};

typedef boost::variant<...> FittableMeasurement;

Measurement<loc1,phi> m1;
Measurement<loc0,loc1> m2;

std::vector<FittableMeasurement> vMeasurements = {m1,m2};

struct print_visitor: public boost::static_visitor<>
{
   template<typename T>
   void operator()(const T& m) const
   {
      std::cout << "Generic measurement with " << m.parameters() << std::endl;
   }

   void operator()(const Measurement<phi>& m) const
   {
     std::cout << "Phi measurement: " << m.parameters() << std::endl;
   }
};

print_visitor v;
for(const auto& m: vMeasurements)
   boost::apply_visitor(v,m);
```

generic measurement definition

all possible measurement types

type which can hold any possible concrete measurement

ATS binding geometry link - TrkDetElementBase

- Connects any favourite geometry module to ATS

```
/** Identifier */
virtual Identifier identify() const = 0;

/**Identifier hash*/
virtual IdentifierHash identifyHash() const = 0;

/**Return local to global transform*/
virtual const Amg::Transform3D & transform() const = 0;

/**Return local to global transform associated with this identifier*/
virtual const Amg::Transform3D & transform(const Identifier& id) const = 0;

/**Return surface associated with this detector element*/
virtual const Surface & surface () const = 0;

/**Return surface associated with this identifier, which should come from the
PrepRawData object
(i.e. Surface s = PRD.detElement().surface( PRD.identify() ).
This is only really relevant for the TRT (where there are several
surfaces per detector element). For other detector elements it will just
return surface() ... the Identifier is ignored.
*/
virtual const Surface & surface (const Identifier& id) const = 0;

/** Returns the full list of all detection surfaces associated to this detector element */
virtual const std::vector<const Surface*>& surfaces() const = 0;
```

Parameter definitions

c14fce75a886... a-tracking-sw / Core / CoreUtils / CoreUtils / **ParameterDefinitions.h**

add parameter definition enum

Christian Gumpert authored 10 minutes ago

0a98030e

Browse Files »

ParameterDefinitions.h 870 Bytes

RAW | BLAME | HISTORY | PERMALINK | EDIT | REPLACE | DELETE

```
#ifndef ATS_PARAMETERDEFINITIONS_H
#define ATS_PARAMETERDEFINITIONS_H 1

#ifdef ATS_PARAMETER_DEFINITIONS_PLUGIN

#include ATS_PARAMETER_DEFINITIONS_PLUGIN

#else

namespace Ats
{
  enum ParDef : unsigned int
  {
    eLOC_1   = 0, ///< first coordinate in local surface frame
    eLOC_2   = 1, ///< second coordinate in local surface frame
    eLOC_R = eLOC_1,
    eLOC_PHI = eLOC_2,
    eLOC_RPHI = eLOC_1,
    eLOC_Z = eLOC_2,
    eLOC_X = eLOC_1,
    eLOC_Y = eLOC_2,
    eLOC_D0 = eLOC_1,
    eLOC_Z0 = eLOC_2,
    ePHI     = 2, ///< phi direction of momentum in global frame
    eTHETA   = 3, ///< theta direction of momentum in global frame
    eQOP     = 4,  ///< charge/momentum for charged tracks, for neutral tracks it is 1/momentum
    NGlobalPar
  };
}  // end of namespace Ats

#endif // ATS_PARAMETER_DEFINITIONS_PLUGIN

#endif //  ATS_PARAMETERDEFINITIONS_H
```

possibility to define custom parameter definition

could have been: enum **class** ParDef
but then parameters[Ats::ePHI] not possible
(needs cast to unsigned int)

same for all possible parameter definitions
→ enforced by unit test

must be present

C.Gumpert

Parameter policy

c14fce75a886... a-tracking-sw / EventData / EventDataUtils / EventDataUtils / **ParameterPolicy.h**

add coordinate transformations and parameter policy

Christian Gumpert authored 9 minutes ago

c14fce75

Browse Files »

ParameterPolicy.h 1.15 KB

RAW BLAME HISTORY PERMALINK EDIT REPLACE DELETE

```
#ifndef ATS_DEFAULTPARAMETERPOLICY_H
#define ATS_DEFAULTPARAMETERPOLICY_H 1

// STL include(s)
#include <cmath>

#include "CoreUtils/ParameterDefinitions.h"
#include "EventDataUtils/ParameterTypes.h"

#ifdef ATS_PARAMETER_POLICY_PLUGIN

#include ATS_PARAMETER_POLICY_PLUGIN

#else

namespace Ats
{
  typedef ParDef ParID_t;
  typedef double ParValue_t;

  template<ParID_t>
  struct par_type;

  template<>
  struct par_type<ParDef::eLOC_1>
  {
    typedef local_parameter type;
  };

  template<>
  struct par_type<ParDef::eLOC_2>
  {
    typedef local_parameter type;
  };
```

C.Gumpert

Coordinate transformations

c14fce75a886... a-tracking-sw / EventData / EventDataUtils / EventDataUtils / **CoordinateTransformations.h**

add coordinate transformations and parameter policy

Christian Gumpert authored 15 minutes ago

c14fce75

Browse Files »

CoordinateTransformations.h 1.86 KB

RAW | BLAME | HISTORY | PERMALINK | EDIT | REPLACE | DELETE

```
#ifndef ATS_COORDINATETRANSFORMATIONS_H
#define ATS_COORDINATETRANSFORMATIONS_H 1

// ATS include(s)
#include "Surfaces/Surface.h"
#include "EventDataUtils/ParameterPolicy.h"

#ifdef ATS_COORDINATE_TRANSFORM_PLUGIN

#include ATS_COORDINATE_TRANSFORM_PLUGIN

#else

namespace Ats
{
  struct coordinate_transformation
  {
    typedef AtsVector<ParValue_t,Ats::NGlobalPar> ParVector_t;

    static AtsVectorD<3> parameters2globalPosition(const ParVector_t& pars,const Surface& s)
    {
      AtsVectorD<3> globalPosition;
      s.localToGlobal(AtsVectorD<2>(pars(Ats::eLOC_1),pars(Ats::eLOC_2)),globalPosition,globalPosition);

      return globalPosition;
    }

    static AtsVectorD<3> parameters2globalMomentum(const ParVector_t& pars)
    {
      AtsVectorD<3> momentum;
      double p = fabs(1./pars(Ats::eQOP));
      double phi = pars(Ats::ePHI);
      double theta = pars(Ats::eTHETA);
      momentum << p * sin(theta) * cos(phi), p * sin(theta) * sin(phi), p * cos(theta);

      return momentum;
    }
```

C.Gumpert