SPONSORED BY THE

Federal Ministry of Education and Research

EP-R&D

Programme on Technologies for Future Experiments

CERN

2022 ACTS Workshop

`detray` - Overview and Status

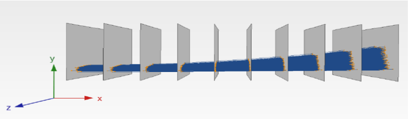
Joana Niermann

27.09.2022

The detray Project

The ACTS Kalman Filter on GPU

- Investigation of intra- and inter-track parallelization.
- Speedup of up to 4.6 towards multithreaded CPU, for more than 1000 tracks.
- Polymorphic geometry cannot be transferred to CUDA kernels.
- Telescope geometry and single surface type propagation are not very realistic.



detray powered by acts

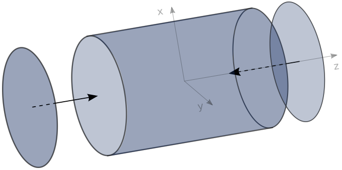
General Considerations

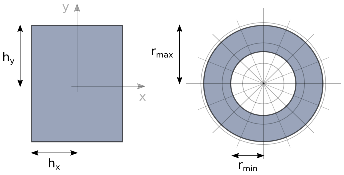
- Geometry classes without runtime polymorphism (no virtual function calls).
- Flat container structure using vecmem, with index based data linking.
- Implementation of core package usable in host and device code.

The detray Geometry Model

Building Blocks

- **Volumes**: logical containers for surfaces, defined by their boundary (portal) surfaces.
- **Surfaces**: Placed by transforms and defined by boundary masks in local coordinates.
- **Masks**: Define the shape types by defining local coordinates and extent of surfaces.
- **Portals**: Surfaces that tie volumes together through links (No static difference to sensitive surfaces).
- **Material:** Added to a surface in same way as a mask (link + tuple unrolling). Many predefined materials available.





No runtime polymorphism: Every type needs its own container: Compile time unrolling of mask container.

Heterogeneous Computing Model

Implementation in `detray`

- Goal: outsource many-track navigation to device.
- Need to handle host-device memory transfers.
- Core classes templated on `STL` vs. vecmem containers.
- The geometry data structures are built host side and memory allocation strategy is determined by *vecmem memory resources*.

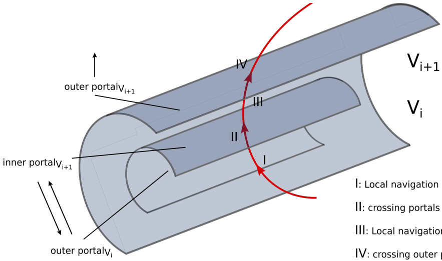
```
#include <vecmem/containers/vector.hpp>

// Transform store using managed memory
vecmem::cuda::managed_memory_resource mng_mr;

// Build with host vector type
transform_store<vecmem::vector> store(mng_mr);

// Get store view object
auto sv = detray::get_data(store);

// Run the kernel
test_kernel<<<block_dim, thread_dim>>>(sv);
```

```
#include <vecmem/containers/device_vector.hpp>

// Kernel-side construction
__global__ void test_kernel(store_view sv) {
  // Build with device vector type
  transform_store<vecmem::device_vector> store(sv);

  // Do something
}
```

Track State Propagation

Main Classes

- **Propagator:** steers the workflow between the stepper, navigator and the actors.
- **Navigator:** Moves between detector volumes and resolves next candidate surface.
- **Stepper:** Advances the track-state/covariance through the geometry.
- **Actors/Aborters:** Perform various tasks during propagation and watch termination criteria.



Geometry Navigation

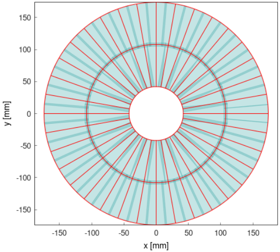
Trust-based candidate evaluation

- . . . cache line-surface intersections. *trust levels* determine update method:
- ***Full trust***: Do nothing.
- ***High trust***: Only update the current next target surface.
- ***Fair trust***: Update all entries and sort again.
- ***No trust***: (Re-)initialize the entire (current) volume, i.e. fill cache and sort by distance.

⇒ Stepper/actors can lower trust level to influence navigation update policy.

Local Navigation in a Volume

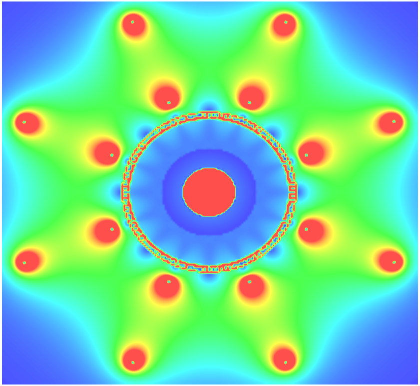
- Accelerators provide neighbourhood lookups during navigation candidate search.
- Navigate local neighborhood, before reassuming volume navigation
- Abstract interface towards navigation (iterator based)



Runge-Kutta Stepper

Numerical Integration

- Inhomogeneous B-Field: No track solution in closed form (e.g. helix)
- Adaptive Runge-Kutta Algorithm for Integration
- Gets the distance to next target surface and adjusts stepsize according to integration error (B-field)
- Transport the track parameter covariance in the same way



Summary - Status

Project Status

- Implementation of tracking geometry description: Index based linking, no runtime polymorphism, using flat containers.
- Fully featured prototype (toy) detector, modelled after ACTS generic detector's pixel detector.
- Successful porting of the core implementation to device, using `vecmem`.
- Successful track state propagation with covariance transport through toy detector (homogeneous B-field).
- Integration of covfie library.
- Integration of actsvg.

Summary - Outlook

WIP

- Prepared detector and indexing to reference grids and potentially other accelerator data structures on a per volume basis.
- New grid implementation for local navigation.
- Comprehensive local coordinate system implementation, including corresponding Jacobians.
- Material Interaction actor.
- Implement inhomogeneous B-field in the toy detector as first step.

Outlook

- Rigorous testing (benchmarks , physics performance . . .)
- Interface to read ACTS tracking geometry implementations.
- More subdetector geometry support/appropriate accelerator data structures.
- Extend backend support, e.g. SYCL.
- Further investigation of host-side optimization (vectorization, multi-threading of propagation).

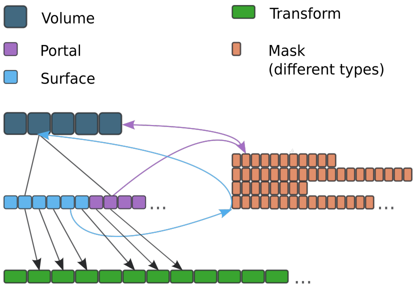
Backup

Detray Container Structure

In ACTS: Jagged memory layout of volumes containing layers (might be removed), containing surfaces.

Linking by Index

- Volumes keep index ranges into surface/portal containers.
- Surfaces/Portals keep indices into the transform and mask containers.
- Portals link to adjacent volume and next surfaces finder (local grid).
- Surfaces link back to mother volume.



⇒ Only transforms and masks contain geometric data, all other classes are uniquely used for container indexing.

Toy Detector - The TML Pixel Detector

Toy Detector

- Implement a small geometry, independent from io module
- All links are manually checked for consistency

The toy detector contains:

- A beampipe ($r = 27\,\mathrm{mm}$)
- An inner layer ($r_{\min} = 27\,\mathrm{mm}$, $r_{\max} = 38\,\mathrm{mm}$) with 224 pixel module surfaces
- A gap volume ($r_{\min} = 38\,\mathrm{mm}$, $r_{\max} = 64\,\mathrm{mm}$)
- An outer layer ($r_{\min} = 64\,\mathrm{mm}$, $r_{\max} = 80\,\mathrm{mm}$) with 448 pixel module surfaces
- Add grid will be added for local navigation.

$\Rightarrow$ Provides a reliable, dynamically generated geometry that can be used for testing and rapid development.

$\Rightarrow$ Complexity (number of barrel and endcap layers) can be configured for easier debugging.

Geometry Validation

Ray Scan

- Shoot straight line rays through detector setup
- Record every intersection, together with associated volume index.
- Sort by distance and check for consistent crossing of adjacent portals.

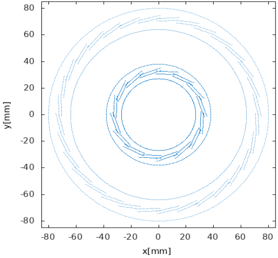
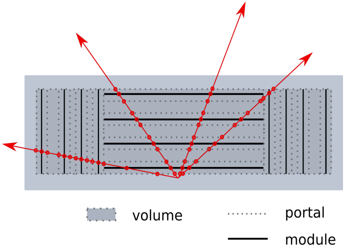


Figure 1: Intersections produced by ray scan. Two inner barrel layers modelled after ACTS Generic Detector (TrackML challenge).



Geometry Linking Validation

- Compare ray scan with geometry linking graph.
- Provides a coarse, but automated check of geometric setup.

Navigation Validation

- Shoot ray/helix, but this time follow with navigator.
- Compare the entire intersection trace with the objects encountered by navigator.

See also: Fig2(TrackML) `https://sites.google.com/site/trackmlparticle/`

The detray Actor Model

```
// initialize the navigation
navigator.init(propagation);

// Run while there is a heartbeat
while (propagation.heartbeat) {

  // Take the step
  stepper.step(propagation);

  // And check the status
  navigator.update(propagation);

  // Run all registered actors
  run_actors(propagation.actor_states, propagation);
}
```

What is an actor in `detray`?

- Callable that performs a task after every step.
- Has a per track state, where results can be passed.
- Can be plugged in at compile time.
- In `detray`: Aborters are actors

Implementation

- Actors can 'observe' other actors, i.e. additionally act on their subject's state.
- Observing actors can be observed by other actors and so forth (resolved at compile time!).
- Observer is being handed subject's state by actor chain
  $\Rightarrow$ no need to know subject's state type and fetch it.
- Greater flexibility in testing different setups

$\Rightarrow$ Currently implemented: Navigation policies, pathlimit aborter, propagator inspectors.

Actor Chain Implementation

Overview of actor implementation:

```
/// Base class actor implementation
struct actor {
  /// Tag whether this is a composite type
  struct is_comp_actor :
                public std::false_type {};

  /// Defines the actors state
  struct state {};
};
```

```
// Actor with observers
template <class actor_impl_t = actor,
          typename... observers>
class composite_actor final :
                           public actor_impl_t {
 struct is_comp_actor : public std::true_type{};
 // Implement this actor
 using actor_type = actor_impl_t;
 // Actor implementation + notify call
 void operator()(...) const { [...] notify(...);}

 private:
 // Call all observers
 void notify(...) const {...}
};
```

Building a chain:

```
// Define types
...
using observer_lvl1 = composite_actor<dtuple, print_actor, example_actor_t, observer_lvl2>;
using chain = composite_actor<dtuple, example_actor_t, observer_lvl1>;

// Aggregate actor states to be able to pass them through the chain
auto actor_states = std::tie(example_actor_t::state, print_actor::state);

// Run the chain
actor_chain<dtuple, chain> run_chain{};
run_chain(actor_states, prop_state);
```