SPONSORED BY THE
Federal Ministry
of Education
and Research

CERN

ACTS Parallelization Meeting

`detray` - Actor Implementation

Joana Niermann[1,2]

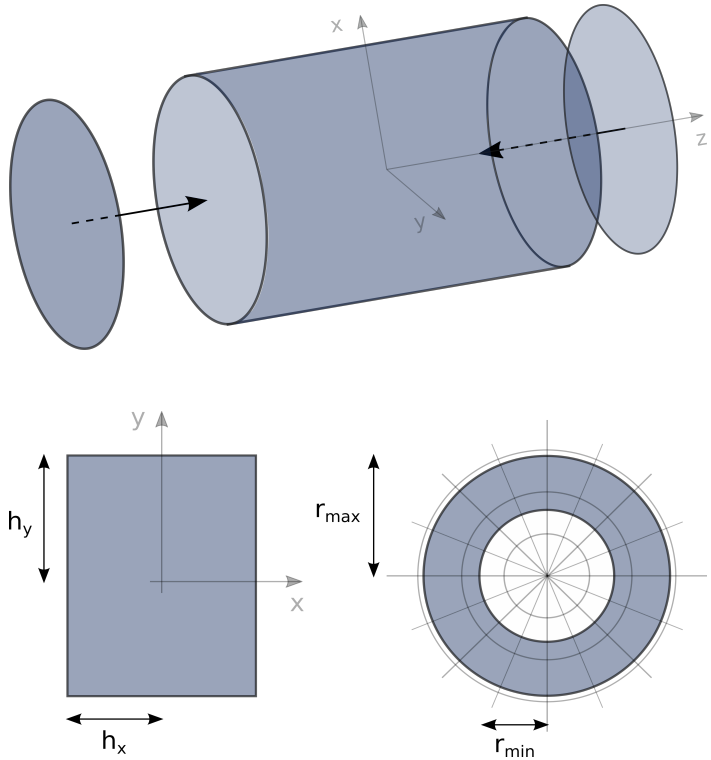
13.05.2022

[1] CERN
[2] II. Physikalisches Institut, Georg-August-Universität Göttingen

The detray Geometry Model

Building Blocks

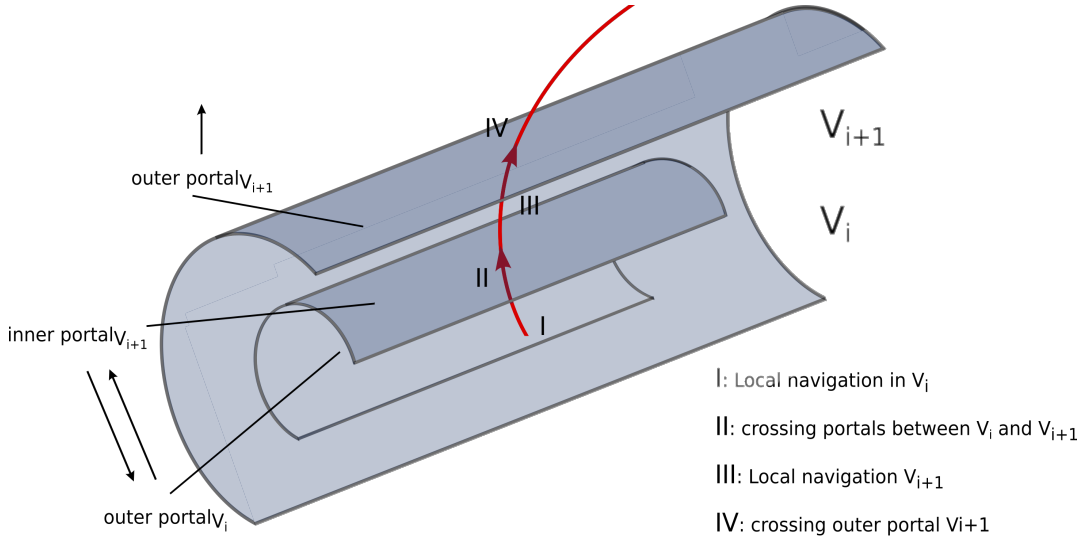
- ***Volumes***: containers for surfaces, defined by their boundary surfaces (portals).
- ***Surfaces***: Placed by transformations and defined by boundary masks.
- ***Masks***: Define the shape types by providing local coordinates and extent of surfaces.
- ***Portals***: Surfaces that tie volumes together through links.



Supported masks: Rectangles, trapezoids, ring/discs, cylinders and an annulus shape (***No runtime polymorphism:***).

Navigation Model

- ***Propagator***: steers the workflow between the stepper and navigator.
- ***Stepper***: Advances the track state through the geometry.
- ***Navigator***: Provides the next candidate surface and its distance.



Volumes in outer layers can contain thousands of surfaces: Implement local navigation.

Navigation Model - Implementation

The Navigator as a data structure . . .

- . . . is a cache around line-surface intersections.
- Different *trust levels* determine how that cache is updated after track state changes.
- ***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.

$\Rightarrow$ A call to the navigator update function restores *full trust*, otherwise aborts the propagation.

Navigation Policy

Navigation policies

- After an actor modifies the track state, determine severity of change.
- Actors can decide for themselves how to reduce the trust level.
- No actor can raise the trust level!

⇒ Navigation policies are actors and can be plugged in e.g. as observing actors.

Example: Current stepper-default-policy

- Uses step constraints to estimate severity of track state change.
- Reduce trust level when a constraint has been hit. Might also resolve type of constraint in the future.

⇒ Also available: `guided_navigation` [and `always_init`].

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);
```

Full example available at detray#248

Summary

What's new

- Some groundwork for adding actors/aborters in `detray`, including the modeling of interdependencies.
- Added flexibility in how to configure behaviour of navigation updates.
- Additional tuning points

Backup