Next generation geometry model for Tracking in ACTS

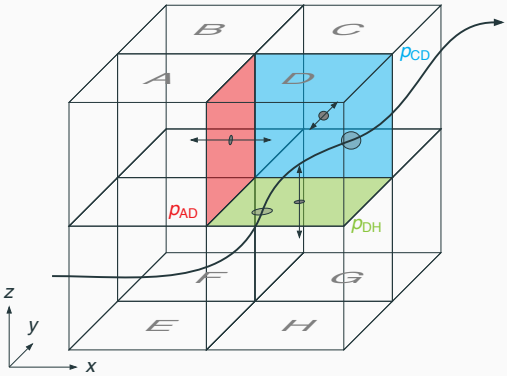
Paul Gessinger, Andreas Salzburger
CERN
2024-10-23 - CHEP 2024 Kraków

Introduction

- ACTS[1] is an experiment-agnostic track reconstruction framework
- Grew out of ATLAS software, has evolved significantly since
- Includes a dedicated tracking geometry model
- Ships with a set of plugins for `DD4hep`, `GDML`, `TGeo`

[1] A Common Tracking Software Project, Comput Softw Big Sci 6, 8 (2022)

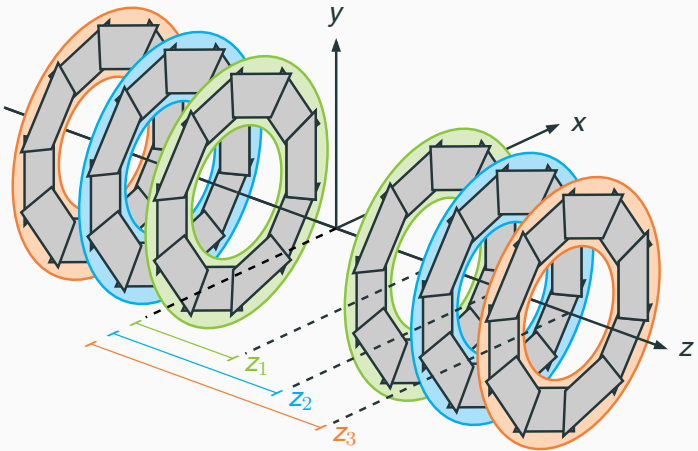
Navigation model

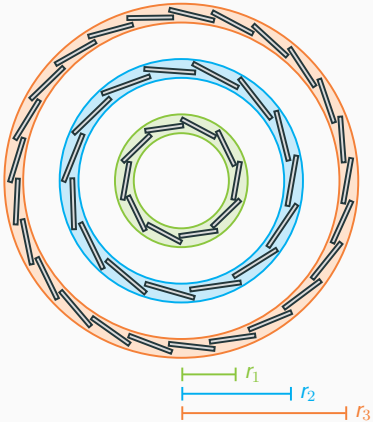


- Dense, fully connected **volumes**
- Volumes are connected by **portals**
- Navigation looks up target volume when reaching portal
- Volume **content** depends on experiment / detector
- Nested volumes can model complex geometries

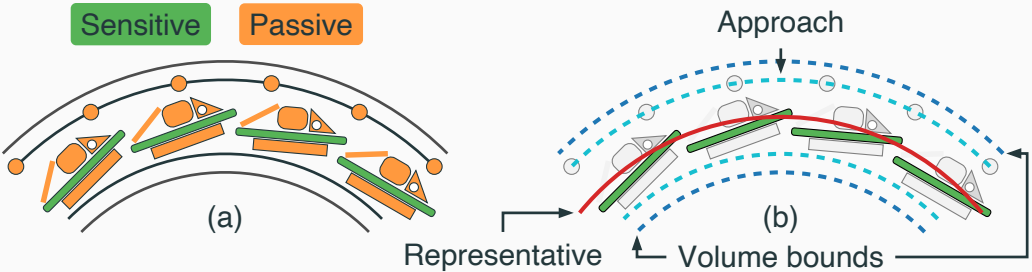
Sensor layers

- Modeled around ”barrel-endcap” style trackers: **layers of sensors**



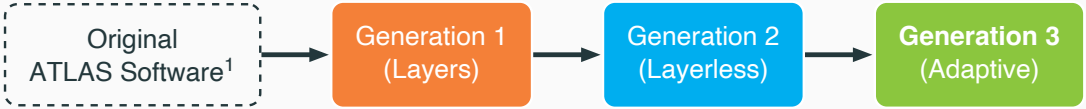


Sensor layers ii



- Assumption of *thin* layers
- Discard detailed structures, keep material carrier surfaces
- Project passive material onto carrier surfaces for material effects

Geometry model evolution



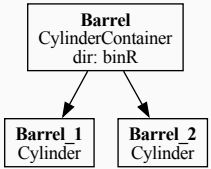
- **Original geometry model**: layers are first-class concept
  - Restricts flexibility for other layouts
  - Adds additional complexity to navigation
- **2nd Generation** experimental geometry model
  - Drops layers in favor of *layer-volumes*
  - Validates approach of registering local navigation inside volumes
  - Demonstrates conversion to `detray` [2] GPU geometry model for use with `traccc` [1]
- **3rd Generation**: combination of both
  - Layer-volumes with local navigation policies
  - Rewrote *geometry construction* from the ground up
  - Extensible, composable and flexible

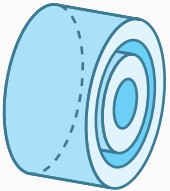
[1] 2404.06335, [2] J. Phys.: Conf. Ser. 2438 012026 [3] B. Yeo's Talk on Wednesday in Track 3

Blueprint tree

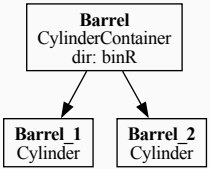
- Central concept for new geometry construction: **blueprint tree**
- Nodes can represent **subsystems, tracker layers**, etc.
- Build phases are propagated down the tree: nodes can take decisions based on their children e.g.
  - Synchronize volume sizes
  - Register volume connections via portals
- **Extensible** with custom nodes

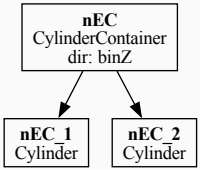
Blueprint tree: example

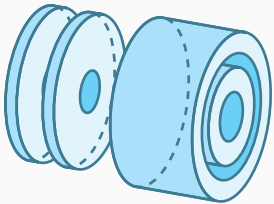




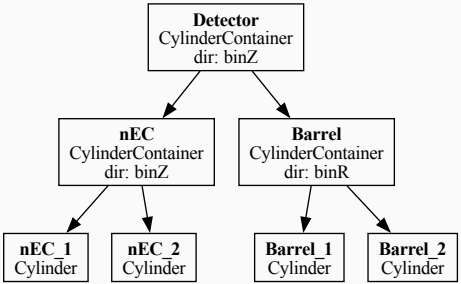
Blueprint tree: example

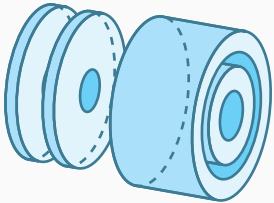




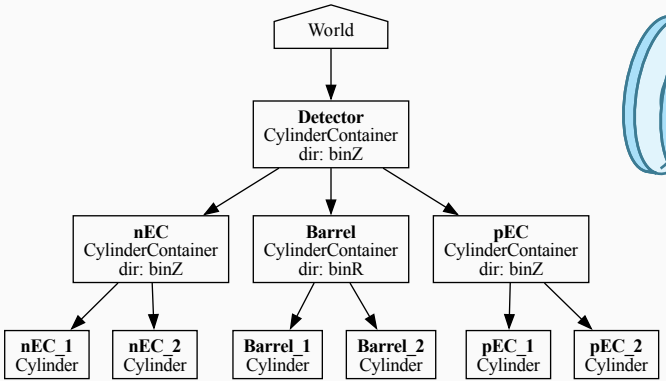


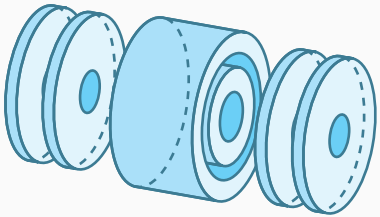
Blueprint tree: example





Blueprint tree: example





Construction phases

1. **Build**: Construct volume representation + compute final sizing

Construction phases

1. **Build**: Construct volume representation + compute final sizing
2. **Connect**: Create and connect portals at volume boundaries

Construction phases

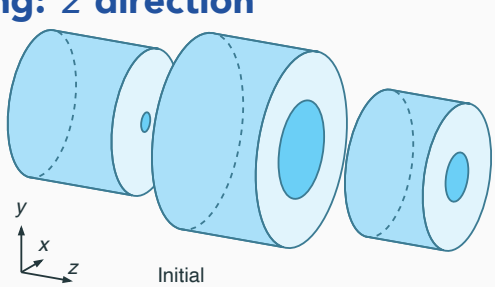
1. **Build**: Construct volume representation + compute final sizing
2. **Connect**: Create and connect portals at volume boundaries
3. **Finalize**: Register portals with volumes + create acceleration structures

Build

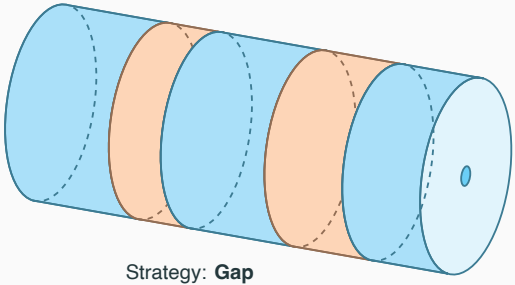
Volume stacks

- Arrangement of volumes along (local) axes
- Cylinder volumes: *z* and *r* direction
- Configurable **attachment strategy** and **resize strategy**
- No shrinking, only expansion!

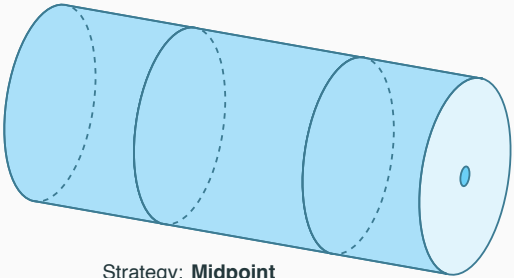
Sizing: z direction



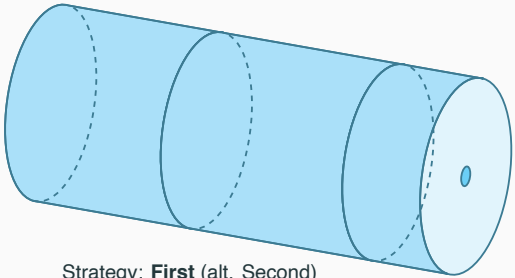
Initial



Strategy: **Gap**

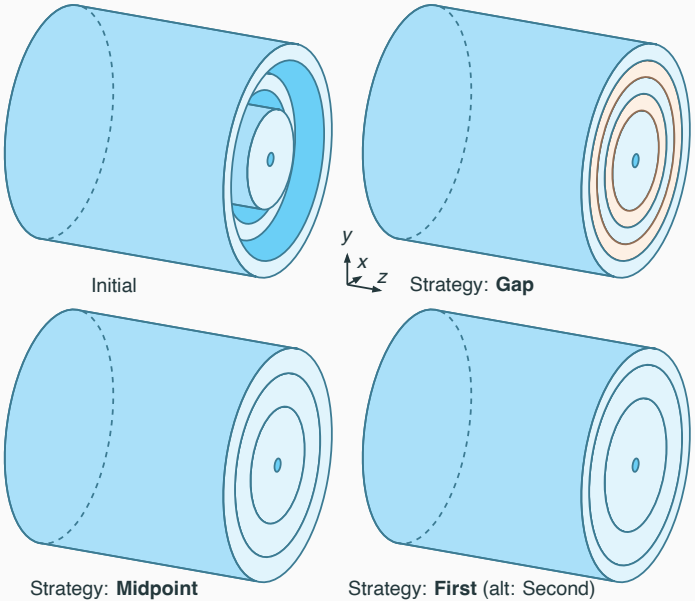


Strategy: **Midpoint**



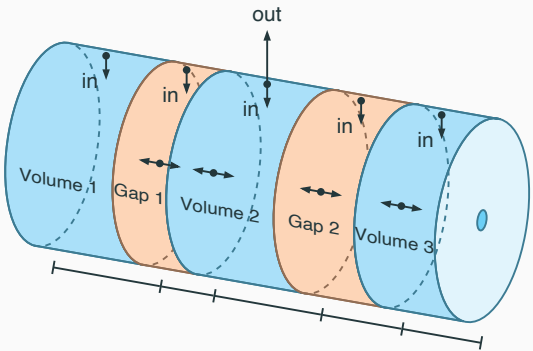
Strategy: **First** (alt. Second)

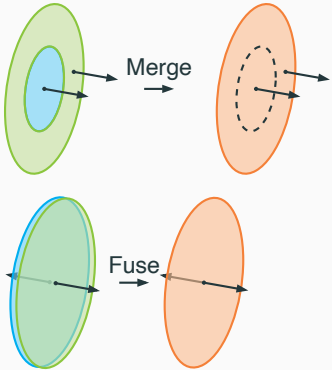
Sizing: *r* direction



Connect

Portal construction and registration

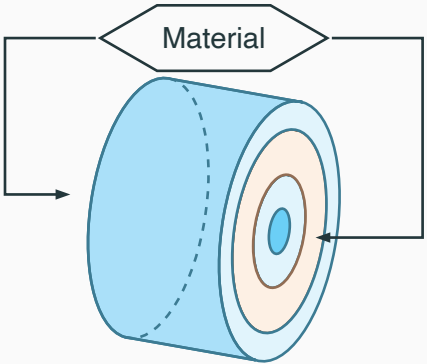


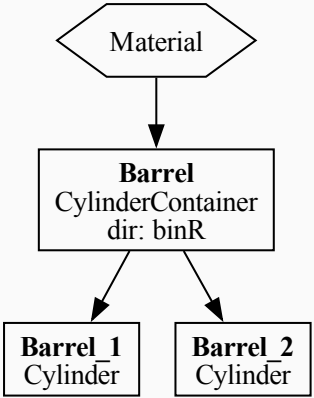


- Portal construction from volume boundaries after sizing
- Nodes can take specific action (e.g. cylinder container)
- Portal accumulation strategy configurable, e.g. suitable for `detray`

Material

- **Material node** can be inserted at any point in the tree
- Configured to mark up surfaces to **carry material**
- Applied to **final portal surfaces**

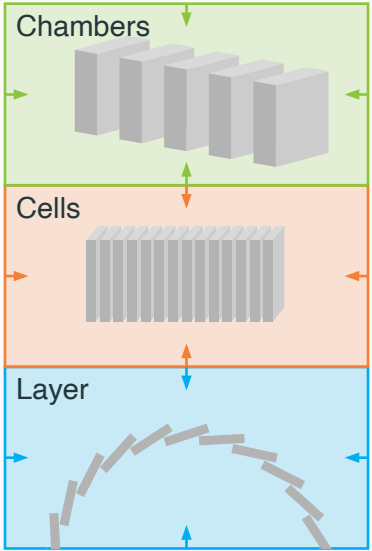




Finalize

Navigation structures

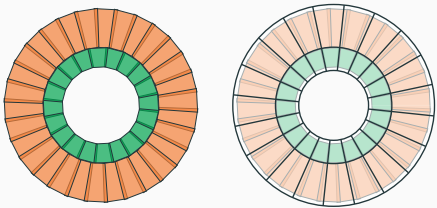
- Original navigation model: **coupled to basic geometry objects**
- Focused on layer-based navigation only
- Now: flexible **local navigation policies** assigned to each volume
- Allows accommodating arbitrary acceleration structures: bin lookup, bounding box searches, etc.



Navigation policies

```
using namespace Acts;
auto factory = NavigationPolicyFactory::make()
  .add<TryAllPortalNavigationPolicy>()
  .add<SurfaceArrayNavigationPolicy>(SurfaceArrayNavigationPolicy::Config{
    .layerType = SurfaceArrayNavigationPolicy::LayerType::Disc,
    .bins = {10, 10}
  });
```

- Composable, extensible, and configurable
- Constructed with **final volumes** (size + content)
- Policies can optimize for local structure, e.g. **model a layer!**

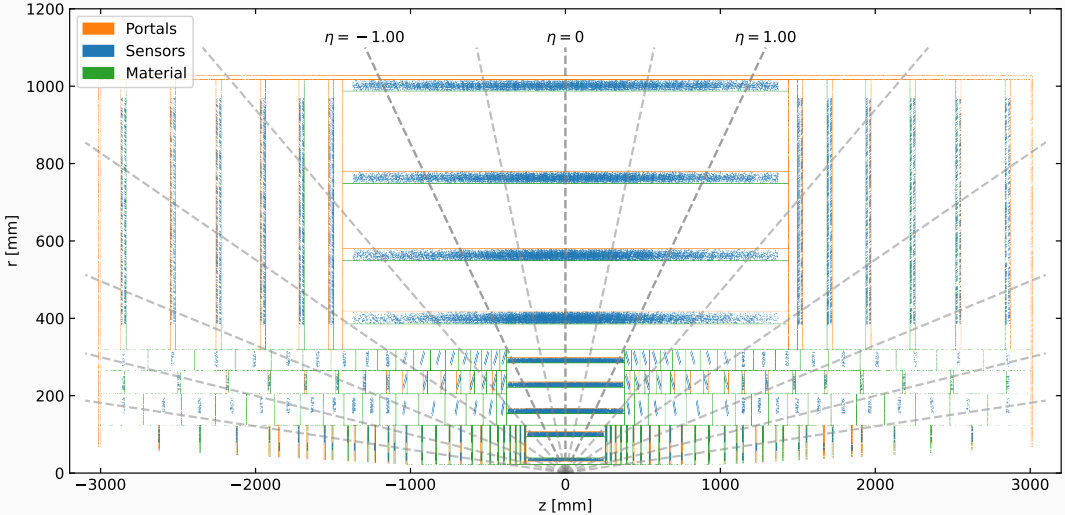


Non-trivial example: `GeoModel` detector

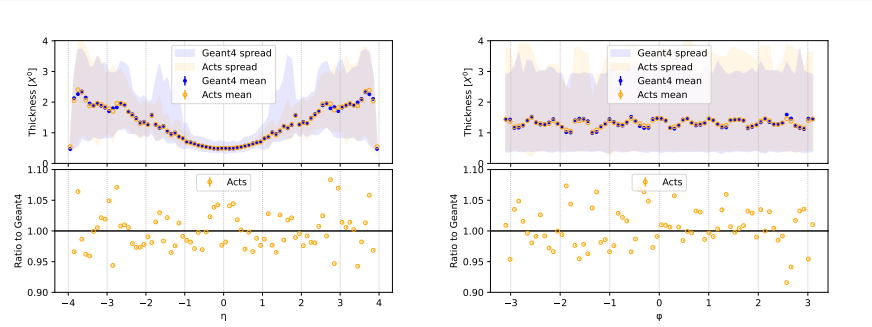
GeoModel

- `GeoModel` is a detector description package
- Convertible to `Geant4` for full simulation
- Verify updated geometry model using non-trivial example implemented in `GeoModel`
- Procedure:
  1. Sort sensitive elements into desired layer grouping
  2. Produce blueprint tree using python code
  3. Introduce material nodes in *optimal* locations
  4. Register surface-grid navigation policies for layers

GeoModel : ray navigation test



Material mapping



- Tested and verified material mapping procedure
- Produces mapping result compatible with expectation prior to detailed optimization

Summary

- Combination of **layerless** geometry model with original classes
- Incorporates **lessons learned** and improvements
- Rewrite of the geometry construction using **blueprint tree**
- Demonstration of `GeoModel` detector **validates concepts**

Next steps

- Update geometry plugins to build geometry via **automatically created blueprint tree**
- Add support for **telescope-like** detectors
- **Adapt numerical integration + navigation** on top of new structure
- Adapt `detray` **geometry conversion** from blueprint (concepts map 1:1)

Backup

Blueprint tree definition

Python

```python
base = acts.Transform3.Identity()
root = acts.RootBlueprintNode(envelope=env(r=[10 * mm, 10 * mm]))
with root.CylinderContainer("Detector", direction=bv.binZ) as det:
  with det.CylinderContainer("Barrel", direction=bv.binR) as barrel:
    for i in range(0, 2):
        r = 25 * mm + i * 50 * mm
        bounds = acts.CylinderVolumeBounds(r, r + 20 * mm, 200 * mm)
        barrel.addStaticVolume(base, bounds, name=f"Barrel_{i+1}")
  with det.CylinderContainer("nEC", direction=bv.binZ) as ec:
    for i in range(0, 2):
        z = -200 * mm - i * 200 * mm
        bounds = acts.CylinderVolumeBounds(100 * mm, 150 * mm, 50 * mm)
        trf = base * acts.Translation3(acts.Vector3(0, 0, z))
        ec.addStaticVolume(trf, bounds, name=f"nEC_{i+1}")
  with det.CylinderContainer("pEC", direction=bv.binZ) as ec:
    for i in range(1, 3):
        z = 200 * mm + i * 200 * mm
        bounds = acts.CylinderVolumeBounds(100 * mm, 150 * mm, 50 * mm)
        trf = base * acts.Translation3(acts.Vector3(0, 0, z))
        ec.addStaticVolume(trf, bounds, name=f"pEC_{i+1}")
```

Equivalent code possible with C++ API

Navigation policies

Python

```
polType = acts.SurfaceArrayNavigationPolicy
layerType = polType.LayerType.Cylinder
factory = (
  acts.NavigationPolicyFactory.make()
    .add(acts.TryAllPortalNavigationPolicy)
    .add(
        polType,
        polType.Config(
            layerType=layerType,
            bins=(10, 10),
        ),
    )
)
```

```
RootBlueprintNode root{{.envelope{{.r = {10_mm, 10_mm}}}}};
root.addCylinderContainer("Detector", binZ, [&](auto& det) {
  det.addCylinderContainer("Barrel", binR, [&](auto& barrel) {
    for (std::size_t i = 0; i < 2; i++) {
      double r = 25_mm + i * 50_mm;
      auto bounds = std::make_shared<CylinderVolumeBounds>(r, r + 20_mm, 200_mm);
      barrel.addStaticVolume(Transform3::Identity(), bounds,
                             "Barrel_" + std::to_string(i + 1));
    }
  });
  det.addCylinderContainer("nEC", binZ, [&](auto& ec) {
    for (std::size_t i = 0; i < 2; i++) {
      auto bounds = std::make_shared<CylinderVolumeBounds>(100_mm, 150_mm, 50_mm);
      Transform3 trf{Translation3{Vector3{0, 0, -200_mm - i * 200_mm}}};
      ec.addStaticVolume(trf, bounds, "nEC_" + std::to_string(i + 1));
    }
  });
  det.addCylinderContainer("pEC", binZ, [&](auto& ec) {
    for (std::size_t i = 1; i < 3; i++) {
      auto bounds = std::make_shared<CylinderVolumeBounds>(100_mm, 150_mm, 50_mm);
      Transform3 trf{Translation3{Vector3{0, 0, 200_mm + i * 200_mm}}};
      ec.addStaticVolume(trf, bounds, "pEC_" + std::to_string(i + 1));
    }
  });
});
```

GeoModel: ray navigation test