Geometry and environment setup in CORSIKA 8

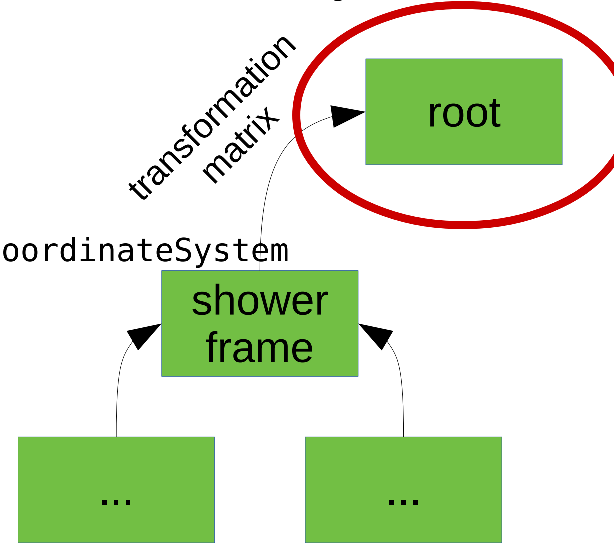
Maximilian Reininghaus

Geometry: preliminaries

- code to be found in *corsika/Framework/Geometry/*
- example of usage in *Documentation/Examples/geometry_example.cc*
- important: distinguish between vectors/points and their components/coordinates w.r.t. coordinate systems (CS)!
- Defining new vectors/points from components/coordinates requires stating a CS!
- After that, you don’t need to care about CS anymore, except if you explicitly want to obtain components/coordinates. (please try to avoid that!)

Geometry: `CoordinateSystem`

coordinate systems



How to define a new CS:

- `cs.translate(QuantityVector<length_d>) → CoordinateSystem`
- `cs.rotate(QuantityVector<arb. dim.> axis, double angle) → CoordinateSystem`
- `cs.RotateToZ(Vector<arb. Dim.> newZAxis) → CoordinateSystem`

  in the new CS, the z-axis is aligned with newZAxis

  → useful for inferfaces with event generators
- `cs.translateAndRotate(QuantityVector<length_d> translation,`

  `QuantityVector<arb. dim.> axis, double angle) → CoordinateSystem`
- If you define a new CS, make sure its lifetime extends beyond the objects which use it! (sorry for the inconvenience...)

Geometry: Points and Vectors

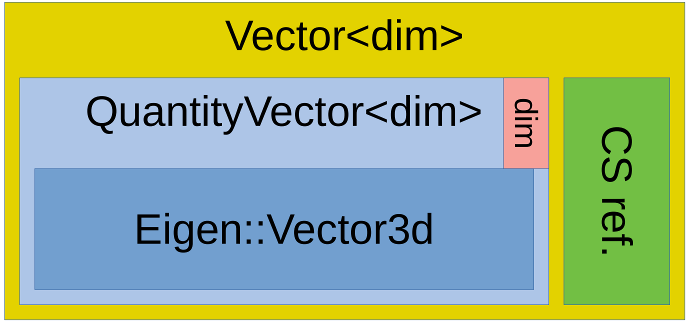
- `QuantityVector<dim>` is just a wrapper of a 3d vector with support for units (dimensions)
- can be printed directly: `std::cout << qVector → (1 2 3) m`
- `Vector<dim>` is a 3d vector
  - definition with components in a given CS:

    ```
    Vector(CoordinateSystem const& cs, Quantity x, Quantity y, Quantity z)
    ```

    e.g. `Vector<speed_d> v(rootCS, 1_m/1_s, 3_km/2_ms, constants::c)`
- `Point` is a point in space, coordinates can only be lengths
  - definition with coordinates in a given CS:

    ```
    Point(CoordinateSystem const&, LengthType,
          LengthType, LengthType)
    ```

    e.g. `Point p(rootCS, 4_m, 5_m, 6_m)`



Geometry: Points and Vectors

- `vector.GetComponents(CoordinateSystem&) → QuantityVector<dim>`
- `point.GetCoordinates(CoordinateSystem&) → QuantityVector<length_d>`
- Arithmetic:
  - `Point – Point → Vector<length_d>`
  - `Point + Vector<length_d> → Point`
- Vectors are invariant under translations, points are not!

Environment

- `Environment` class is templated, template parameter describes interface to which properties are available, e.g. density, composition, refractive index,…
- currently the only option is `environment::IMediumModel` (virtual)
  - `GetMassDensity(Point) → MassDensityType`
  - `IntegratedGrammage(Line, LengthType) → GrammageType`
  - `ArclengthFromGrammage(Line, GrammageType) → LengthType`
  - `GetNuclearComposition() → NuclearComposition`
-
```
NuclearComposition(std::vector<particles::Code>,
                   std::vector<float>)
```

Environment: VolumeTreeNode

- create nodes

  Environment<...>::CreateNode<TShape>(...arguments for TShape…) → unique_ptr of VolumeTreeNode<...>

  e.g. Environment<...>::CreateNode<Sphere>(centerPoint, radius)

- set the medium model of a node

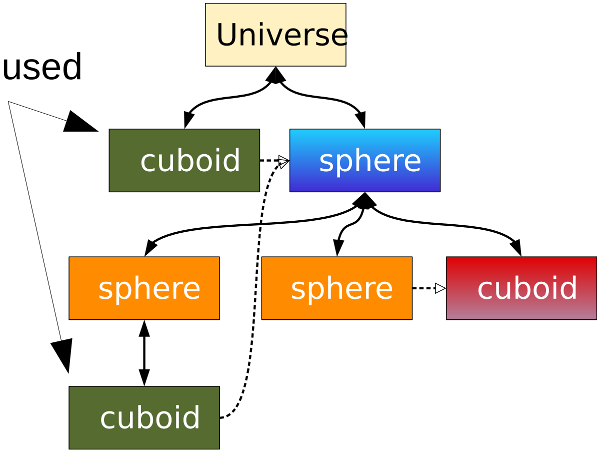
  node->SetModelProperties(...) → shared_ptr, can be reused

- assemble nodes in tree

  outerNode->AddNode(std::move(innerNode))

  innerNode no longer usable!

- top node: environment.GetUniverse()



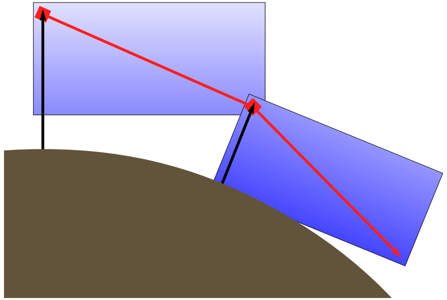
Environment: density models

- most simple model: `HomogeneousModel(MassDensityType, NuclearComposition)`
- analytically treatable: `FlatExponentialModel(Point p0, Vector<dimless> axis, MassDensityType rho0, LengthType lambda, Nucl.Comp.)`

$$\varrho(r) = \varrho_0 \exp\left(\frac{1}{\lambda}(r - p_0) \cdot \vec{a}\right)$$

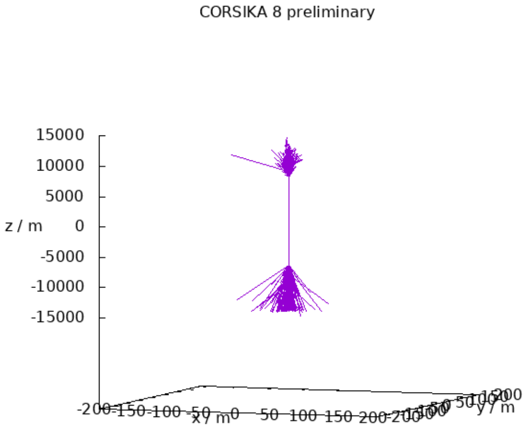
- *sliding planar approximation* for spherical distribution: `SlidingPlanarExponential`

$$\varrho(r) = \varrho_0 \exp\left(\frac{|p_0 - r|}{\lambda}\right)$$



Example

- checkout branch *workshop*
- open *Documentation/Examples/workshop_example.cc*
- *advice: enable floating point exceptions!*


CORSIKA 8 preliminary
z / m
x / m
y / m