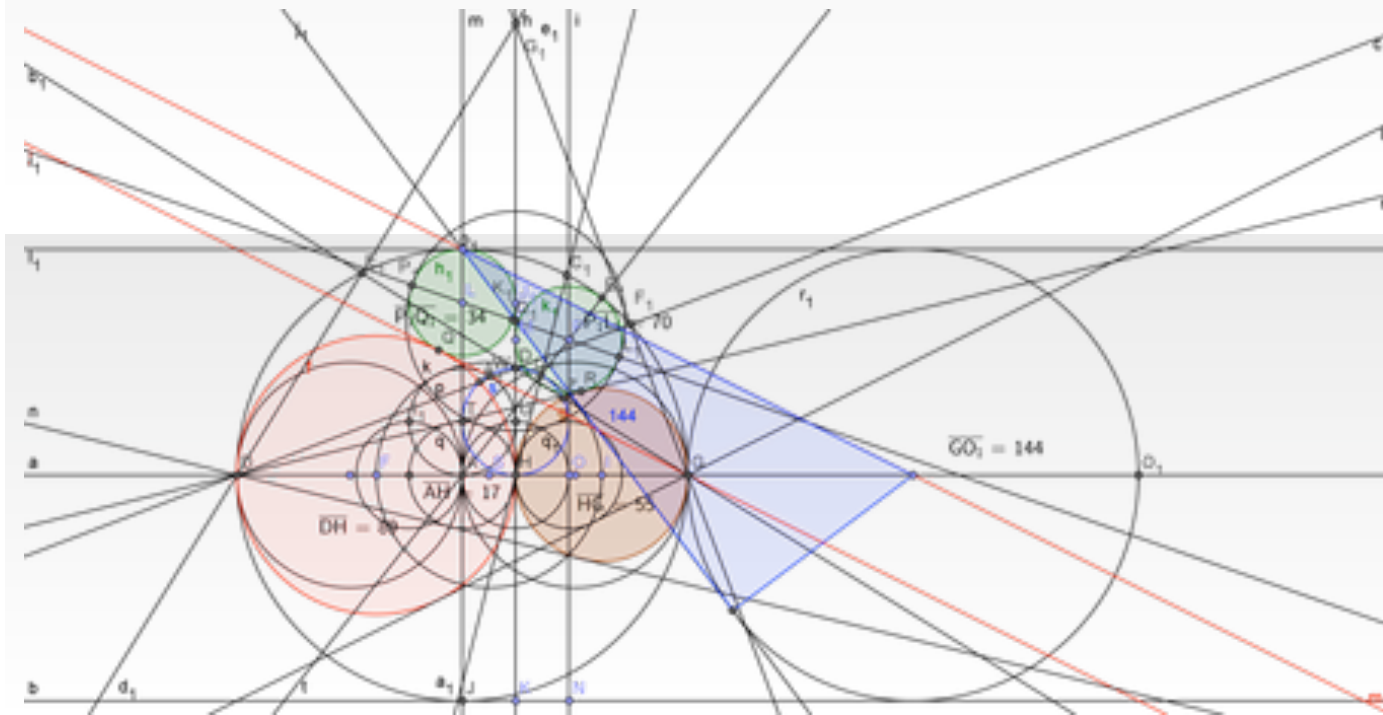


PIERRE
AUGER
OBSERVATORY

The Geometry classes



Lukas Nellen

ICN-UNAM

lukas@nucleares.unam.mx

PAPIIT IN114924

Quick Overview

- Abstract **Affine-Linear** Geometry
- Use components for concrete representations
 - I/O, Plots, ...
- Most Geometry is in Utilities
- Some high-level functionality in Framework
 - CoordinateRegistry

Abstract (coordinate free) geometry

- It is possible to do geometry without referring to coordinates or coordinate systems.

$$\cos(\phi) = \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| |\vec{v}|}$$

- Have to use coordinate system to create object or to extract (most) information.
- Internal representation uses coordinates.

Geometry objects

- Cartesian Geometry objects and transformations
 - Point, Vector, AxialVector
 - Line, Plane
 - CoordinateSystemPtr
 - Operations
 - Sum, difference, size, scalar and cross products, angle
 - Distance, intersection, projection
- Geodesy objects and transformation to/from Cartesian objects
 - UTMPoint
 - Lat/Long/Height
 - ReferenceEllipsoid

Coordinate systems

- Central to the Geometry package
- Needed to establish basis:
 - Create object from components
 - Extract components from object
 - Internal representation
- `CoordinateSystem` objects are always handled through `CoordinateSystemPtr`'s
- The `CoordinateSystemPtr` is a smart pointer which handles ownership and use
 - Keep CS alive as long as it used by some object
 - Free memory automatically when a CS is no longer needed

Coordinate Systems #2

- All coordinate systems are right-handed
 - x-axis points east
 - y-axis points north
 - z-axis points up
- NB: this is also true in coordinate system created for a shower
 - The shower plane is rotated relative to the ground to be orthogonal to the shower axis
 - Z points up, i.e., opposite to the direction of the primary particle

Coordinate Systems #3

- ◎ Coordinate systems are relative to another coordinate system
- ◎ One absolute (root) coordinate system fixed by convention
- ◎ A few coordinate system pre-defined
 - ◎ Earth-Centered, Earth Fixed (ECEF) co-rotating coordinate system,
our **absolute root coordinate system**
 - ◎ Malargüe site
 - ◎ Pampa Amarilla
 - ◎ CLF, XLF

Coordinate Systems #4

- Use factory functions to create coordinate systems
 - `CoordinateSystem::Translation(...)`
 - `CoordinateSystem::Rotation(...)`
 - `AugerCoordinateSystem::Create(...)`
- You never create a `CoordinateSystem` object
- `CoordinateSystems` passed through a `CoordinateSystemPtr`
- Some components of detector or shower provide functions to create special coordinate systems
 - Eyes
 - SD station
 - Shower

Coordinate Systems #5

- Keep coordinate systems as long as you need them
- Cache as needed
- In operations, coordinate systems are compared by object, not by internal components
 - much faster
 - can miss unnecessary transformations
- Example from `sdet::Station`:

```
const
utl::CoordinateSystemPtr Station::GetLocalCoordinateSystem() const {
    if (!fReferenceSystem) {
        fReferenceSystem =
            fwk::LocalCoordinateSystem::Create(GetPosition());
    }
    return fReferenceSystem;
}
```

Points and Vector

- Points, vectors, and axial vectors have many properties in common, but they differ in
 - Valid operations (e.g., you cannot add two points)
 - Transformation properties
- The geometry package uses the strong typing of C++ to detect errors in the use of geometrical objects

Points, Vectors, and Coordinate Systems

- Coordinate of points and vectors can be given and queried in Cartesian, cylindrical, and spherical coordinates.
- Coordinates only make sense together with a coordinate system. Therefore, you have to specify a coordinate system when constructing or querying geometry objects.
- You have to write
`const Point origin(0, 0, 0, CS);`

Multiple return value: boost tuple

- You can obtain the coordinates of a point or vector in a single subroutine call using boost tuples and ties:

```
#include <boost/tuple/tuple.hpp>
Point pt(...);
CoordinateSystemPtr myCoordinateSystem;
double x;
double y;
double z;

...

boost::tie(x, y, z) = pt.GetCoordinates(myCoordinateSystem);
```

- More efficient than using GetX/GetY/GetZ separately

Operations

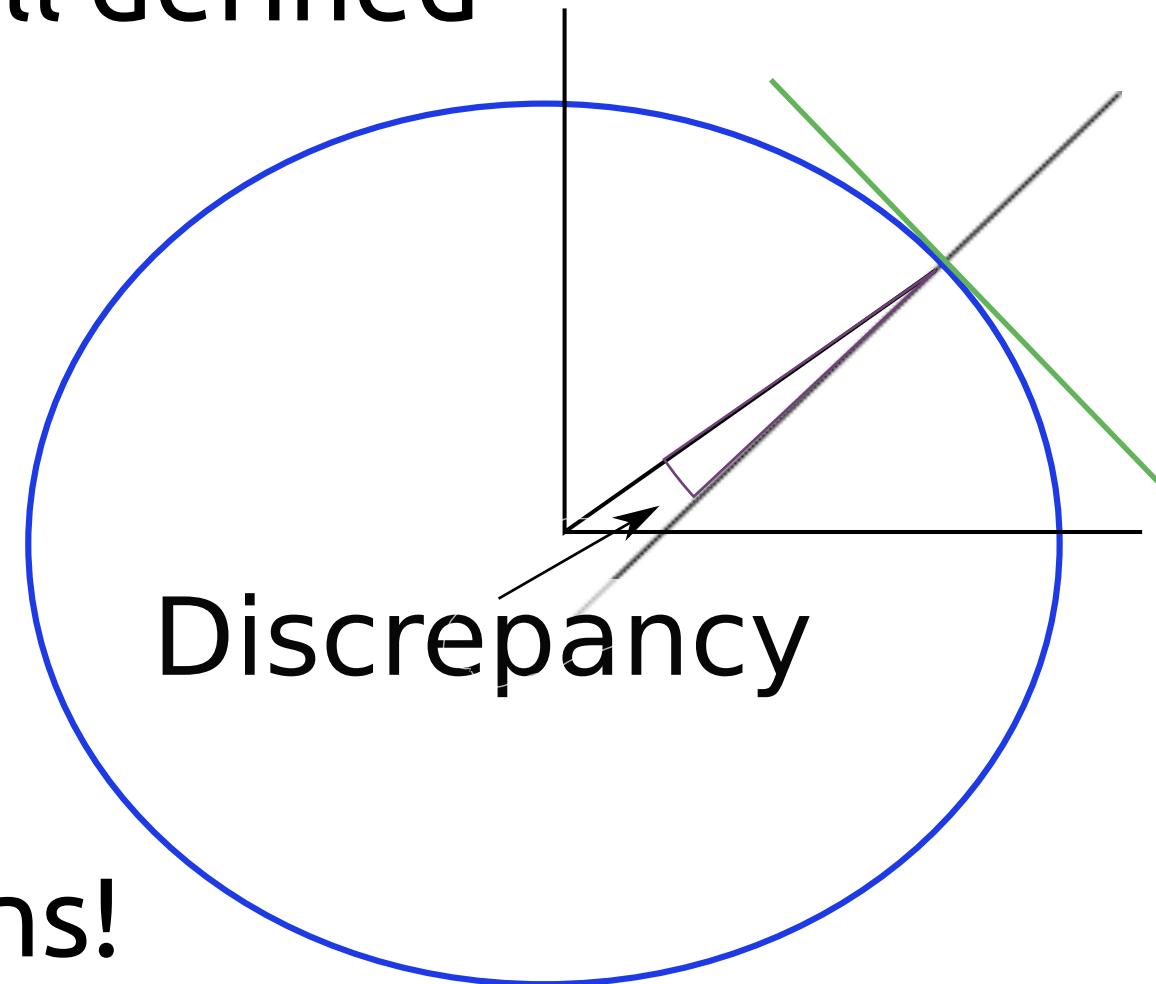
- All operations expected on points and vectors supported
 - Sum, difference, scaling
 - scalar and cross products
 - Magnitude of vector
- Points only support operations in an affine space
 - $\text{Point} - \text{Point} = \text{Vector}$
 - $\text{Point} \pm \text{Vector} = \text{Point}$
- Operations involving planes and lines
 - In `utl/GeometryUtilities.h`
 - Normalized angle
 - Intersection, Distance
 - Perpendicular Plane

Active transformations

- Active transformations can be used to translate and rotate objects
- Transformations require a coordinate system for their definition
- They can be used, e.g., when cloning an object in a different position and orientation like pixels in a camera

Geodetic transformations

- Points can be converted to and from geodetic coordinates:
 - Latitude, longitude, height
 - UTM (northing, easting, height, zone, band)
- These operations require the specification of a reference ellipsoid to be well defined
- NB: In geodesy, the latitude is defined as the angle relative to the local vertical, not the angle to the centre of the earth
- Use the right transformations!



Object for Geodesy

- `ReferenceEllipsoid`
 - provides the transformation to and from latitude/longitude/height
 - has a built-in registry for well-known ellipsoids (currently only wgs84)
- `UTMPoint`
 - provides the transformation to and from UTM coordinates

Local Coordinate Systems

- The Auger conventions define a local coordinate system in a point as
 - the coordinate system that is tangent to the reference ellipsoid in the point
 - x pointing east
 - y pointing north
 - z pointing up
- This functionality is in the Framework, not in Utilities.
 - Needs detector to determine the site
 - Left-over from the Auger-North dreams...
- Remember the caveat for shower coordinates

Technical: Forward declarations

- The geometry makes intense use of templates with traits and policies.
 - Flexible
 - Slows down compilation
- Have to include files with `-fwd` in the name to get forward declarations
- You cannot just declare
`class Point;`
- To get a forward declaration, use
`#include <utl/Point-fwd.h>`

Technical: Testing

- The Geometry package evolved together with a complete set of unit tests
- Some functions in the interface of a class exist only to support testing, not for normal use. They are marked as such in the Doxygen documentation.
Do not use them – they might disappear in the future
- The unit test exercise and therefore document the interface of the Geometry package. Consider the test part of the documentation of the package.

Suggested exercises

- Set the coordinates of a point in one coordinate system and extract them in another.
- Verify that the sum of angles of a triangle is π , using the `CosAngle(v, v)` function.
- Verify that the earth is curved:
- define the vector $(0, 0, 1)$ once in the Los Leones local coordinate systems and then in Coihueco. Extract the angle between the vectors.