The Geometry Package of the Pierre Auger Offline software

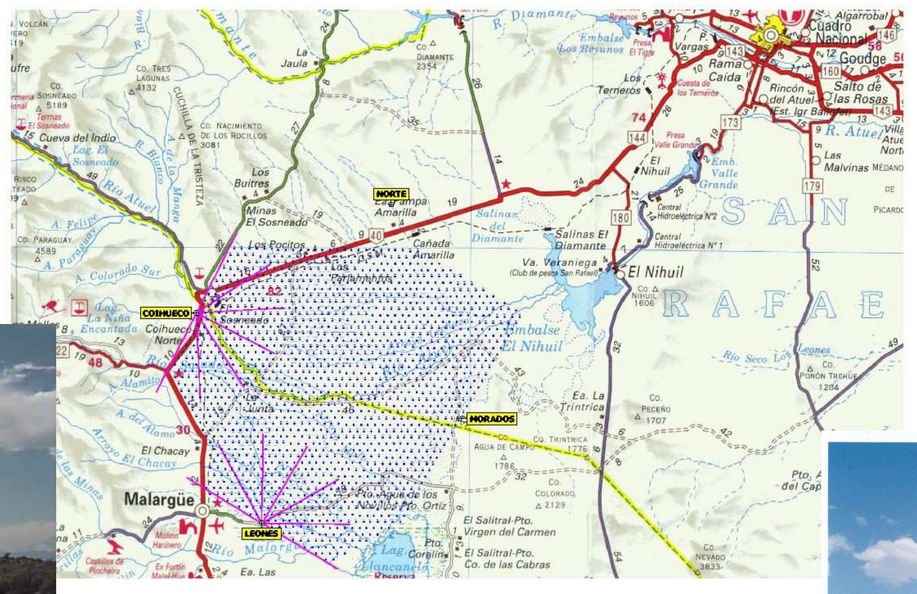
Lukas Nellen (I. DE CIENCIAS NUCLEARES, UNAM)
Stefano Argiro (University of Torino)
Thomas Paul (Northeastern University)
Troy Porter (Louisiana State University)
Luis Prado Jr (State University of Campinas)

Why a new geometry package?

The Pierre Auger Observatory consists of two, semi-autonomous detection systems for cosmic ray air showers: the surface array and the fluorescence detectors.
The Observatory covers an area of 3000 km^2, corresponding to a linear extend of $\approx$ 60 km.

As a consequence, one has to be careful in the treatment of geometrical quantities:
- The earth's curvature is non-negligible (0.5 deg)
- Each component of the detector has a different, natural and preferred coordinate system.
- Surveying work uses geodesic coordinate systems, like the Universal Transverse Mercator (UTM) grid, which are unfamiliar to physicists.

The map of the southern site in Malargüe, Argentina



A surface detector station

Data Integration requires agreement about geometry

A fluorescence detector building

Goals of the geometry package

Provide an abstract, coordinate system independent geometry library without compromising usability.
Handle geodesic and astronomical coordinate transformations.

Why abstract geometry?

Traditionally, one relies on conventions for dealing with coordinate systems in scientific code, at least in the interface between different software modules.

Problems of relying on conventions:
- Conventions can be violated, ignored, or changed without updating all the code
- Imposing conventions globally can lead to the use of non-optimal coordinate systems
- Requiring conventions in the interface only can introduce two coordinate transformations on module boundaries: Internal$_1$ to convention to internal$_2$

In an abstract geometry, coordinates are hidden in the private part of the object. To achieve coordinate-system independence, one has to
- Carry coordinate system information in every object
- Compare and, if necessary transform, coordinates before performing operations.

The price of coordinate system independence is that the user has to specify a coordinate system for
- Setting coordinates
- Retrieving coordinates

The advantage of coordinate system independence is the ease with which one can combine information from different parts of the experiment without worrying about convention. It encourages the use of specialised, optimised coordinate systems, leaving transformations to the geometry package. The consequence is cleaner user code. Arbitrarily complicated chains of transformations can be set up and tracked transparently, thereby removing a common source of errors.

The Geometry package is:

Part of the offline software of the Pierre Auger Observatory

Implemented in C++

Split in a lower-level, generic part
higher-level, Auger specific part

The lower level part provides classes for:
- Coordinate Systems
- Vectors, Points, Axial Vectors
- Active transformations

- UTM coordinates
- Reference Ellipsoids (parametrisation of the shape of the Earth)

The lower level part also implements standard operations:
- Sum, difference, scaling
- vector size, point distance, angle
- Object creation in Cartesian, cylindrical, and polar coordinates

The higher level part implements Auger specific functions and conventions:
- Factories for local coordinate systems
- Registry of well-known coordinate systems

Contact: Lukas Nellen, ICN-UNAM, Mexico; e-mail: lukas@nuclecu.unam.mx

The Geometry Package of the Pierre Auger Offline software (2)

PIERRE AUGER OBSERVATORY

Some implementation details

Every object contains a (private) set of coordinates and a reference to the coordinate system used to define them.

Coordinate systems are shared, using a (reference counted) smart pointer **`CoordinateSystemPtr`** to guarantee
- that a coordinate system is kept alive as long as some object uses it
- that the coordinate system is deleted once nobody uses it.

Coordinate systems are defined relative to other coordinate systems. The transformation between two system results in a chain of transformations. When transforming, such a chain is evaluated only once and cached, that way reducing the overhead when transforming a large number of objects to a given coordinate system.

When (not) to transform coordinates

The internal representation of all objects participating in an operation has to be in the same coordinate system to be able to perform the operation. If necessary, some objects have their internal representation transformed.

Guarantee: No transformation will occur if all objects are already in the same coordinate system.

Optimisation help: the user can manually force the transformation of the representation of an object into a given coordinate system to prevent unwanted, uncontrolled coordinate transformations later.

```
Particle*
CorsikaShowerFileParticleIterator::GetOneParticle()
{
  for (;;) {
    const Block::ParticleData* corsikaParticle =
GetOneParticleRecord();

    // end of particle list
    if (!corsikaParticle) return 0;

    int particleId =
      Corsika::CorsikaToPDG(int(corsikaParticle-
>fDescription/1000));

    // skip unknown particles...
    if (particleId == Particle::eUndefined) continue;

    fCurrentParticle =
      Particle(particleId,
               Particle::eShower,
               Point(corsikaParticle->fX*cm,
                     corsikaParticle->fY*cm,
                     0.*cm,
                     GetCorsikaCoordinateSystem()),
               Vector(corsikaParticle->fPx*GeV,
                      corsikaParticle->fPy*GeV,
                      -corsikaParticle->fPz*GeV,
                      GetCorsikaCoordinateSystem()),
               TimeInterval(corsikaParticle->fTorZ*ns -
fTimeOffset),
               corsikaParticle->fWeight);
    return &fCurrentParticle;
  } // for (;;)
}
```

Routine to process information from an external air shower simulator. Note the use of a special Corsika coordinate system, set up according to the conventions of Corsika, which allows us to create points and vectors directly from the coordinates stored by Corsika.

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

...

boost::tie(x, y, x) = pt.GetCoordinatesmyCoordinateSystem);
```

Using BOOST tuples and ties, we can retrieve all three coordinates of a point in a single subroutine call.

Geodesy

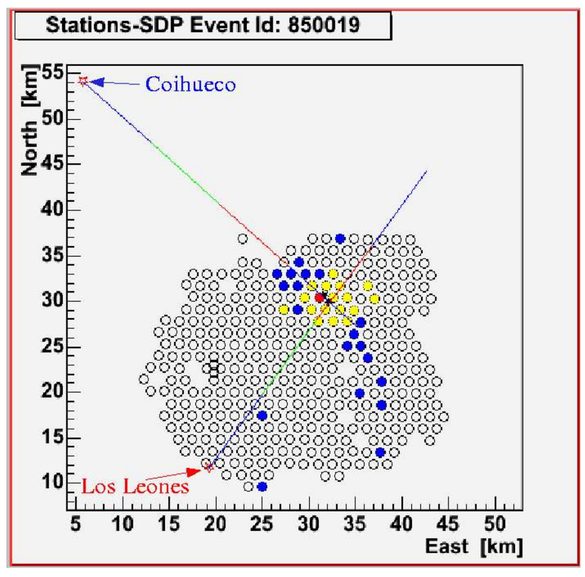
Surveying data, e.g., the exact positions of all detectors, is provided in Universal Transverse Mercator (UTM) coordinates. To avoid having these curved coordinates in physics code, transformations between Cartesian and (curved) Geodesic coordinates. The information about the ellipsoid used to approximate the shape of the Earth is provided as another class

Experience with the geometry package

The geometry package is used throughout the offline software of the Auger Observatory. Combining data has proven very simple.

The strong typing of C++ helps to detect illegal operations, e.g., the summing of two points.

The user feedback has been very positive. We have not indications of any performance problems induced by the abstract geometry.



Stereo reconstruction of an event seen from two fluorescence detectors. The shower detector planes are determined locally using coordinates optimised for one detector station. The geometry package makes combining the information easy.

Testing and QA:

The geometry package is developed with a comprehensive test of unit tests. They turned out to be extremely helpful for delivering a high quality package.

The tests are part of the test suite for the Auger Offline software. They can be run stand-alone during the development and maintenance of the geometry package

Next steps:

- Extend the package to include
  Lines, planes, ...
  astronomical coordinates

- Full benchmarking and profiling

The geometry package uses two external libraries:
- BOOST for smart pointers and tuples
- CLHEP for the underlying implementation of coordinate-system dependent geometry.