A platform for experiments with geodetic software

Thomas Knudsen, SDFI, Copenhagen, Denmark, thokn@sdfi.dk

Rationale

Rust Geodesy (RG) is a platform for experiments with geodetic software, originally intended for experiments with PROJ data flow models.

RG implements an overall architecture much like PROJ's, but by a radical re-design, a much reduced selection of non-essential functionality, and the use of an implementation language, Rust, wonderfully fit for the purpose, it has been possible to keep the implementation limited to less than 5000 lines of code (compared to PROJ's 400000).

Hence, it is much easier to test alternative data flow models and architectural/conceptual abstractions in RG than in PROJ (hear my talk in session 5, Wednes-day for an example of why the latter is essential).

RG is not intended as a replacement for PROJ, but as a platform for experiments, where ideas for future PROJ development can be tested and iterated on in a compact code environment, where new conceptual ideas may have a better chance of staying clear cut, and hence provide a blueprint of inspiration for potential reimplementation in PROJ's much larger and somewhat more hostile code landscape.

Why Rust?

The PROJ data flow architecture is in bad need of a redesign. But changing the data flow architecture is equivalent to changing the entire system, so the obvious solution is to build a new data flow scaffolding, then porting the existing functional blocks to that scaffolding.

After two unsuccesful attempts in C, and two in C++ (the two languages already used in PROJ), I decided to take a radical detour into a totally different language. And Rust seemed to fit the bill: Very different than the C family, but also sufficiently alike, and with the added bonus of being type safe, having good traction, very helpful compiler warnings, and an extremely helpful user community. The result was Rust Geodesy.

Viewing RG as "another PROJ" will lead to bad disappointment: RG is a platform for experiments, not an attempt at something operational.

You may, however, through RG catch a weak mirage of a potential shape of jazz to come for the PROJ internal dataflow.

Essentials

RG exposes a rich set of geometric geodesy primitives, including geodesics, directly at the API level. Also:

- Macros and user defined operators fully supported at the API and command line level
- Rationalized pipeline syntax: Higher clarity through less verbosity
- Clear and explicit syntax for coordinate order and unit adaptation through the `adapt` operator
- Strict four dimensional data flow architecture, for simple extensibility and maintenance

The actual transformation functionality of RG is, however, minimal: At time of writing, it includes just a few low level operations, including:

- The three, six, seven, and fourteen-parameter versions of the Helmert transformation
- Horizontal and vertical grid shift operations
- Helmert's companion, the cartesian/geographic coordinate conversion
- The full and abridged versions of the Molodensky transformation
- Three widely used conformal projections: The Mercator, the Transverse Mercator, and the Lambert Conformal Conic projection
- The `adapt` operator, which mediates between various conventions for coordinate units and order

While this is sufficient to test the architecture, and while supporting the most important transformation primitives and three of the most used map projections, it is a far cry from PROJ's enormous gamut of supported map projections (which, however, is partially supported through a bridge to the proj projection program).

So fundamentally, RG is a geodesy, rather than cartography library. And while PROJ benefits from four decades of reality hardening, RG, being a platform for experiments, does not even consider development in the direction of operational robustness. RG is however, from the ground up built for built for multithreaded and parallel execution, hence streamlining a large class of functionality, that has been retrofitted onto PROJ.

Examples

1. Simplified pipeline syntax

PROJ:

```
proj=pipeline
     step proj=cart ellps=intl
     step proj=helmert x=-87 y=-96 z=-120
     step proj=cart inv ellps=GRS80
```

RG:

```
cart ellps=intl | helmert x=-87 y=-96 z=-120 | cart inv
```

2. Coding in Rust

```
fn main() {
   // [0] Conventional shorthand for accessing the major functionality
   use geodesy::preamble::*;

   // [1] Build some context
   let mut ctx = Minimal::default();

   // [2] Obtain a handle to the utm-operator
   let utm32 = ctx.op("utm zone=32").unwrap();

   // [3] Coordinates of some Scandinavian capitals
   let copenhagen = Coord::geo(55., 12., 0., 0.);
   let stockholm  = Coord::geo(59., 18., 0., 0.);

   // [4] Put the coordinates into an array
   let mut data = [copenhagen, stockholm];

   // [5] Then do the forward conversion, i.e. geo -> utm
   ctx.apply(utm32, Fwd, &mut data);
   println!({:?}, data);

   // [6] And go back, i.e. utm -> geo
   ctx.apply(utm32, Inv, &mut data);
   println!({:?}, data);
}
```