# TURTLE: a C library for an optimistic stepping through a topography

*V. Niess*[1]

IRN Terascale

*21th May 2019*

Annecy-le-vieux, France

(pdf) (html)

[1] Université Clermont Auvergne, CNRS/IN2P3, LPC, F-63000 Clermont-Ferrand, France

# The problematic

*Render a detailed topography (~**1-10 m**) over large scales (~**10-100 km**), for the navigation of Monte Carlo particles with any interaction length.*

## Some use cases

- The *backward* Monte Carlo transport of atmospheric μ, e.g. for muography computations.

  > A single μ can have an energy varying from MeV to TeV.

- The coupled $\nu_\tau - \tau$ transport, for $\nu$-astronomy, e.g. for the GRAND experiment.

> In both cases, efficiently navigating through large scale topography data is a key issue for the Monte Carlo performances.

A dedicated library was developped for this purpose: TURTLE

# Topography data

Let us consider Topography data as a *regular* 2D grid of the ground *elevation*, stored over 2 bytes (`int16_t`) per node.

> *Regular* in geographic coordinates, **not** in the ~~lab frame~~.

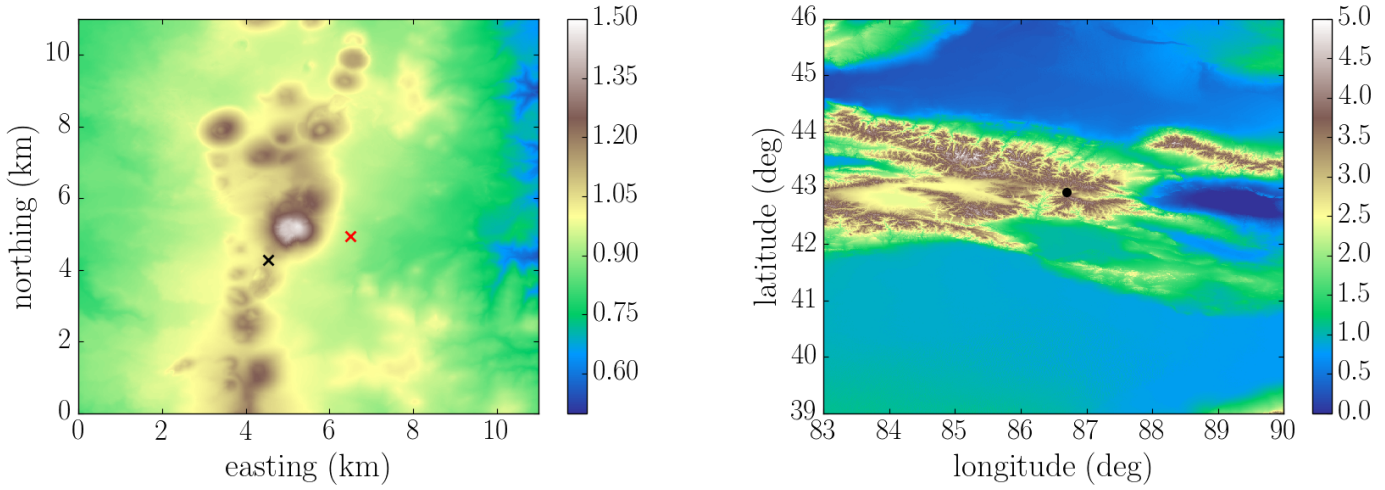
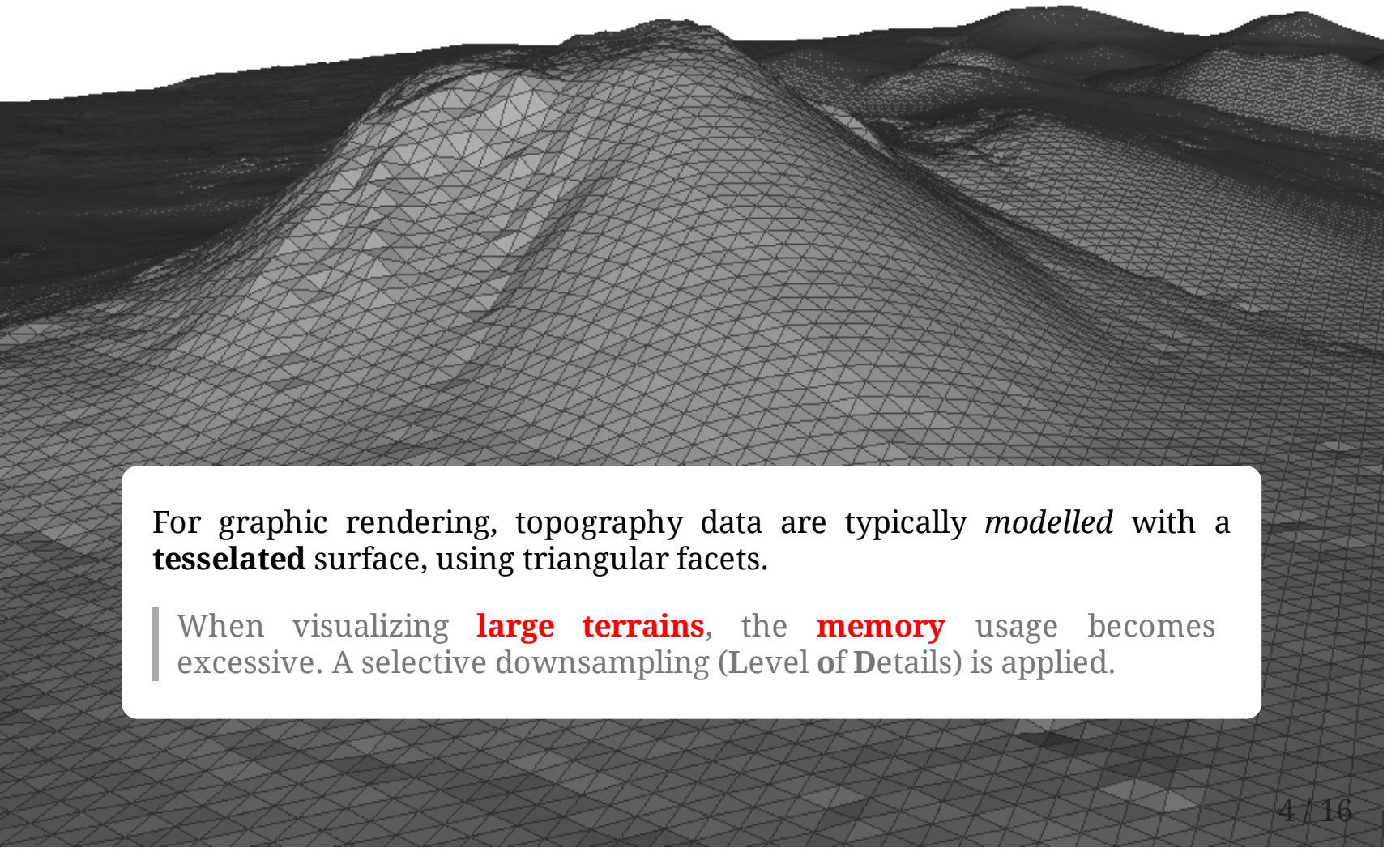



*Figure 1: examples of topography data. Left: LIDAR of the Chaîne des Puys area, France (2.5 m resolution, **40 MB**). Right: SRTMGL1 (v3) tiles of the Tian Shan mountains, China (30 m resolution, **1 GB**).*

# Terrain modelling



For graphic rendering, topography data are typically *modelled* with a **tesselated** surface, using triangular facets.

> When visualizing **large terrains**, the **memory** usage becomes excessive. A selective downsampling (**L**evel **o**f **D**etails) is applied.

# The ray tracing problem

- A Monte Carlo *geometry* is represented by a hierarchy of volumes and/or surfaces.

> Navigating through the geometry requires **intersecting** with a line (all) *the bounding surfaces* of the current volume, and of its contained children. This is a **ray tracing** problem.

- When the number of surfaces is large, e.g. in the case of a terrain, a *brute force* linear scan is inefficient CPU wise: $\mathcal{O}(N)$.

> **Optimisation** techniques are applied. For example a Bounding Volume Hierarchy (BVH), consisting in sorting the volumes as a binary tree according to bounding boxes: $\mathcal{O}(\ln N)$.

> Such techniques however significantly increase the memory usage, w.r.t. to the topography data, which is problematic for large terrains (~$10^9$ nodes, or more).

# But particles ain't no rays

In a HEP Monte Carlo, particles **don't** follow *straight lines*, e.g. they *scatter*. This is modelled by discrete straight steps where the particle **direction changes**.

> If the number of MC steps is large, tracing through the full terrain at each step is inefficient.

*Figure 2: illustrative example of a MC trajectory. Each dot corresponds to the end (start) of a MC step.*

# Divide and rule

A solution, used e.g. in medical Physics, is to pave the terrain with a Polyhedral mesh, i.e. with sub-volumes. For non interacting particles, it allows to traverse the terrain in $\mathcal{O}(\sqrt{N})$ steps.

> A simplified local geometry can be used. Inside a polyhedron, the ray tracing problem is limited to a few neighbouring nodes.

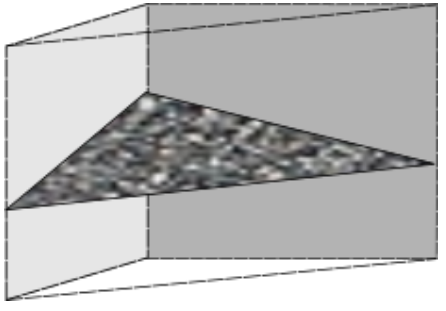


*Figure 3: example of tetrahedral mesh (prisms) built on a terrain Tessellation. The original triangular facet is the textured one.*

# The *optimistic* ray tracing algorithm

- An **alternative** approach is to proceed by **trials and errors**, starting from an **initial guess** of the distance to the topography, along the flight direction.
- A simple **initial guess** is provided by the vertical distance to the ground, $s_g = |z_0 - z_g|$.
- This leads to the following *optimistic algorithm*:

1. Perform a tentative step of length $\max(\alpha|z_0 - z_g|, s_{min})$, along the flight direction.
2. If the end volume differs from the start one, draw a line between both positions and lookup the topography crossing point *in between*, using a binary search.

> $\alpha$ and $s_{min}$ are **tunning parameters**, to fit depending on the terrain slopes and the desired accuracy.

# Benchmark: tracing the rock thickness

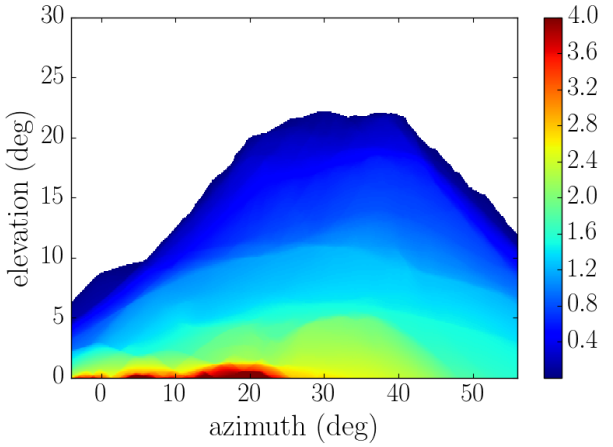
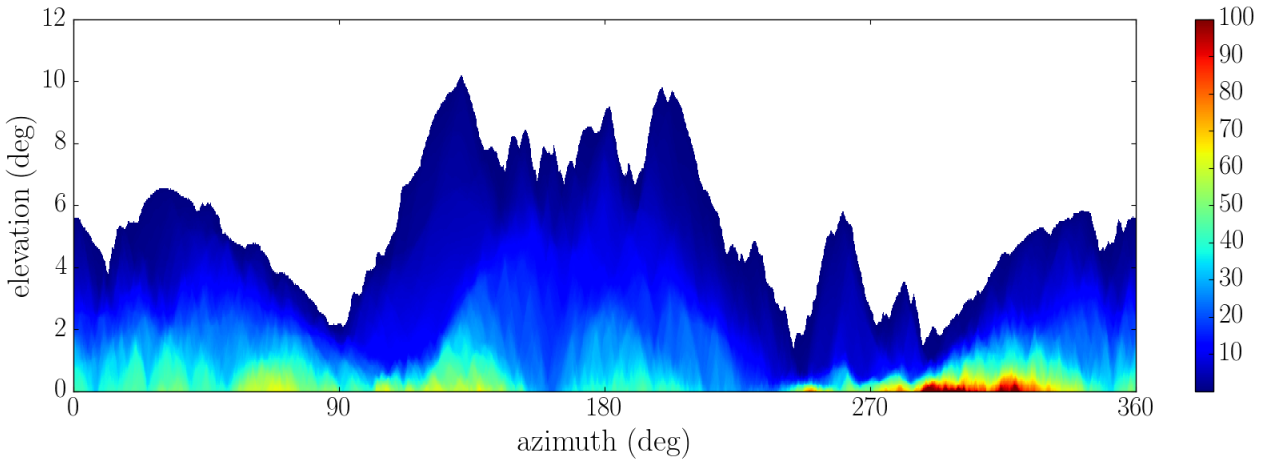



*Figure 4: map of the rock thickness (km) along various lines of sight.* ***Left****: Col de Ceyssat view (France).* ***Bottom****: Ulastai view (China).*



# Accuracy of the *optimistic* tracing

Using the *optimistic* tracing, the average error (**7-9 µm**) on the rock thickness is well below the elevation accuracy of topography data (**~10 cm**).

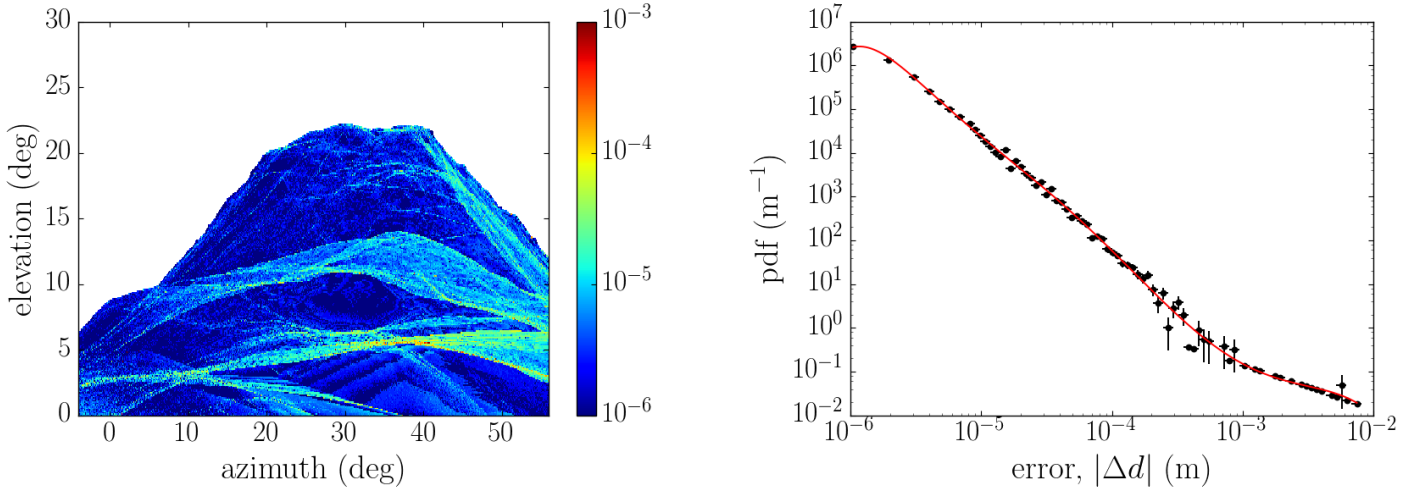



*Figure 5: error on the rock thickness (m) for the Col de Ceyssat view, using the **tuned** optimistic algorithm. Left: map of the error. Right: distribution over all lines of sight.*

# Impact of the terrain modelling

The terrain modelling can induce large differences on the rock thickness, of the order of the spacing between grid nodes.

> Using triangular facets or a bilinear model, the average difference on the rock thickness is of **0.1 m** (**1 m**) for the Col de Ceyssat (Ulastai) view.

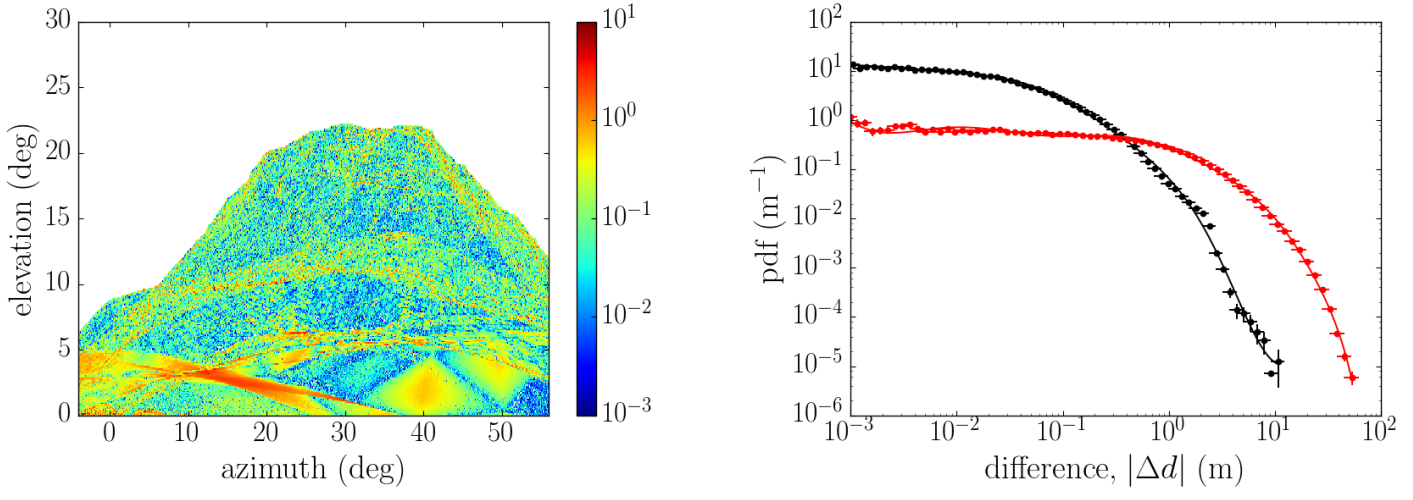



*Figure 6: differences on the rock thickness (m) for the Col de Ceyssat view, using triangular facets or a bilinear model. Left: map of the difference. Right: distribution over all lines of sight.*

# The TURTLE library

- Open source (**C99**) available from GitHub under **GNU LGPL-3.0** license.

> Depends only on the C89 standard library and (optionally) on `libpng` and `libtiff` for specific data formats.

- **O**bject **O**riented API allowing to manipulate *some* topography maps (`turtle_map`) as well as globals elevation models (`turtle_stack`).
- The `turtle_ecef` (`turtle_projection`) functions provide coordinates transforms between *geocentric* and geodetic (cartographic) coordinates.
- The `turtle_stepper` object allows to navigate through topography layers using *geocentric* coordinates. The *optimistic* algorithm is used under the hood.

> **Zero memory cost** : the geometry is built on the fly from the initial (`int16_t`) elevation data, using a bilinear interpolation.

*arXiv:1904.03435*, *submitted to CPC*

# TURTLE performances: ray tracing

> Using the *optimistic* tracing, the cpu time for computing the rock thickness depends strongly on the direction of observation. The average value over all lines if sight is of ~**280 µs**.

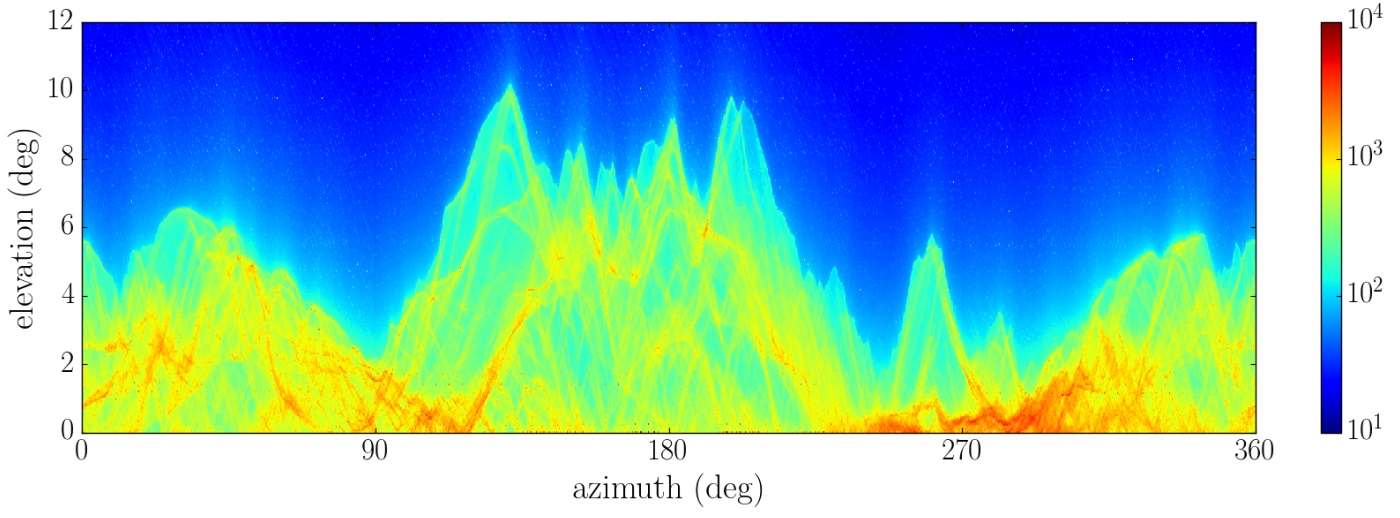



*Figure 7: map of the cpu time (µs) required for computing the rock thickness, as seen from Ulastai, with the TURTLE library.*

# Comparison of performances: ray tracing

> The ray tracing performances of TURTLE have been compared to a tetrahedral grid approach and a BVH one, using the CGAL library.

> For a pure ray tracing problem, the BVH should perform the best, provided that there is enough memory (**256 GB** for $10^9$ nodes).

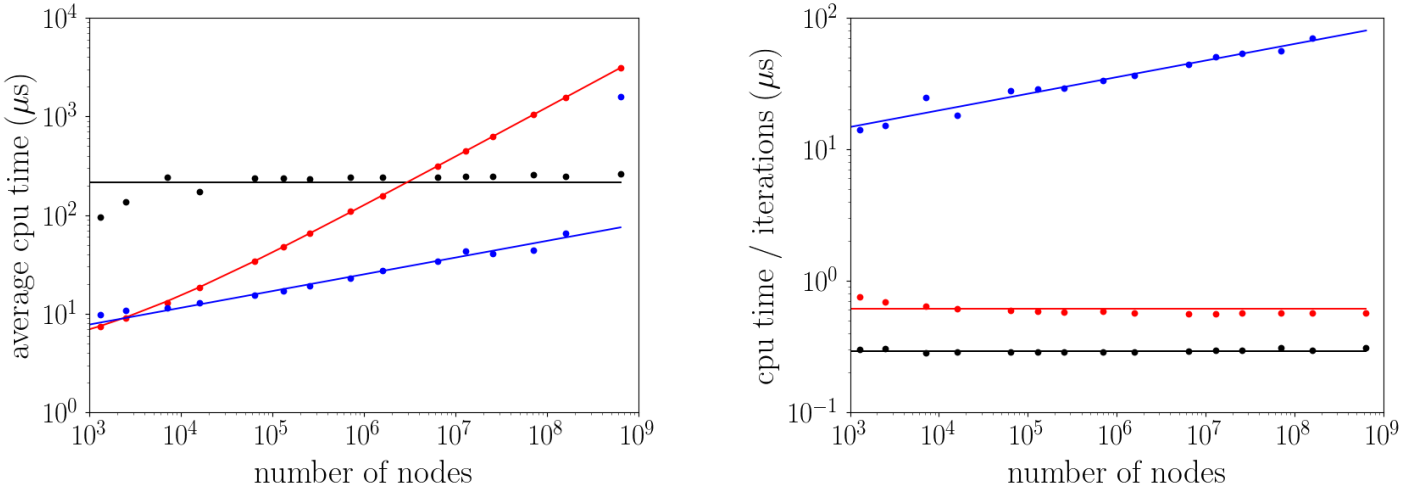



*Figure 8: cpu time (μs) as function of the number of topography data nodes for TURTLE, the tetrahedral grid and the BVH. Left: average time per line of sight. Right: average time per iteration.*

# TURTLE performances: Monte Carlo

> The atmospheric μ flux was computed using PUMAS and TURTLE, for various locations and lines of sight.

The **average slow down** due to the topography is only a factor of **~2**

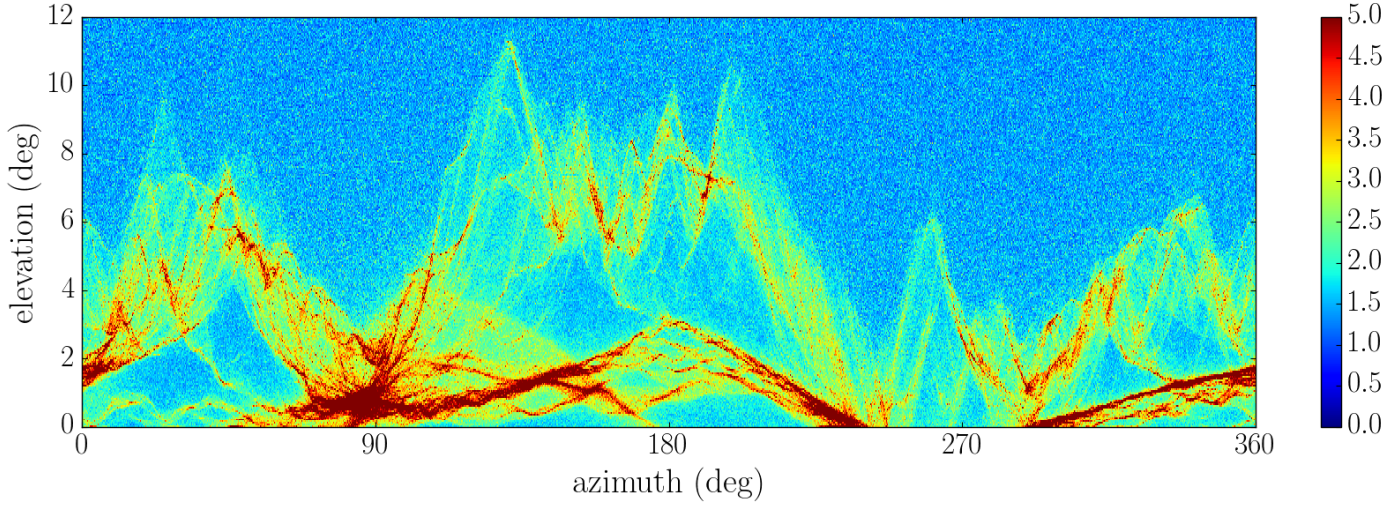



*Figure 9: slow-down due to the topography resolution when computing the atmospheric μ flux seen from the Ulastai site.*

# Summary

- TURTLE is a small (<100 kB) `C` library providing utilities for efficiently navigating through large sets of topography data, in a Monte Carlo.
- TURTLE implements the ***optimistic* algorithm**.
  - It does not guarantee an exact resolution of the terrain model. However the typical errors are well below the accuracy of topography measurements.
  - It is straight forward to represent the topography surface by a higher level model than triangular facets.

    > TURTLE implements a bilinear interpolation on the fly, providing extra accuracy with no extra memory cost.

  - The traversal time does not depend on the number of data nodes. Only on the travelled distance and the height w.r.t. the topography.
- TURTLE has a *preliminary* **binding** for Geant4 and an *incomplete* one for Python.