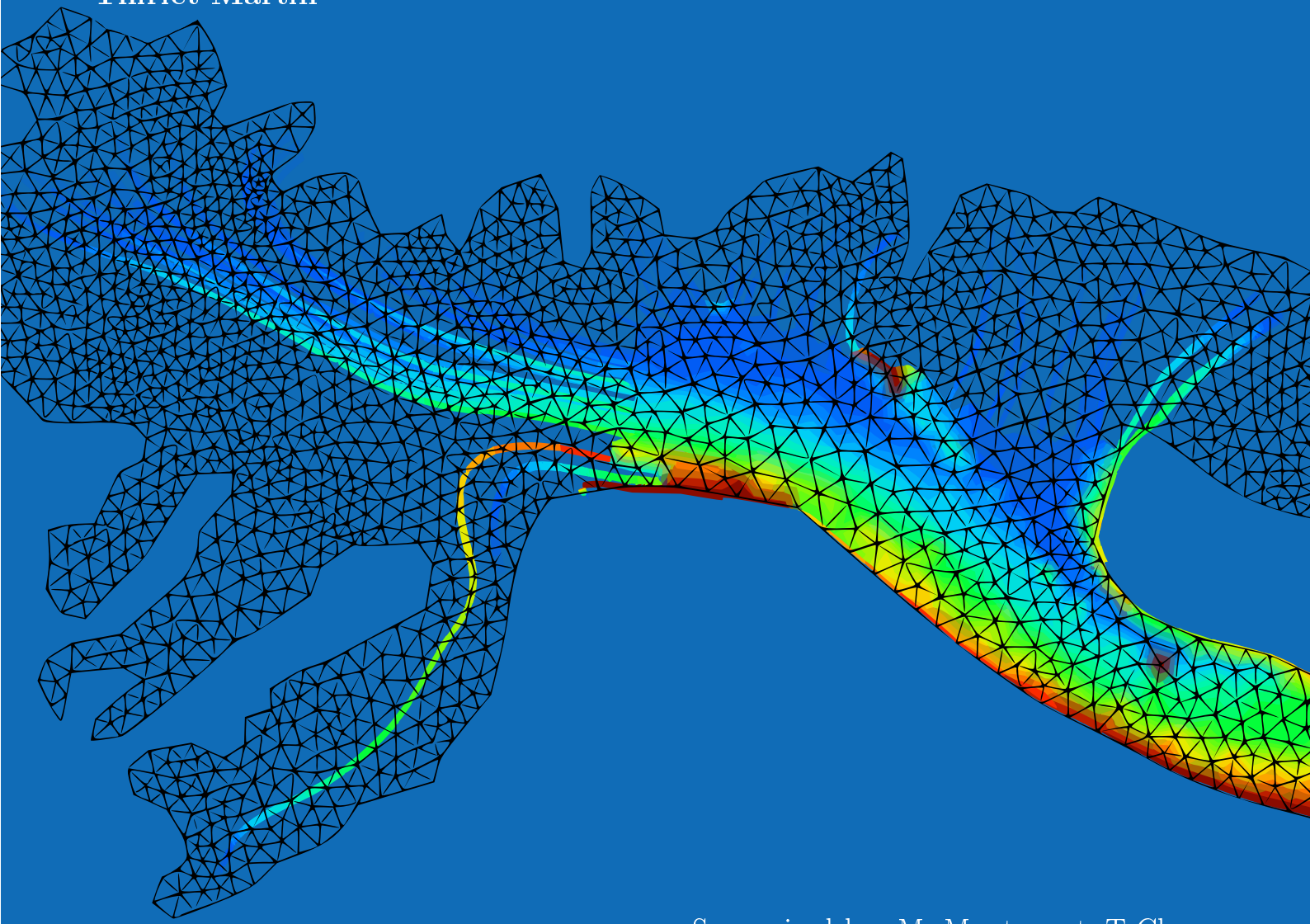
CryoDyn - Glaciers and ice sheets dynamics

Modeling Mechanical History along Flow Lines in Glacier or Ice Cap to study the Impact of Viscoplastic Anisotropy on Ice modeling

Thiriet Martin



Supervised by: M. Montagnat, T.Chauve

Institut des Géosciences de l'Environment

école normale supérieure paris—saclay

Contents

List of symbols and Abbreviations

General Notations and Functions

$\bullet$: scalar value

$\underline{\bullet}$: vector value

$\underline{\underline{\bullet}}$: tensor value

$\cdot$: scalar product

$\times$: matrix or tensor product

$|| \bullet ||$: norm

$\prod_{i=0}^{n} \bullet_i$: product from 0 to n

cos, sin, tan: trigonometric functions

$\sqrt{\bullet}$: square root

$\frac{\partial \bullet}{\partial \bullet}$: partial derivative

$\bullet^{-1}$: inverse of a function or matrix

$\bullet^{T}$: transposition of a matrix

$\otimes$: vectorical product

Coordinates

x, y, z: coordinates in the global basis

a_1, a_2, a_3: coordinates in the local basis

Fields and Physical Quantities

$\underline{v}$: velocity field

$\underline{v_i}$: nodal velocity vector

$v_{\%}$: ratio of velocities

$\mathbb{E}$: strain

$\dot{\mathbb{E}}$: strain rate

$\mathbb{F}$: deformation tensor

$\underline{\underline{\mathbb{L}}}$: gradient of the velocity field

A_{relat}: relative anisotropy factor

$\sigma_{\bullet}$: standard deviation of $\bullet$

$\overline{\bullet}$: mean of $\bullet$

Geometry and Shape Functions

$N_i()$: shape functions of an element

D_N: matrix of the derivatives of the shape functions

Positions and Paths

$\underline{x_n}$: position of the particle at time increment n

$\underline{t_n}$: tangent vector to the flow line

$\underline{dl}$: small vector

Φ: function that defines the particle position

Rotation and Transformation Matrices

R_i: rotation matrices

P: matrix of change of basis

ϕ, θ, ψ: rotation angles

Mathematical and Matrix Operations

$\underline{\underline{J}}$: Jacobian matrix

$\underline{\underline{\underline{grad}}}$: gradient function

X: variable in polynomial equations

m_{ij}: component (i, j) of matrix M

$\bullet^D$: deviatoric part of a tensor

$\mathrm{Tr}(\bullet)$: trace of a matrix

Variables and Increments

a, b: general variables

dt: time increment

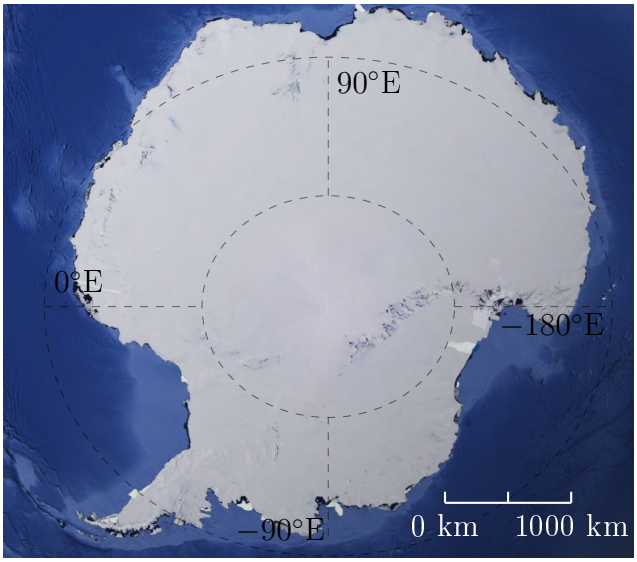
1 Introduction to models of alpine glaciers and ice caps

1.1 Overview of alpine glaciers and ice caps

Alpine glaciers and ice caps comprise the majority of Earth's freshwater resources, with approximately 97% of the planet's freshwater stored within these frozen reserves (Barry, 2011). In recent years, global temperature increases have accelerated the melting of these ice masses, contributing significantly to sea level rise. It is estimated that the meltwater from glaciers and ice caps currently adds about 1.48 mm/year to global sea levels (Jacob et al., 2012).

Within the cryosphere, ice structures are generally categorized into two major types: ice caps and glaciers. Ice caps are extensive sheets of ice predominantly located at polar regions, where persistent snowfall accumulates and compacts over time through processes such as compression and recrystallization. This accumulation gradually transforms into dense ice layers, which, due to their massive weight, flow outward like a very slow-moving fluid toward the edges of the ice sheet, where they eventually melt or calve into the sea or on land. These ice sheets are substantial, often measuring up to more than 3 km, and constitute the bulk of the cryosphere (see Figure 1).

(a) Satellite image of Greenland with scale reference.



(b) Satellite image of Antarctica with scale reference.

Figure 1: The two primary polar ice sheets on Earth (GoogleEarth, 2024).

The second type of ice structure, glaciers they are typically smaller than ice caps and are most often located in mountainous regions. Glaciers exhibit a similar water cycle to ice caps, where snow accumulates in an upper area known as the accumulation zone, compresses into ice, and flows downward to the ablation zone, where melting occurs (see Figure 2). Glaciers are generally less extensive than ice caps, with lengths often measured in kilometers and thicknesses reaching up to several hundred meters.

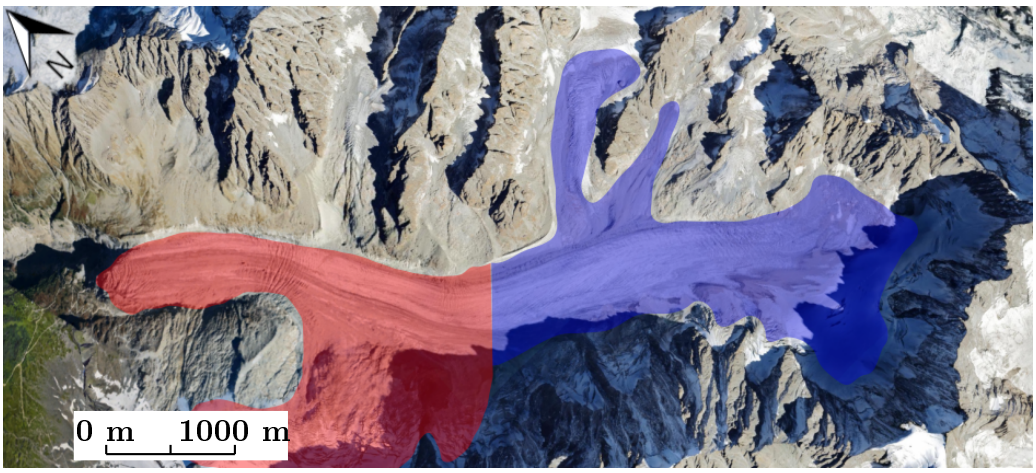


Figure 2: Satellite image of Argentière Glacier (GoogleEarth, 2024). The ablation area is marked in red at the glacier's lower regions, while the accumulation area is shown in blue.

A distinctive feature of glaciers is their temperature variability, as they exist at a wide range of altitudes—from low-elevation glaciers like Engabreen in Norway, at approximately 20 meters, to high-altitude glaciers such as Khumbu Glacier in Nepal, reaching altitudes of up to 7600 meters. This diversity in altitude results in significant temperature differences among glaciers, leading to their classification into two main types: cold glaciers, where the ice temperature remains below 0°C, and temperate glaciers, where the ice temperature is at or near 0°C throughout. The combination of the two aspects is also possible with a glacier that is cold in one area and temperate in the other ares.

1.2 Modeling Ice Flow Using Elmer/Ice

Various software tools are available for modeling the dynamics of ice sheets and glaciers, with Elmer/Ice being a particularly versatile and widely used option. Elmer/Ice is a specialized module of the open-source Elmer finite element software, developed specifically for glaciological applications (Elmer). Written in FORTRAN 90 and C, Elmer/Ice is highly competitive with commercial modeling software in terms of computational efficiency and flexibility.

Elmer/Ice provides a range of customizable options, allowing users to select different physical laws for each aspect of the glacier model, as outlined by Thomas ZWINGER (2019). To accurately simulate ice flow, a suitable flow law must be defined to characterize the ice's rheological behavior. Additionally, a basal friction law is required to model the glacier-bed interface conditions. Various approximations of the velocity fields can also be employed to simplify computations, depending on the model's objectives and computational resources.

1.2.1 Determining Glacier Geometry

The initial step in modeling glaciers or ice caps is to define the geometry of the ice mass, which is determined by the top and bottom surfaces of the glacier. The top surface, representing the glacier's exposed surface, can be accurately mapped using digital elevation models (DEMs). These models are often generated through advanced technologies like light detection and ranging (LiDAR) from drones or airplanes, providing high-precision surface data, as demonstrated by Gindraux et al. (2017).

The bottom surface, or bedrock topography, poses a greater challenge in data acquisition since direct measurements are typically unfeasible. For relatively thin ice layers with minimal impurities, ground-penetrating radar (GPR) is an effective method, as demonstrated by Collins et al. (1989). GPR works by emitting radio waves through the ice, these waves reflect upon reaching the bedrock, and the returned signals are then used to calculate the ice thickness. However, GPR accuracy diminishes if the ice contains fractures or impurities, which can cause noise in the reflected signal, thereby reducing the precision of the bedrock topography. Additionally, for thicker ice masses, GPR is less effective because radio waves tend to be absorbed and scattered, resulting in limited accuracy for deep ice.

An alternative method involves drilling with a ice core drill or a hot water system a boreholes across the glacier or ice cap, providing localized measurements with a high level of certainty. While boreholes offer precise data points and allow for numerous mesurements and experiments for other research fields, they are labor-intensive and do not provide comprehensive coverage of the glacier's bed. The drilling process also becomes increasingly challenging at greater depths, making this technique impractical for extensive ice caps or glaciers it is thus used to precise other methodes of data aquisition.

The most commonly adopted approach combines surface velocity measurements, mass balance data, and boundary condition assumptions to construct a finite element model of the glacier. Using this model, an inverse method can be applied to estimate ice thickness. This approach leverages accessible data, such as surface velocity fields (often derived from digital image correlation) and mass balance measurements obtained through field surveys. Although this method provides a practical solution for estimating ice thickness, it is sensitive to the assumptions and approximations made in the inverse modeling process, as highlighted by Michel et al. (2013).

1.2.2 Time Evolution in Glacier Simulations

In glacier simulations, different approaches are employed to approximate the time evolution of the glacier's state. The simplest and most computationally efficient approach is to construct a steady-state model. In a steady-state simulation, time dependency is ignored, and the glacier is assumed to be in equilibrium, with a constant geometry and stabilized velocity fields obtained after a hypothetical infinite relaxation time. These simulations are accurate only if the glacier has a constant mass balanced, meaning the volume of water entering and exiting the glacier is equal. Steady-state models do not incorporate time-varying data, such as annual precipitation rates or temperature changes.

A more sophisticated and realistic approach for simulating glaciers and ice caps is to use a transient model, as exemplified in the case of Argentière Glacier (Gilbert et al., 2023). In transient simulations, the glacier does not reach equilibrium, instead, its evolution is calculated step-by-step from a specified initial condition, allowing for continuous changes over time. Transient models require a detailed and dynamic mass balance to apply appropriate conditions at each time step, along with a time-dependent temperature profile. Unlike steady-state models, transient simulations involve a deforming mesh that adapts to changes in glacier geometry over time. Given that glaciers can experience significant deformations, remeshing may be necessary after a certain number of time steps to maintain accuracy. In addition, masks are applied to exclude areas that have melted from subsequent computations. Although transient models are more complex to implement, they provide more accurate and realistic results by accounting for temporal variations in glacier conditions.

1.2.3 Basal Friction Law

The choice of basal friction law plays a crucial role in accurately modeling glacier and ice cap dynamics, as it significantly affects the interaction between the glacier base and the underlying bedrock. Basal friction depends on factors such as the bedrock shape (Lliboutry, 1979) and surface roughness (Budd et al., 1979). Obtaining experimental data to accurately constrain the basal friction law is challenging, which makes selecting an appropriate law and determining its parameters complex. Additionally, when using inverse modeling to estimate the glacier bed's digital elevation model (DEM), the basal friction law often becomes a tuning parameter, potentially obscuring other physical processes such as dynamic recrystallization.
Elmer/Ice provides several basal friction laws that can be implemented in finite element models, each with unique characteristics suited to different glaciological conditions:

Linear Law: The linear basal friction law is a straightforward model based on Coulomb's principles (Coulomb, 1785). It assumes a direct linear relationship between basal shear stress and sliding velocity of the ice over the bed. This law is suitable for scenarios where basal sliding occurs over smooth bed surfaces with minimal topographical variation. However, its simplicity can be limiting in complex glacier systems with rough or highly variable bed conditions, where basal friction may vary significantly.

Weertman Law: Proposed by Weertman (1957), this foundational sliding law models basal friction as a non-linear function of effective pressure at the bed and sliding velocity. The Weertman law includes power-law terms that capture the complex, non-linear relationship between shear stress and velocity. This approach is particularly useful for glaciers with irregular bedrock topography, where sliding behavior depends on pressure variations and meltwater presence. The Weertman law is widely used in glaciology and provides a basis for more advanced friction laws.

Modified Coulomb Law: Gagliardini et al. (2007) extended the traditional Coulomb law to better model glaciers with debris-laden bases. In this formulation, basal friction is not solely dependent on sliding velocity but also on basal effective pressure, making it well-suited for temperate glaciers where subglacial water and debris affect sliding dynamics. The modified Coulomb law adapts to varying bed conditions and provides a more realistic representation of sliding at the glacier-bed interface, especially in scenarios with fluctuating basal water pressure.

Budd Law: Introduced by Budd et al. (1979), this law incorporates bed roughness and surface melt conditions into the basal friction calculation. Budd's law includes terms for both rugosity and sliding velocity, enabling it to account for friction effects due to bedrock irregularities. This model is particularly advantageous for glaciers with rough or complex bed topography, as it provides a more realistic friction profile when the glacier bed is highly variable. Additionally, Budd's law accounts for meltwater influence, which can reduce basal friction and enhance sliding, especially in warmer climates or during melt seasons.

Tsai Law: Developed byTsai et al. (2015), this modern friction law addresses some limitations of earlier models by incorporating hydrological processes at the glacier bed. Tsai's model considers feedback mechanisms between basal water pressure, ice velocity, and effective pressure, allowing for a more accurate simulation of glaciers influenced by hydrodynamic variations. This law is ideal for transient models that capture short-term fluctuations in glacier velocity associated with seasonal or event-driven meltwater pulses, making it particularly relevant for regions where glacier movement is sensitive to water input changes.

Each of these basal friction laws offers distinct advantages depending on the glacier's environ-

mental conditions and the simulation's objectives. The choice of law should align with the specific dynamics of the glacier or ice cap being modeled to achieve an accurate representation of basal sliding behavior.

1.2.4 Approximation of the vertical Velocity Field

After selecting the flow law and establishing boundary conditions, it is necessary to choose the appropriate equations to represent glacier flow. Although the full Stokes equations offer a complete 3D solution to fluid mechanics problems, they are often too computationally intensive for large-scale glacier modeling. Therefore, several approximations are typically employed to reduce complexity by solving a 2D problem and then calculating the third velocity component (the vertical or z velocity) using various techniques:

Full Stokes Approximation: The full Stokes approach (Stokes, 2007) provides the most precise solution by solving the entire set of Stokes equations without simplifications, accounting for all stress components and three-dimensional flow dynamics. This approach is well-suited for modeling glaciers with intricate geometries or varying bed topographies, where any simplifications could omit critical dynamics. Although it offers the most accurate z-velocity field calculation, the full Stokes approximation demands substantial computational resources, making it impractical for large-scale or long-duration simulations. Consequently, this method is generally applied in high-fidelity studies focused on specific glacier regions rather than entire ice sheets or glaciers.

Shallow Shelf Approximation (SSA): The SSA, as presented by Morlighem et al. (2013), is a 2D simplification that assumes horizontal stress gradients dominate over vertical shear stresses, ideal for fast-flowing glaciers and ice streams where horizontal flow is the primary driver. SSA is particularly applicable to ice shelves and floating glacier tongues, where basal drag is minimal and vertical shearing can be ignored. This approximation accelerates computations while accurately capturing horizontal flow characteristics; however, it is less effective for areas with significant basal friction or where accurate vertical velocity (z-velocity) representation is required.

Shallow Ice Approximation (SIA): The SIA, formalized by Hutter (2017), is another 2D approach tailored for glaciers where vertical shear stresses play a dominant role, such as those found on steep slopes with high basal drag. SIA assumes that horizontal stress gradients are negligible compared to vertical ones, making it suitable for slow-moving glaciers where vertical shearing is the primary flow mechanism. While efficient and useful for cases with substantial bed friction, the SIA is limited in its applicability to fast-flowing glaciers or floating ice, as it does not capture horizontal stress effects well. This simplification allows efficient z-velocity calculations in regions where vertical shearing governs ice movement.

Improved Shallow Ice Approximation with Lateral Drag (ISCAL): The ISCAL approximation, developed for use in the Elmer/Ice framework (elmerice, 2017), enhances the basic SIA by including lateral drag effects and permitting longitudinal stress coupling along the glacier's length. ISCAL refines SIA by accounting for lateral and basal drag interactions, making it more accurate for glaciers with variable bed conditions or lateral constraints. By incorporating these coupling effects, ISCAL offers a more realistic approximation of the 3D flow while retaining the computational efficiency of a 2D model. This approach strikes a balance between the simplicity of SIA and the detail provided by the full Stokes solution, making it suitable for glaciers with mixed basal characteristics and moderate sliding.

Each of these approximations presents trade-offs between computational efficiency and physical accuracy. The choice of method should align with the specific goals and conditions of the glacier model to ensure appropriate representation of the z-velocity field.

1.2.5 Flow Law

Once the mesh, boundary conditions, and governing flow law have been established, it is necessary to select an appropriate material law to represent ice behavior. The choice of this law will vary depending on the desired accuracy and complexity of the model.

1.2.5.1 Anisotropy of single crystal Ice

Unlike many materials, which tend to be isotropic and possess a cubic crystal structure, ice has a hexagonal crystal structure, as illustrated in Figure 3. Due to this hexagonal structure, ice crystals have a preferred orientation, represented by a specific direction in space, commonly denoted by the $\underline{c}$-axis.

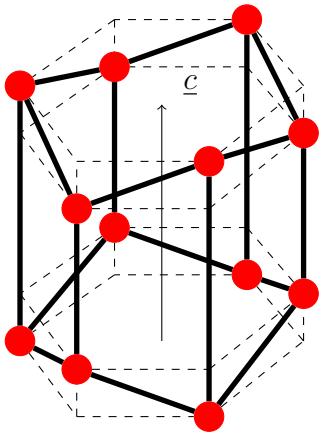
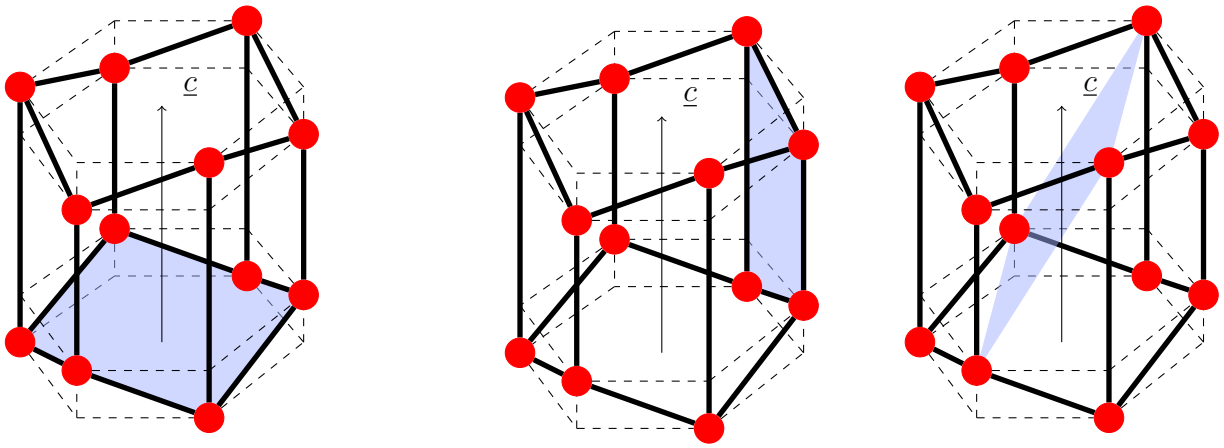


Figure 3: Hexagonal structure of an ice crystal with oxygens atoms in red

This hexagonal structure gives ice its highly anisotropic properties, enabling deformation along three split systems. Each slip system contains multiple slip planes, one of those slip planes is represend in Figure 4 for each type of slip system. These three primary deformation systems define the possible modes of viscoplastic deformation within an ice monocrystal; any other type of deformation will essentially be a combination of the activation of slip planes in these 3 slip systems.



(a) Basal slip plane (b) Prism slip plane (c) Pyramidal slip plane

Figure 4: Slip planes of ice crystals

The anisotropic nature of ice is a result the activation of the different slip systems, each requiring different stress levels to activate. Experimental studies, such as those reviewed by Duval et al. (1983), have demonstrated that deformation along the basal plane is approximately 1000 times faster than non-basal deformation at a given stress level (see Figure 5), underscoring the importance of the $\underline{c}$-axis orientation in ice deformation. This directionality indicates that the primary deformation occurs in planes orthogonal to the $\underline{c}$-axis, which results in a strong viscoplastic anisotropy for the ice monocrystal.

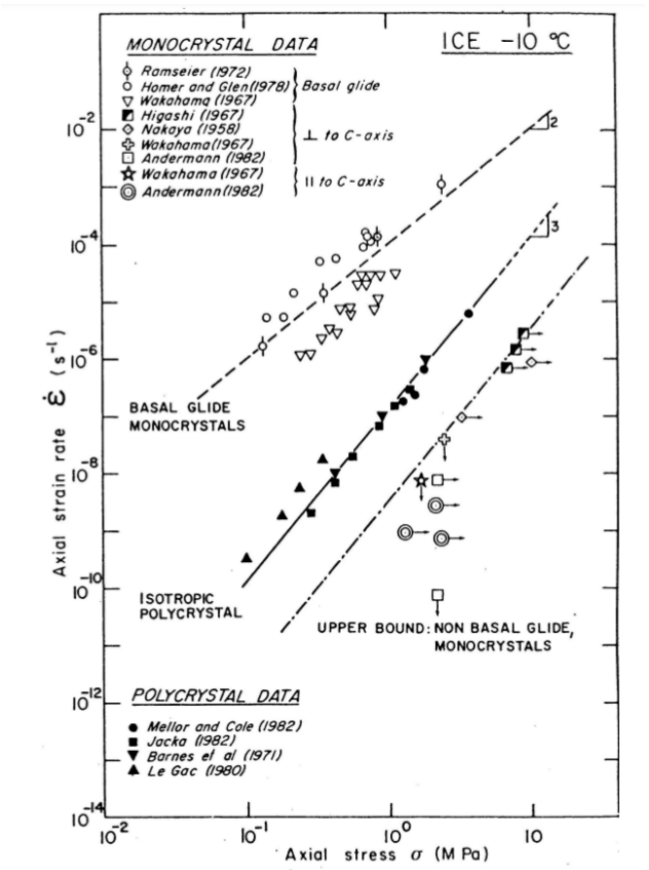


Figure 5: Strain rate as a function of stress for basal and non-basal deformations (Duval et al., 1983)

To accurately model the deformation of ice monocrystals, a flow law developed by Meyssonnier and Philip (1996), which incorporates three distinct deformation planes, can be applied. This approach provides a relationship between the deviatoric strain rate $\underline{\underline{D}}$ and the deviatoric stress $\underline{\underline{S}}$. Known as the continuous transverse isotropic (CTI) model, this law assigns unique properties to each of the three principal axes. In the CTI model, the orientation of the ice crystal is represented by the tensor $M = \underline{c} \otimes \underline{c}$. Several parameters, including the Glen exponent n, viscosity η_n, and anisotropy factors $\alpha_1, \alpha_2, \alpha_3$, are employed to define the velocity law. With these parameters, the resulting flow law can be expressed as follows.

$$\begin{aligned} \underline{\underline{S}} &= \eta_n^* \left(2\alpha_1 \underline{\underline{D}} + 2\alpha_2 \underline{\underline{M}}^D \mathrm{Tr}\left(\underline{\underline{M}}\,\underline{\underline{D}}\right) + \alpha_3 \left(\underline{\underline{M}}\,\underline{\underline{D}} + \underline{\underline{D}}\,\underline{\underline{M}}\right)^D \right) \\ \eta_n^* &= 2\eta_n \left(\alpha_1 \mathrm{Tr}\left(\underline{\underline{D}}^2\right) + \alpha_2 \mathrm{Tr}\left(\underline{\underline{M}}\,\underline{\underline{D}}\right)^2 + \alpha_3 \mathrm{Tr}\left(\underline{\underline{M}}\,\underline{\underline{D}}^2\right) \right)^{\frac{1-n}{2n}} \end{aligned} \tag{1}$$

This law is then able to predict deformations in ice crystall has is has been tested in Mansuy et al. (2002).

1.2.5.2 Polycrystalline Nature of Ice

Although ice monocrystals display strong anisotropy, glacial ice typically exists in a polycrystalline form, composed of many individual ice crystals with varying orientations. The alignment of crystals in polycrystalline ice can be visualized using crossed polarized light, producing colorful images that reflect the orientation of individual grains, as shown in Figure 6.

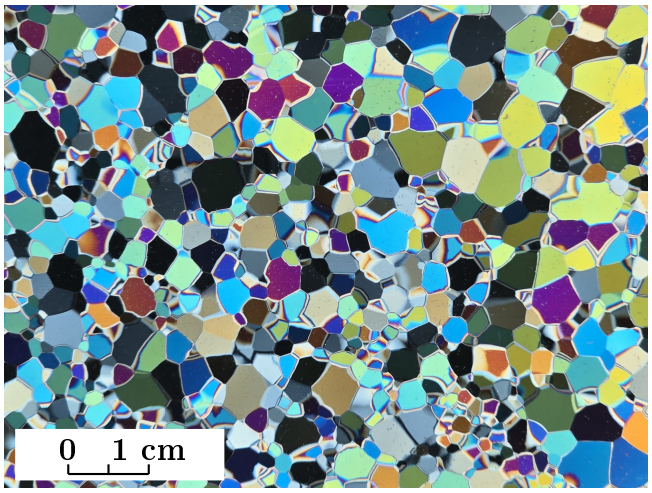


Figure 6: Image of polycrystalline ice viewed through crossed polarized light

Due to the anisotropic nature of individual ice grains, polycrystalline ice can also exhibit anisotropic behavior depending on the orientation distribution of these grains. Pole figures are commonly used to present a statistical overview of crystal orientations in polycrystalline ice, offering insight into its overall texture and behavior .
These figures are created by projecting the $\underline{c}$ axis of each ice crystal onto a disk, similar to a stereographic projection, as shown in Figure 7.

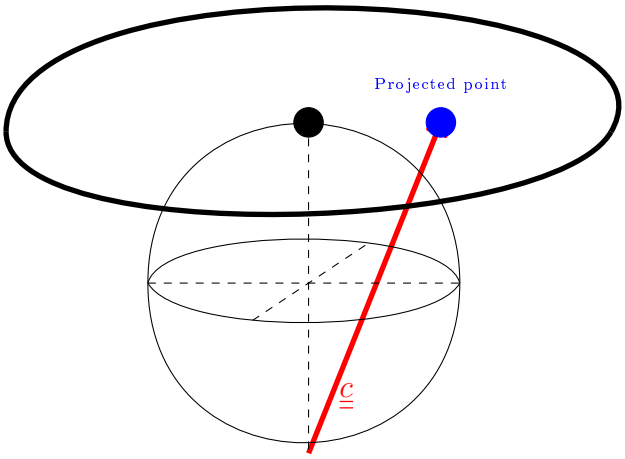
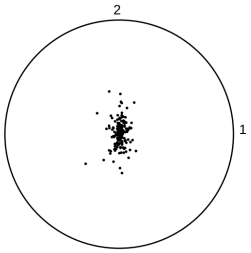
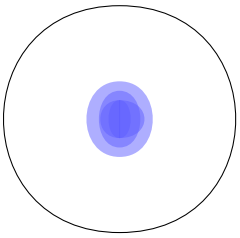


Figure 7: Example of a stereographic projection

This projection (see Figure 8a) can then be processed to display a continuous map of the orientations by plotting the density of points in a local area (see Figure 8b), this process make the representation easier to understand and allows for comparaison between different textures.



(a) Mesured points projected (Montagnat, 2001)



(b) Density of points

Figure 8: Examples of pole figures for uniaxial compression textures

1.2.5.3 Texture évolution of ice under deformation

Under deformation, polycrystalline ice undergoes texture evolution. This phenomenon can be observed, for instance, in deep ice cores where, despite low temperatures at the top of the core that prevent other processes, the ice texture changes due to deformation. This effect is evident in the EPICA ice core at Dome C (Durand et al., 2009). Along the ice core, the texture evolves, beginning with a predominantly uniform orientation, then converging toward a single maximum, as shown in Figure 9.

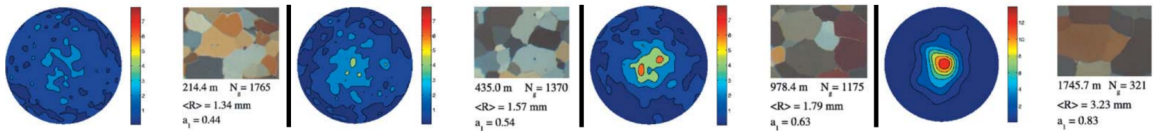


Figure 9: Pole figures showing texture evolution in the EPICA ice core at different depths (Durand et al., 2009)

The evolution of ice texture due solely to deformation can be described using the Continuous Transverse Isotropic (CTI) equations developed in Gillet-Chaulet et al. (2005).

In addition to the CTI equations describing ice flow, the rotation of the $\underline{c}$ axis can be represented by equations from (Gillet-Chaulet et al., 2006) and (Gagliardini et al., 2009), where $\underline{c}$ is the axis of the ice crystals, $\underline{\underline{W}}(\underline{u}) = \frac{1}{2}\left(\underline{\underline{grad}}\,(\underline{u}) - \underline{\underline{grad}}\,(\underline{u})^T\right)$ represents the spin rate (with $\underline{u}$ as the deformation field), $\underline{\underline{D}}(\underline{u}) = \frac{1}{2}\left(\underline{\underline{grad}}\,(\underline{u}) + \underline{\underline{grad}}\,(\underline{u})^T\right)$ represents the strain rate, and λ is a parameter of the model.

$$\frac{\partial \underline{c}}{\partial t} = \underline{\underline{W}}\,\underline{c} - \lambda\left(\underline{\underline{D}}\,\underline{c} - \left(\underline{c}^T\,\underline{\underline{D}}\,\underline{c}\right)\underline{c}\right) \tag{2}$$

This equation describes the evolution of the ice structure due to ice flow motion but does not account for other phenomena.

1.2.5.4 Dynamic Recrystallization Processes

When ice experiences sustained stress leading to high strain, as in large ice masses and if the deformations happens at high temperature (close to the melting point) it undergoes dynamic recrystallization, a process that transforms the ice texture over time. Dynamic recrystallization has been observed both in laboratory settings and within natural ice formations (Kipfstuhl et al., 2009). This process primarily occurs during secondary and tertiary creep stages and accelerates significantly as temperatures approach the melting point. The evolution of ice texture under stress has been documented in experiments, as illustrated in Figure 10.

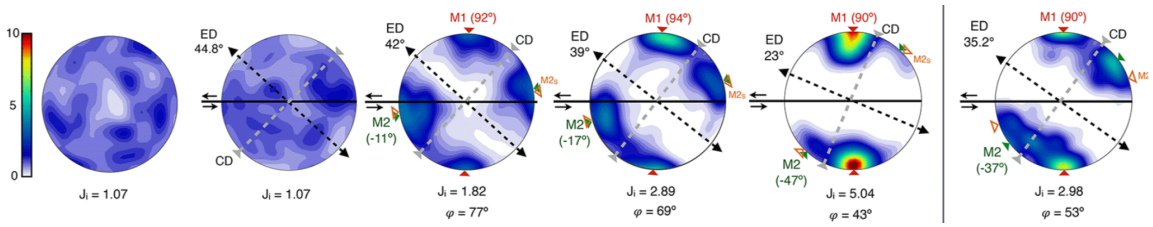


Figure 10: Pole figures showing texture evolution in a shear test (Journaux et al., 2019)

Several numerical models have been developed to simulate dynamic recrystallization with varying levels of alignment to experimental data. Among these, the R^3iCe model, proposed by Chauve et al. (2023), is one of the most accurate. This model uses finite elements to represent individual ice crystals, calculating stresses within each crystal and adjusting the crystal orientation toward the plane of local resolved shear stress.
To achive this task each ice crystall has an attractor (orientation toward the local resolved shear stress) defined by $c_0 = \frac{1}{2}(e_1 \pm e_3)$ with e_i the eigen vectors of $\underline{\underline{S}}$. Then each ice crystall is forced toward this attractor at a certain rate defined by the recrystalisation parameter Γ_{RX}. This can be inclued in the equation 2 like so

$$\frac{\partial \underline{c}}{\partial t} = \underline{\underline{W}}\,\underline{c} - \lambda\left(\underline{\underline{D}}\,\underline{c} - \left(\underline{c}^T\,\underline{\underline{D}}\,\underline{c}\right)\underline{c}\right) + \frac{1}{\Gamma_{RX}}(c_0 - c) \tag{3}$$

The R^3iCe model has been validated across different test cases, demonstrating behavior that closely mimics recrystallization processes observed experimentally. However, due to its computational demands, this model is not feasible for large-scale applications.

1.2.5.5 Approximation of Ice Behavior in Elmer/Ice

Elmer/Ice provides several simplified flow laws or rheologies to approximate the mechanical behavior of ice, each suited to different modeling needs. Choosing an appropriate flow law is essential to balance computational efficiency with realistic simulation results, especially when

dealing with large-scale models. The following flow laws are implemented in Elmer/Ice:

Glen's Law: Glen's flow law, introduced by Glen (1955), is one of the most commonly used constitutive laws in glaciology. It describes a non-linear, power-law relationship between shear stress and strain rate, where stress is raised to an exponent typically around 3. This law effectively represents the creep behavior of ice, where deformation accelerates with increasing stress and temperature. Glen's law is widely applicable for general glacier simulations, especially for steady-state creep under gravitational forces. However, it may not capture details in high-strain regions, such as near calving fronts or in fast-flowing ice streams.

GOLF Law: The General Orthotropic Linear Flow Law (GOLF) developed by Gillet-Chaulet et al. (2005) provides a streamlined approach to model the anisotropic behavior of ice within large-scale ice-sheet simulations. Unlike traditional micro-macro models, which are often complex and computationally demanding, the GOLF law simplifies implementation by assuming that ice behaves as a linearly viscous, orthotropic material. The GOLF model characterizes the fabric of the ice using an orientation distribution function with just two parameters, enabling efficient interpolation of flow parameters for various configurations. This approach allows for adaptable and user-friendly ice flow simulations that take into account the ice rheology.

CAFFE Law: Proposed by Seddik et al. (2008), the CAFFE (Continuum Anisotropic Flow of Field Elements) law is tailored for regions with high strain rates where Glen-type laws may not fully capture the ice behavior. CAFFE introduces advanced constitutive relations to simulate fast-flowing ice features, such as ice streams, shear margins, or calving fronts. It is particularly valuable in dynamic ice shelf and fast-moving glacier models, providing a more accurate depiction of rapid deformation processes, albeit with higher computational demands.

Porous Flow Law: Developed by Gagliardini and Meyssonnier (1997), this flow law accounts for porosity in ice, such as air- or water-filled voids in firn layers or debris-laden ice. By modifying standard flow relations, this law models the reduced stiffness of ice due to the presence of voids, making it suitable for upper glacier layers or ice sheet areas where compaction and permeability impact deformation. The porous law also aids in understanding the densification process and the transition from firn to solid ice.

Damage Model: The damage model, introduced by Krug et al. (2014), incorporates damage mechanics to simulate ice weakening over time due to microcracking and fracturing. This law is particularly relevant in high-stress zones or near calving fronts, where fracturing and damage accumulation significantly impact glacier behavior. The damage model tracks cumulative damage, which influences ice stiffness and flow characteristics. While computationally intensive, this law is essential for simulating calving, rifting, and fracture development, where structural integrity plays a key role.

Each of these flow laws presents unique capabilities and is suited to different conditions and levels of modeling detail. Selecting the appropriate law is key to achieving a realistic balance between computational efficiency and accuracy in glacier and ice cap simulations.

1.3 Objective of the Work

The primary objective of this research is to develop a particle tracking tool capable of post-processing simulation results from flow simulation software, such as Elmer/Ice, track a particule, then extract data allong the flow line and allow for easy computation in the Lagrangian frame of

reference . This tracking tool aims to enable users to identify particle paths with notable stress and strain histories within the glacier flow. Once identified, the data from these trajectories can be transferred to the R^3iCe model, which will then simulate the evolution of ice structure along these flow lines.

The tool is designed to function as an interface between Elmer/Ice and R^3iCe, facilitating predictions of ice textures that may be present at the base of glaciers. By linking flow dynamics with detailed recrystallization modeling, this tool provides a comprehensive approach to understanding the textural changes in ice resulting from its movement and deformation history.

2 Implementation of a Particle Tracking Algorithm Based on Elmer/Ice Output

Numerous Elmer/Ice simulation datasets have been made available by the glaciology community. Given that some of these simulations can take several days to complete on high-performance computing systems, this project focuses on developing a post-processing tool that utilizes the output from existing simulations to trace the paths of selected particles. This approach eliminates the need to re-run complex simulations, allowing results to be post-processed locally on lower-performance hardware, thereby saving both time and computational resources.

2.1 Output Data Format from Elmer/Ice

Elmer/Ice exports simulation data in a non-proprietary .vtu format. This file format contains both the mesh structure and the simulation results, including data at each mesh node and element. The .vtu format is compatible with visualization tools such as kitware, enabling straightforward data examination. In this work, the Python library Schlömer is used for efficient manipulation of the mesh and associated data fields.

In Elmer/Ice glacier simulations, the mesh structure follows a specific arrangement and consists solely of wedge elements, due to the way the mesh is generated. The initial step in mesh creation involves constructing a 2D mesh of the glacier's top surface using triangular elements, as illustrated in Figure 11.

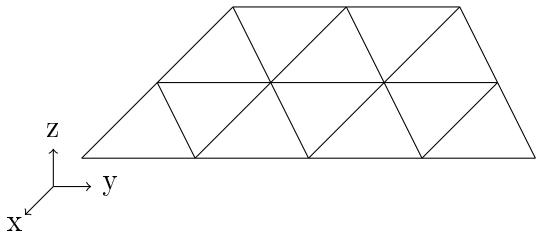


Figure 11: 2D mesh of the glacier's top surface

This 2D mesh is then extruded in the z-direction based on the ice thickness, resulting in a 3D mesh composed entirely of wedge elements. The top and bottom boundary elements are triangular, while the side elements are rectangular, as shown in Figure 12.

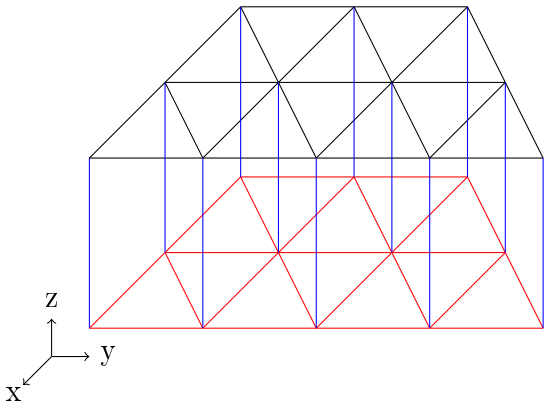


Figure 12: 3D extruded mesh

This extrusion-based mesh generation is a straightforward method for creating a 3D mesh that can be refined locally as needed. However, it does have limitations, as it produces elements

with poor quality on the sides—elements that may be thin yet wide. Such configurations can complicate particle tracking, as illustrated in Figure 13.

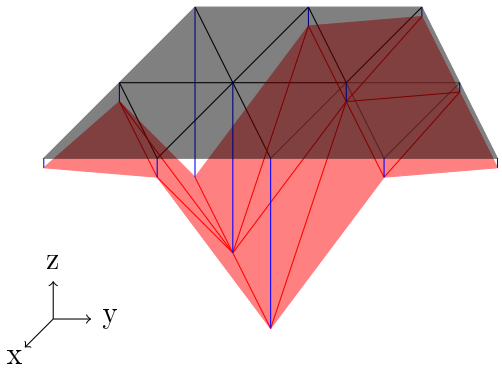


Figure 13: 3D extruded mesh with side elements of poor quality

Once the mesh file is loaded with meshio, users can access an unsorted list of nodes, along with lists for each element type—in this case, triangles, squares, and wedges—each listing the node IDs for that element. Additionally, the simulation's field data, such as node velocities, is readily accessible, providing the essential information required for particle tracking.

2.2 Particle Tracking Inside a Mesh

In this section, we outline the particle tracking method used in this work and explain why it was selected.

Various methods for particle tracking in finite element meshes have been developed, primarily building on the initial method proposed by Cheng et al. (1996). In this approach, the trajectory of a particle is computed based on the initial velocity field at the particle's starting point. An algorithm then determines which mesh face the particle crosses, identifying the subsequent element containing the particle. This method offers high convergence and accuracy in tracking. Further refinements have been made, such as the method by Pokrajac and Lazic (2002), which introduces polynomial interpolation to ensure continuity between elements.

These particle tracking methods are implemented in tools like Elmer/Ice and ParaView, and while they are reliable and efficient, they encounter challenges with complex or low-quality meshes. Specifically, when determining which face the particle passes through, the method involves calculating the vector product between the face and particle position, which is highly sensitive to numerical errors. This sensitivity can lead to inconsistencies in low-quality elements. Additionally, these methods are memory-intensive, making them unsuitable for large-scale models on lower-performance computers.

2.2.1 Integration Scheme

To create a faster and more robust tracking algorithm that can run on lower-performance computers, a different approach was adopted.

The first step is selecting an integration scheme; here, a first-order Euler integration scheme was chosen. This scheme provides the simplest spatial integration but is sufficiently accurate in this context, as the velocity fields between elements exhibit minimal variation. To determine the particle path, the initial position $\underline{x}(t)$, the velocity field $\underline{v}(\underline{x}(t), t)$ at location $\underline{x}(t)$ and time

t, and a time step dt are required. Using this information, the particle's position at the next time step is computed as follows:

$$\underline{x}(t+dt) = \underline{x}(t) + dt \cdot \underline{v}(\underline{x}(t), t) \tag{4}$$

By iteratively applying this formula, the particle's trajectory can be computed over time. While the first-order Euler integration scheme converges more slowly than a second-order scheme—requiring a smaller time step dt to achieve similar error margins—it is appropriate here. Since particles typically pass through multiple consecutive points within an element, and given that the mesh elements are first-order (with a linear velocity field), using a second-order integration scheme would not significantly improve accuracy. Instead, it would increase computational costs unnecessarily.

2.2.2 Locating the Particle Within the Mesh

With the basic principles of the particle tracking method established, we now examine how this approach can be applied to a 3D model within the finite element context. In Equation 4, all elements are available except for the particle's velocity at its initial position. To obtain this velocity, we use shape functions and the nodal velocity values within the element. This requires defining the shape functions for a wedge element.

$$\begin{cases} N_1(a_1,a_2,a_3) = \frac{1}{2}a_1(1-a_3) \\ N_2(a_1,a_2,a_3) = \frac{1}{2}a_2(1-a_3) \\ N_3(a_1,a_2,a_3) = \frac{1}{2}(1-a_1-a_2)(1-a_3) \\ N_4(a_1,a_2,a_3) = \frac{1}{2}a_1(1+a_3) \\ N_5(a_1,a_2,a_3) = \frac{1}{2}a_2(1+a_3) \\ N_6(a_1,a_2,a_3) = \frac{1}{2}(1-a_1-a_2)(1+a_3) \end{cases} \tag{5}$$

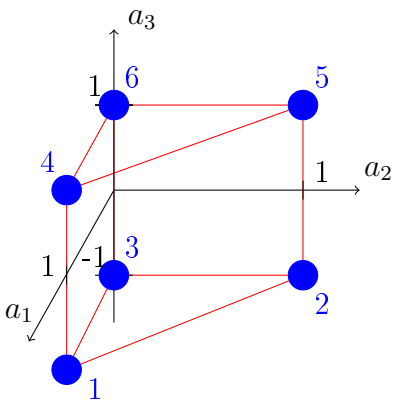


Figure 14: Reference element for a wedge

With the shape functions defined, the velocity at a given point within the element can be computed using the nodal velocities $v_i(t)$:

$$\underline{v}(\underline{x}(t),t) = \begin{pmatrix} \underline{v}_1(t) & \underline{v}_2(t) & \underline{v}_3(t) & \underline{v}_4(t) & \underline{v}_5(t) & \underline{v}_6(t) \end{pmatrix} \cdot \begin{pmatrix} N_1(a_1,a_2,a_3) \\ N_2(a_1,a_2,a_3) \\ N_3(a_1,a_2,a_3) \\ N_4(a_1,a_2,a_3) \\ N_5(a_1,a_2,a_3) \\ N_6(a_1,a_2,a_3) \end{pmatrix} \tag{6}$$

The remaining task is to determine the position of the particle within the reference element, represented by coordinates a_1, a_2, a_3. There are various algorithmic options for this step. In Elmer/Ice's standard approach, the element containing the particle is already known, so it only requires inverting the shape functions to find a_1, a_2, a_3. However, in our case, since the element containing the particle is unknown, we must first identify the wedge where the particle is located.

To locate the wedge, we follow several steps. The first step is to find the nearest wedge to the particle's position, which begins by identifying the nearest node to the particle. Efficiently navigating between nodes requires reorganizing the mesh structure. Typically, the mesh description consists of lists like the example shown in Figure 15.

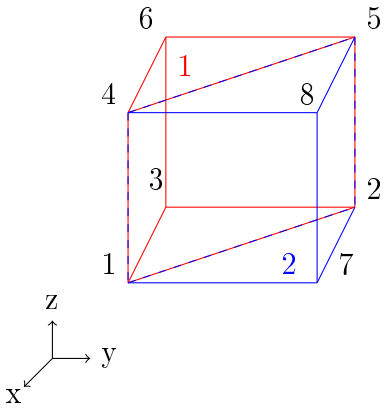


Figure 15: Example mesh used for explanation

Nodes

x	y	z
0	0	0
0.1	0	0
0	0.1	0
0.1	0.1	0
0	0	0.1
0.1	0	0.1
0	0.1	0.1
0.1	0.1	0.1

Triangles

1	2	3
2	3	1
6	7	5
6	7	9
2	3	8

Wedges

1	2	3	4	5	6
2	3	1	6	7	5
2	3	8	6	7	9

This basic structure lacks direct relationships between nodes and elements. By inverting the mesh, we create a list for each node that identifies the connected wedges. This enables efficient navigation through the mesh during particle tracking.

Nodes

x	y	z
0	0	0
0.1	0	0
0	0.1	0
0.1	0.1	0
0	0	0.1
0.1	0	0.1
0	0.1	0.1
0.1	0.1	0.1

Triangles

1	2	3
2	3	1
6	7	5
6	7	9
2	3	8

Wedges

1	2	3	4	5	6
2	3	1	6	7	5
2	3	8	6	7	9

Node Relations

Node	Connected Wedges
1	1
2	1, 2
3	1, 2
4	1
5	1
6	1, 2
7	1, 2
8	2

With this structure, navigating from node to node requires only a few operations. To move from one node to an adjacent node, we consult the Node Relations table to identify the wedges connected to the current node, then select a neighboring node from one of these wedges. This enables rapid mesh traversal, as it bypasses any need for additional computation or searching, given that the relevant indices are pre-determined. The primary computational demand is in the initial mesh inversion, which has a linear complexity with respect to the number of nodes.

Once navigation through the mesh is efficient, the next step is to locate the closest node to a specific point. We start from an initial, random (or strategically chosen) node and compute the distances between this point and neighboring nodes. The closest node is selected, and the algorithm iterates until it converges on the nearest node.

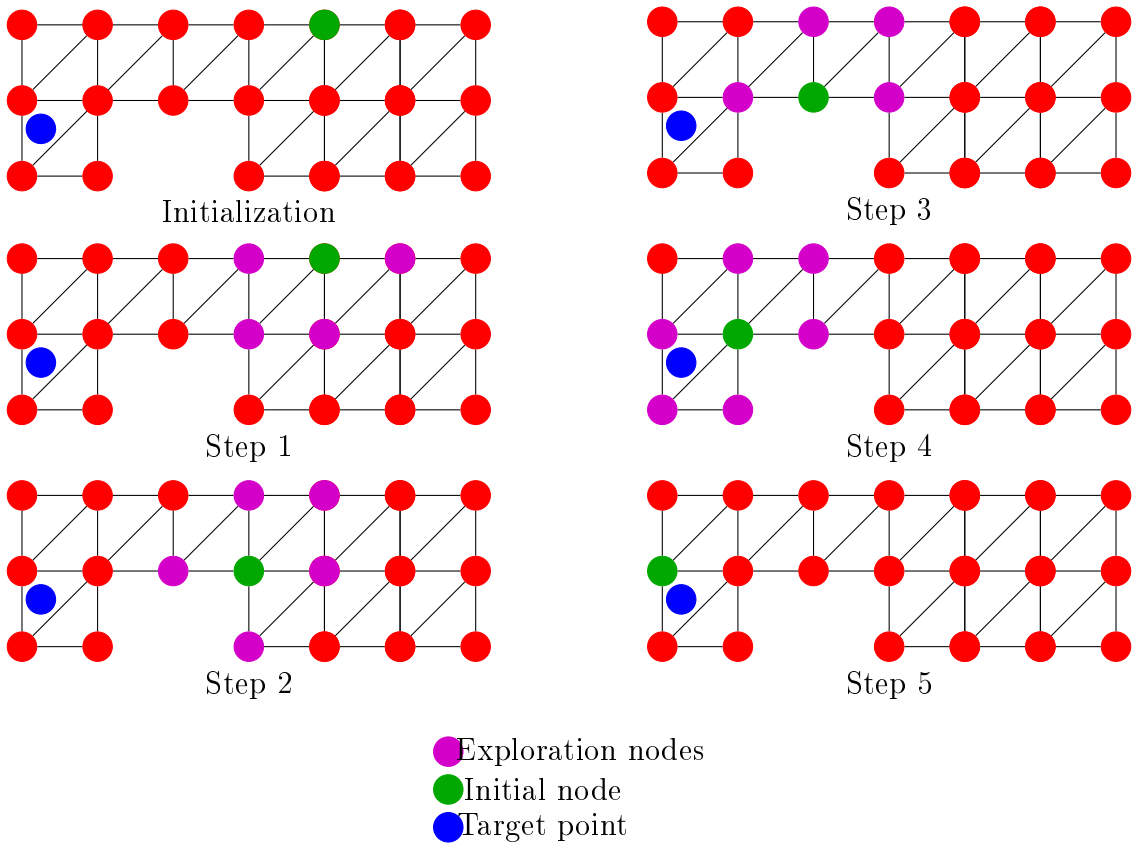


Figure 16: Example of the algorithm for finding the closest node on a simple mesh

This algorithm is guaranteed to converge only in convex regions, which is not always the case here. Therefore, it is possible for the algorithm to get stuck in the mesh. To mitigate this

issue, multiple random starting nodes are used across the mesh, increasing the probability of successful convergence.

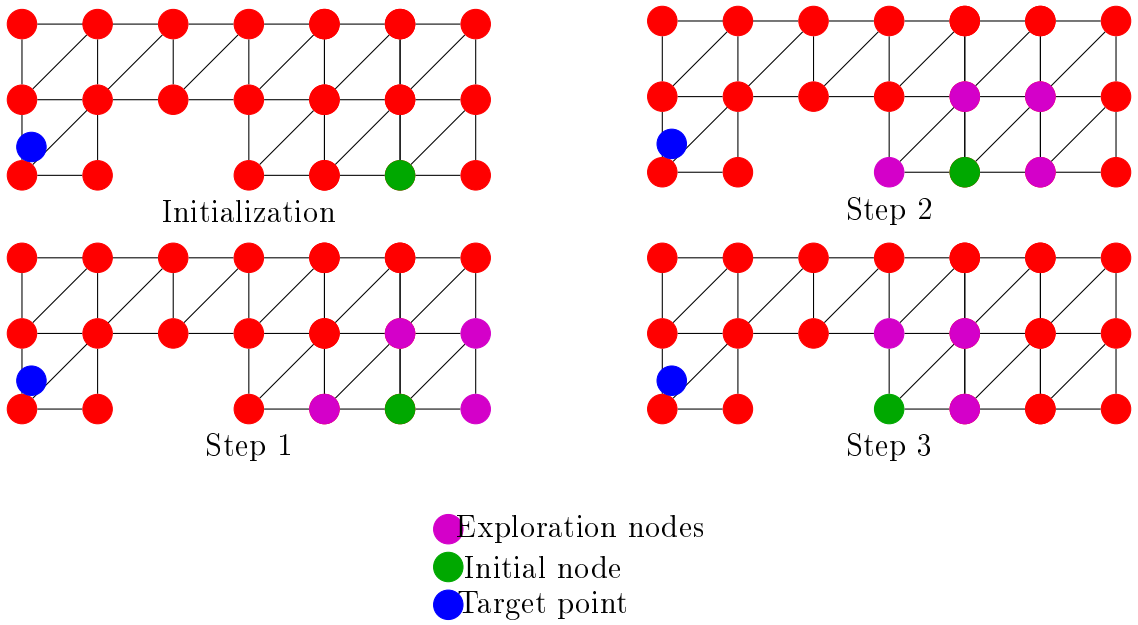


Figure 17: Example of the algorithm encountering a local minimum

With this method, we can reliably locate the closest node to the target point. To identify the element in which the point resides, the algorithm expands from this node, examining each neighboring wedge to determine whether the point is within. The first step is to establish criteria for determining whether a point is inside a wedge.

To verify if a point lies within a wedge, we calculate the point's coordinates in the local basis of the wedge by solving the following system:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 \end{pmatrix} \times \begin{pmatrix} N_1(a_1, a_2, a_3) \\ N_2(a_1, a_2, a_3) \\ N_3(a_1, a_2, a_3) \\ N_4(a_1, a_2, a_3) \\ N_5(a_1, a_2, a_3) \\ N_6(a_1, a_2, a_3) \end{pmatrix} \tag{7}$$

This system is highly nonlinear due to the nonlinearity of the shape functions, making it challenging to solve analytically or numerically. For numerical methods, convergence is particularly problematic in elements of poor quality.

To address this issue, the wedge element is subdivided into three tetrahedra, as shown below:

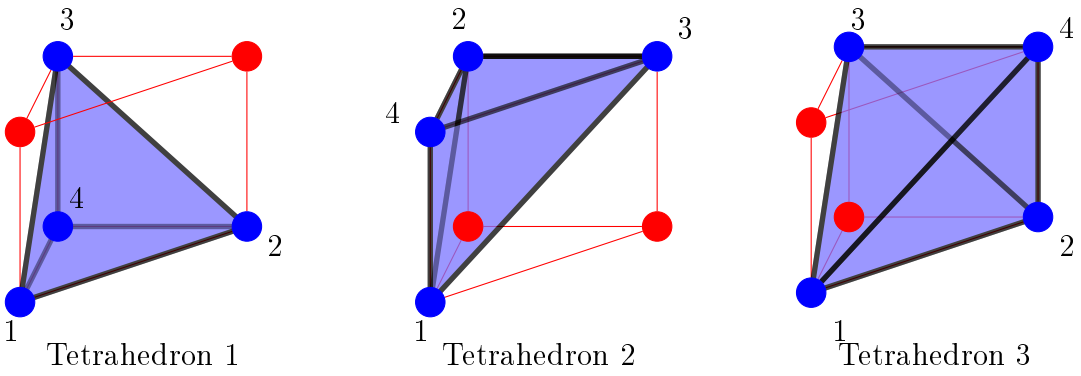


Figure 18: Decomposition of a wedge element into three tetrahedra

This decomposition is not unique, and the node labeling is also flexible. However, this particular decomposition aligns closely with the reference tetrahedral element, simplifying calculations, especially for node 4, which should align with a corner of a square face.

With this approach, if the point lies inside the wedge, it will also be within one of the tetrahedra. Once the point's coordinates within a tetrahedron are known, they can be easily translated to the wedge's base coordinates. While one could verify if the point lies within the wedge by calculating determinants, this method involves similar computational effort but does not yield the exact location within the wedge.

To compute the point's coordinates, we solve the following system:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{pmatrix} \times \begin{pmatrix} N(a_1, a_2, a_3) = a_1 \\ N(a_1, a_2, a_3) = a_2 \\ N(a_1, a_2, a_3) = a_3 \\ N(a_1, a_2, a_3) = 1 - a_1 - a_2 - a_3 \end{pmatrix} \tag{8}$$

This system can be further simplified into a linear system:

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{pmatrix} \times \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ 1 \end{pmatrix} \tag{9}$$

This linear system can be solved directly or iteratively, with iterative solvers generally yielding better results in this context.

Once the local coordinates are obtained, we can determine if the point lies within the tetrahedron by verifying that the following inequalities hold:

$$\begin{aligned} a_1 &\geq 0 \\ a_2 &\geq 0 \\ a_3 &\geq 0 \\ a_1 + a_2 + a_3 &\leq 1 \end{aligned} \tag{10}$$

The point's coordinates within the entire wedge can then be calculated using translation matrices, which map each tetrahedron's coordinates back to the wedge's local basis. The translation matrices for the three tetrahedra can be determined from the shape functions and are as follows:

$$M_{t1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad (11) \qquad M_{t2} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ -1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad (12)$$

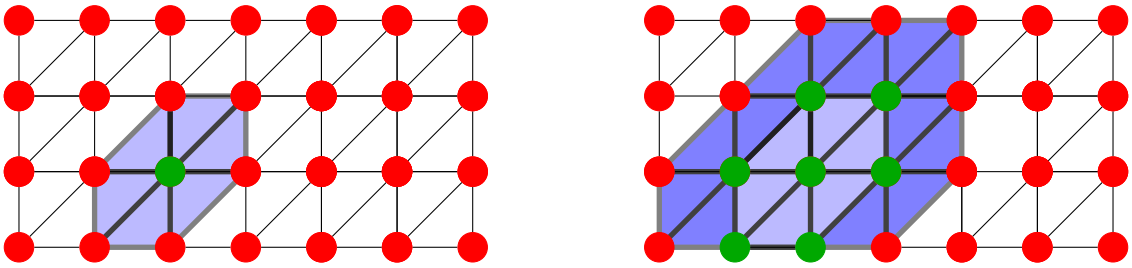
$$M_{t3} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ -1 & -1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad (13)$$

Using these matrices, we can convert coordinates (a_1, a_2, a_3) from one of the tetrahedra to the wedge's local basis as follows:

$$\begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}_{\text{Wedge}} = M_{ti} \times \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}_{\text{Tetrahedron}_i} \quad (14)$$

It should be noted that if a point is not inside any tetrahedron, it might still appear to be within the wedge when examined in the wedge's coordinate system. Therefore, the only reliable criterion for determining if a point is within an element is based on its position in the tetrahedral sub-elements.

With a clear criterion for verifying if a point is inside an element, we can now implement an algorithm to locate the wedge containing the point. This algorithm tests each wedge around the node closest to the point. Although testing all wedges in a given volume around the point is not the most efficient approach, the algorithm optimizes this process by testing wedges in order of proximity, starting from those directly connected to the nearest node and gradually expanding outward. In the first step, the wedges touching the node are tested, followed by those that connect to any node tested in the previous step.



First expansion Second expansion

Figure 19: Example of circular expansion

This approach creates an expanding search pattern in the form of a sphere around the node, which is efficient in meshes with uniform element density. However, glacier meshes often have greater density along the z-axis. To account for this, the expansion pattern is adjusted from a sphere to an oval shape. Instead of expanding from a single central node, the expansion starts in parallel from multiple nodes positioned above and below the initial node, ensuring faster convergence by adapting to mesh density variations. The ovality of the expansion pattern can be set as a ratio of the mesh density in one direction (x or y) to the density along the z-axis.

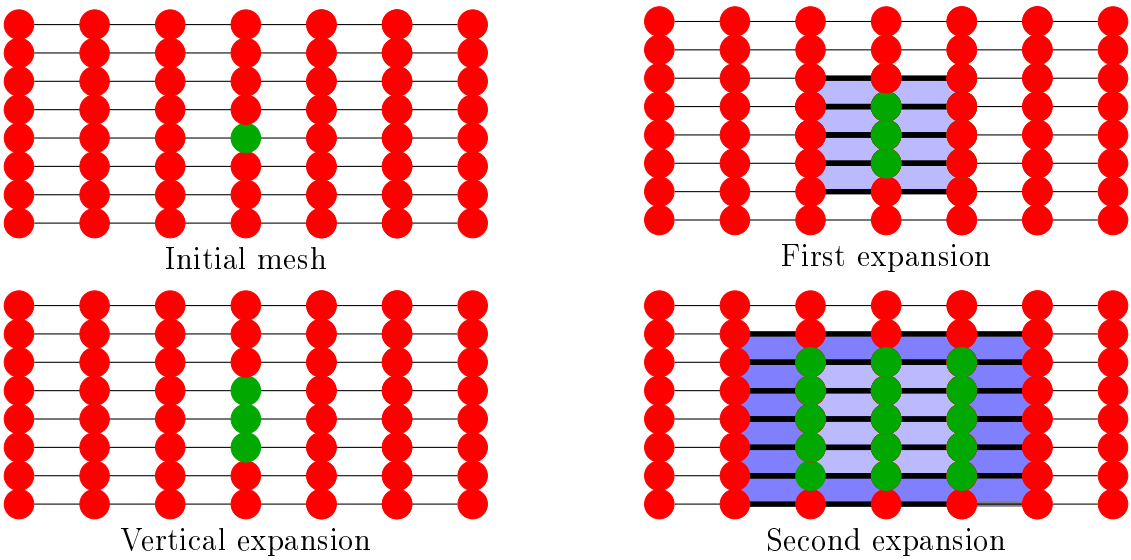


Figure 20: Example of oval expansion

Multiple expansion steps may be required, as the closest node is not always part of the wedge containing the point, as shown in the following example.

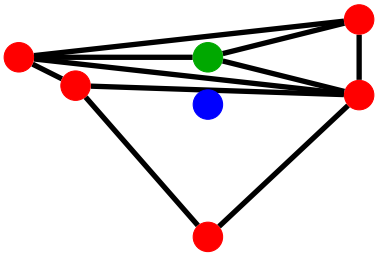


Figure 21: Example where the closest node is not part of the wedge containing the point

With the wedge containing the point identified and local coordinates known, tracking can proceed smoothly in cases where the particle remains within the mesh.

2.3 Boundary Conditions During Tracking

2.3.1 Types of Boundary Conditions Applied

This section explores the boundary conditions we aim to impose on particle tracking within the glacier model. Ideally, this section would be unnecessary if the flow simulation performed by Elmer/Ice perfectly represented the intended flow laws and boundary conditions. For example, an issue that often arises in particle tracking is that a particle may exhibit a velocity that causes it to move through the glacier bed. This should not happen, as the velocity normal to the glacier bed should theoretically be zero. This occurs because boundary conditions are not strictly enforced in the model, and because the tracking algorithm is not continuous. During the final iteration, when a particle is near the glacier surface, the normal velocity component to the boundary is not set to zero, allowing the particle to cross the boundary if the time step is too large.

To address this issue, a practical solution is to adjust the particle's position by either forcing it back within the glacier boundaries or, in some cases, allowing it to disappear.

In the following sections, we detail the boundary conditions for the glacier bed and the glacier surface. These two boundary conditions differ: at the glacier bed, the ice velocity normal to the Earth's surface is assumed to be zero, and we assume no basal melting occurs. At the glacier surface, similar boundary conditions apply, but an additional condition must account for ice accumulation and melting at different points on the glacier surface.

2.3.2 Bottom Boundary Conditions

For the boundary condition at the glacier bed, the goal is to force the particle back into the glacier to account for inaccuracies in the simulation process. The mathematical principle behind this concept is straightforward. If the normal to the bed surface is represented by $\underline{n}$ and the particle's position outside the glacier is $\underline{x_{out}}$, then we can compute the particle's position within the glacier, $\underline{x_{in}}$, as follows:

$$\underline{x_{in}} = \underline{x_{out}} - \underline{n} \cdot \underline{x_{out}} \tag{15}$$

This task becomes more complex when the bed surface is non-planar and is defined using triangular elements. To implement this projection solution in a finite element model, we first need to locate the element where the face is closest to the point $\underline{x_{out}}$. Similar to the previous section, the initial step is to find the nearest node to the point within our mesh. Once this node is identified, the algorithm will start checking if the point is inside any nearby elements. Since our point is outside the mesh, the algorithm will halt once the maximum expansion is reached. For each element checked, the algorithm calculates the distance between $\underline{x_{out}}$ and the nearest point within the element. The wedge with the smallest distance is then selected for the projection.

Once the wedge is selected, by definition, the closest point within the element to another point outside it is the projection of that point onto the element. To determine this closest point in relation to $\underline{x_{out}}$, the wedge is again divided into three tetrahedra. For each tetrahedron, we identify the nearest point, and the closest point among these is taken as the closest point within the wedge.

To calculate this closest point, we divide the space around the tetrahedron into different regions and analytically compute the projection. In Table 1, $\underline{a_{out}}$ represents the coordinates of $\underline{x_{out}}$ in the local basis of the wedge, and $\underline{a_{in}}$ represents the coordinates of the projected point $\underline{x_{in}}$.

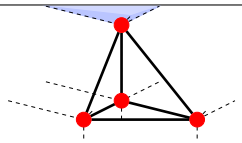
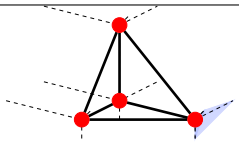
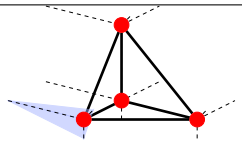
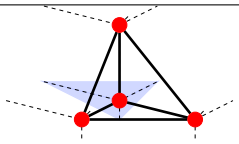
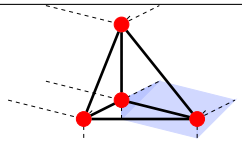
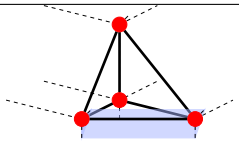
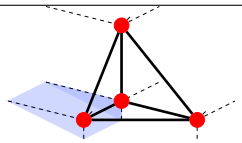
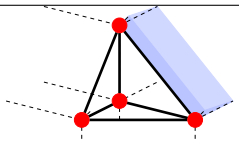
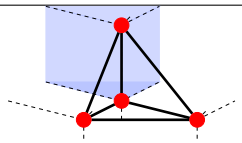
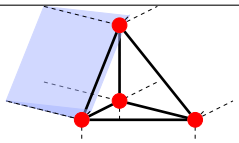
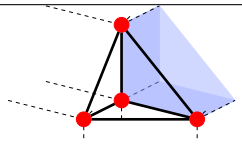
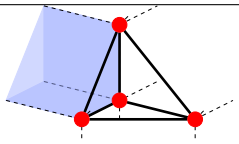
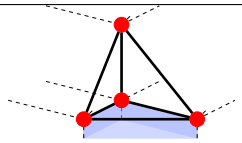
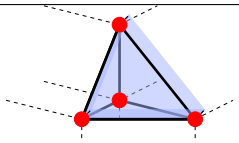
Area of Validity	Projection Formula	Area of Validity	Projection Formula
Projection to a Point			
	$\underline{a_{in}} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$		$\underline{a_{in}} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$
	$\underline{a_{in}} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$		$\underline{a_{in}} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
Projection to a Line			
	$\underline{a_{in}} = \begin{pmatrix} 0 \\ a_{2out} \\ 1 \end{pmatrix}$		$\underline{a_{in}} = \frac{1}{2} \begin{pmatrix} 1 + a_{1out} - a_{2out} \\ 1 - a_{1out} + a_{2out} \\ 0 \end{pmatrix}$
	$\underline{a_{in}} = \begin{pmatrix} a_{1out} \\ 0 \\ 0 \end{pmatrix}$		$\underline{a_{in}} = \frac{1}{2} \begin{pmatrix} 0 \\ 1 + a_{2out} - a_{3out} \\ 1 - a_{2out} + a_{3out} \end{pmatrix}$
	$\underline{a_{in}} = \begin{pmatrix} 0 \\ 0 \\ a_{3out} \end{pmatrix}$		$\underline{a_{in}} = \frac{1}{2} \begin{pmatrix} 1 + a_{1out} - a_{3out} \\ 0 \\ 1 - a_{1out} + a_{3out} \end{pmatrix}$
Projection to a Face			
	$\underline{a_{in}} = \begin{pmatrix} 0 \\ a_{2out} \\ a_{3out} \end{pmatrix}$		$\underline{a_{in}} = \begin{pmatrix} a_{1out} \\ 0 \\ a_{3out} \end{pmatrix}$
	$\underline{a_{in}} = \begin{pmatrix} a_{1out} \\ a_{2out} \\ 0 \end{pmatrix}$		$\underline{a_{in}} = \frac{1}{3} \begin{pmatrix} 1 + 2a_{1out} - a_{2out} - a_{3out} \\ 1 - a_{1out} + 2a_{2out} - a_{3out} \\ 1 - a_{1out} - a_{2out} + 2a_{3out} \end{pmatrix}$

Table 1: Table of projection of a point inside a tetrahedron

Now that we have the coordinates of the projected point in the local basis, we need to convert them to global coordinates to compute the distance to the initial point $\underline{x_{out}}$, using Equation 8. Once we obtain $\underline{x_{in}}$, we can compute the distance d as:

$$d = ||\underline{x_{in}} - \underline{x_{out}}|| \tag{16}$$

This distance is calculated for all wedges around the closest node, and the wedge with the smallest d is selected. The point for the next iteration is set to $\underline{x_{in}}$. This method enables efficient reprojection of a point into the mesh, even in complex and intricate meshes, and provides the local coordinates in the element's basis (derived initially in the tetrahedral basis but converted to the wedge basis via matrix multiplication).

As the particle approaches the glacier bed, where the velocity is predominantly normal to the

surface, it may end up being reprojected repeatedly to nearly the same location. In such cases, it becomes inefficient to continue tracking, as the particle provides no new information and consumes computational resources.

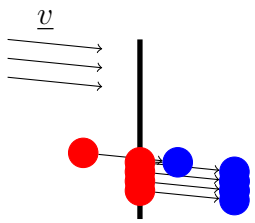


Figure 22: Example of a particle repeatedly reprojected near its initial location

In such cases, a criterion can be introduced where the effective velocity (the norm of the distance between the initial and final positions after reprojection, divided by the time step) is compared to a threshold, such as the minimum velocity in the mesh. If this effective velocity is below the threshold, the particle can be removed from the simulation.

2.3.3 Top Boundary Conditions

The top boundary layer of a glacier can be treated similarly to the bottom layer in terms of particle reprojection. Particles that exit the mesh at the top need to be reprojected back into the glacier. This is particularly important because, in flow simulations, the top surface is defined as a free surface, allowing non-zero normal velocity. The reprojection process follows the same steps as for the bottom boundary, and particles with low velocity are removed in a similar manner.

The main addition for the top boundary layer is that particles are removed if they are in the ablation area and the normal component of their velocity is too high. To implement this, we first define the ablation and accumulation zones, which can be done by setting an altitude threshold that separates the two regions.

Once the altitude threshold is defined, two cases arise. In the accumulation area, if a particle exits the glacier, it is reprojected back into the mesh as before. In the ablation area, however, the normal component of the particle's velocity is calculated as follows:

$$v_{\%} = \frac{\underline{x_{n+1_{out}}} - \underline{x_n}}{||\underline{x_{n+1_{out}}} - \underline{x_n}||} \cdot \frac{\underline{x_{n+1_{out}}} - \underline{x_{n+1_{in}}}}{||\underline{x_{n+1_{out}}} - \underline{x_{n+1_{in}}}||} \tag{17}$$

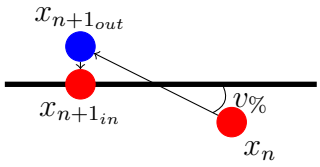


Figure 23: Calculation of particle angle with respect to the surface

If the velocity percentage $v_{\%}$ exceeds a specified threshold, the particle is removed. Allowing particles to continue their path in the ablation area when the angle is low helps mitigate sensitivity to surface roughness in the mesh. Furthermore, setting this threshold to prevent any particle melting enables the simulation of rock dynamics trapped within the ice, offering additional applications for the algorithm.

3 Extraction of Data Along a Flow Line

The particle tracking algorithm enables precise prediction of the trajectory of any given initial particle, which provides valuable insight into the glacier's internal flow. However, tracking data alone is insufficient for some purposes, such as determining the cumulative deformation of a particle, which offers clues about the final ice texture. To obtain this information, additional parameters need to be integrated alongside the trajectory.

3.1 Computing the Strain Rate

Since Elmer/Ice does not compute the strain rate by default, it must be derived from the velocity field. By definition, we know that:

$$\underline{\underline{\dot{\mathbb{E}}}} = \frac{1}{2}\left(\underline{\underline{grad}}(\underline{v}) + \underline{\underline{grad}}(\underline{v})^T\right) \tag{18}$$

To calculate the gradient of the velocity field, we first compute the derivatives of the shape functions, which can be expressed in matrix form as follows:

$$D_N(a_1,a_2,a_3) = \begin{pmatrix} \frac{\partial N_1}{\partial a_1} & \frac{\partial N_1}{\partial a_2} & \frac{\partial N_1}{\partial a_3} \\ \frac{\partial N_2}{\partial a_1} & \frac{\partial N_2}{\partial a_2} & \frac{\partial N_2}{\partial a_3} \\ \frac{\partial N_3}{\partial a_1} & \frac{\partial N_3}{\partial a_2} & \frac{\partial N_3}{\partial a_3} \\ \frac{\partial N_4}{\partial a_1} & \frac{\partial N_4}{\partial a_2} & \frac{\partial N_4}{\partial a_3} \\ \frac{\partial N_5}{\partial a_1} & \frac{\partial N_5}{\partial a_2} & \frac{\partial N_5}{\partial a_3} \\ \frac{\partial N_6}{\partial a_1} & \frac{\partial N_6}{\partial a_2} & \frac{\partial N_6}{\partial a_3} \end{pmatrix} = \begin{pmatrix} \frac{1}{2}(1-a_3) & 0 & -\frac{1}{2}a_1 \\ 0 & \frac{1}{2}(1-a_3) & -\frac{1}{2}a_2 \\ -\frac{1}{2}(1-a_3) & -\frac{1}{2}(1-a_3) & -\frac{1}{2}(1-a_1-a_2) \\ \frac{1}{2}(1+a_3) & 0 & \frac{1}{2}a_1 \\ 0 & \frac{1}{2}(1+a_3) & \frac{1}{2}a_2 \\ -\frac{1}{2}(1+a_3) & -\frac{1}{2}(1+a_3) & \frac{1}{2}(1-a_1-a_2) \end{pmatrix} \tag{19}$$

Once the shape function derivatives are computed, the Jacobian for the complete wedge can be derived as:

$$\underline{\underline{J}} = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \\ z_1 & z_2 & z_3 & z_4 & z_5 & z_6 \end{pmatrix} \times D_N(a_1,a_2,a_3) \tag{20}$$

With the Jacobian computed, it can then be numerically inverted, allowing the velocity gradient to be calculated as follows:

$$\underline{\underline{grad}}(\underline{v}) = D_N(a_1,a_2,a_3) \times \underline{\underline{J}}^{-1} \times \begin{pmatrix} \underline{v_1} & \underline{v_2} & \underline{v_3} & \underline{v_4} & \underline{v_5} & \underline{v_6} \end{pmatrix} \tag{21}$$

By implementing these equations, the strain rate field can be computed at each node, enabling a comprehensive understanding of strain rates throughout the model.

3.2 Computing the Flow Line Local Basis Rotations

The objective in this section is to compute the rotation matrix between the main reference frame and the local frame of reference associated with the particle. First, we define the rotation angles between the trajectory and the reference frame, allowing us to create a rotation matrix that facilitates transformations between the two frames.

For the rotation angles, we will use the Euler 321 sequence, as it is straightforward to implement in our case. The tangent vector to the flow line can be calculated by normalizing the vector between two consecutive points:

$$\underline{t_n} = \frac{\underline{x_{n+1}} - \underline{x_n}}{||\underline{x_{n+1}} - \underline{x_n}||} \tag{22}$$

When performing linear interpolation as shown here, this method utilizes all available data.

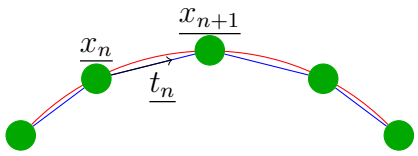


Figure 24: Tangent to the path

The rotation matrices associated with the 321 sequence in this case are:

$$\begin{aligned} R_1 &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{pmatrix} \\ R_2 &= \begin{pmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \\ R_3 &= \begin{pmatrix} \cos(\psi) & \sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{23}$$

Since we only have the tangent to the flow line, we can determine only two rotations, as any rotation around the tangential vector has no effect. To simplify, we assume $\phi = 0$. This assumption can be verified for ice flow simulations by observing that particles around a central one do not orbit. With this approximation, the rotation matrix becomes:

$$\begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\sin(\psi) & \cos(\psi) & 0 \\ \sin(\theta)\cos(\psi) & \sin(\theta)\sin(\psi) & \cos(\theta) \end{pmatrix} \tag{24}$$

If we want the tangential vector to be a rotation of the vector $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, we can express this rotation as:

$$\begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} = \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\sin(\psi) & \cos(\psi) & 0 \\ \sin(\theta)\cos(\psi) & \sin(\theta)\sin(\psi) & \cos(\theta) \end{pmatrix} \times \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{25}$$

From this, the angles can be calculated as follows:

$$\begin{aligned} \psi &= -\sin^{-1}(t_y) \\ \theta &= \tan^{-1}\left(\frac{t_x}{t_z}\right) \end{aligned} \tag{26}$$

After calculating the angles, they are filtered to ensure a continuous motion of the reference frame, as particle tracking does not always guarantee continuity.

We can now create rotation matrices to switch between the global and local frames of reference:

$$\begin{aligned} R_{\text{global}\to\text{local}} &= \begin{pmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ -\sin(\psi) & \cos(\psi) & 0 \\ \sin(\theta)\cos(\psi) & \sin(\theta)\sin(\psi) & \cos(\theta) \end{pmatrix} \\ R_{\text{local}\to\text{global}} &= \begin{pmatrix} \cos(\theta)\cos(\psi) & -\sin(\psi) & \sin(\theta)\cos(\psi) \\ \cos(\theta)\sin(\psi) & \cos(\psi) & \sin(\theta)\sin(\psi) \\ -\sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \end{aligned} \tag{27}$$

Now, if we have a tensor in the global basis, such as the strain rate tensor, it can be converted to the local basis as follows:

$$\underline{\underline{\mathbb{E}_{\text{local}}}} = R^T_{\text{global}\to\text{local}} \times \underline{\underline{\mathbb{E}_{\text{global}}}} \times R_{\text{global}\to\text{local}} \tag{28}$$

3.3 Integration of the Strain

Once the strain in the local basis is computed, it can be integrated. However, this integration cannot be performed step-by-step as it would be for small deformations, because the following formula is only valid under small deformation assumptions:

$$\underline{\underline{\mathbb{E}}}_{t+1} = \mathbb{E}_t + \dot{\underline{\underline{\mathbb{E}}}}(t)\, dt \tag{29}$$

3.3.1 Method for Strain Integration

To accurately compute strain in the context of high deformations, we return to the definition of strain. The strain tensor $\mathbb{E}$ can be expressed in terms of the deformation tensor as follows:

$$\underline{\underline{\mathbb{E}}} = \frac{1}{2}\left(\underline{\underline{\mathbb{F}}}^T \times \underline{\underline{\mathbb{F}}} - \mathbb{I}\right) \tag{30}$$

The deformation tensor, $\underline{\underline{\mathbb{F}}}$, can be defined in terms of an initial vector $\underline{dl_0}$ and its deformed state $\underline{dl}$:

$$\underline{dl} = \underline{\underline{\mathbb{F}}}\, \underline{dl_0} \tag{31}$$

If we consider multiple deformation steps applied sequentially, then we have:

$$\begin{aligned} \underline{dl_1} &= \underline{\underline{\mathbb{F}}}_1\, \underline{dl_0} \\ \underline{dl_2} &= \underline{\underline{\mathbb{F}}}_2\, \underline{dl_1} \\ &\vdots \\ \underline{dl_{n-1}} &= \underline{\underline{\mathbb{F}}}_{n-1}\, \underline{dl_{n-2}} \\ \underline{dl_n} &= \underline{\underline{\mathbb{F}}}_n\, \underline{dl_{n-1}} \end{aligned} \tag{32}$$

These steps can be combined to obtain:

$$\underline{dl_n} = \prod_{i=1}^{n} \underline{\underline{\mathbb{F}}}_i\, \underline{dl_0} \tag{33}$$

By rearranging the definition of the strain tensor, we arrive at:

$$\underline{\underline{\mathbb{F}}}_{\text{tot}} = \prod_{i=1}^{n} \underline{\underline{\mathbb{F}}}_i = \prod_{i=1}^{n} \sqrt{2\underline{\underline{\mathbb{E}}}_i + \mathbb{I}} \tag{34}$$

Injecting this relation into the strain definition gives the total strain:

$$\underline{\underline{\mathbb{E}}}_{\text{tot}} = \frac{1}{2}\left(\prod_{i=1}^{n}(2\underline{\underline{\mathbb{E}}}_i + \mathbb{I}) - \mathbb{I}\right) \tag{35}$$

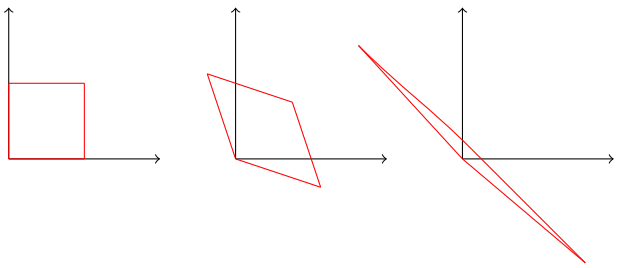
The strain tensor can also be expressed in terms of the strain rate tensor:

$$\underline{\underline{\mathbb{E}}}_{\text{tot}} = \frac{1}{2}\left(\prod_{i=1}^{n}(2\dot{\underline{\underline{\mathbb{E}}}}_i\, dt + \mathbb{I}) - \mathbb{I}\right) \tag{36}$$

This method allows us to compute the total strain of an ice particle under high deformation conditions. Additionally, the eigenvalues and eigenvectors of this matrix can be calculated to provide further insight into the deformation characteristics.

3.3.2 Method for High Precision in the Eigenvalues of Strain

A challenge remains with this method when applied numerically, especially under conditions of very high deformation. This issue arises in two key situations when attempting to use the integrated strain results. First, when calculating the volume deformation of the particle, the results are often inaccurate, despite our prior knowledge that the volume should remain constant, as we assume ice to be incompressible. Second, when computing the eigenvalues of the strain, we often find that only the first eigenvalue retains high precision, while the others may be significantly inaccurate. This problem occurs because the particle undergoes substantial deformation, which, from a mathematical perspective, means that calculating the eigenvalues involves subtracting large values to obtain small, precise results—a process that requires an impractical level of numerical precision in the strain tensor.



Initial particule Deformed particule Final deformation

Figure 25: Very high deformation of the particle

To address this, we need a method that integrates strain along the flow path while diagonalizing at each step to maintain precision and preserve incompressibility.

Starting from the final formula derived previously, we construct a matrix P_{tot}, representing the basis in which the total deformation tensor $\underline{\underline{\mathbb{F}}}_{tot}$ is diagonal. With this, we can express $\underline{\underline{\mathbb{F}}}_{tot}$ as:

$$\underline{\underline{\mathbb{F}}}_{tot} = P_{tot}\, \underline{\underline{\mathbb{F}}}^{d}_{tot}\, P^{T}_{tot} = \prod_{i=1}^{n} \underline{\underline{\mathbb{F}}}_i \tag{37}$$

Instead of using the matrix inverse, we use its transpose here, since $\underline{\underline{\mathbb{F}}}_{tot}$ is symmetric and real, making it orthogonally diagonalizable.

Additionally, we diagonalize the incremental strain tensors $\underline{\underline{\mathbb{F}}}_i$ with their respective transformation matrices P_i as follows:

$$\underline{\underline{\mathbb{F}}}_i = P_i \, \underline{\underline{\mathbb{F}}}_i^d \, P_i^T \tag{38}$$

This result can then be substituted into the previous formula:

$$\underline{\underline{\mathbb{F}}}_{tot} = P_{tot} \, \underline{\underline{\mathbb{F}}}_{tot}^d \, P_{tot}^T = \prod_{i=1}^{n} P_i \, \underline{\underline{\mathbb{F}}}_i^d \, P_i^T \tag{39}$$

Rewriting this formula iteratively, we obtain:

$$P_{tot_{i+1}} \, \underline{\underline{\mathbb{F}}}_{tot_{i+1}}^d \, P_{tot_{i+1}}^T = P_{tot_i} \, \underline{\underline{\mathbb{F}}}_{tot_i}^d \, P_{tot_i}^T \, \underline{\underline{\mathbb{F}}}_i \tag{40}$$

By multiplying on the right side by $\mathbb{I} = P_{tot_i} P_{tot_i}^T$, we get:

$$P_{tot_{i+1}} \, \underline{\underline{\mathbb{F}}}_{tot_{i+1}}^d \, P_{tot_{i+1}}^T = P_{tot_i} \, \underline{\underline{\mathbb{F}}}_{tot_i}^d \, P_{tot_i}^T \, \underline{\underline{\mathbb{F}}}_i \, P_{tot_i} P_{tot_i}^T \tag{41}$$

The remaining task is to diagonalize the product $\underline{\underline{\mathbb{F}}}_{tot_i}^d (P_{tot_i}^T \underline{\underline{\mathbb{F}}}_i P_{tot_i})$, which consists of a diagonal matrix and a symmetric matrix with real values. This is feasible since $\underline{\underline{\mathbb{F}}}$ is symmetric, and the change of basis preserves this property. Additionally, the determinant of each $\underline{\underline{\mathbb{F}}}$ tensor should be one (reflecting volume conservation), so the determinant of $(P_{tot_i}^T \underline{\underline{\mathbb{F}}}_i P_{tot_i})$ is also one.

To diagonalize the product of a diagonal matrix D and a symmetric matrix S without directly multiplying them (to avoid loss of precision), we can calculate the characteristic polynomial of their product. For a 3x3 matrix, one form of the characteristic polynomial is:

$$X^3 - \mathrm{tr}(DM)X^2 - \frac{1}{2}\left(\mathrm{tr}((DM)^2) - \mathrm{tr}(DM)^2\right) X - \det(DM) \tag{42}$$

Applying this with the previous assumptions, we obtain:

$$a = \mathrm{tr}(DM) = \lambda_1 + \lambda_2 + \lambda_3 = d_{11}m_{11} + d_{22}m_{22} + d_{33}m_{33} \tag{43}$$

$$b = \frac{1}{2}\left(\mathrm{tr}((DM)^2) - \mathrm{tr}(DM)^2\right) = \lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1 \tag{44}$$

$$= d_{11}d_{22}(m_{11}m_{22} - m_{12}^2) + d_{11}d_{33}(m_{11}m_{33} - m_{13}^2) + d_{33}d_{22}(m_{33}m_{22} - m_{32}^2) \tag{45}$$

$$1 = \det(DM) = \lambda_1\lambda_2\lambda_3 \tag{46}$$

Thus, the problem reduces to finding the roots of the following polynomial:

$$X^3 - aX^2 + bX - 1 \tag{47}$$

This polynomial generally takes the following shape in our case:

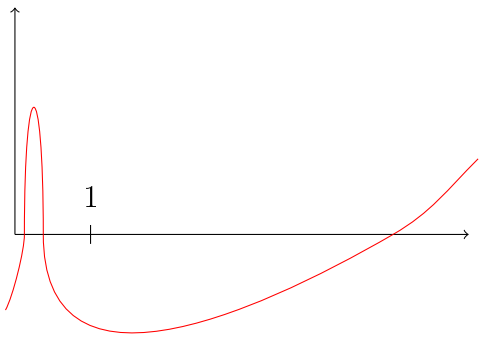


Figure 26: Characteristic polynomial of the matrix DM

However, finding the roots of this polynomial can still lead to low precision in the smaller eigenvalues, as the polynomial is quite steep around the two largest eigenvalues. To minimize the influence of the largest eigenvalue on the two smaller ones, we reduce the polynomial's order by treating the X^3 term as negligible in the interval $[0, 1]$. This simplification yields:

$$X^2 - \frac{b}{a}X + \frac{1}{a} \tag{48}$$

The ratio $\frac{b}{a}$ can be computed analytically, simplifying under the assumptions $d_{11}m_{11} \gg d_{22}m_{22}$ and $d_{11}m_{11} \gg d_{33}m_{33}$, which hold since all values of M are of similar order and d_{11} is the highest eigenvalue:

$$\frac{b}{a} = \frac{d_{11}d_{22}(m_{11}m_{22} - m_{12}^2) + d_{11}d_{33}(m_{11}m_{33} - m_{13}^2) + d_{33}d_{22}(m_{33}m_{22} - m_{32}^2)}{d_{11}m_{11} + d_{22}m_{22} + d_{33}m_{33}} \tag{49}$$

$$\approx d_{22}\left(m_{22} - \frac{m_{12}^2}{m_{11}}\right) + d_{33}\left(m_{33} - \frac{m_{13}^2}{m_{11}}\right) \tag{50}$$

With this approach, we can compute the lower eigenvalues with high precision, as this simplified polynomial maintains the same lower eigenvalues but has a gentler gradient near the roots.

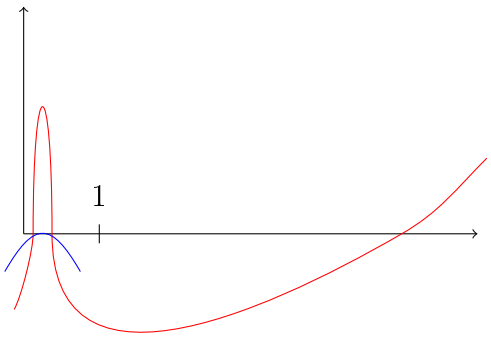


Figure 27: Characteristic polynomial of matrix DM and approximation for smaller roots

This approach addresses cases where one eigenvalue is very large and the other two are relatively small, representing a particle deforming into a needle shape. For cases where the particle deforms into a sheet shape, resulting in one small eigenvalue and two large ones, the function behaves as follows:

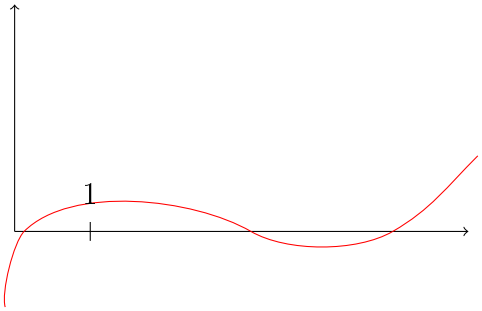


Figure 28: Characteristic polynomial with one small eigenvalue

In this case, we apply a similar simplification based on our new assumptions, yielding the following polynomial for root finding:

$$X - \frac{1}{b} \tag{51}$$

This is a first-order approximation of our polynomial, visualized below:

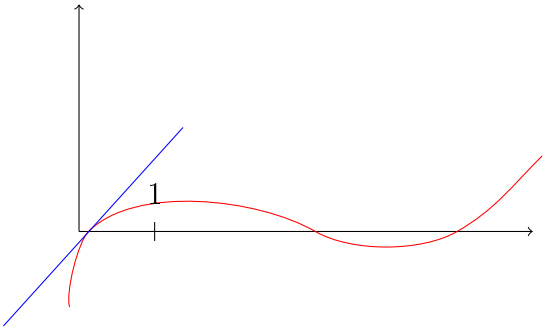


Figure 29: Characteristic polynomial with one small eigenvalue

Since the choice between methods depends on the assumptions' validity and applies only under high deformation, we compute the roots of the original polynomial using all methods and then reinsert the results into the polynomial. The root closest to zero is selected, ensuring that when deformation resembles a needle, the smaller roots are from the second-order polynomial, and when deformation resembles a sheet, the smaller roots are from the first-order polynomial.

The remaining task is to compute the eigenvectors. To avoid computing the inverse of the basis change matrix, we find an orthogonal basis of unitary norm eigenvectors.

First, we find the eigenvectors with unitary norm by solving:

$$DMX - \lambda_i X + (||X|| - 1)\mathbb{I} = 0 \tag{52}$$

Once the three eigenvectors are obtained, we use the QR decomposition algorithm to orthogonalize our matrix of eigenvectors. This decomposition yields a matrix P such that $PP^T = \mathbb{I}$ and $P\underline{\underline{\mathbb{F}}}^d_{tot_i}(P^T_{tot_i}\underline{\underline{\mathbb{F}}}_i P_{tot_i})P$ is a diagonal matrix.
We then set:

$$\begin{aligned} P_{tot_{i+1}} &= P_{tot_i} P \\ \underline{\underline{\mathbb{F}}}^d_{tot_{i+1}} &= \underline{\underline{\mathbb{F}}}^d_{tot_i}(P^T_{tot_i}\underline{\underline{\mathbb{F}}}_i P_{tot_i}) \end{aligned} \tag{53}$$

This process can be run iteratively until the final step, where we obtain the strain tensor in its diagonal basis, along with the corresponding eigenvectors.

4 Implementation of the Particle Tracking Algorithm in a Python Library

With the theoretical basis for particle tracking established, the method can now be implemented in a user-friendly Python library designed to model particle flow paths within a glacier. This Python library processes output results from Elmer/Ice in either vtu (single-core simulation) or pvtu (parallelized simulation) file formats. It opens these files, tracks user-specified particles, performs operations on fields tracked along the flow path, and includes a visualization interface for examining various scalar and vector fields.

This tool is available to everyone on IGE's GitLab repository (Martin, 2024).

4.1 Mesh Tools

In this section, we describe the tools available for working with different meshes and their various uses.

The first tool is a function that converts pvtu files to vtu file types. The pvtu file format is used for parallelized simulations, where the pvtu file references multiple files, each representing a different part of the mesh, along with the relationships between boundary nodes. Converting these multiple parallelized files into a single vtu file standardizes inputs and speeds up the program, as working with multiple files and meshes is time-consuming. Additionally, the parallelization options chosen for this library do not depend on mesh partitioning.

Other mesh-related tools include a function to invert the mesh (establishing node-to-wedge relationships), a parallelized tool to compute the strain rate at each node, and a tool to compute strain at each node based on Glen's law. Additional tools allow users to select nodes within a specified area or volume, facilitating access to starting nodes for the particle tracking algorithm. There is also a function to save modifications to the mesh, enabling users to add data (such as strain and stress) that was not computed during the flow simulation. Lastly, several internal tools support these functions, though they are not intended for direct user interaction.

4.2 Tracking Tool

Once the mesh has been imported, the tracking tools can be utilized. The first tool allows the user to track particles in the velocity field. This program is parallelized at the particle level, with each particle representing a separate instance, as they are independent of one another. The core of this algorithm is written in C++ for enhanced speed, while an internal interface between C++ functions and Python functions makes the process seamless and transparent to the user. Notably, tracking can be performed forward or backward in time, allowing users to place particles at the end of their trajectories and follow them backward. If particle tracking is done in reverse time, a function reverses all tracked fields to restore chronological order.

Additional tools allow users to add fields to a particle's data, such as calculating the particle's age along the path, determining deformation eigenvalues and eigenvectors, and computing various anisotropy factors. Mathematical tools are also available to decompose a tensor field into its primary components. Since each particle is represented as a dictionary, data extraction for further post-processing or simplified representation is straightforward.

The tool also supports transient particle tracking simulations. By providing multiple meshes to the code, it can automatically switch from one velocity field to another at the correct time for each particle. This method, while straightforward, efficiently handles changing velocities over time. Although linear or higher-order velocity interpolation could be implemented, it is generally unnecessary since velocity variations between simulations are minimal.

Finally, computed tracking data can be saved by the user for use in other software (e.g., files are exported in a txt format and can be opened with spreadsheet applications such as Excel).

4.3 Representation Tools

The final main capability of IceTrackPy is its ability to represent the different fields tracked along particle trajectories. To achieve this, it uses a GPU-accelerated library named Vispy, which enables fast and smooth rendering of complex visual elements, ideal for representing intricate glaciological data. Vispy allows for both interactive 3D plotting and 2D projections, the latter being more suitable for inclusion in written reports.

4.3.1 Visualizing Data Along Particle Trajectories

The first step is to represent the mesh. A convenient function automatically plots the mesh, displaying only the top and bottom layers. This approach avoids overlaying excessive information, making the visualizations clearer.

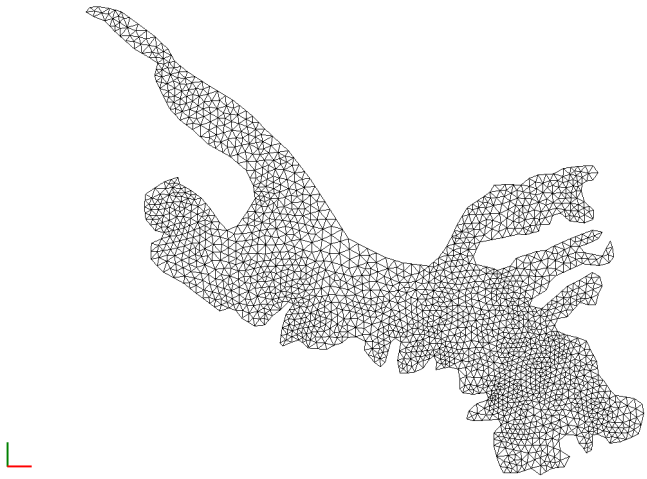


Figure 30: Mesh of the glacier in a 2D projection

Next, particle trajectories can be added. This can be done in a solid color or using different fields to visualize data along each trajectory at each time step.

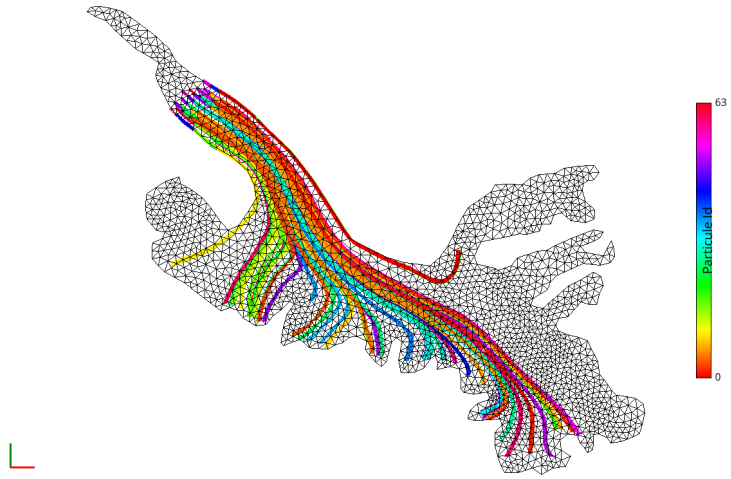


Figure 31: Representation of particle trajectories with different colors for each particle

If the user wishes to plot a scalar field, a color scale can be added to help visualize the data and understand its range.

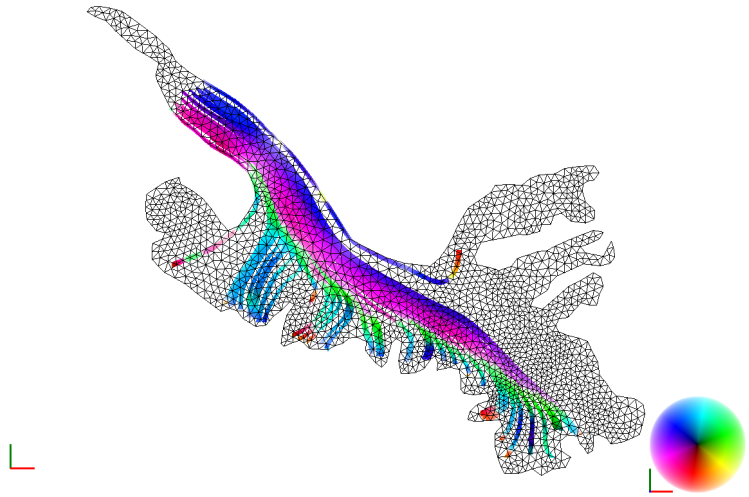
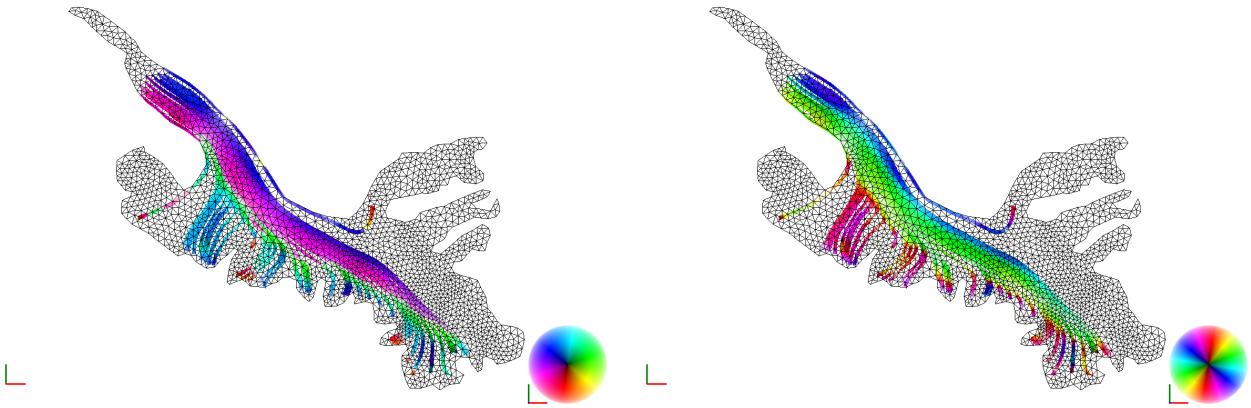


Figure 32: Representation of a scalar value (ice age) along the trajectories

For vector fields of unitary norm, a color wheel can be used, where each color represents an orientation. This allows the orientation of the vector to be visualized along its path.



(a) With non-symmetric color wheel (b) With symmetric color wheel

Figure 33: Representation of a vector (first eigenvector) along the trajectories

4.3.2 Visualizing Data on a Surface

Another option for data visualization is to plot data on a surface, such as the surface where particles emerge from the ice at the glacier's terminus. This approach is particularly useful for identifying areas with the greatest diversity in anisotropy, for example. To achieve this, the reverse-time tracking functionality can be utilized. By strategically placing points, a 2D mesh of the surface can be created and used to represent data, as shown below:

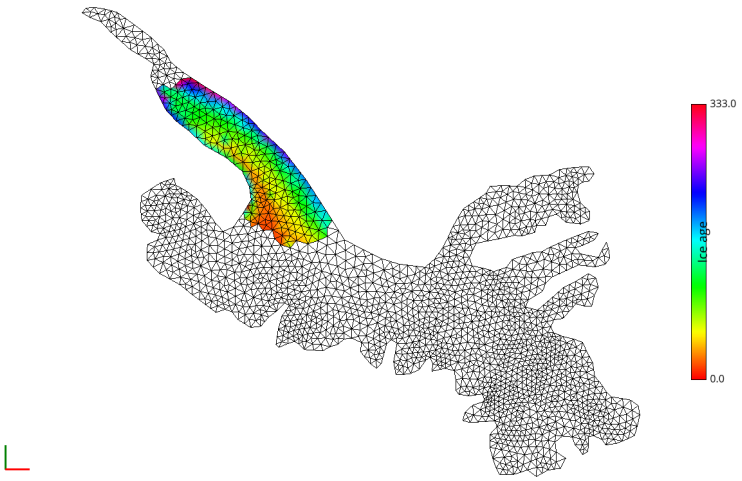
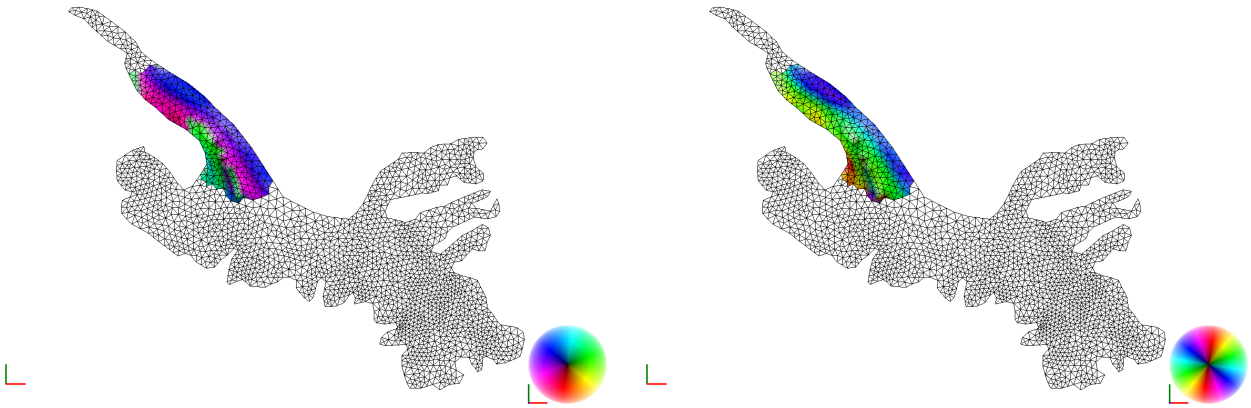


Figure 34: Representation of a scalar value (ice age) on a surface

This function can also be used to represent vector fields, such as the orientation of principal deformation for each particle. The vector representation offers various symmetries depending on the chosen color map, so an option has been added to adjust the orientation of the representation. This allows users to align the color map's symmetry with, for instance, the symmetry of the glacier, enhancing the effectiveness of the color mapping.



(a) With non-symmetric color wheel

(b) With symmetric color wheel

Figure 35: Representation of a vector (first eigenvector) at the end of the particle's path

4.4 Creation of a Test Case

To verify and compare the functionality of IceTrackPy with other commercial solutions, a test case has been created. This test case allows for the validation of each component of the code with a known solution, enabling users to evaluate various features in a flexible and straightforward context. This simplifies debugging and testing processes.

The test must satisfy several criteria: it must respect all physical laws implemented in the particle tracking software, such as incompressibility; it should represent glacier flow characteristics, including particle accumulation at the top and melting at the bottom; and it must have a fully analytical solution to facilitate direct comparison with the tracking software. Additionally, the mesh should be structured similarly to how it is implemented in Elmer/Ice so that IceTrackPy can open this test case.

4.4.1 Velocity Field Justification

To meet the above criteria, the following velocity field was chosen:

$$\underline{v}(x,y,z,t) = \begin{pmatrix} v_x \\ -y\frac{v_z}{H}\left(1-\frac{2x}{L}\right) \\ z\frac{v_z}{H}\left(1-\frac{2x}{L}\right) \end{pmatrix} \tag{54}$$

This simple velocity field allows us to model all the different aspects used by IceTrackPy. The field appears as follows:

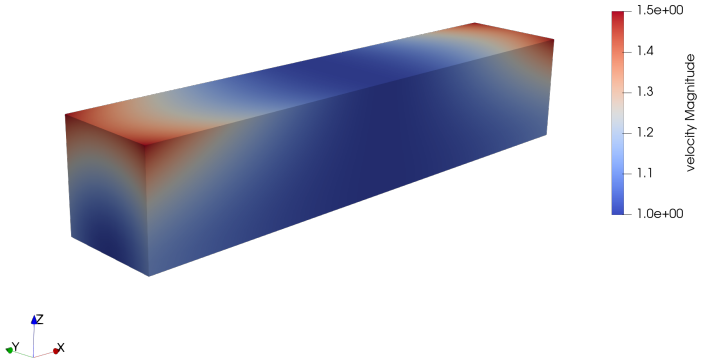
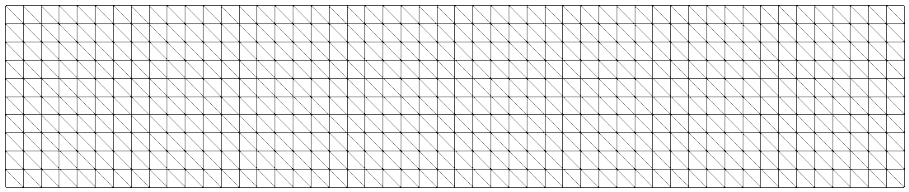
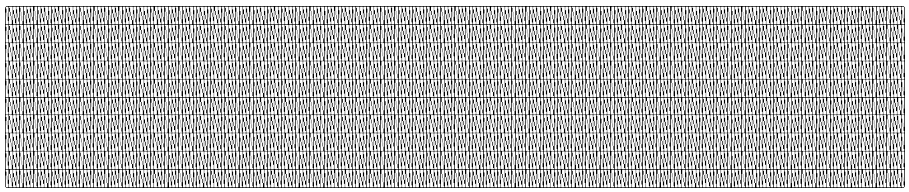


Figure 36: Velocity field norm

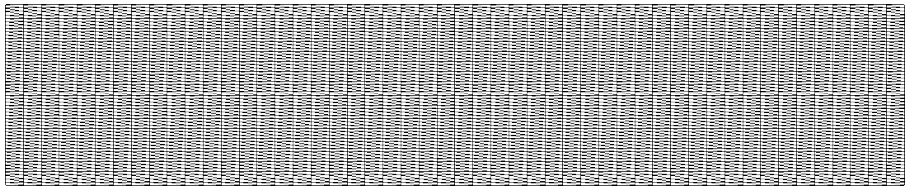
This velocity field is then applied to a mesh created in a similar manner to an Elmer/Ice mesh. The element shape and density can be adjusted to explore different settings for particle tracking.



(a) 1 element per meter for x, y, z (1/1/1)



(b) 1 element per meter for y, z and 5 elements per meter for x (5/1/1)



(c) 1 element per meter for x, z and 5 elements per meter for y (1/5/1)

Figure 37: Different meshes with varying densities in different directions

4.4.2 Integration of the Different Fields

Now that the velocity field is defined, it can be integrated to find the particle trajectories, resulting in:

$$\Phi(x_0, y_0, z_0, t) = \begin{cases} x(t) = v_x t + x_0 \\ y(t) = y_0 e^{\left(-v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} \\ z(t) = z_0 e^{\left(v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} \end{cases} \tag{55}$$

The next task is to compute the strain rate field. To do this, the first step is calculating the gradient of the velocity field as follows:

We can calculate the strain rate from the previously computed values, knowing that:

$$\underline{\underline{\mathbb{L}}} = \underline{\underline{grad}}(v) = \begin{pmatrix} 0 & 0 & 0 \\ \frac{2v_x y}{HL} & -\frac{v_z}{H}\left(1 - \frac{2x}{L}\right) & 0 \\ -\frac{2v_z z}{HL} & 0 & \frac{v_z}{H}\left(1 - \frac{2x}{L}\right) \end{pmatrix} \tag{56}$$

Using the definition of the strain rate:

$$\dot{\underline{\underline{\mathbb{E}}}} = \frac{1}{2}(\underline{\underline{\mathbb{L}}} + \underline{\underline{\mathbb{L}}}^T) \tag{57}$$

we can compute the strain rate as:

$$\dot{\underline{\underline{\mathbb{E}}}} = \begin{pmatrix} 0 & \frac{yv_z}{LH} & -\frac{zv_z}{LH} \\ \frac{yv_z}{LH} & -\frac{v_z}{H}\left(1 - \frac{2x}{L}\right) & 0 \\ -\frac{zv_z}{LH} & 0 & \frac{v_z}{H}\left(1 - \frac{2x}{L}\right) \end{pmatrix} \tag{58}$$

To compare all tracking parameters, we also need to compute the strain. First, we define the deformation tensor:

$$\underline{\underline{\mathbb{F}}} = \underline{\underline{grad}}(\Phi) = \begin{pmatrix} 1 & 0 & 0 \\ y_0 \frac{2v_z t}{HL} e^{\left(-v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} & e^{\left(-v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} & 0 \\ -z_0 \frac{2v_z t}{HL} e^{\left(v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} & 0 & e^{\left(v_z \frac{(L-2x_0)t - v_x t^2}{HL}\right)} \end{pmatrix} \tag{59}$$

We simplify this with the following notations:

$$a = \frac{2v_z t}{HL} \tag{60}$$

$$b = v_z \frac{(L - 2x_0)t - v_x t^2}{HL} \tag{61}$$

Using these, we can calculate the strain tensor $\underline{\underline{\mathbb{E}}}$:

$$\underline{\underline{\mathbb{E}}} = \frac{1}{2} \begin{pmatrix} a^2(y_0^2 e^{-2b} + z_0^2 e^{2b}) & y_0 a e^{-2b} & -z_0 a e^{2b} \\ y_0 a e^{-2b} & e^{-2b} - 1 & 0 \\ -z_0 a e^{2b} & 0 & e^{2b} - 1 \end{pmatrix} \tag{62}$$

With these computations, we have all the variables necessary to test IceTrackPy and compare it with other commercial software solutions in various ways.

4.5 Testing and Comparison with Elmer/Ice Particle Tracking

In this section, we will compare IceTrackPy's tracking algorithm with Elmer/Ice's built-in particle tracking tool.

4.5.1 Working Principle of Elmer/Ice Particle Tracking

Elmer/Ice includes a tool that allows users to track particles during the simulation, so it is useful to first understand its operation. When tracking a particle with Elmer/Ice, the particle must initially be assigned to a node. The trajectory is then integrated (with various order options), and the vectorial product between the trajectory and each face of the wedge is computed to determine which face the particle crosses. The algorithm ensures that the particle crosses only one element boundary (adjusting the time step if necessary), and then solves the nonlinear system to find the particle's coordinates in the local basis, allowing the algorithm to obtain the new velocity and continue.

A primary issue with this algorithm arises when the vectorial product between the trajectory and the face is nearly parallel, leading to errors where the particle is not located in the correct element, resulting in a tracking failure. Additionally, in multi-processed simulations, Elmer/Ice does not reassemble the mesh into a single file, which can cause particles to be misplaced in the mesh and prevent the algorithm from converging.

4.5.2 Error Comparison Between the Two Programs

In this section, we will conduct tests in a consistent manner. Multiple particles will be launched from the same position for each tracking program, and the field values of interest will be evaluated at regular intervals (e.g., one year, two years). This approach allows us to track the error for each particle along its trajectory. Since examining all variables simultaneously is complex, we will focus on the average, minimum, and maximum errors.

On the Elmer/Ice side, we have opted to use first-order integration for comparability with IceTrackPy (ITP). This setup leads to the following particle paths:

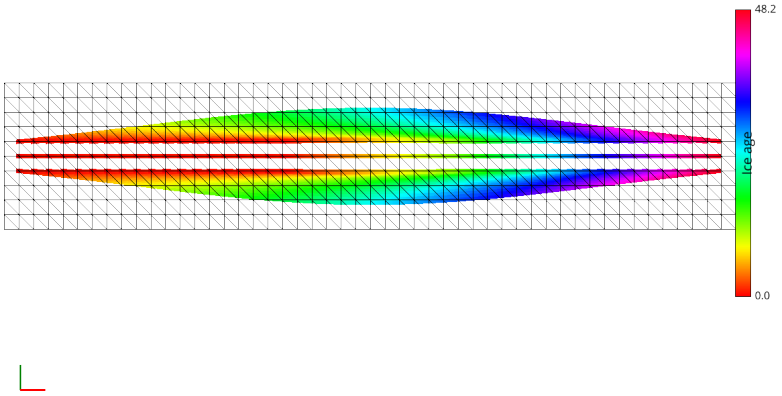


Figure 38: Trajectories of the particles plotted with ice age

The first step is to assess mesh sensitivity for both algorithms. For this, we created seven meshes, beginning with an initial mesh at one element per meter in each direction, then increasing to 10 and 20 elements per meter for each direction while maintaining one element per meter in the other directions.

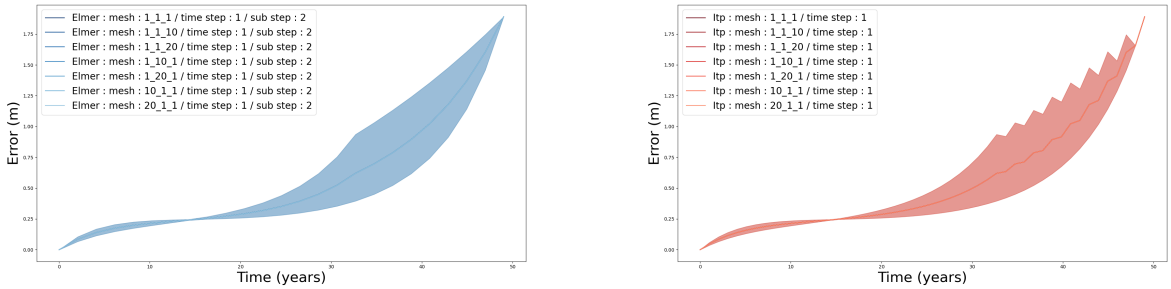


Figure 39: Mesh sensitivity for Elmer/Ice and IceTrackPy

From this figure, we observe that both solutions exhibit low sensitivity to mesh density in this simplified model, an important property for particle tracking software. Additionally, we note that, over time, the maximum and minimum values converge as the number of particles decreases. After 35 years, only one particle remains, causing the minimum and maximum values to coincide.

This mesh-independence has been tested for multiple integration times, yielding consistent results: these algorithms are truly independent of mesh refinement.

Next, we compare the two algorithms for different integration times (choosing 0.1-year and 1-year integration time steps here).

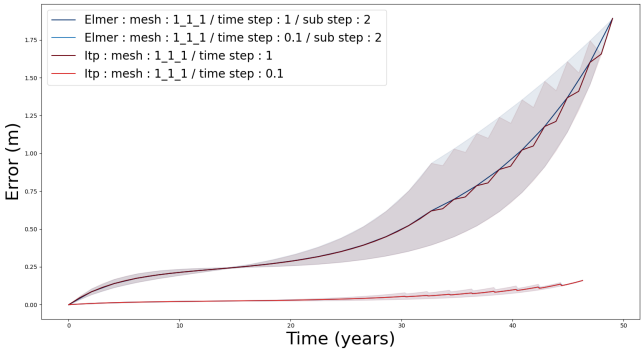


Figure 40: Impact of time step count

From this figure, we observe that both algorithms exhibit identical error at a given time step, as expected, since this reflects the minimum possible error for the specified time step and integration order.

Since Elmer/Ice is not equipped to integrate the strain rate for each particle, the only remaining variable for comparison is speed. Thus, for each possible mesh configuration and time step, the computation time for both algorithms was recorded and can be represented in a bar graph.

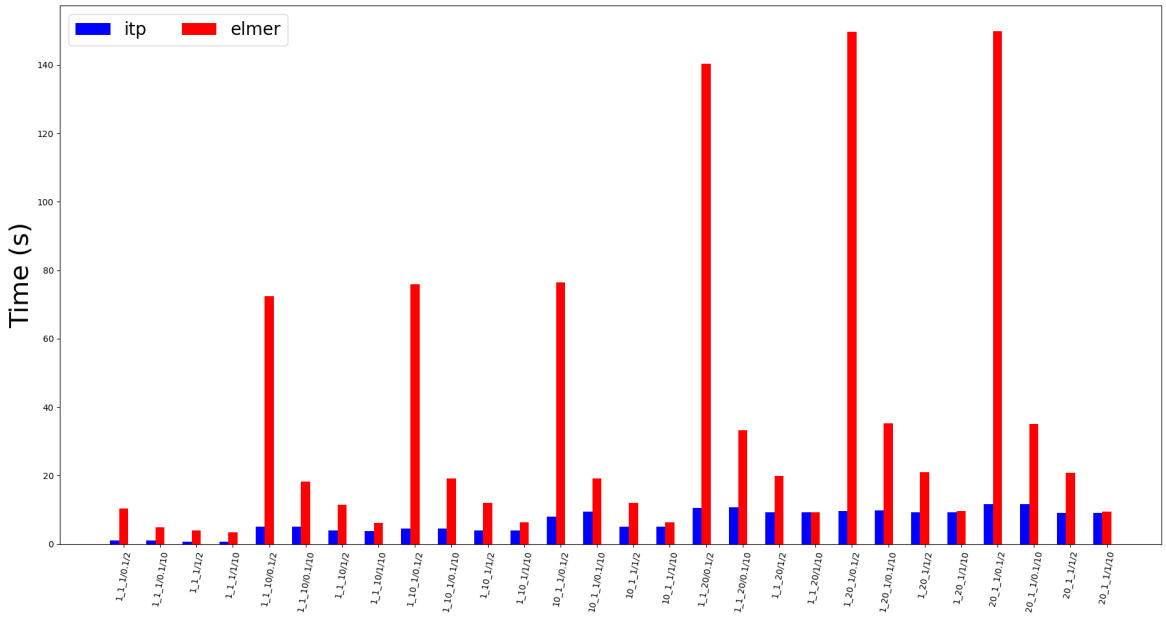
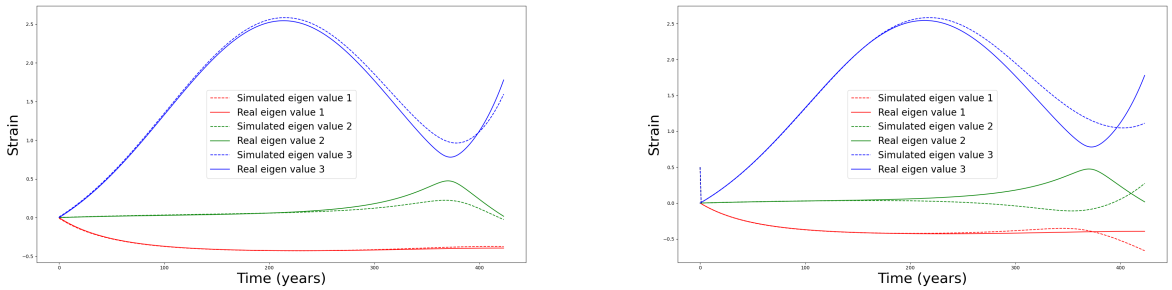


Figure 41: Speed comparison between the two algorithms

Next, a comparison can be made between IceTrackPy and the test case for strain computation. First, we examine the evolution of eigenvalues over time, using the improved method to reduce numerical error alongside the classical method for computing strain.



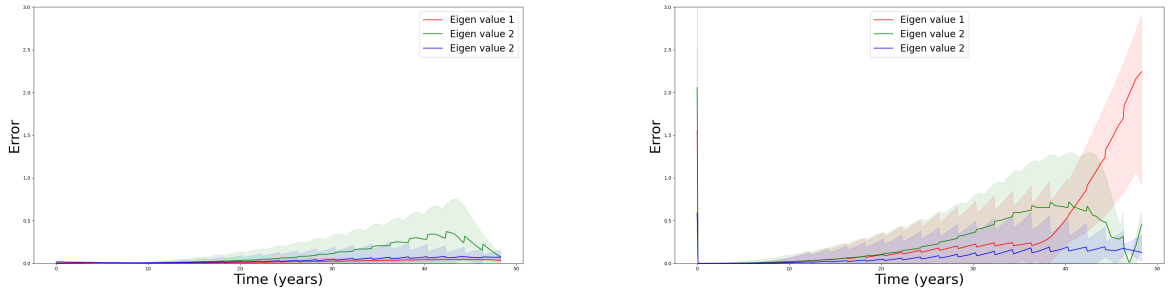
(a) Improved method for computing strain

(b) Classical method for computing strain

Figure 42: Evolution of strain in the test case

As shown in the graphs, the improved method provides significantly better results, even though it is not flawless; after 400 iterations, numerical errors start accumulating, leading to a slight deviation from the expected values. However, this improved algorithm does not diverge completely after multiple iterations, unlike the classical method, which shows noticeable divergence on the first eigenvalue after 375 iterations.

This example illustrates the performance on a single particle. To further validate, we compute the mean, minimum, and maximum errors across all particles in this case, resulting in the following graphs. Here, the error is calculated by taking the difference between the true value and the integrated value, divided by the maximum value along the path for the true strain, allowing all errors to be plotted on the same graph.



(a) Improved method for computing strain

(b) Classical method for computing strain

Figure 43: Error evolution in strain computation for the test case

Once again, we observe that the improved algorithm performs better than the classical method, providing higher precision for the final eigenvalues. This demonstrates that IceTrackPy's tracking tools are as accurate as commercial solutions for particle tracking, and that its strain integration tools offer enhanced reliability over traditional methods.

4.6 Application on the Argentière Glacier

In this section, we apply the tools developed previously to the Argentière Glacier.

4.6.1 Steady-State Model

The first step was to perform a simulation using Elmer/Ice of the Argentière Glacier, as provided in the resources available from Elmer/Ice. This resulted in the following velocity field for the glacier:

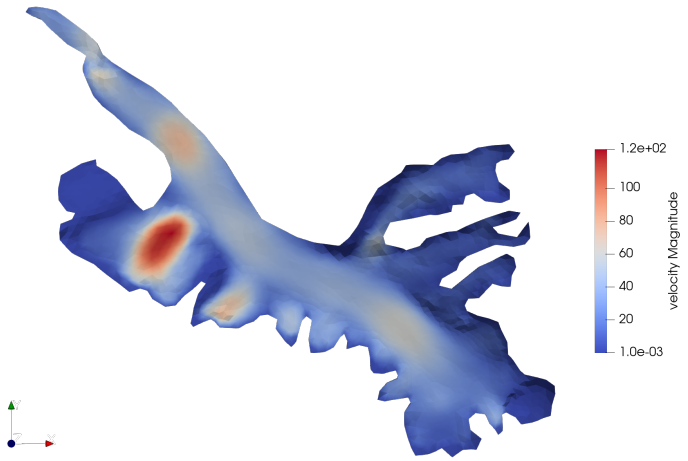


Figure 44: Velocity field from Elmer/Ice based on the 1907 glacier topography

4.6.2 Steady-State Tracking and Data Visualization

With the velocity field established, the particle tracking software can then be executed. All parameter details can be explored in the IceTrackPy examples. The first step is to plot the ice age along the trajectory to identify any inconsistencies in the tracking.

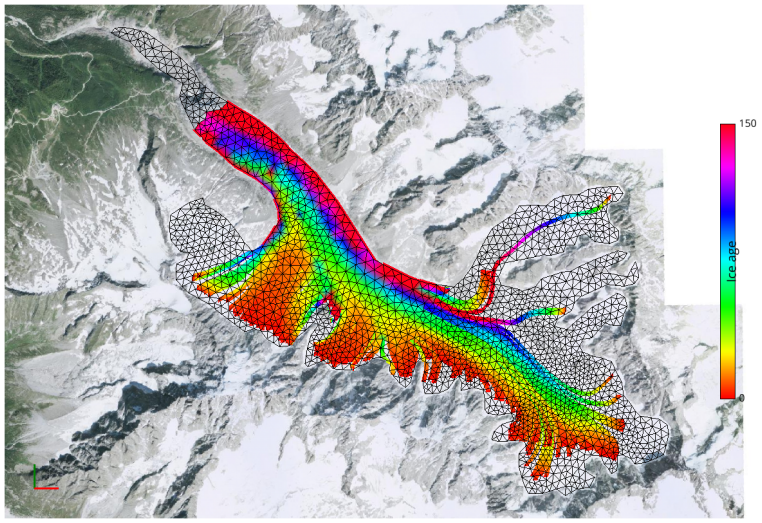


Figure 45: Representation of ice age along the path and in the ablation area where melting occurs

Next, we examine the direction of principal deformation by plotting the orientation of the primary eigenvector.

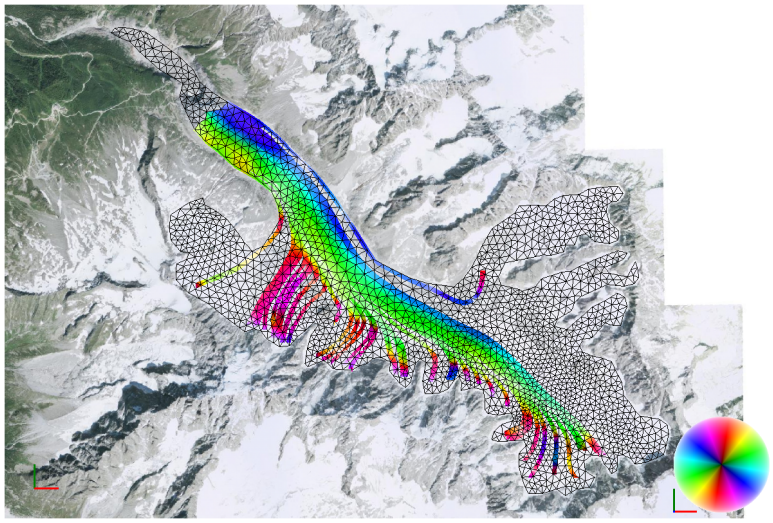


Figure 46: Representation of the orientation of principal deformation for particles at the end of their trajectories

Finally, we analyze different anisotropy factors, such as relative anisotropy, which is defined as:

$$A_{\text{rel}} = \frac{\sigma_{\epsilon_i}}{\overline{\epsilon_i}} \tag{63}$$

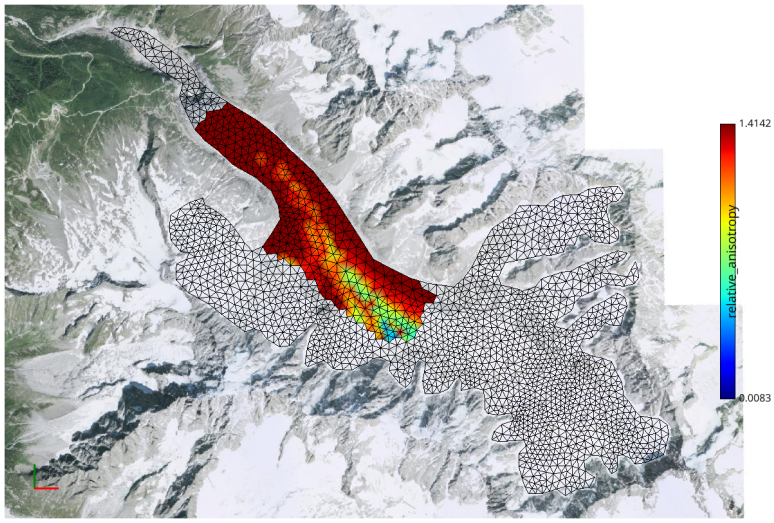
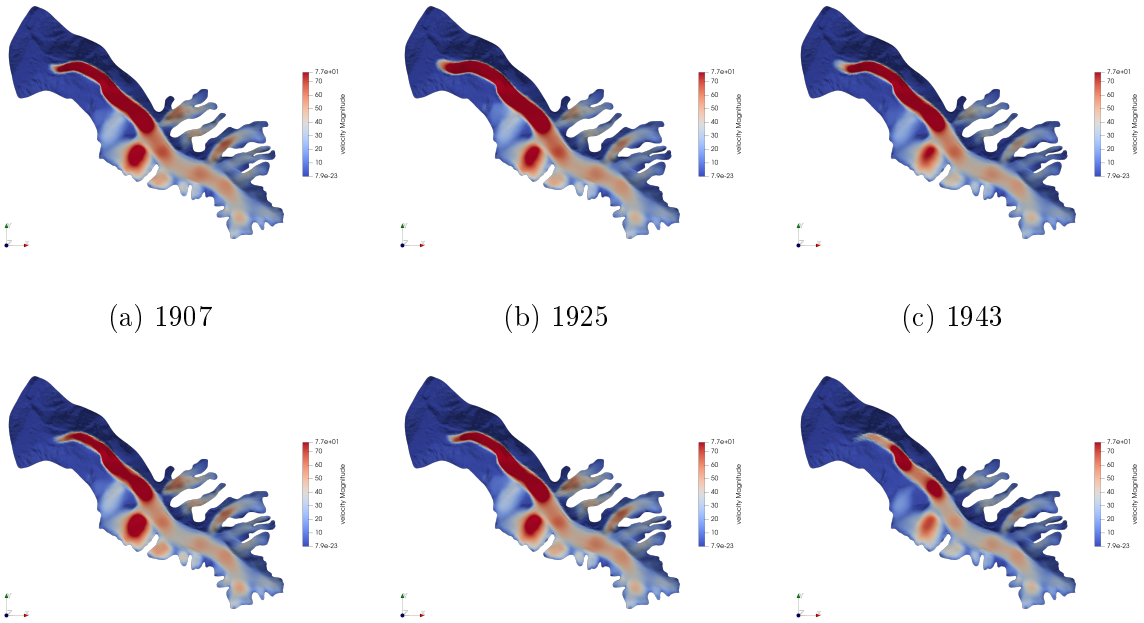


Figure 47: Representation of the relative anisotropy for particles at the end of their trajectories

Using these visualizations, we can then select specific particles for further analysis.

4.6.3 Transient Case Model

IceTrackPy can also be applied to a transient case. In this scenario, the simulation was previously run, and only the output files from this simulation were provided. The following images display the evolution of the glacier's velocity field over time:



(a) 1907 (b) 1925 (c) 1943

(d) 1961 (e) 1979 (f) 1997

Figure 48: Evolution of the velocity field at different times

4.6.4 Transient Case Tracking and Data Visualization

All time steps were imported into IceTrackPy, and the particle tracking algorithm was executed. Various fields were then visualized, starting with the ice age along the particle path.

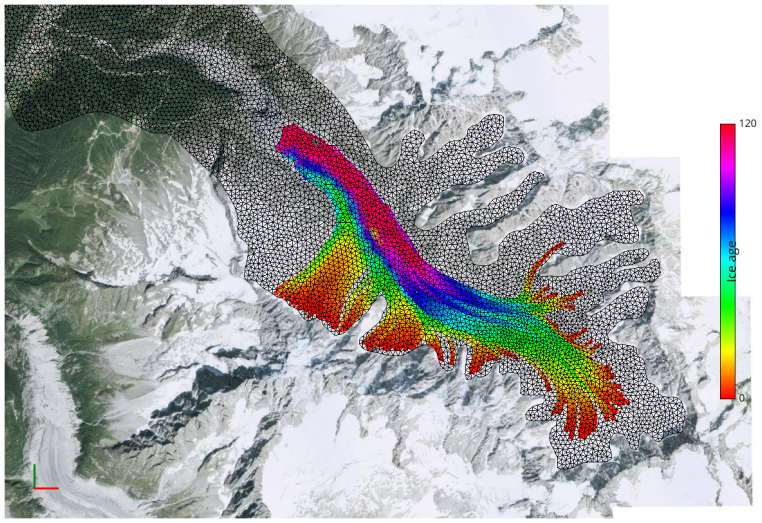


Figure 49: Representation of the age of ice along the path and in the ablation area where it melts

Next, the direction of principal deformation was examined by plotting the orientation of the largest eigenvector.

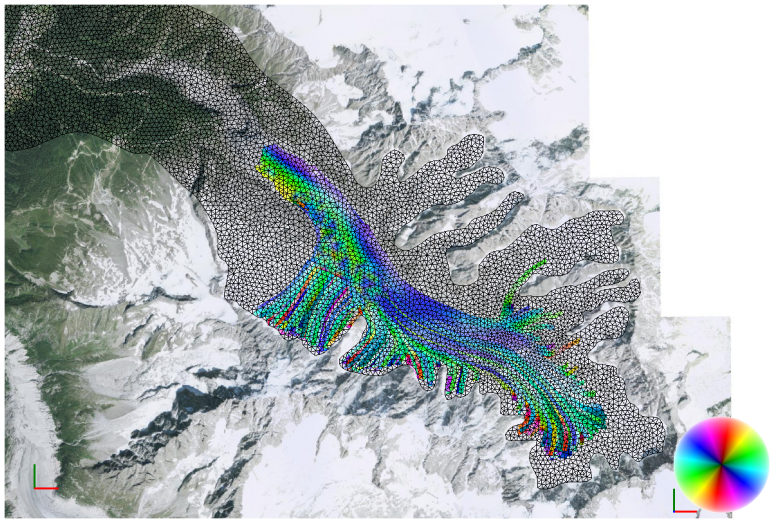


Figure 50: Representation of the orientation of principal deformation for particles at the end of their trajectories

Finally, we analyzed various anisotropy factors, such as relative anisotropy:

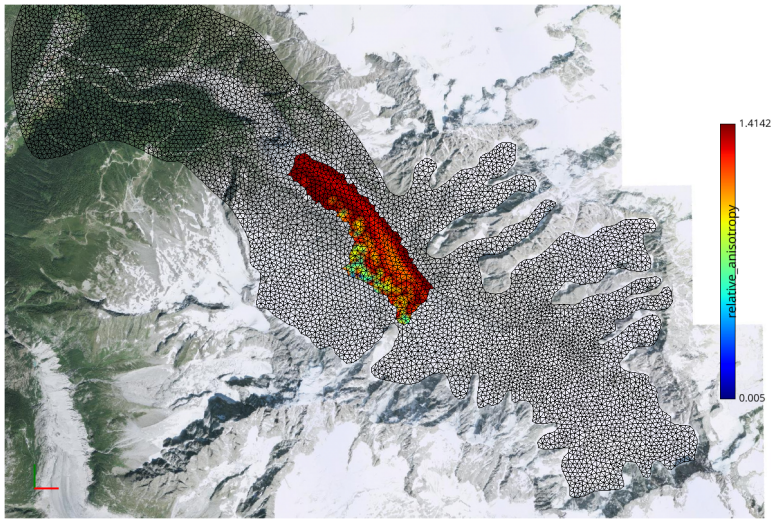


Figure 51: Representation of the relative anisotropy for particles at the end of their trajectories

Using these visualizations, we can then select specific particles for further analysis.

5 Utilization of Flow Line Data to Model Ice Rheology

The primary objective of this particle tracking tool is to capture the trajectory and stress history of particles along their flow path. This extracted data can then be imported into R^3iCe to predict the rheological properties of ice that are likely to be found in the glacier.

5.1 Desired Characteristics in the Flow Line

The first parameter of interest is the variation of stress along the particle's path. However, it is essential that these stresses remain relatively simple, as complex stress histories increase the uncertainty in predicting the final ice texture. Thus, we aim to identify trajectories where the particle experiences distinct stress conditions:

- **Traction-dominated path:** Identify trajectories where the particle is primarily under tensile stress. When input into R^3iCe, this type of stress history is expected to produce specific textures.
- **Compression-dominated path:** Find trajectories where the particle predominantly experiences compressive stress. Comparing the resulting textures from R^3iCe with expected patterns will help validate the model.
- **Shear-dominated path:** Locate paths where the particle undergoes significant shear. This is important as recrystallization mechanisms significantly influence texture evolution under shear. Observing the resulting texture on glacier samples would allow for further comparison and validation.

By isolating and analyzing these specific types of stress-dominated trajectories, we aim to better understand and predict the textural evolution of ice along flow lines in a glacier.

5.2 Identified Flow Lines on the Argentière Glacier

The following examples highlight specific flow lines where particles predominantly experience different types of stress, useful for modeling distinct texture evolutions:

- **Traction-dominated flow line:** A path where the particle is primarily under tensile stress.

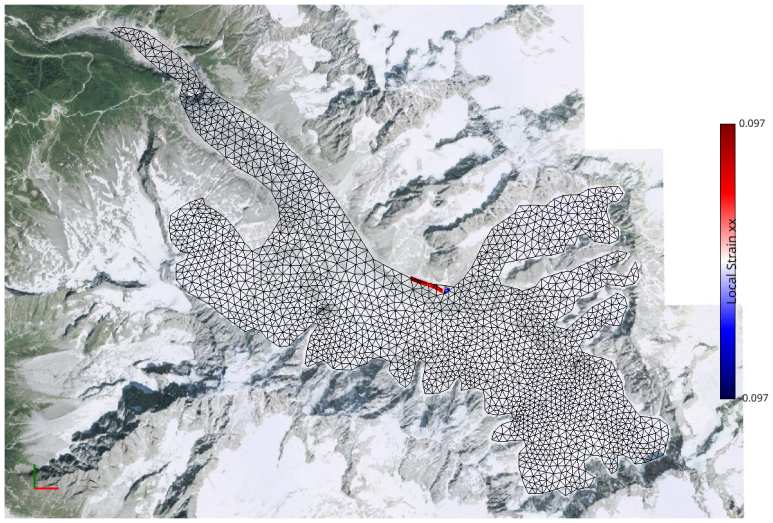


Figure 52: Flow path of a particle under tensile stress

- **Compression-dominated flow line:** A path where the particle mainly undergoes compressive stress.

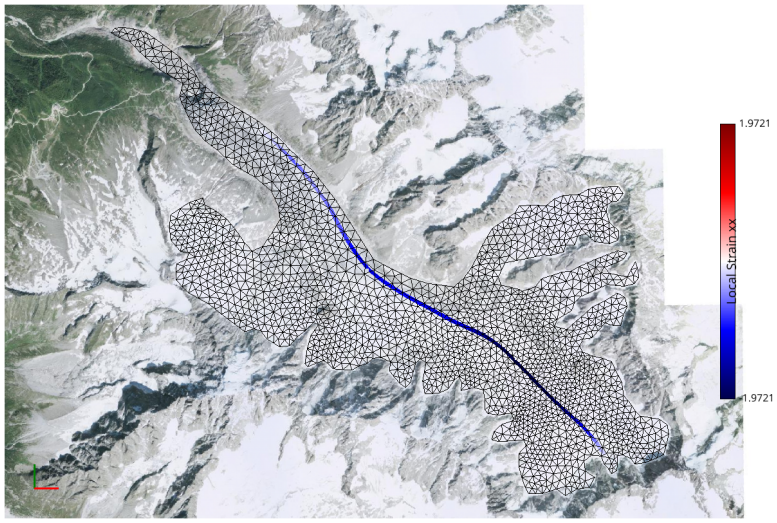


Figure 53: Flow path of a particle under compressive stress

- **Shear-dominated flow line:** A path where shear is the primary stress acting on the particle.

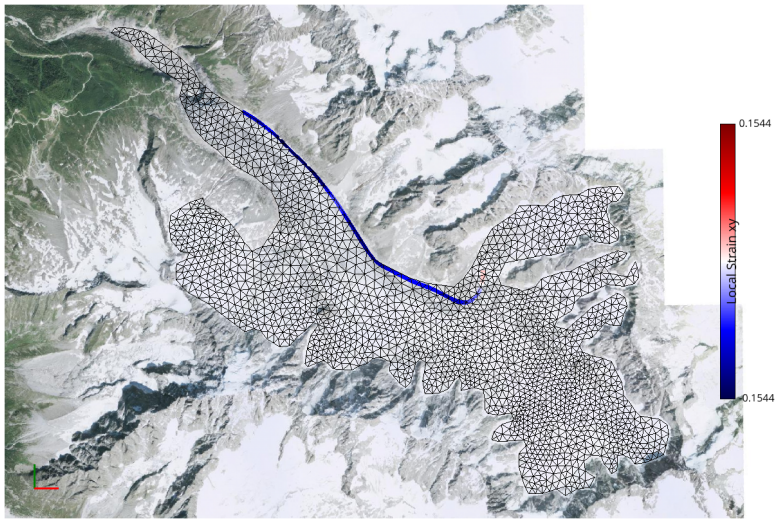


Figure 54: Flow path of a particle under shear stress

5.3 Data Export for R^{3}iCe

After identifying the flow line, it can be exported for use in R^3iCe to simulate texture evolution. The exported data includes the eigenvalues of the strain rate tensor and the rotation matrix between the global and local coordinate systems at each time step. This information allows R^3iCe to simulate texture transformations accurately along the particle's path.

The simulation begins by orienting the initial ice texture according to the rotation matrix at each step, then applying boundary conditions on all sides of a cube representing the particle. The simulation then runs for one iteration, after which the resulting texture is reoriented into the global frame using the inverse of the rotation matrix. This process is repeated for each time step in the particle's trajectory, thereby reconstructing the texture evolution along the flow path.

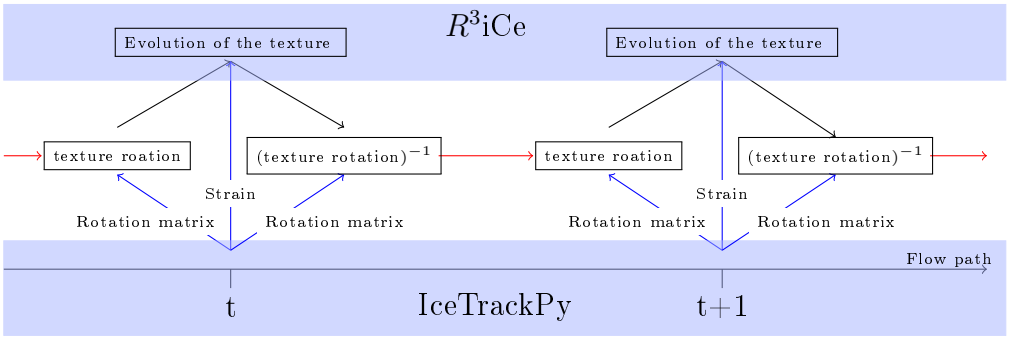


Figure 55: Implementation of IceTrackPy for texture evolution computation

6 Conclusion

This thesis presents a comprehensive framework for modeling the trajectory and deformation of particles in glacial environments, employing particle tracking and flow line analysis through IceTrackPy, a custom-developed tool interfacing with Elmer/Ice and R^3iCe. The work addresses key challenges in glaciology, particularly the need for accurate, efficient particle tracking in numerical simulations of glacier flow. By leveraging a custom particle tracking algorithm, this research not only ensures independence from high-performance computing requirements but

also enhances tracking reliability across varied mesh densities, as confirmed by comparisons with commercial solutions.

The framework developed here offers a novel integration of flow line data with high-fidelity strain and rheology analysis, enabling the investigation of complex stress and strain histories along a particle's path. Through carefully designed test cases, the robustness of IceTrackPy was validated against known analytical solutions, revealing improved accuracy in strain computation over standard methods, even under high deformation conditions. Additionally, the tool's visualization capabilities provide new insights into the spatial distribution of key rheological properties, such as strain anisotropy and principal deformation directions, throughout a glacier.

Applied to real-world data from the Argentière Glacier, the framework demonstrated its capacity to track particles over transient and steady-state models, elucidating important patterns in ice age, deformation orientation, and anisotropy distribution. These insights contribute to our understanding of glacial flow dynamics and serve as a foundation for further study into ice rheology, particularly by interfacing particle stress and strain histories with R^3iCe. The outcome is a predictive model that can be adapted to various glaciers, providing insights into their unique flow characteristics and potential future behaviors.

In conclusion, this thesis advances the capabilities of particle tracking and deformation modeling in glaciology, offering a versatile, validated tool for glacier simulation analysis. The integration of IceTrackPy with Elmer/Ice opens avenues for enhanced rheological studies and fosters a deeper understanding of glacier dynamics. Future work may extend these methods by incorporating higher-order velocity interpolations, adaptive meshing, or advanced recrystallization models, further refining our ability to predict ice behavior under changing environmental conditions.

References

Roger G. Barry. The cryosphere – past, present, and future: a review of the frozen water resources of the world. *Polar Geography*, 34(4):219–227, 2011. doi: 10.1080/1088937X.2011.638146.

WF Budd, PL Keage, and NA Blundy. Empirical studies of ice sliding. *Journal of glaciology*, 23(89):157–170, 1979. doi: https://doi.org/10.3189/S0022143000029804.

Thomas Chauve, Maurine Montagnat, Véronique Dansereau, Pierre Saramito, Kévin Fourteau, and Andrea Tommasi. A physically-based formulation for texture evolution during dynamic recrystallization. a case study for ice. *Comptes Rendus. Mécanique*, 2023. doi: https://doi.org/10.5194/egusphere-egu24-7333.

H-P Cheng, J-R Cheng, and G-T Yeh. A particle tracking technique for the lagrangian–eulerian finite element method in multi-dimensions. *International Journal for Numerical Methods in Engineering*, 39(7):1115–1136, 1996. doi: https://doi.org/10.1002.

ME Collins, JA Doolittle, and RV Rourke. Mapping depth to bedrock on a glaciated landscape with ground-penetrating radar. *Soil Science Society of America Journal*, 53(6):1806–1812, 1989. doi: https://doi.org/10.2136/sssaj1989.03615995005300060032x.

Charles Augustin Coulomb. *Théorie des machines simples: en ayant égard au frottement de leurs parties, et à la roideur de cordages.* 1785.

Gaël Durand, Anders Svensson, Asbjørn Persson, Olivier Gagliardini, Fabien Gillet-Chaulet, Jesper Sjolte, Maurine Montagnat, and Dorthe Dahl-Jensen. Evolution of the texture along the epica dome c ice core. 68(Supplement):91–105, 2009.

Paul Duval, MF Ashby, and I Anderman. Rate-controlling processes in the creep of polycrystalline ice. *The Journal of Physical Chemistry*, 87(21):4066–4074, 1983. doi: https://doi.org/10.1021/j100244a014.

Elmer. Elmer. doi: https://doi.org/10.5281/zenodo.7892181. URL `https://github.com/ElmerCSC`.

elmerice. Iscal (ice sheet coupled approximation levels). *ElmerIce*, 2017. URL `https://elmerfem.org/elmerice/wiki/doku.php?id=solvers:iscal`.

Olivier Gagliardini and Jacques Meyssonnier. Flow simulation of a firn-covered cold glacier. *Annals of Glaciology*, 24:242–248, 1997. doi: https://doi.org/10.3189/S0260305500012246.

Olivier Gagliardini, D Cohen, P Råback, and Thomas Zwinger. Finite-element modeling of subglacial cavities and related friction law. *Journal of Geophysical Research: Earth Surface*, 112(F2), 2007. doi: https://doi.org/10.1029/2006JF000576.

Olivier Gagliardini, Fabien Gillel-Chaulet, and Maurine Montagnat. A review of anisotropic polar ice models: from crystal to ice-sheet flow models. 68(Supplement):149–166, 2009.

A Gilbert, F Gimbert, O Gagliardini, and C Vincent. Inferring the basal friction law from long term changes of glacier length, thickness and velocity on an alpine glacier. *Geophysical Research Letters*, 50(16):e2023GL104503, 2023. doi: https://doi.org/10.1029/2023GL104503.

Fabie Gillet-Chaulet, Olivier Gagliardini, Jacques Meyssonnier, Maurine Montagnat, and Olivier Castelnau. A user-friendly anisotropic flow law for ice-sheet modeling. *Journal of glaciology*, 51(172):3–14, 2005. doi: https://doi.org/10.3189/172756505781829584.

Fabien Gillet-Chaulet, Olivier Gagliardini, Jacques Meyssonnier, Thomas Zwinger, and Juha Ruokolainen. Flow-induced anisotropy in polar ice and related ice-sheet flow modelling. *Journal of non-newtonian fluid mechanics*, 134(1-3):33–43, 2006. doi: https://doi.org/10.1016/j.jnnfm.2005.11.005.

Saskia Gindraux, Ruedi Boesch, and Daniel Farinotti. Accuracy assessment of digital surface models from unmanned aerial vehicles' imagery on glaciers. *Remote Sensing*, 9(2):186, 2017. doi: https://doi.org/10.3390/rs9020186.

John W Glen. The creep of polycrystalline ice. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 228(1175):519–538, 1955. doi: https://doi.org/10.1098/rspa.1955.0066.

GoogleEarth. 2024. URL `https://earth.google.com/web/`.

Kolumban Hutter. *Theoretical glaciology: material science of ice and the mechanics of glaciers and ice sheets*, volume 1. Springer, 2017.

Thomas Jacob, John Wahr, W Tad Pfeffer, and Sean Swenson. Recent contributions of glaciers and ice caps to sea level rise. *Nature*, 482(7386):514–518, 2012. doi: https://doi.org/10.1038/nature10847.

Baptiste Journaux, Thomas Chauve, Maurine Montagnat, Andrea Tommasi, Fabrice Barou, David Mainprice, and Léa Gest. Recrystallization processes, microstructure and crystallographic preferred orientation evolution in polycrystalline ice during high-temperature simple shear. *The Cryosphere*, 13(5):1495–1511, 2019. doi: https://doi.org/10.5194/tc-13-1495-2019.

Sepp Kipfstuhl, Sérgio H Faria, Nobuhiko Azuma, Johannes Freitag, Ilka Hamann, Patrik Kaufmann, Heinrich Miller, Karin Weiler, and Frank Wilhelms. Evidence of dynamic recrystallization in polar firn. *Journal of Geophysical Research: Solid Earth*, 114(B5), 2009. doi: https://doi.org/10.1029/2008JB005583.

kitware. paraview. doi: 10.1007/978-3-030-90539-2_33. URL `https://github.com/Kitware/ParaView`.

JWOGJ Krug, J Weiss, O Gagliardini, and G Durand. Combining damage and fracture mechanics to model calving. *The Cryosphere*, 8(6):2101–2117, 2014. doi: https://doi.org/10.5194/tc-8-2101-2014.

L. Lliboutry. Local friction laws for glaciers: A critical review and new openings. *Journal of Glaciology*, 23(89):67–95, 1979. doi: 10.3189/S0022143000029750.

Ph Mansuy, J Meyssonnier, and A Philip. Localization of deformation in polycrystalline ice: experiments and numerical simulations with a simple grain model. *Computational Materials Science*, 25(1-2):142–150, 2002. doi: https://doi.org/10.1016/S0927-0256(02)00258.

THIRIET Martin. Icetrackpy. 2024. URL `gricad-gitlab.univ-grenoble-alpes.fr`.

Jacques Meyssonnier and Armelle Philip. A model for the tangent viscous behaviour of anisotropic polar ice. *Annals of Glaciology*, 23:253–261, 1996. doi: https://doi.org/10.3189/S0260305500013513.

Laurent Michel, Marco Picasso, Daniel Farinotti, Andreas Bauder, Martin Funk, and Heinz Blatter. Estimating the ice thickness of mountain glaciers with an inverse approach using surface topography and mass-balance. *Inverse Problems*, 29(3):035002, 2013. doi: 10.1088/0266-5611/29/3/035002.

Maurine Montagnat. *Contribution à l'étude du comportement viscoplastique des glaces des calottes polaires: modes de déformation et simulation du développement des textures*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2001.

M Morlighem, H Seroussi, E Larour, and E Rignot. Inversion of basal friction in antarctica using exact and incomplete adjoints of a higher-order model. *Journal of Geophysical Research: Earth Surface*, 118(3):1746–1753, 2013. doi: https://doi.org/10.1002/jgrf.20125.

Dubravka Pokrajac and Ranko Lazic. An efficient algorithm for high accuracy particle tracking in finite elements. *Advances in Water Resources*, 25(4):353–369, 2002. doi: https://doi.org/10.1016/S0309-1708(02)00012-X.

Nico Schlömer. meshio. doi: https://doi.org/110.21105/joss.02369. URL `https://github.com/nschloe/meshio?tab=readme-ov-file`.

Hakime Seddik, Ralf Greve, Luca Placidi, Ilka Hamann, and Olivier Gagliardini. Application of a continuum-mechanical model for the flow of anisotropic polar ice to the edml core, antarctica. *Journal of Glaciology*, 54(187):631–642, 2008. doi: https://doi.org/10.3189/002214308786570755.

George Gabriel Stokes. On the theories of the internal friction of fluids in motion, and of the equilibrium and motion of elastic solids. 2007. doi: https://doi.org/10.1190/1.9781560801931.ch3e.

Peter RABACK Olivier GAGLIARDINI Thomas ZWINGER. Elmer/ice course. 2019. doi: 10.5281/zenodo.7892180.

Victor C Tsai, Andrew L Stewart, and Andrew F Thompson. Marine ice-sheet profiles and stability under coulomb basal conditions. *Journal of Glaciology*, 61(226):205–215, 2015. doi: https://doi.org/10.3189/2015JoG14J221.

Johannes Weertman. On the sliding of glaciers. *Journal of glaciology*, 3(21):33–38, 1957. doi: https://doi.org/10.3189/S0022143000024709.