



## Elmer/Ice – New Generation Ice Sheet Model



# Elmer/Ice – 2D glacier toy model

Thomas Zwinger



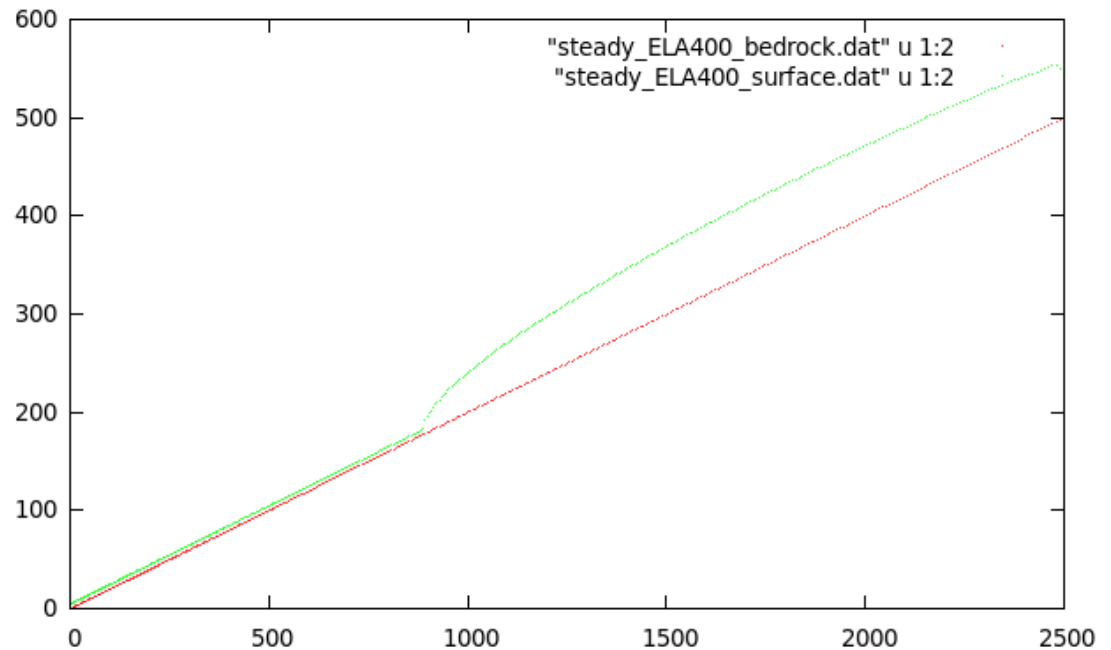
# DIAGNOSTIC RUN

Starting from a given point-distribution (DEM) in 2D we show how to

- Build the geometry and mesh it
- Set up runs on fixed geometry
- Introduce sliding
- Introduce heat transfer (thermo-mechanical coupling)
- Write a simple MATC function

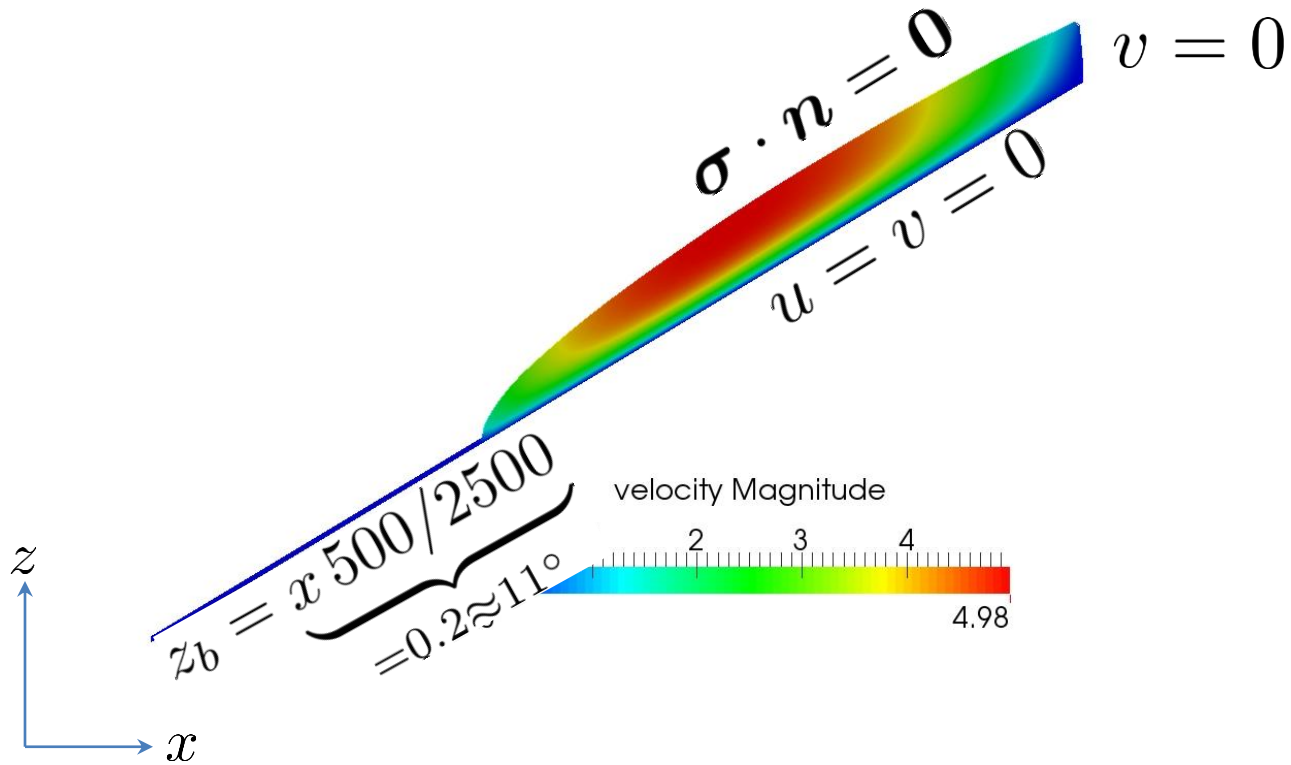
# The diagnostic problem

- We start from a distribution of surface and bedrock points that have been created driving a prognostic run into steady state



- The distributions are given in the files:  
`steady_ELA400_bedrock.dat` , `steady_ELA400_surface.dat`

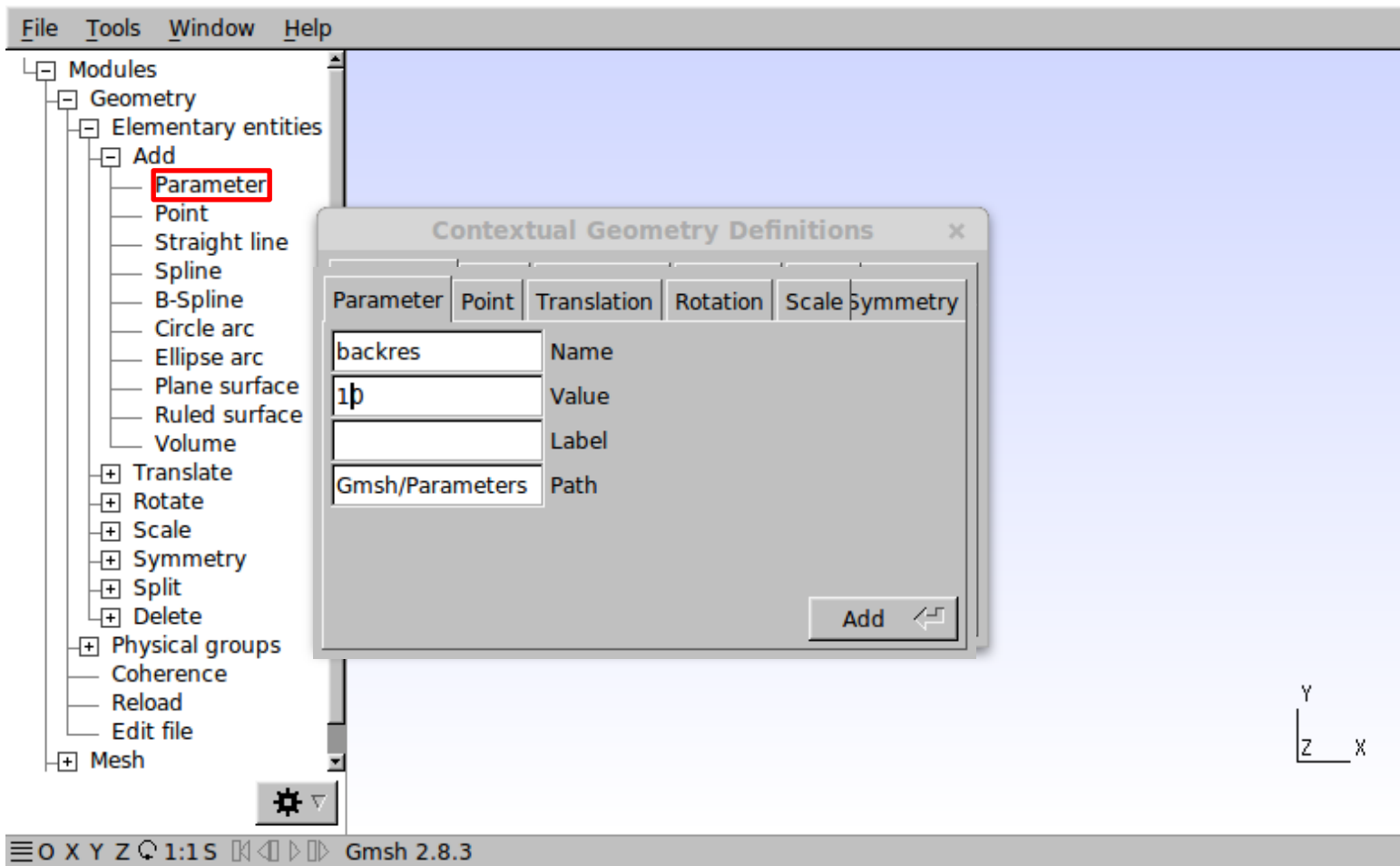
# The diagnostic problem



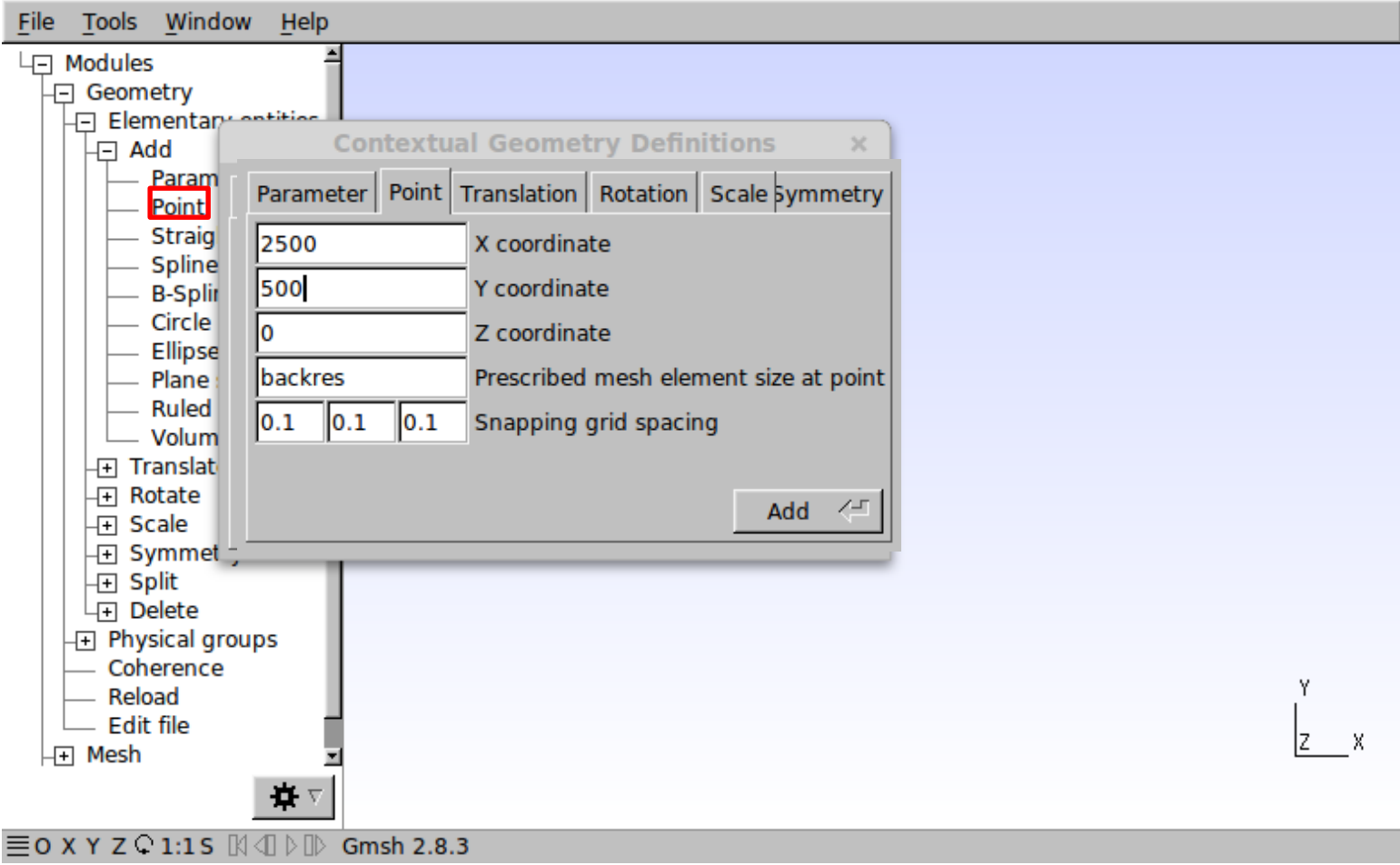
# The Mesh

- Using Gmsh
- Simply launch by:
  - \$ **gmsh mytestglacier.geo &**
  - Don't use the existing one, **testglacier.geo**, since we want to keep it as a backup, should this one fail

# The Mesh

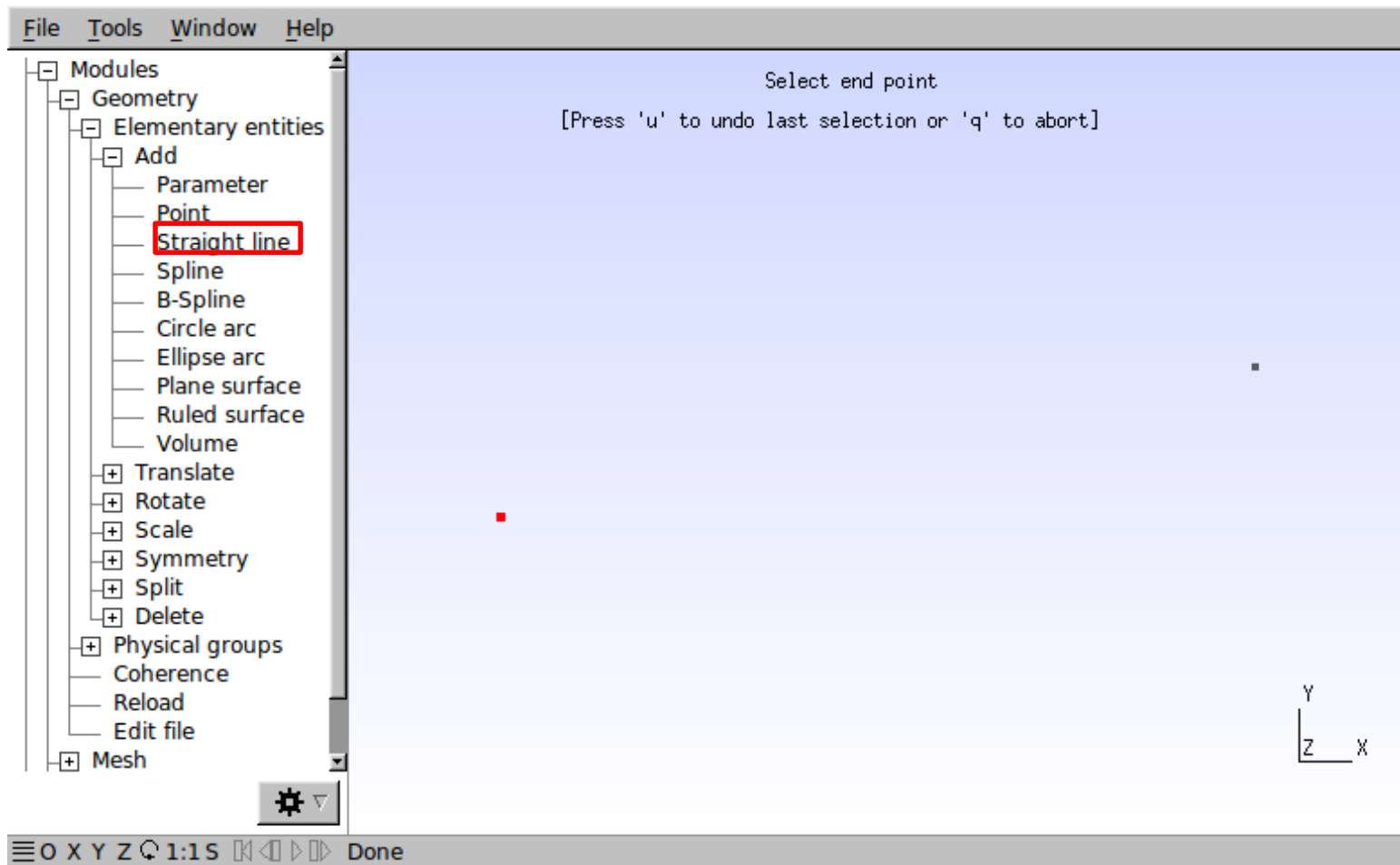


# The Mesh

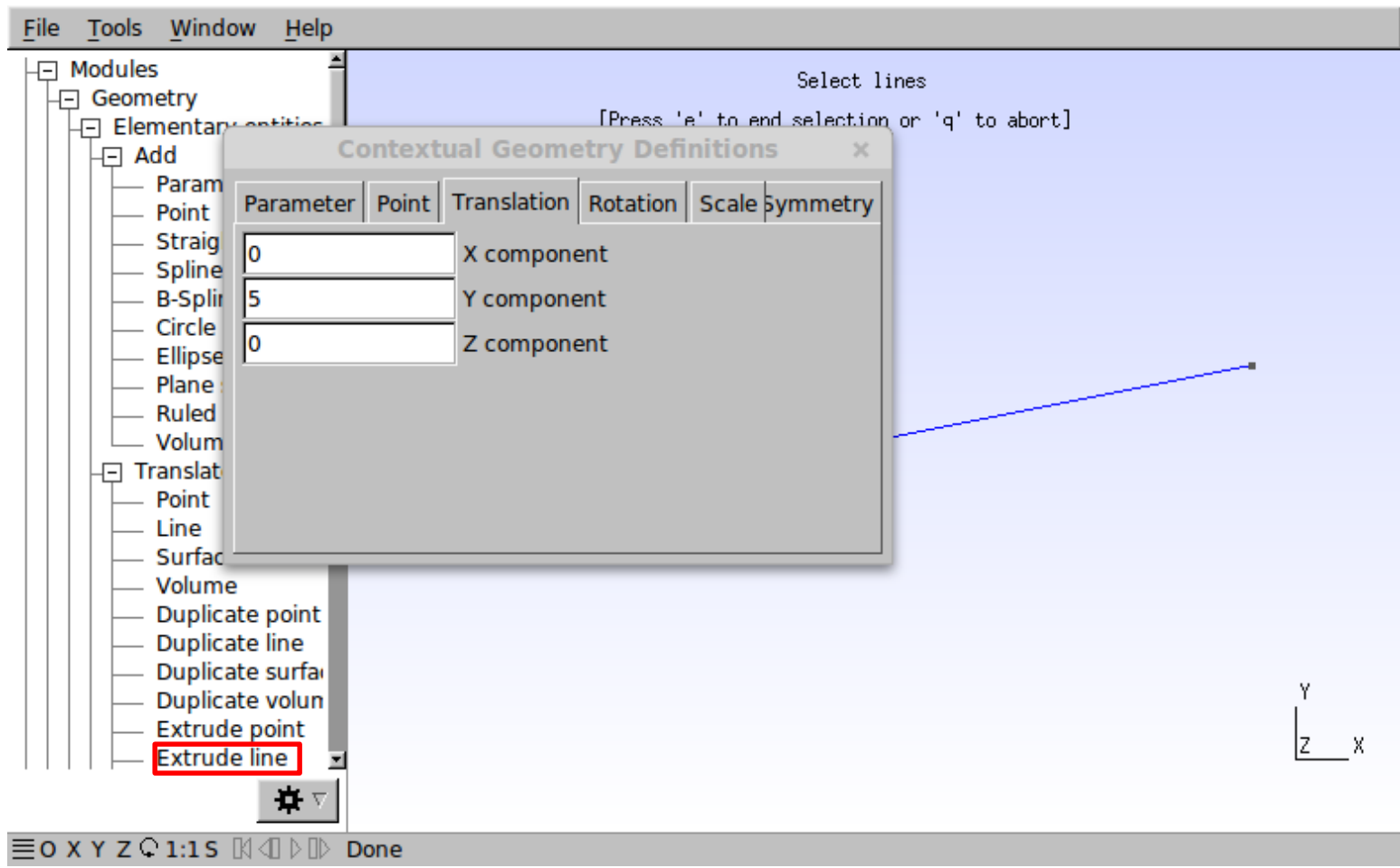




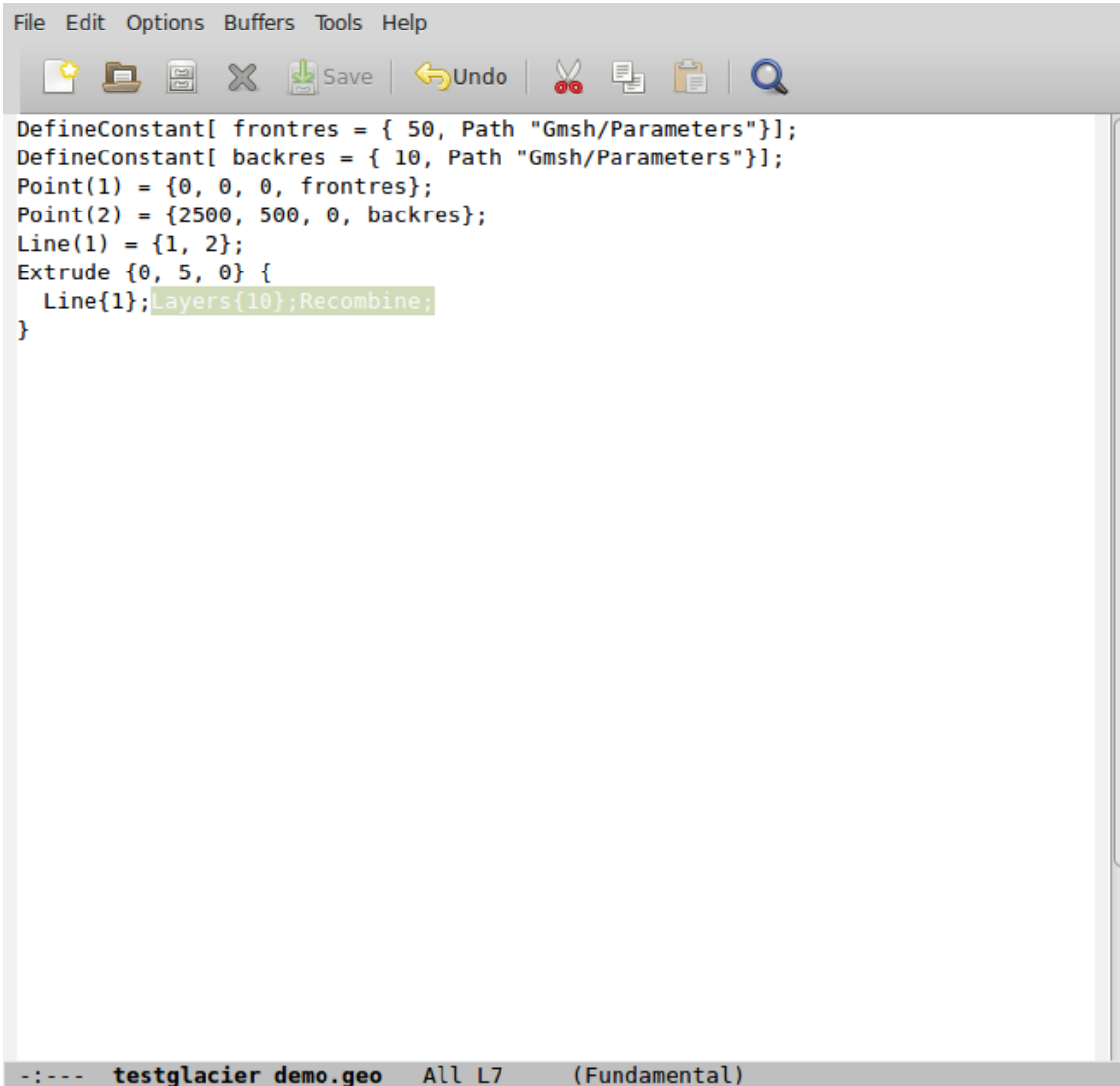
# The Mesh



# The Mesh



# The Mesh



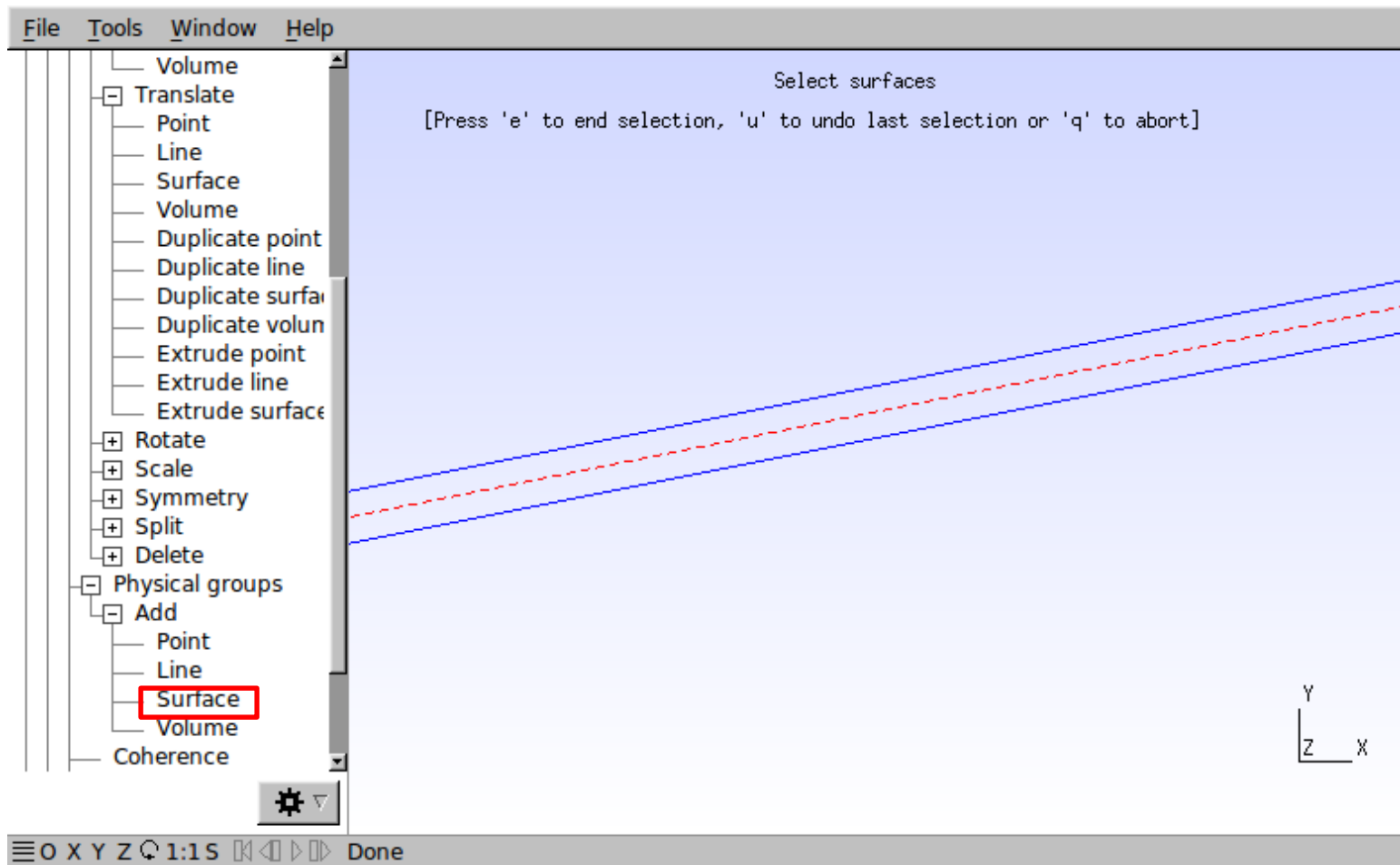
The screenshot shows the Gmsh software interface. The menu bar includes File, Edit, Options, Buffers, Tools, and Help. The toolbar contains icons for opening files, saving, undo, cut, copy, paste, and search. The main text area contains the following Gmsh script code:

```
DefineConstant[ frontres = { 50, Path "Gmsh/Parameters"}];
DefineConstant[ backres = { 10, Path "Gmsh/Parameters"}];
Point(1) = {0, 0, 0, frontres};
Point(2) = {2500, 500, 0, backres};
Line(1) = {1, 2};
Extrude {0, 5, 0} {
  Line{1};Layers{10};Recombine;
}
```

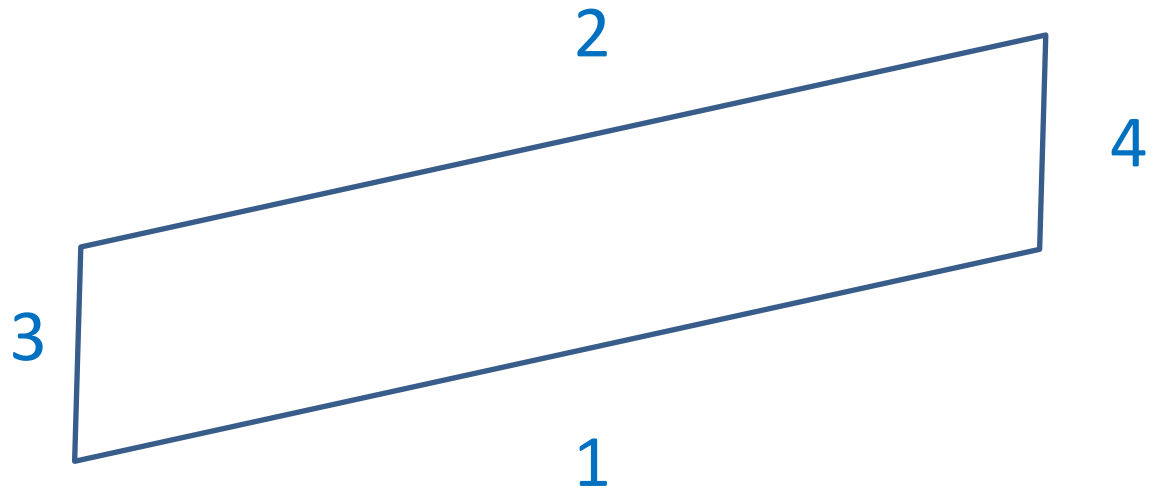
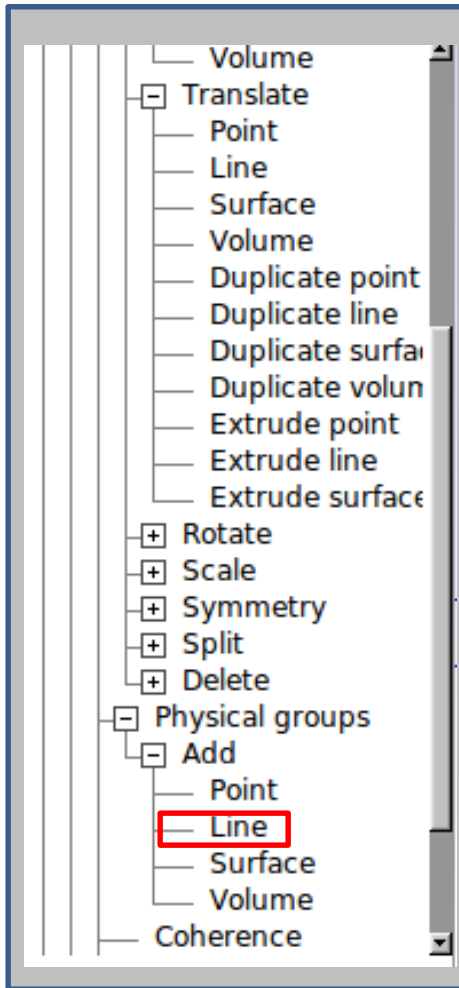
The status bar at the bottom shows the file name `testglacier_demo.geo`, the current layer is `All L7`, and the mesh is `(Fundamental)`.

- Open the file `mytestglacier.geo`

# The Mesh



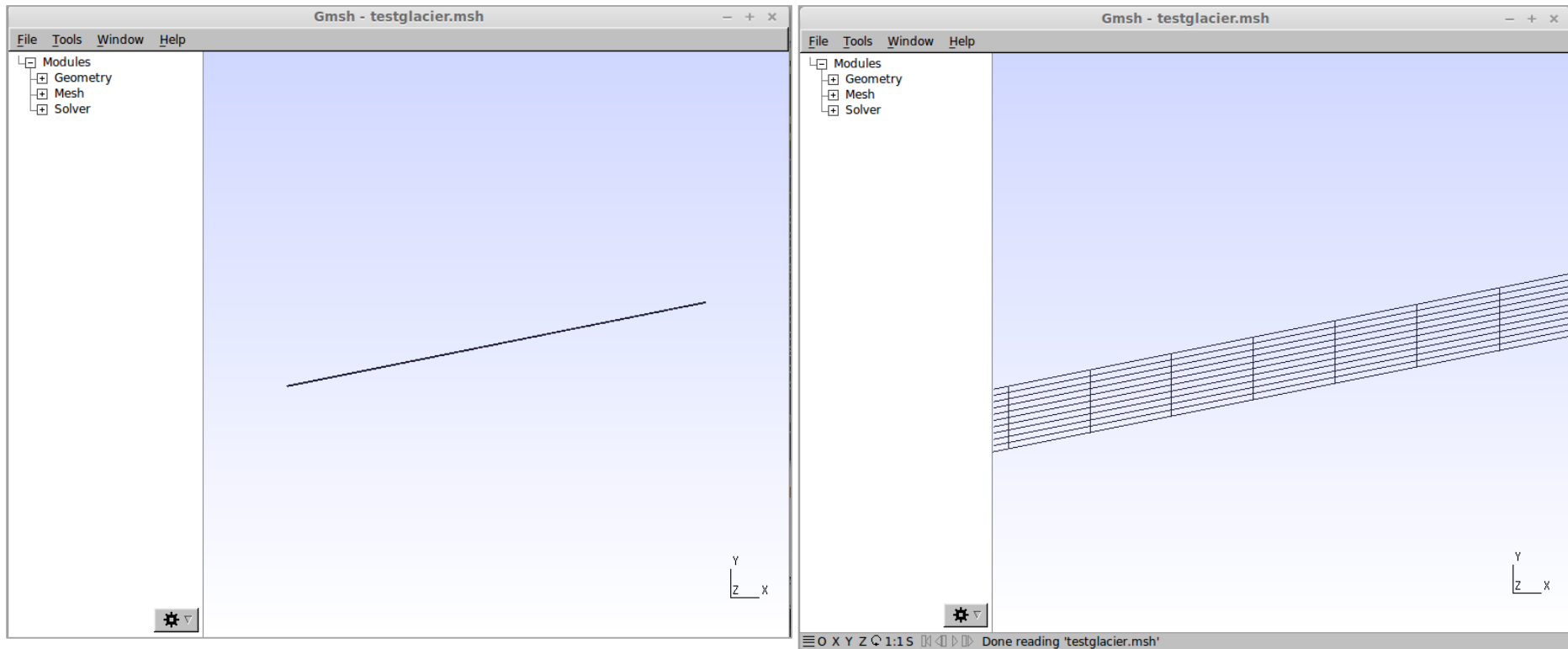
# The Mesh



- You have to zoom (mouse wheel) in and out of the model
- and translate (right mouse button)
- Select boundary and press “e”

# The diagnostic problem

- We use a  $\sim 11$  deg inclined rectangular mesh (produced with Gmsh) of unit-height (load the ready-made file)



# The diagnostic problem

- Open the Gmsh file (if not already opened):

```
$ gmsh testglacier.geo
```

- Go to **Mesh** and press the **2D** button
- **Save** the mesh
- Use ElmerGrid to convert the mesh:

```
> ElmerGrid 14 2 testglacier.msh\  
-autoclean -order 0.1 1.0 0.01
```

Needed to  
clean up  
geometry

Orders the numbering in x  
y z -directions (highest  
number fastest)

# The diagnostic problem

- We will do a diagnostic simulation, i.e., we ignore the time derivative in ANY equation
  - Stokes anyhow has no explicit time dependence
  - That also means, that the surface velocity distribution is a result of the given geometry and cannot be prescribed
- Open the Solver Input File (SIF)

```
$ emacs Stokes_diagnostic.sif &
```



# The diagnostic problem

```
!echo on
Header
  !CHECK KEYWORDS Warn
  Mesh DB "." "testglacier"
  Include Path ""
  Results Directory ""
End
```

This declares our mesh; capital/small letters matter

```
Simulation
  Max Output Level = 4
  Coordinate System = "Cartesian 2D"
  Coordinate Mapping(3) = 1 2 3
  Simulation Type = "Steady"
  Steady State Max Iterations = 1
  Output Intervals = 1
  Output File = "Stokes_ELA400_diagnostic.result"
  Post File = "Stokes_ELA400_diagnostic.vtu" ! use .ep suffix for legacy format
  Initialize Dirichlet Conditions = Logical False
End
```

The coordinate system (inkl. Dimension)

Steady State = diagnostic

# The diagnostic problem

```
Body 1  
  Name = "Glacier"  
  Body Force = 1  
  Equation = 1  
  Material = 1  
  Initial Condition = 1  
End
```

Assigns the Equation/Material/Body Force/and Initial condition to a body

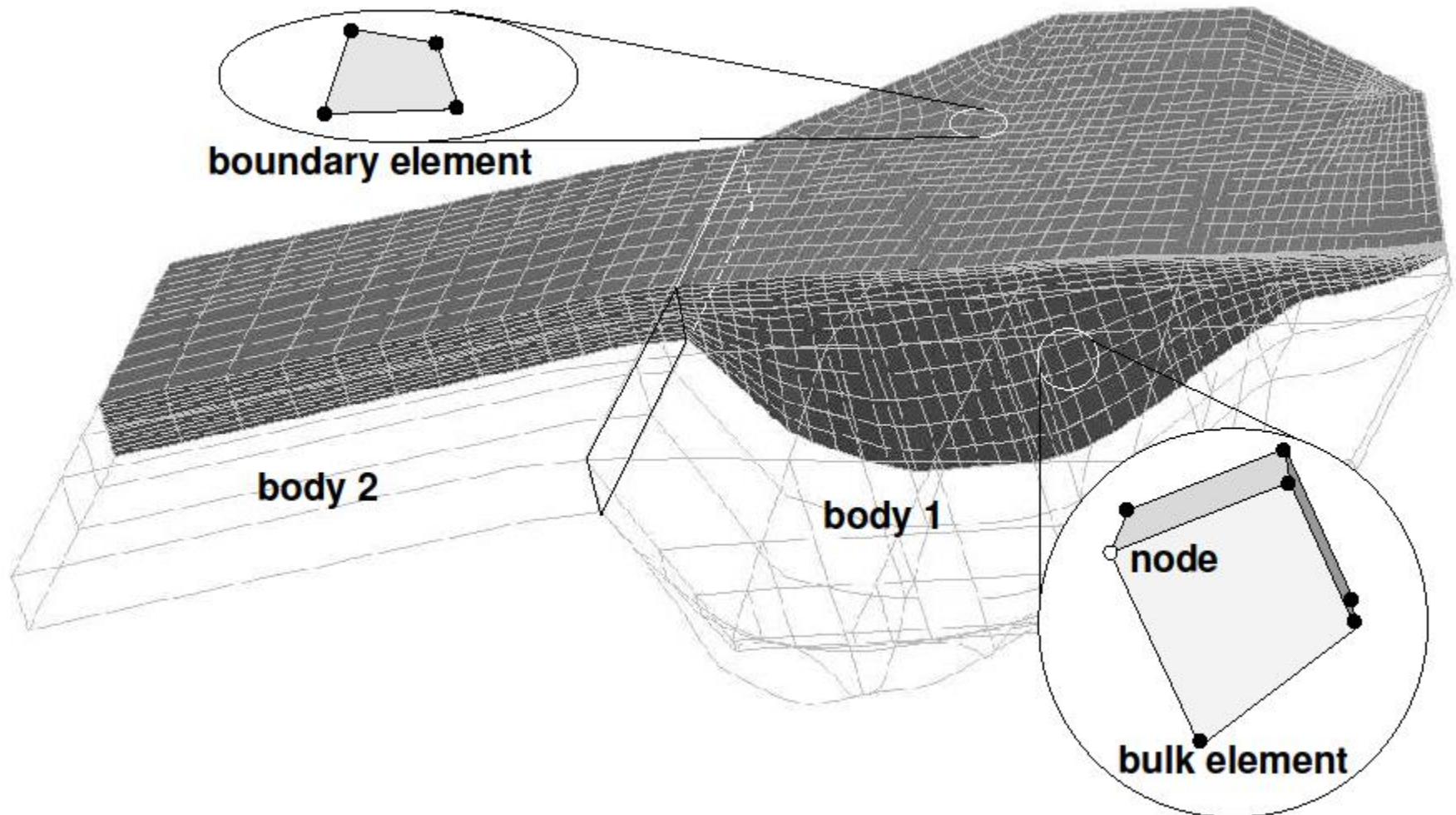
```
Equation 1  
  Name = "Equation1"  
  Convection = "computed"  
  Flow Solution Name = String "Flow Solution"  
  Active Solvers(3) = 1 2 3  
End
```

The Equation for Body 1 (see above); declares set of Solvers

```
Initial Condition 1  
  Velocity 1 = 0.0  
  Velocity 2 = 0.0  
  Pressure = 0.0  
  Depth = Real 0.0  
End
```

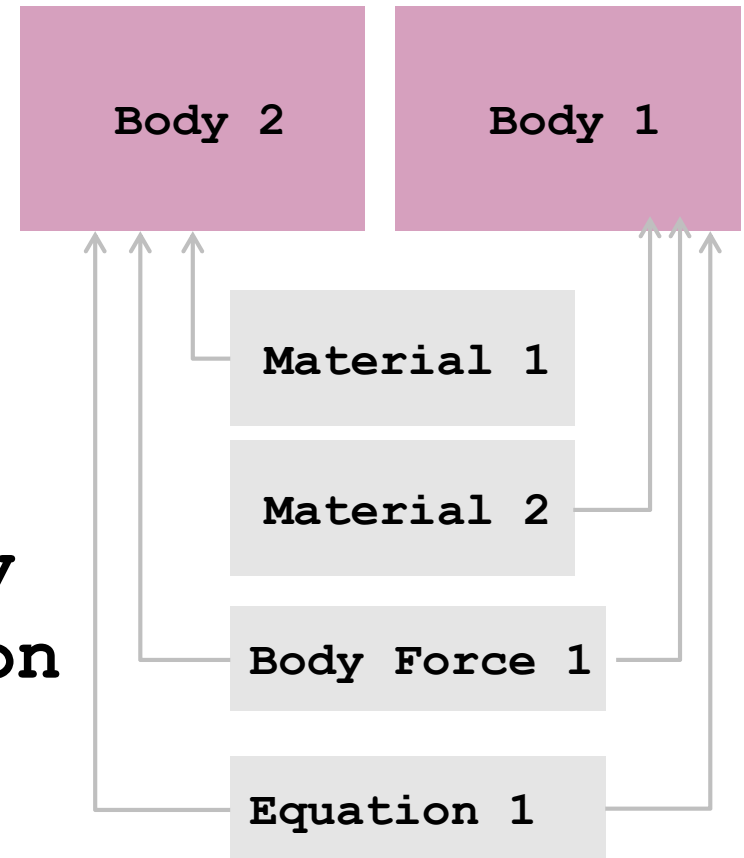
Well, as the name suggests: initial values for variables

# On Bodies and Boundaries



# On Bodies and Boundaries

- Each **Body** has to have an **Equation** and **Material** assigned
  - **Body Force**, **Initial Condition** optional
- Two bodies can have the same **Material/Equation/Body Force/Initial Condition** section assigned



# The diagnostic problem

```
! maps DEM's at the very beginning
! to originally rectangular mesh
! see Top and Bottom Surface in BC's
Solver 1
  Exec Solver = "before Simulation"
  Equation = "MapCoordinate"
  Procedure = "StructuredMeshMapper" "StructuredMeshMapper"
  Active Coordinate = Integer 2 ! the mesh-update is y-direction
! For time being this is currently externally allocated
  Mesh Velocity Variable = String "Mesh Velocity 2"
! The 1st value is special as the mesh velocity could be unrealistically high
  Mesh Velocity First Zero = Logical True
! The accuracy applied to vector-projections
  Dot Product Tolerance = Real 0.01
End
```

The primary criterion for order of execution is the **Exec Solver** keyword, thereafter the numbering

This solver simply projects the shape given in the input files before the run (see Exec Solver keyword) to the initially flat mesh; See **Top Surface** and **Bottom Surface** keywords later

# The diagnostic problem

## Solver 2

```
Equation = "Flowdepth"  
Exec Solver = "Before Simulation"  
Procedure = File "ElmerIceSolvers" "FlowDepthSolver"  
Variable = String "Depth"  
Variable DOFs = 1  
Linear System Solver = "Direct"  
Linear System Direct Method = "UMFPACK"  
Linear System Max Iterations = 200  
Linear System Convergence Tolerance = 1.0E-09  
Linear System Abort Not Converged = False  
Linear System Preconditioning = "ILU0"  
Linear System Residual Output = 1  
Gradient = Real -1.0E00  
Exported Variable 1 = -dofs 3 "Mesh Velocity"  
! switch that to True, if you want to have  
! free surface gradients to be computed  
!-----  
Calc Free Surface = Logical True  
! the name for the exported (if not existing) added variable  
! the gradients will be stored in variables with the base  
! name given and "Grad1" and (in 3 dimensions) "Grad2" added,  
! so in our case "FreeSurfGrad1" and "FreeSurfGrad2"  
! again, if those variables did not exist,  
! automatically created  
!-----  
Freesurf Name = String "FreeSurf"
```

End

Flow Depth this time for post processing, only,  
on generally unstructured mesh (will be  
replaced by structured version)

# The diagnostic problem

```
! the central part of the problem: the Stokes solver
Solver 3
! Exec Solver = "Never" # uncommenting would switch this off
Equation = "Navier-Stokes"
Optimize Bandwidth = Logical True
! direct solver
Linear System Solver = Direct
Linear System Direct Method = "UMFPACK"
! alternative to above - Krylov subspace iterative solution
! Linear System Solver = "Iterative"
! Linear System Iterative Method = "GCR" !or "BICGStab"
Linear System Max Iterations = 5000
Linear System Convergence Tolerance = 1.0E-06
Linear System Abort Not Converged = False
Linear System Preconditioning = "ILU1"
Linear System Residual Output = 1

Steady State Convergence Tolerance = 1.0E-05
! Stabilization Method can be [Stabilized,P2/P1,Bubbles]
Stabilization Method = Stabilized

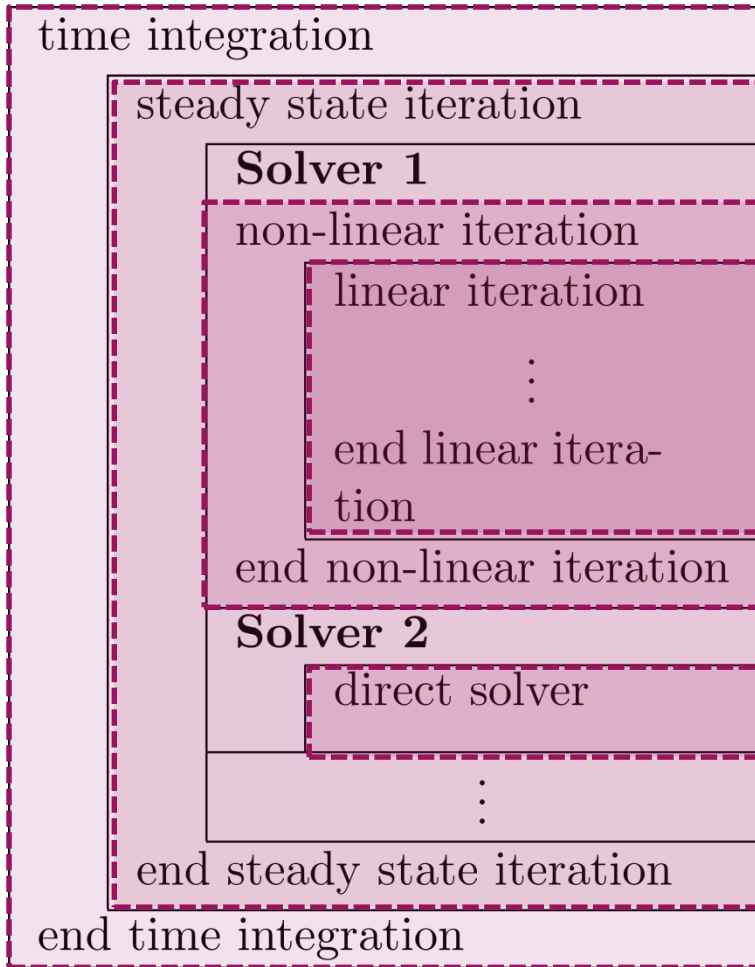
Nonlinear System Convergence Tolerance = 1.0E-04
Nonlinear System Convergence Measure = Solution
Nonlinear System Max Iterations = 50
Nonlinear System Newton After Iterations = 3
Nonlinear System Newton After Tolerance = 1.0E-01
! Nonlinear System Relaxation Factor = 0.75
End
```

Linear System Solver keyword chooses type of solution of the linearized problem

You need that in Stokes and also in PDE's with significant amount of convection

Account for non-linearity of the rheology

# On iteration methods



1. Timestep Intervals
2. Steady State Max Iterations
3. Nonlinear Max Iterations
4. Linear System Max Iterations
4. Linear System Convergence Tolerance
3. Nonlinear System Convergence Tolerance
2. Steady State Convergence Tolerance
- 1.



# The diagnostic problem

```

! we use m-yr-MPa system 1 yr = 31556926.0 sec
Material 1
Name = "ice-ice-baby"
Density = Real $910.0*1.0E-06*(31556926.0)^(-2.0)
!-----
! viscosity stuff
!-----
Viscosity Model = String "Glen"
! Viscosity has to be set to a dummy value
! to avoid warning output from Elmer
Viscosity = Real 1.0
Glen Exponent = Real 3.0
Critical Shear Rate = Real 1.0e-10
! Rate factors (Paterson value in MPa^-3a^-1)
Rate Factor 1 = Real 1.258e13
Rate Factor 2 = Real 6.046e28
! these are in SI units - no problem, as long as
! the gas constant also is
Activation Energy 1 = Real 60e3
Activation Energy 2 = Real 139e3
Glen Enhancement Factor = Real 1.0
    
```

This is for scaling reasons (see next slide)

$$D_{ij} = A\tau_e^{n-1}S_{ij} \quad ; \quad S_{ij} = A^{-1/n}I_{D_2}^{(1-n)/n}D_{ij}$$

where  $I_{D_2}^2 = D_{ij}D_{ij}/2$  and  $\dot{\gamma}^2 = 2D_{ij}D_{ij}$

$$A = A(T') = A_0 \exp^{-Q/RT'}$$

# On the choice of units

Elmer(/Ice) does not assume any choice of units. This is on you, BUT, units have to be consistent amongst each other and with the mesh geometry units.

The order of magnitude in numbers do not change results, as matrix is pivoted

For the Stokes problem, one should give values for:

- the density:  $\rho$  ( $= 910 \text{ kg/m}^3$ )
- the gravity:  $g$  ( $= 9.81 \text{ m s}^{-2}$ )
- the viscosity:  $\eta_0$  ( $\text{Pa s}^{1/n}$ ) ( $1 \text{ Pa} = 1 \text{ kg s}^{-2} \text{ m}^{-1}$ )

kg – m – s [SI] : velocity in m/s and time-step in seconds



kg – m – a : velocity in m/a and timesteps in years



1 a = 31 557 600 s

MPa – m – a : velocity in m/a and Stress in MPa



(What we have in our SIF)

# On the choice of units

To give you an example: for ISMIP tests A-D, the value for the constants would be

- the density:  $\rho = 910 \text{ kg/m}^3$
- the gravity:  $g = 9.81 \text{ m s}^{-2}$
- the fluidity:  $A = 10^{-16} \text{ Pa}^{-3} \text{ a}^{-1}$

	USI kg - m - s		kg - m - a		MPa - m - a	
$g =$	9.81	$\text{m} / \text{s}^2$	9.7692E+15	$\text{m} / \text{a}^2$	9.7692E+15	$\text{m} / \text{a}^2$
$\rho =$	910	$\text{kg} / \text{m}^3$	910	$\text{kg} / \text{m}^3$	9.1380E-19	$\text{MPa m}^{-2} \text{ a}^2$
$A =$	3.1689E-24	$\text{kg}^{-3} \text{ m}^3 \text{ s}^5$	1.0126E-61	$\text{kg}^{-3} \text{ m}^3 \text{ a}^5$	100	$\text{MPa}^{-3} \text{ a}^{-1}$
$\eta =$	5.4037E+07	$\text{kg m}^{-1} \text{ s}^{-5/3}$	1.7029E+20	$\text{kg m}^{-1} \text{ a}^{-5/3}$	0.1710	$\text{MPa a}^{1/3}$

# The diagnostic problem

```
! the variable taken to evaluate the Arrhenius law
! in general this should be the temperature relative
! to pressure melting point. The suggestion below plugs
! in the correct value obtained with TemperateIceSolver
! Temperature Field Variable = String "Temp Homologous"
! the temperature to switch between the
! two regimes in the flow law
Limit Temperature = Real -10.0
! In case there is no temperature variable (which here is the case)
Constant Temperature = Real -3.0
! Heat transfer stuff (will come later)
!Temp Heat Capacity = Variable Temp
! Real MATC "capacity(tx)*(31556926.0)^(2.0)"
!Temp Heat Conductivity = Variable Temp
! Real MATC "conductivity(tx)*31556926.0*1.0E-06"
!Temp Upper Limit = Variable Depth
! Real MATC "273.15 - 9.8E-08 * tx * 910.0 * 9.81" !-> this is the correct
! solution of the pressure melting point with respect to the hydrostatic overburden at t
! the point
End

Body Force 1
Name = "BodyForce1"
Heat Source = 1
Flow BodyForce 1 = Real 0.0
Flow BodyForce 2 = Real $-9.81 * (31556926.0)^(2.0) !MPa - a - m
End
```

We set our glacier to be at -3 C

Now commented, needed later

Gravity, scaled to deliver results in  
m/a and MPa

# The diagnostic problem

- Boundary conditions:
  - using array function for reading surfaces
  - **Real [cubic]** expects two columned row:  
 $X_1 \quad Z_1$   
 $X_2 \quad Z_2$   
...
  - **include** just inserts external file (length)
  - Right values interpolated by matching interval of left values for input variable

```
Boundary Condition 1
Name = "bedrock"
Target Boundaries = 1
Compute Normals = Logical True
! include the bedrock DEM, which has two columns
Bottom Surface = Variable Coordinate 1
Real cubic
    include "steady_ELA400_bedrock.dat"
End
Velocity 1 = Real 0.0e0
Velocity 2 = Real 0.0e0
End

Boundary Condition 2
Name = "sides"
Target Boundaries(2) = 3 4 ! combine left and right boundary
Velocity 1 = Real 0.0e0
End

Boundary Condition 3
Name = "surface"
Target Boundaries = 2
! include the surface DEM, which has two columns
Top Surface = Variable Coordinate 1
Real cubic
    include "steady_ELA400_surface.dat"
End
Depth = Real 0.0
End
```

# The diagnostic problem

- Now, run the case:

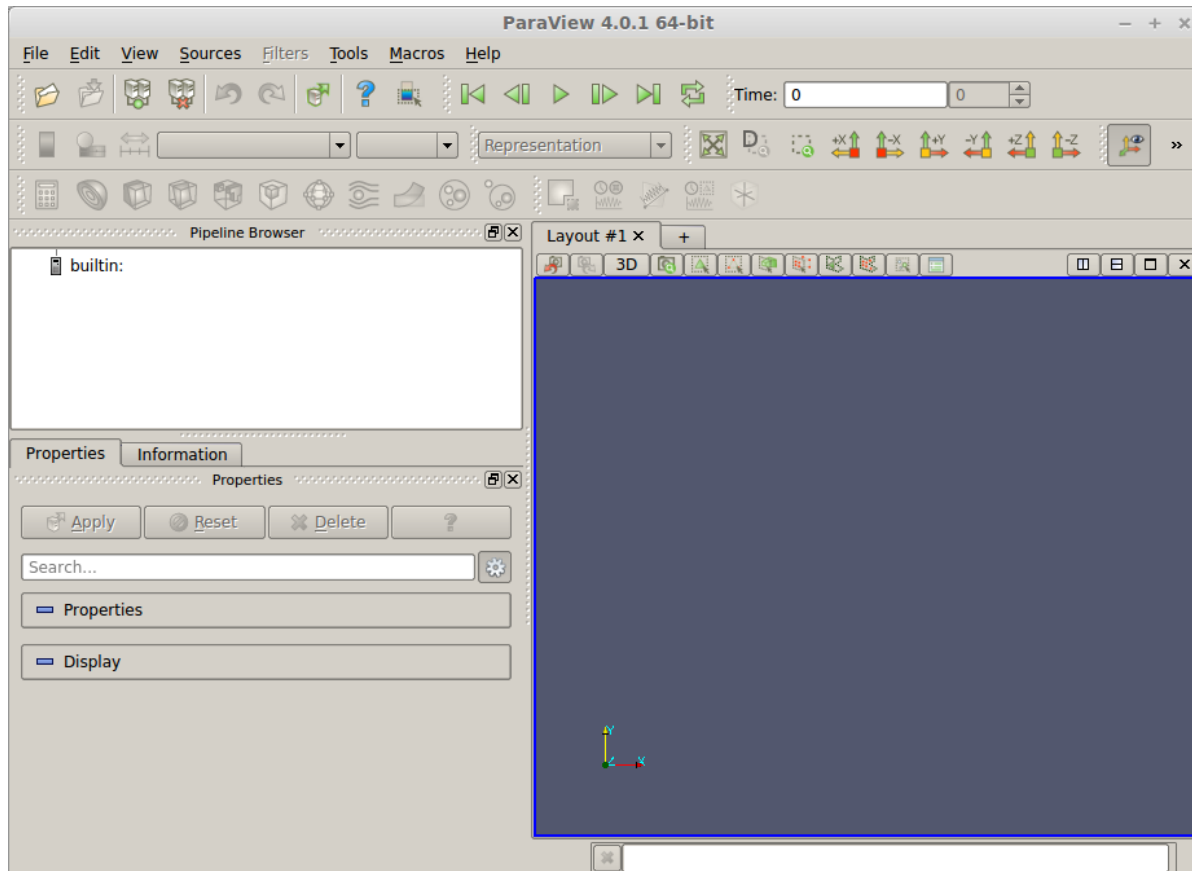
```
$ ElmerSolver Stokes_diagnostic.sif
```

– You will see the convergence history displayed:

```
FlowSolve: -----  
FlowSolve:  NAVIER-STOKES ITERATION                23  
FlowSolve: -----  
FlowSolve:  
FlowSolve: Starting Assembly...  
FlowSolve: Assembly done  
FlowSolve: Dirichlet conditions done  
ComputeChange: NS (ITER=23) (NRM,RELC): ( 1.6112696  
0.90361030E-03 ) :: navier-stokes  
FlowSolve: iter:    23 Assembly: (s)    0.26    6.04  
FlowSolve: iter:    23 Solve:    (s)    0.11    2.62  
FlowSolve: Result Norm      :    1.6112695610649261  
FlowSolve: Relative Change  :    9.0361030224648782E-004
```

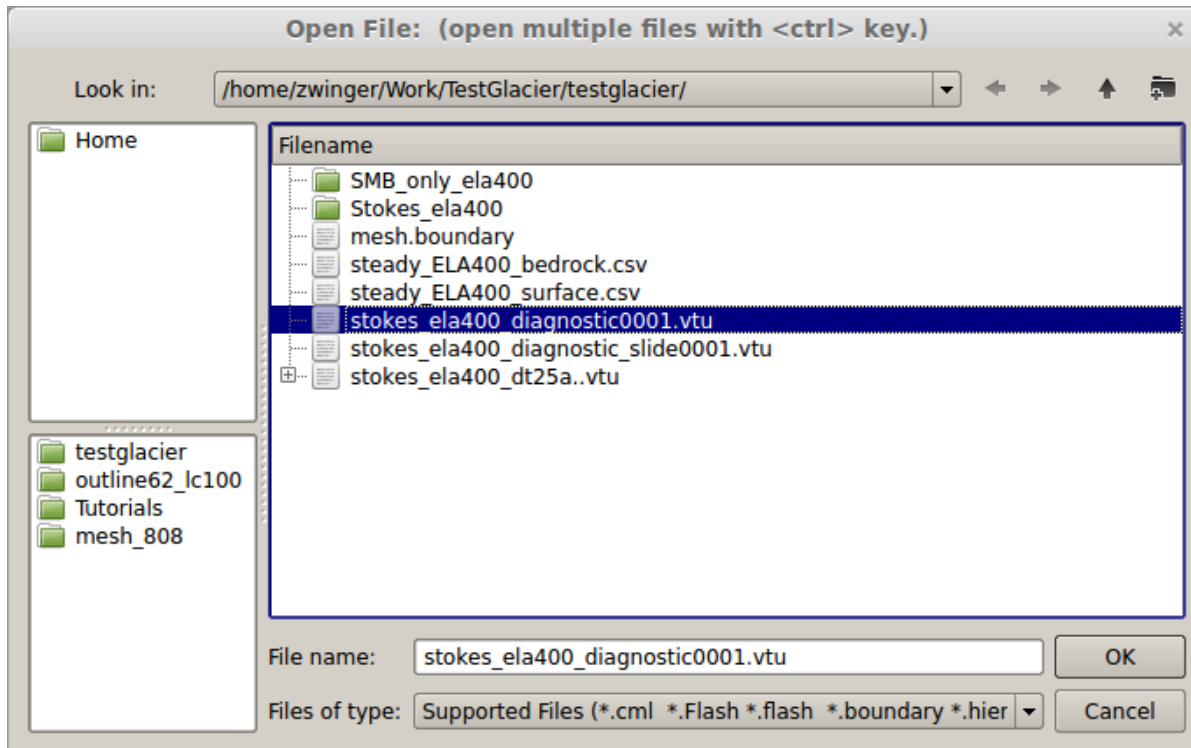
# The diagnostic problem

- Post-processing using ParaView: `$ paraview`



# The diagnostic problem

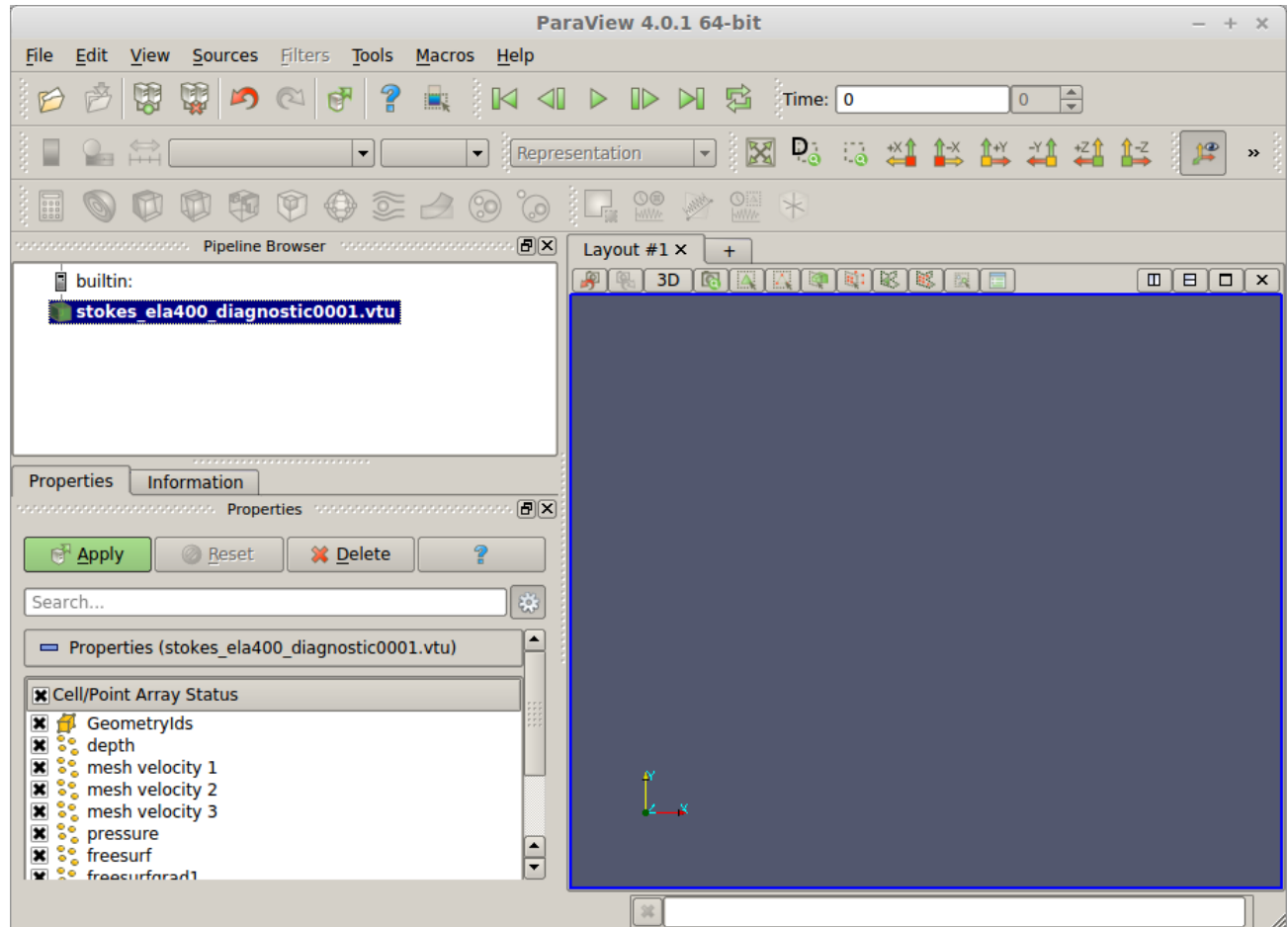
- **File → Open** stokes\_ela400\_diagnostic0001.vtu





# The diagnostic problem

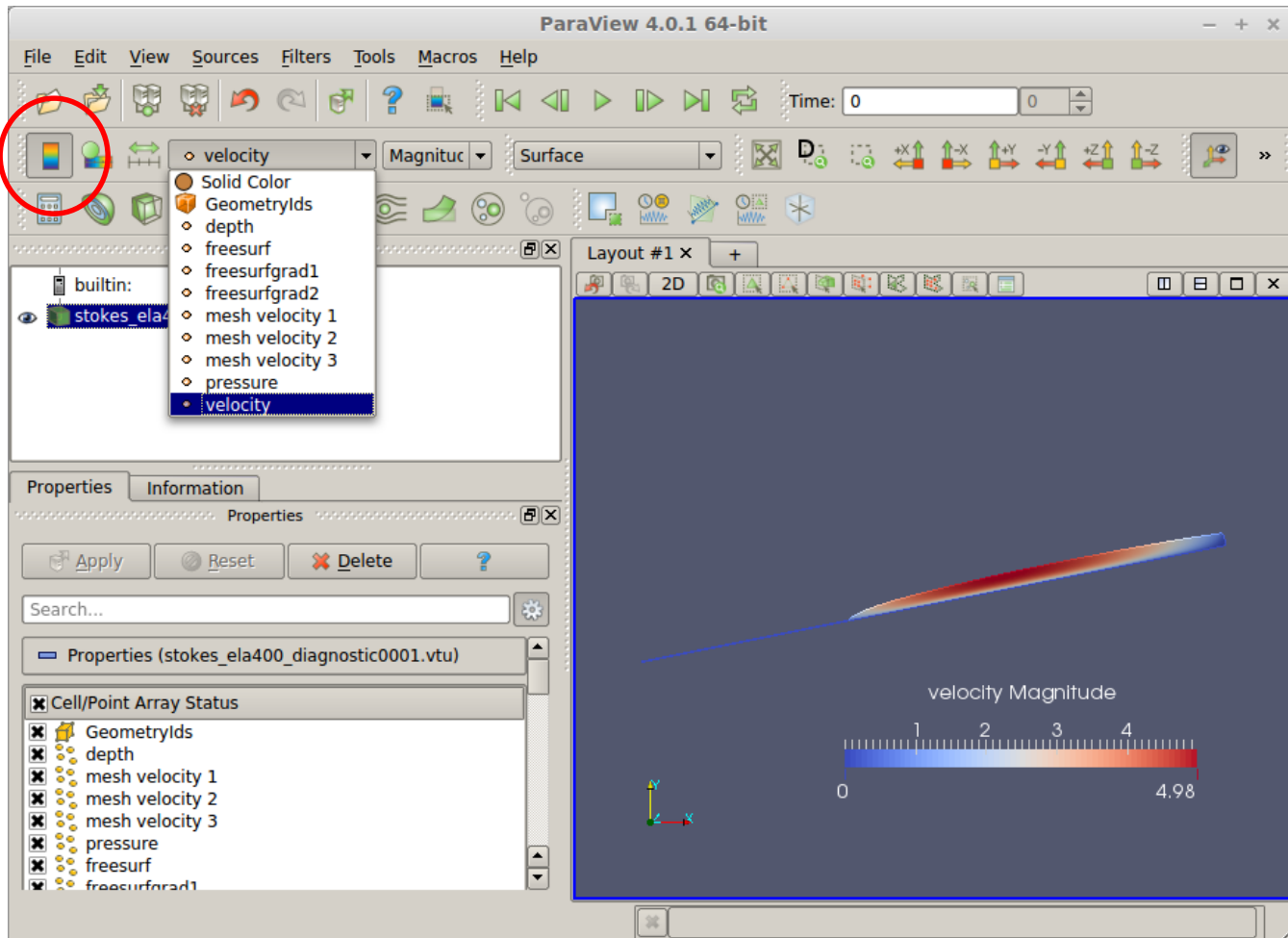
- **Apply**



# The diagnostic problem

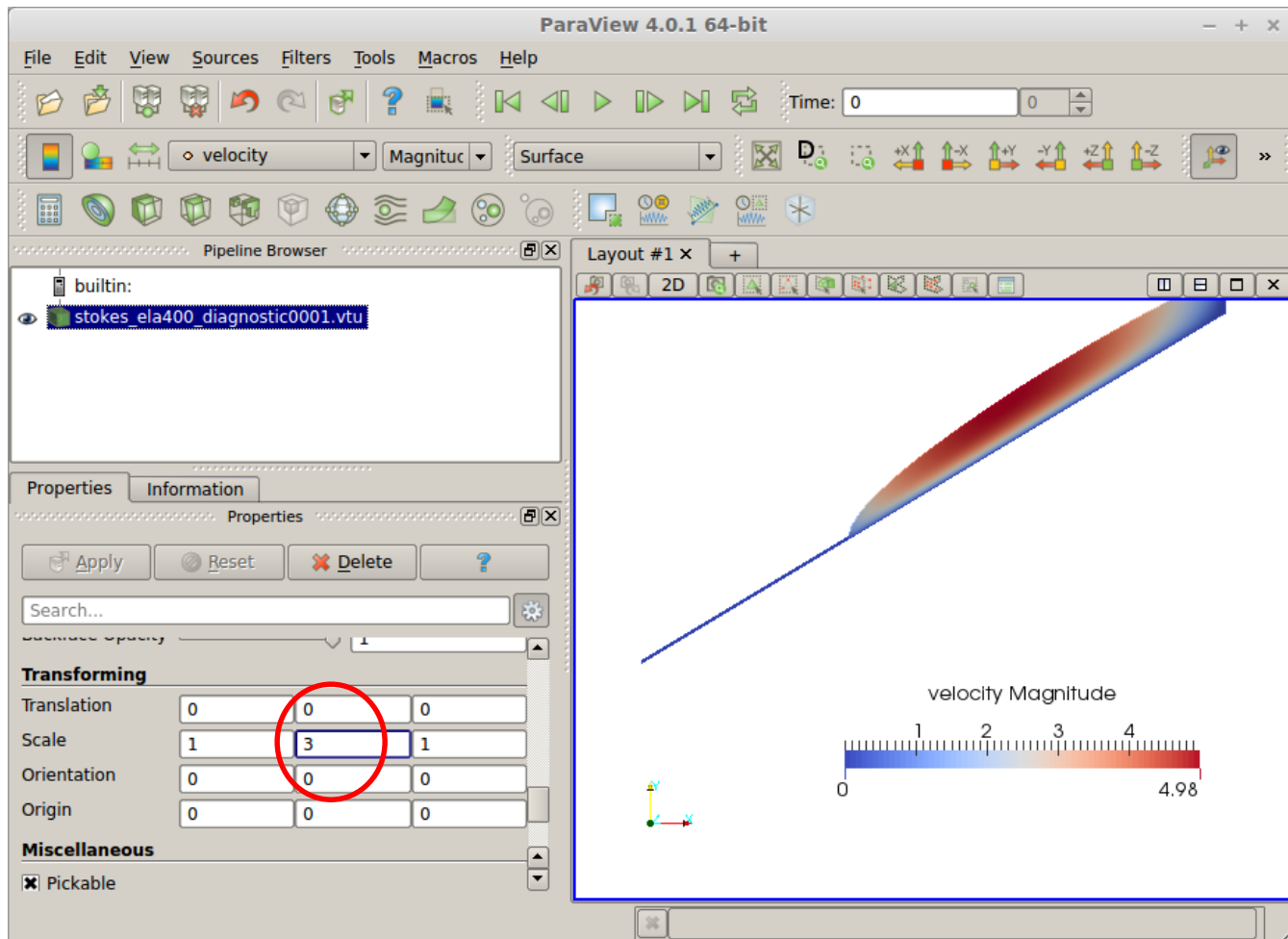
- Change to **velocity**

Press to  
activate  
colour  
bar



# The diagnostic problem

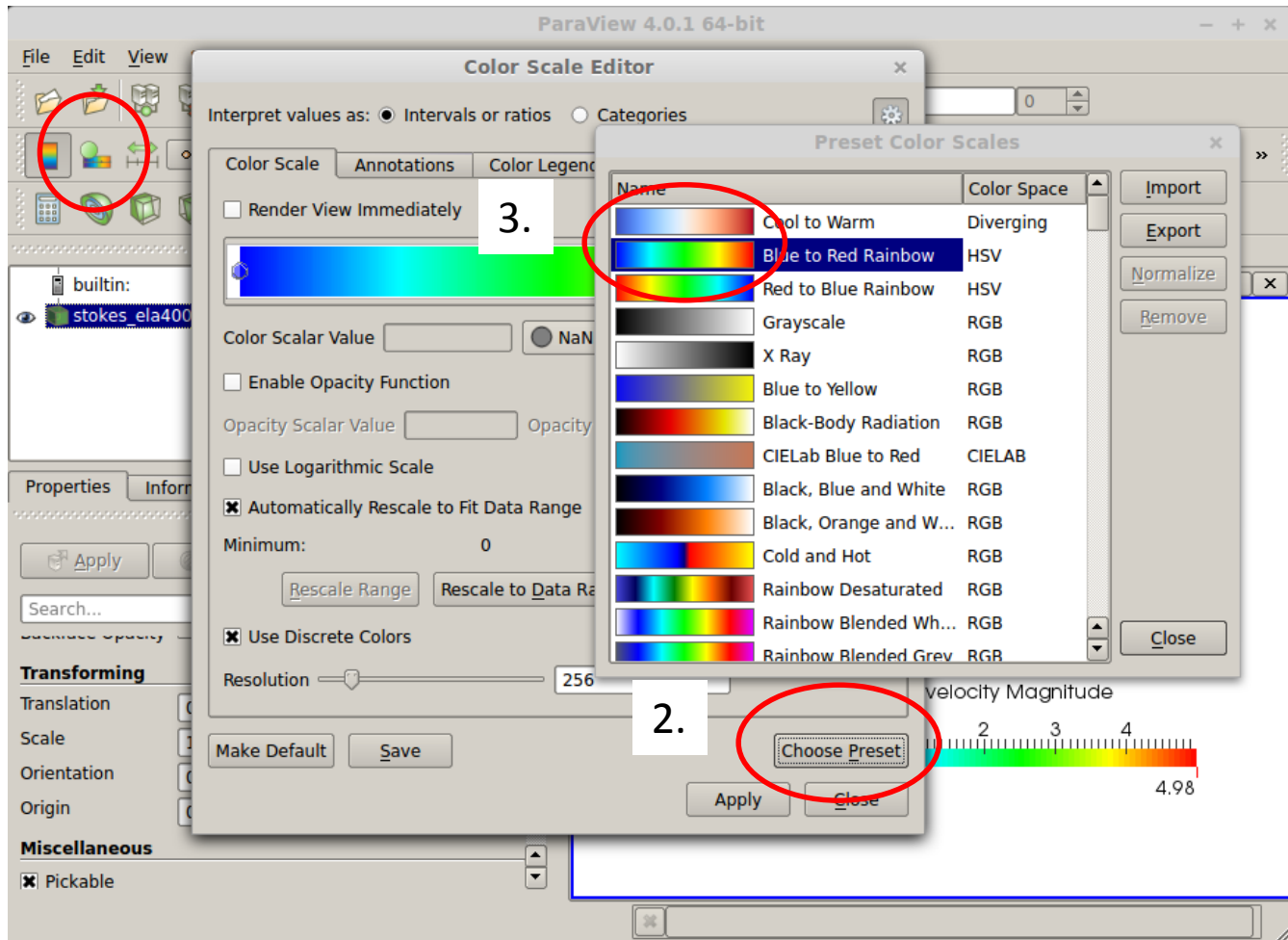
- Scale



# The diagnostic problem

- Change colours

1.



# Sliding

- Different sliding laws in Elmer
- Simplest: Linear Weertman  $\tau = \beta^2 \mathbf{u}$ 
  - This is formulated for the traction  $\tau$  and velocity  $\mathbf{u}$  in tangential plane
- In order to define properties in normal-tangential coordinates: **Normal-Tangential Velocity = True**
- $\beta^{-2}$  is the **slip Coefficient** {2,3} (for the tangential directions 2 and 3) (for 3D, in 2d only direction 2)
- Setting normal velocity to zero (no-penetration)  
**Velocity 1 = 0.0**

# Sliding

- Now we introduce sliding
  - We deploy a sliding zone between  $z=300$  and  $400\text{m}$

```
Boundary Condition 1
  Name = "bedrock"
  Target Boundaries = 1
  Compute Normals = Logical True
  ! include the bedrock DEM, which has two columns
  Bottom Surface = Variable Coordinate 1
  Real cubic
    include "steady_ELA400_bedrock.dat"
  End
  Normal-Tangential Velocity = True
  Velocity 1 = Real 0.0e0
  Slip Coefficient 2 = Variable Coordinate 2
    Real MATC "(1.0 - (tx > 300.0)*(tx < 400.0))*1000.0 + 1.0/100.0"
  End
```

Use normal-tangential coordinate system

Definition of slip Coefficient

# Sliding

```
! Flow Depth still for postprocessing, only,  
! now replaced by structured version  
Solver 2  
Equation = "HeightDepth"  
Procedure = "StructuredProjectToPlane" "StructuredProjectToPlane"  
Active Coordinate = Integer 2  
Operator 1 = depth  
Operator 2 = height  
End
```

Replace the **FlowDepth** Solver with this one. This solver simply uses the vertically structured mesh to inquire the Depth/Height without solving a PDE (much cheaper).

# Sliding

- Restart from previous run (improved initial guess)

```
Simulation
Max Output Level = 4
Coordinate System = "Cartesian 2D"
Coordinate Mapping(3) = 1 2 3
Simulation Type = "Steady"
Steady State Max Iterations = 1
Output Intervals = 1
Output File = "Stokes_ELA400_diagnostic_slide.result"
Post File = "Stokes_ELA400_diagnostic_slide.vtu"
Initialize Dirichlet Conditions = Logical False
! Restart from previous run
Restart File = "Stokes_ELA400_diagnostic.result"
Restart Position = 0 Take last entry
End
```



# The diagnostic problem

- Now, run the case:

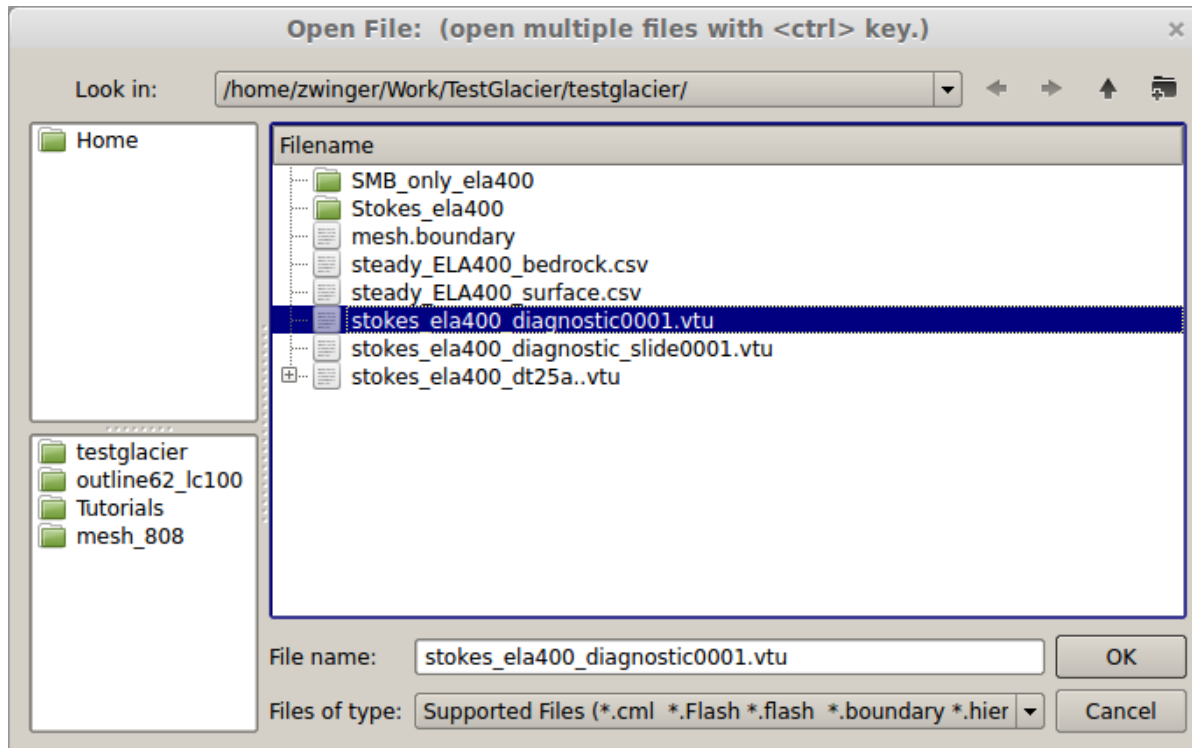
```
$ ElmerSolver Stokes_diagnostic_slide.sif
```

– Converged much earlier:

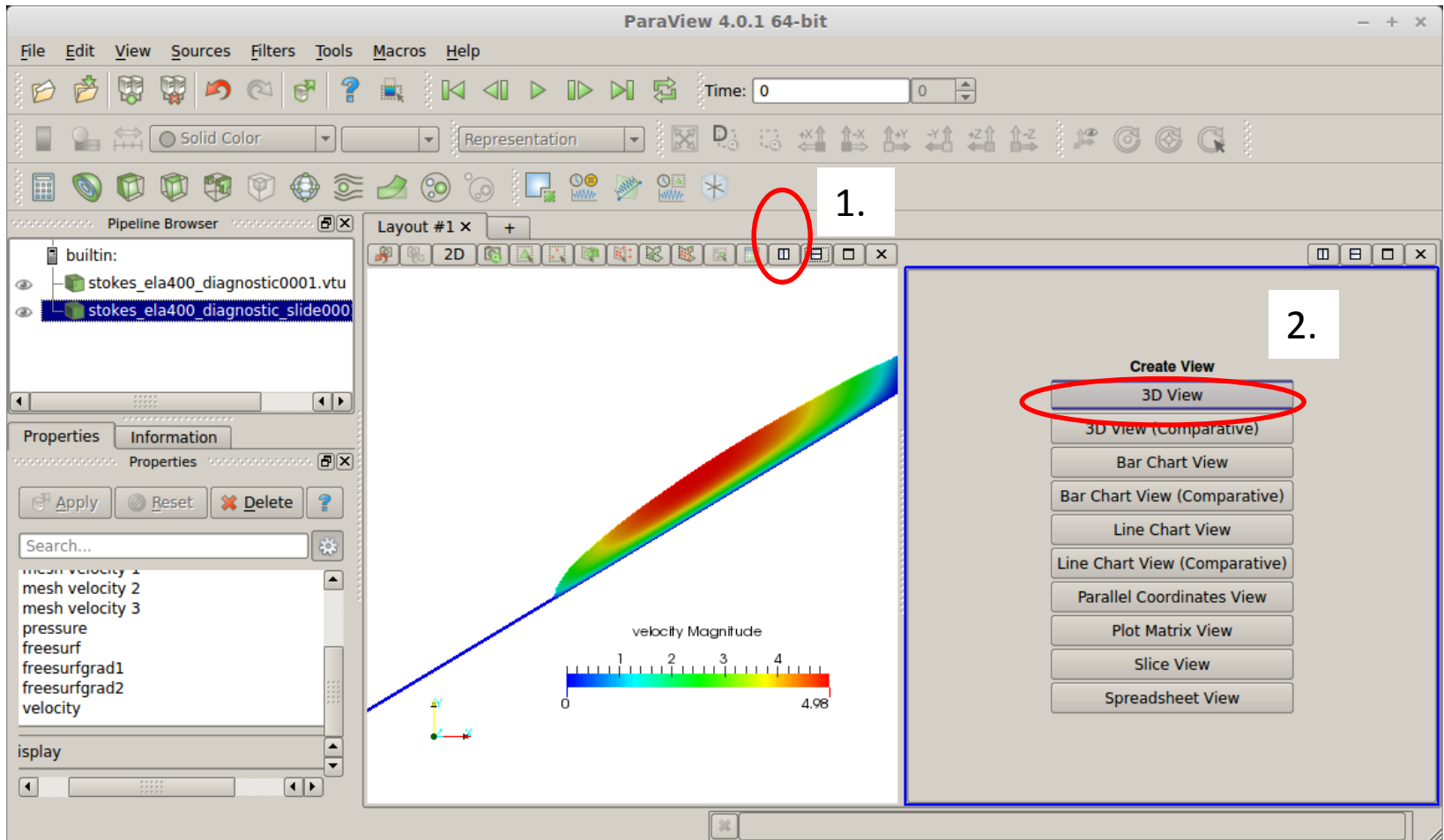
```
FlowSolve: -----  
FlowSolve:  NAVIER-STOKES ITERATION                12  
FlowSolve: -----  
FlowSolve:  
FlowSolve: Starting Assembly...  
FlowSolve: Assembly done  
FlowSolve: Dirichlet conditions done  
ComputeChange: NS (ITER=12) (NRM,RELC): ( 3.4915753  
0.34732117E-05 ) :: navier-stokes  
FlowSolve: iter:   12 Assembly: (s)      0.32    3.53  
FlowSolve: iter:   12 Solve:    (s)      0.12    1.38  
FlowSolve: Result Norm      :      3.4915753430899730  
FlowSolve: Relative Change  :      3.4732116934487441E-006  
ComputeChange: SS (ITER=1) (NRM,RELC): ( 3.4915753  
2.0000000      ) :: navier-stokes
```

# Sliding

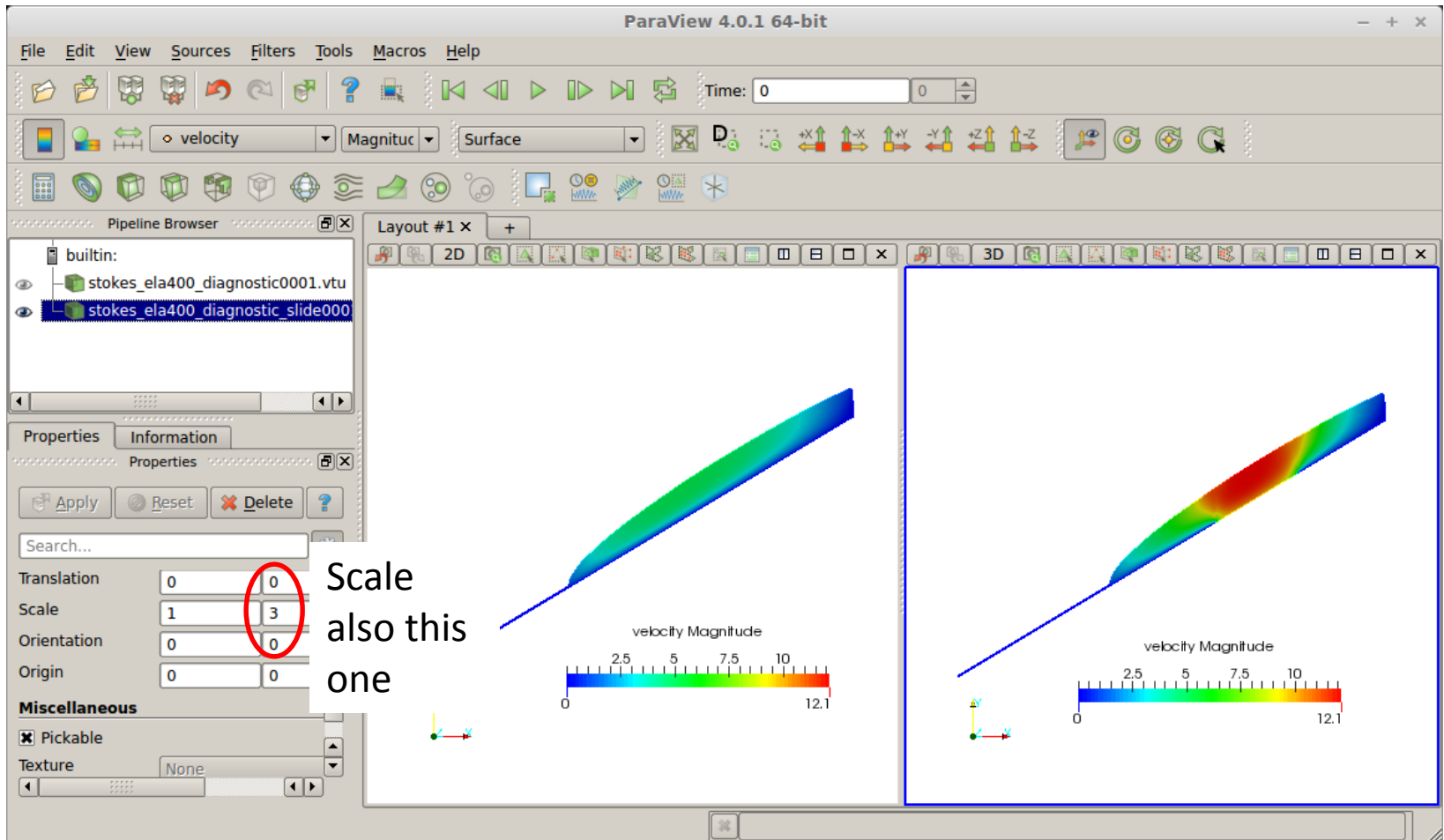
- Load parallel to previous file
- **File → Open**  
`stokes_ela400_diagnostic_slide0001.vtu`



# Sliding



# Sliding



# Heat transfer

- Adding heat transfer:
  - Add `ElmerIceSolvers` `TemperateIceSolver` with variable name `Temp` (see next slide)
  - Surface temperature distribution: linear from 273.15 K at  $z=0\text{m}$  to 263.15 K at  $z=1000\text{m}$

```
Temp = Variable Coordinate 2
  Real
    0.0    273.15
    1000.0 263.15
  End
```

- Geothermal heat flux of 200 mW  $\text{m}^{-2}$  at bedrock

```
Temp Flux BC = Logical True
Temp Heat Flux = Real $ 0.200 * (31556926.0)*1.0E-06
```

# Heat transfer

Solver 4

```
Equation = String "Homologous Temperature Equation"  
Procedure = File "ElmerIceSolvers" "TemperateIceSolver"  
Variable = String "Temp"  
Variable DOFs = 1  
Stabilize = True  
Optimize Bandwidth = Logical True  
Linear System Solver = "Iterative"  
Linear System Direct Method = UMFPACK  
Linear System Convergence Tolerance = 1.0E-06  
Linear System Abort Not Converged = False  
Linear System Preconditioning = "ILU1"  
Linear System Residual Output = 0  
Nonlinear System Convergence Tolerance = 1.0E-05  
Nonlinear System Max Iterations = 100  
Nonlinear System Relaxation Factor = Real 9.999E-01  
Steady State Convergence Tolerance = 1.0E-04
```

End

# Heat transfer

- Material parameters in Material section

```
Material 1
...
! Heat transfer stuff
Temp Heat Capacity = Variable Temp
  Real MATC "capacity(tx) * (31556926.0) ^ (2.0) "

Temp Heat Conductivity = Variable Temp
  Real MATC "conductivity(tx) * 31556926.0 * 1.0E-06"
End
```

– Capacity:  $c(T) = 146.3 + (7.253 \cdot T[\text{K}])$

– Conductivity:  $\kappa(T) = 9.828 \exp(-5.7 \times 10^{-3} \cdot T[\text{K}])$

# Heat transfer

- Material parameters in Material section

```
!! conductivity
$ function conductivity(T) { _conductivity=9.828*exp(-5.7E-03*T) }
!! capacity
$ function capacity(T) { _capacity=146.3+(7.253*T) }
```

- Using defined MATC-functions for

– Capacity:  $c(T) = 146.3 + (7.253 \cdot T[\text{K}])$

– Conductivity:  $\kappa(T) = 9.828 \exp(-5.7 \times 10^{-3} \cdot T[\text{K}])$



# Heat transfer

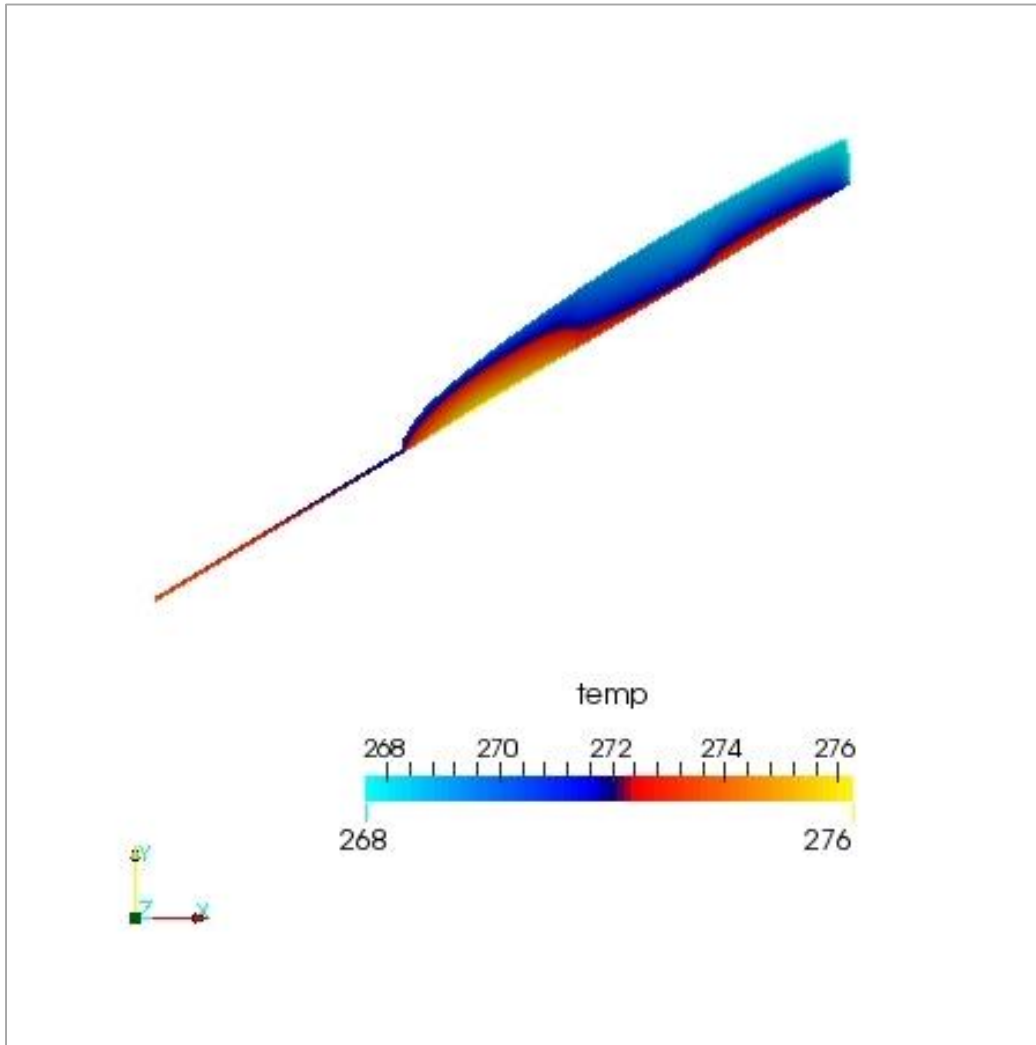
- Now, run the case:

```
$ ElmerSolver Stokes_diagnostic_temp.sif
```

- It goes pretty quick, as we only have one-way coupling and hence don't even execute the Stokes solver

```
Solver 3  
  Exec Solver = "Never" ! we have a solution from previous case  
  Equation = "Navier-Stokes"
```

# Heat transfer



- Due to high geothermal heatflux we have areas above pressure melting point
- We have to account for this

# Heat transfer

- Constrained heat transfer:
  - Including following lines in Solver section  
ElmerIceSolvers TemperatureIce

```
! the contact algorithm (aka Dirichlet algorithm)
!-----
Apply Dirichlet = Logical True
! those two variables are needed in order to store
! the relative or homologous temperature as well
! as the residual
!-----
Exported Variable 1 = String "Temp Homologous"
Exported Variable 1 DOFs = 1
Exported Variable 2 = String "Temp Residual"
Exported Variable 2 DOFs = 1
```

# Heat transfer

- Constrained heat transfer:
  - Also introduce the upper limit for the temperature (a.k.a. pressure melting point) in the Material section

```
Temp Upper Limit = Variable Depth  
Real MATC "273.15 - 9.8E-08 * tx * 910.0 * 9.81"
```

$$T_{\text{pm}} = T_0 + \beta_{\text{c}} p$$

$$p \approx \rho_{\text{ice}} g d$$

# Heat transfer

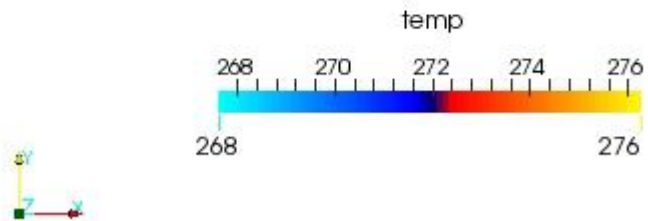
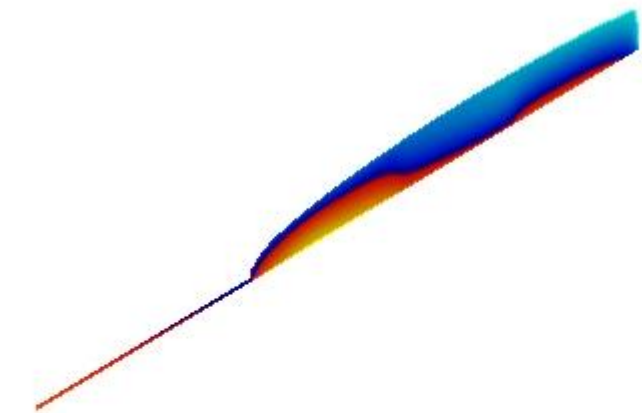
- Now, run the case:

```
$ ElmerSolver \
    Stokes_diagnostic_temp_constrained.sif
```

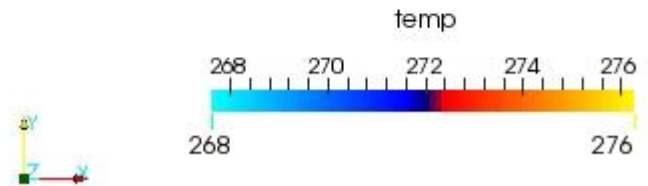
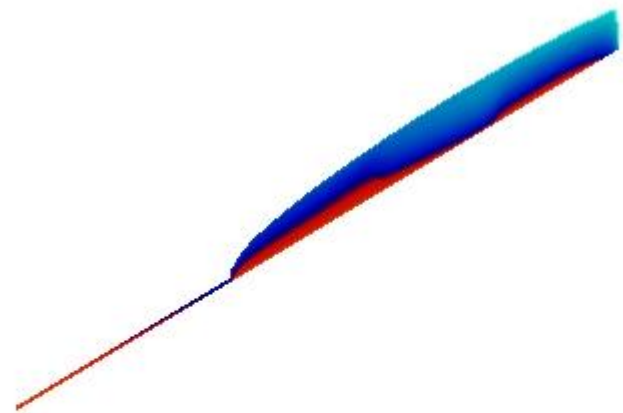
- Already from the norm (~ averaged nodal values) it comes clear that values are in general now lower

```
TemperateIceSolver (temp): iter:      5 Assembly: (s)      1.36      6.77
TemperateIceSolver (temp): iter:      5 Solve:      (s)      0.00      0.01
TemperateIceSolver (temp): Result Norm   :      271.78121462656480
TemperateIceSolver (temp): Relative Change :
5.0215061382786350E-006
ComputeChange: SS (ITER=1) (NRM,RELC): ( 271.78121      2.0000000
) :: homologous temperature equation
```

# Heat transfer



Unconstrained



Constrained

# Heat transfer

- Thermo-mechanically coupled simulation:
  - We have to iterate between Stokes and HTEq.

```
Steady State Max Iterations = 20
```

- Coupling to viscosity in Material section

```
! the variable taken to evaluate the Arrhenius law  
! in general this should be the temperature relative  
! to pressure melting point. The suggestion below plugs  
! in the correct value obtained with TemperateIceSolver  
Temperature Field Variable = String "Temp Homologous"
```

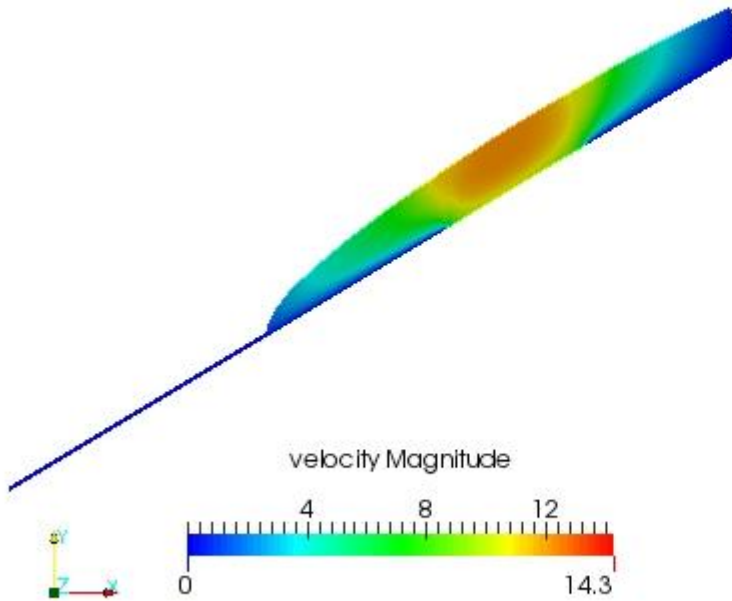
# Newton Iterations

- We need Picard (=fixed-point) iterations instead of Newton iterations at the beginning of each new non-linear iteration loop

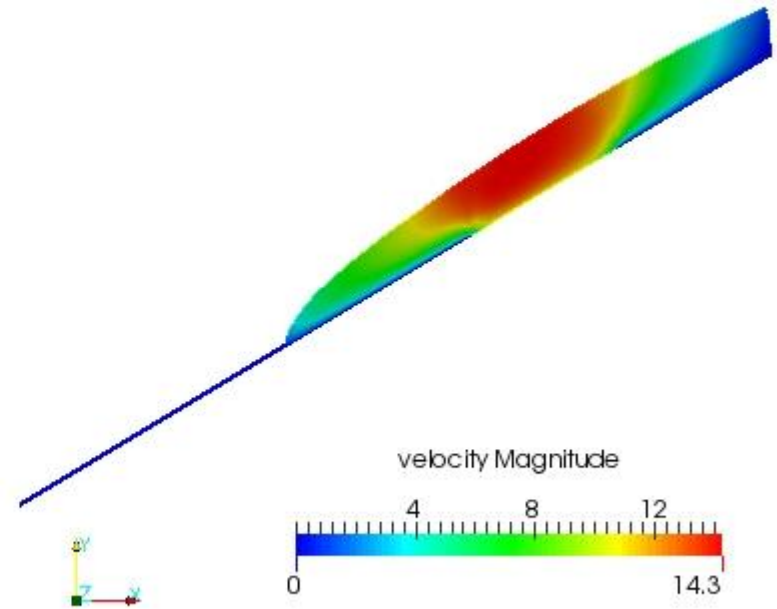
```
Solver 1
! Exec Solver = "Never"
Equation = "Navier-Stokes"
...
Nonlinear System Reset Newton = Logical True
!Nonlinear System Relaxation Factor = 0.75
End
```



# Heat transfer



Uncoupled (constant T)



Thermo-mechanically coupled

# PROGNOSTIC RUN

Starting from a deglaciated situation we show

- How to move to a transient run, i.e., introduce the
  - Free surface solution
  - Including coupling to climate via prescribing an accumulation/ablation function
- How to write a less simple MATC function

# The prognostic problem

- Glacier with  $\sim 11$  deg constant inclination
- Standard accumulation/ablation function

$$a(z) = \lambda z + a(z = 0)$$

- Or in terms of ELA (equilibrium line altitude):

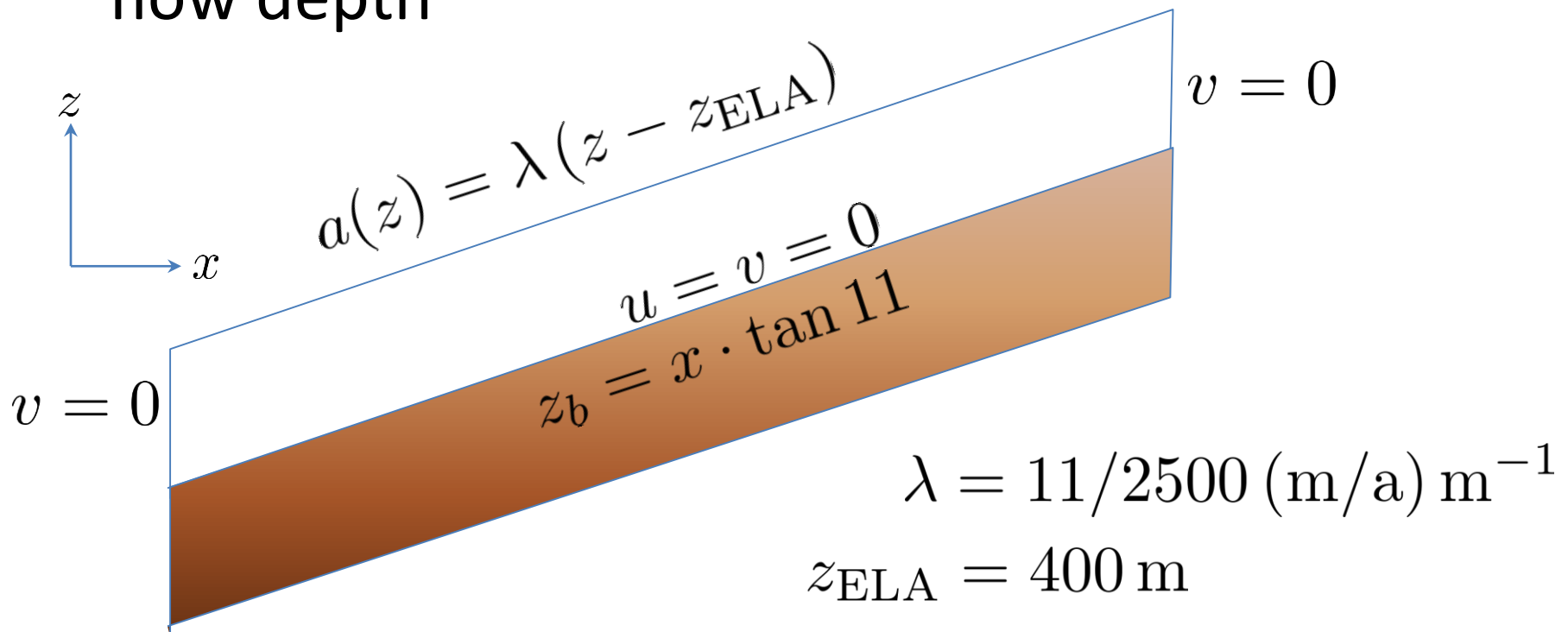
$$a_{\text{ELA}} = \lambda z_{\text{ELA}} + a_0 = 0$$

- We know lapse rate,  $\lambda$ , and  $z_{\text{ELA}}$  and have to define

$$a_0 = -\lambda z_{\text{ELA}}$$

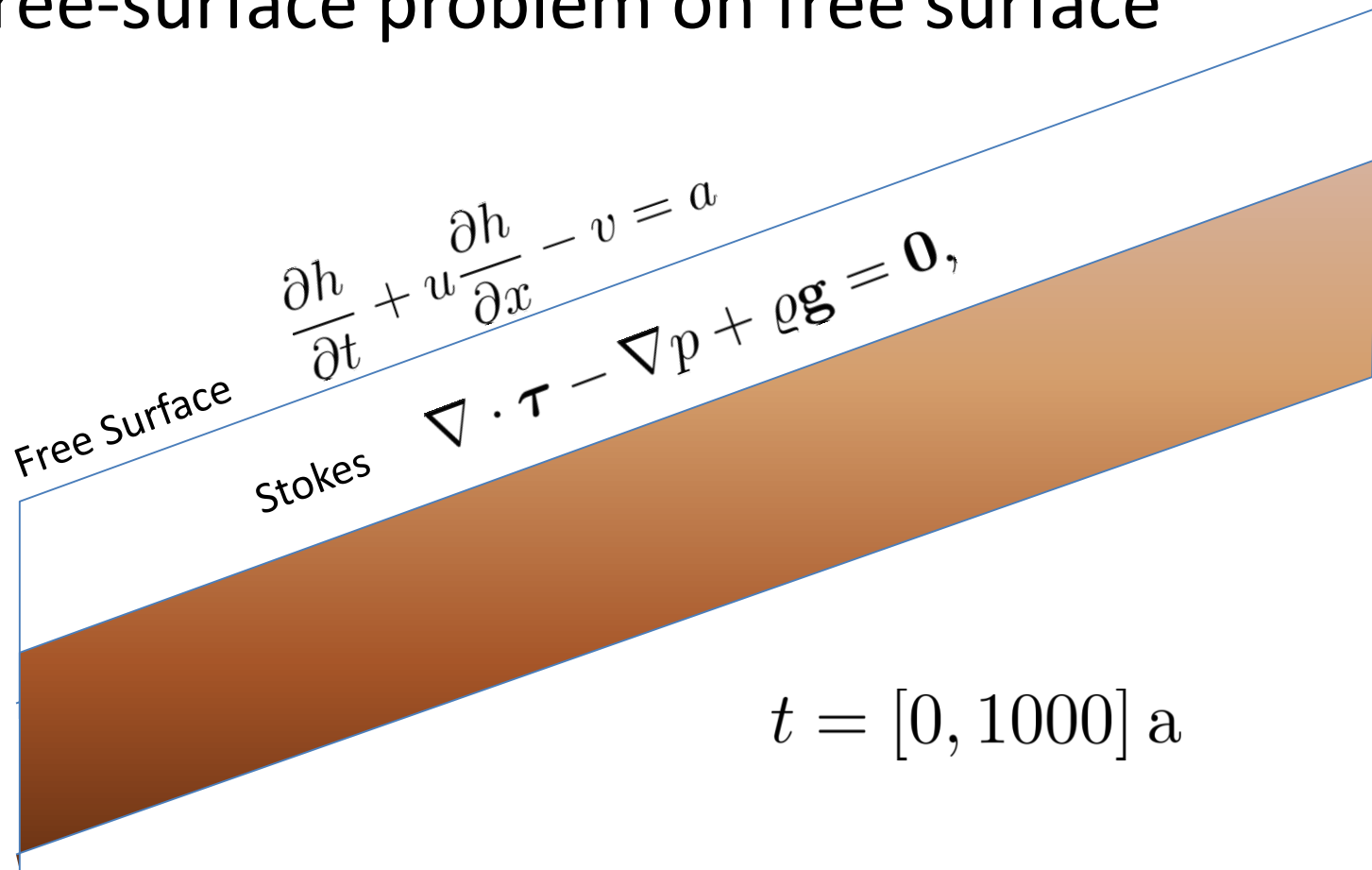
# The Problem

- From  $x=[0 : 2500]$ ,  $z=[0:500]$
- Setting mesh with 10 vertical levels with 5m flow depth



# The Problem

- Flow problem (Navier-Stokes) in ice
- Free-surface problem on free surface



# Free Surface Equation

Solver 4

```
Equation = String "Free Surface"
```

```
Procedure = File "FreeSurfaceSolver" "FreeSurfaceSolver"
```

```
Exec Solver = always
```

```
Variable = String "Zs"
```

```
Variable DOFs = 1
```

```
! needed for evaluating the contact pressure
```

```
Exported Variable 1 = -dofs 1 "Zs Residual"
```

```
! needed for storing the initial shape (needed for updates)
```

```
Exported Variable 2 = -dofs 1 "RefZs"
```

```
Procedure = "FreeSurfaceSolver" "FreeSurfaceSolver"
```

```
! This would take the constrained points out of solution
```

```
! Use in serial run, only
```

```
! Before Linsolve = "EliminateDirichlet" "EliminateDirichlet"
```

# Free Surface Equation

```
Linear System Solver = Iterative
Linear System Max Iterations = 1500
Linear System Iterative Method = BiCGStab
Linear System Preconditioning = ILU0
Linear System Convergence Tolerance = Real 1.0e-7
Linear System Abort Not Converged = False
Linear System Residual Output = 1
Nonlinear System Max Iterations = 100
Nonlinear System Convergence Tolerance = 1.0e-6
Nonlinear System Relaxation Factor = 0.60
Steady State Convergence Tolerance = 1.0e-03
Stabilization Method = Bubbles
! Apply contact problem
Apply Dirichlet = Logical True
```

End

# Free Surface Equation

```
Body 2
```

```
  Name = "Surface"
```

```
  Body Force = 2
```

```
  Equation = 2
```

```
  Material = 2
```

```
  Initial Condition = 2
```

```
End
```

```
Equation 2
```

```
  Name = "Equation2"
```

```
  Convection = "computed"
```

```
  Active Solvers(1) = 3
```

```
  Flow Solution Name = String "Flow Solution"
```

```
End
```



# Free Surface Equation

```
Boundary Condition 3
```

```
Name = "surface"
```

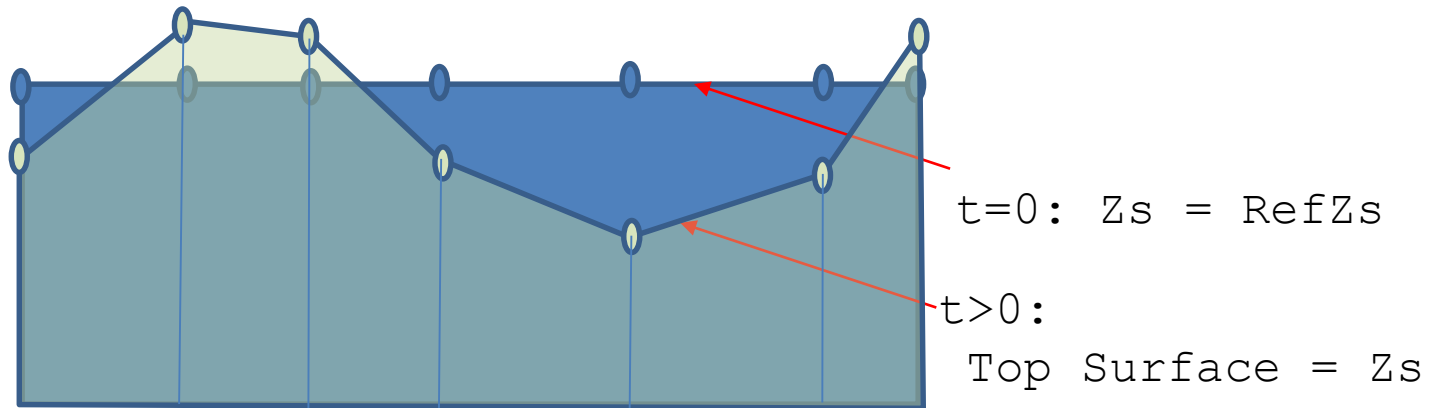
```
Top Surface = Equals "Zs"
```

```
Target Boundaries = 2
```

```
Body ID = 2
```

```
Depth = Real 0.0
```

```
End
```



# Free Surface Equation

- Starting with same values for both variables

```
Initial Condition 2
```

```
Zs = Equals Coordinate 2
```

```
RefZs = Equals Coordinate 2
```

```
End
```

- Using the latter to keep minimal height

```
Material 2
```

```
Min Zs = Variable RefZs
```

```
Real MATC "tx - 0.1"
```

```
Max Zs = Variable RefZs
```

```
Real MATC "tx + 600.0"
```

```
End
```

# Free Surface Equation

- And here comes the coupling to climate

```
Body Force 2
```

```
  Name = "Climate"
```

```
  Zs Accumulation Flux 1 = Real 0.0e0
```

```
  Zs Accumulation Flux 2 = Variable Coordinate 1, Coordinate 2
```

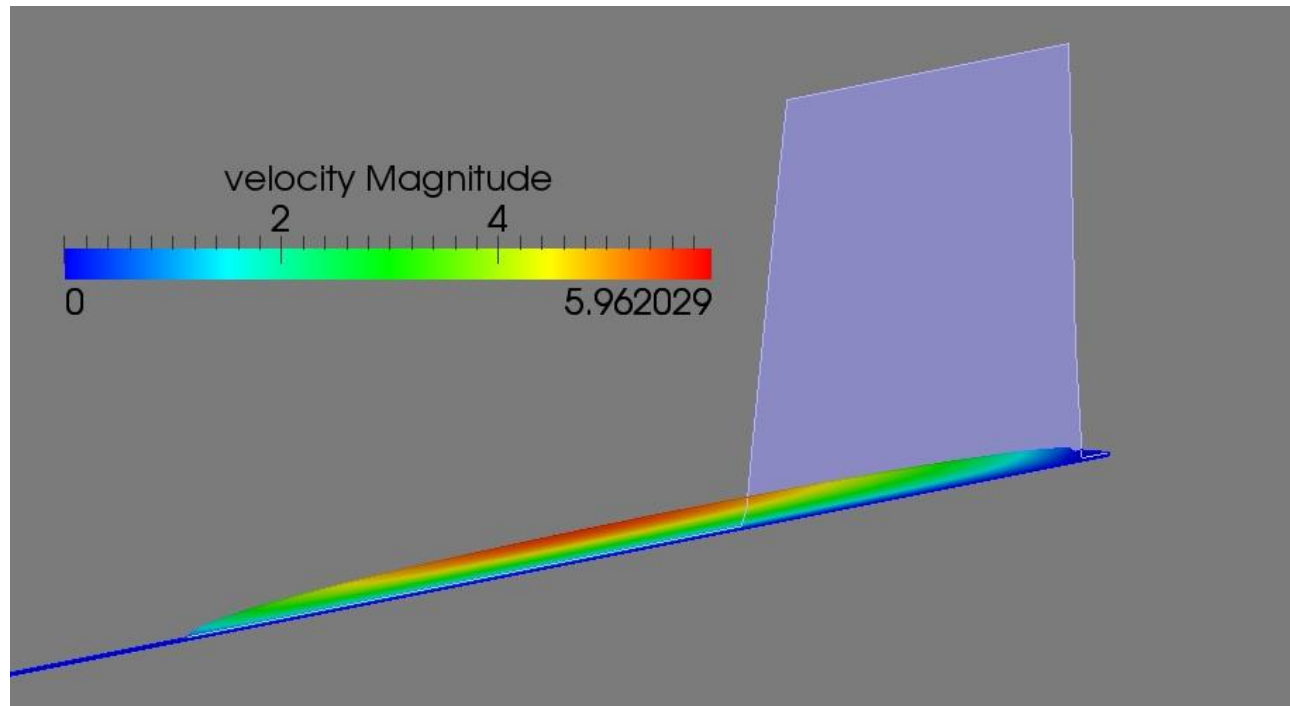
```
    Real MATC "accum(tx)"
```

```
End
```

```
$ function accum(X) {\
  lapserate = (11.0/2750.0);\
  ela = 400.0;\
  atsl = -ela*lapserate;\
  if (X(0) > 2500)\
    {_accum = 0.0;}\
  else\
    { _accum = lapserate*X(1) + atsl;}\
}
```

# The Solution

- Starting with no-flow problem, i.e., only surface mass balance, simply by setting Convection = “none” and (saves time) not executing Navier-Stokes, compare to run with coupled flow
- \$ **ElmerSolver Stokes\_prognostic.sif**



# USER DEFINED FUNCTION

In a follow-up session, by changing the previous setup we show

- How to write, compile and include a self-written user defined function
- How to introduce time changing variables

# User Defined Function

- Replace the MATC function with a user defined function (UDF)
- All UDF's have the same header in Elmer(/Ice):

```
FUNCTION getAccumulation( Model, Node, InputArray) RESULT (accum)
  ! provides you with most Elmer functionality
  USE DefUtils
  ! saves you from stupid errors
  IMPLICIT NONE
  ! the external variables
  !-----

  TYPE(Model_t) :: Model      ! the access point to everything
  about the model
  INTEGER :: Node             ! the current Node number
  REAL(KIND=dp) :: InputArray(2) ! Contains the arguments passed
  to the function
  REAL(KIND=dp) :: accum      ! the result
  !-----
```

# User Defined Function

```
!-----  
! internal variables  
!-----  
REAL(KIND=dp) :: lapserate, ela0, dElaDt, elaT, accumulationAtSl,&  
    inittime, time, elevation, cutoff, offset  
LOGICAL :: FirstTime=.TRUE.  
! Remember this value  
SAVE FirstTime, inittime  
  
! lets hard-code our values (if we have time we can later make them being read  
from SIF)  
lapserate = 11.0_dp/2750.0_dp  
ela0 = 400.0_dp  
dElaDt = -0.1_dp  
cutoff = 600.0_dp  
offset = 1500.0  
  
! copy input (should match the arguments!)  
elevation = InputArray(1)  
time = InputArray(2)  
WRITE (Message, '(A,E10.2,A,E10.2)') "elevation=", elevation, "time=", time  
CALL INFO("getAccumulation", Message, Level=9)
```

# User Defined Function

```
! store the initial time, to be sure to have relative times
IF (FirstTime) THEN
    inittime = time
    FirstTime = .FALSE.
END IF

! get change of ELA with time
IF (time > offset) THEN
    elaT = ela0 - dElaDt * (time - offset)
ELSE
    elaT = ela0
END IF

! lets do the math
accumulationAtSl = -elaT*lapserate
IF (elevation > cutoff) elevation = cutoff
accum = lapserate*elevation + accumulationAtSl

RETURN

END FUNCTION getAccumulation
```



# User Defined Function

The body-force section changes to:

```
Body Force 2
  Name = "Climate"
  Zs Accumulation Flux 1 = Real 0.0e0
  Zs Accumulation Flux 2 = Variable Coordinate 2, Time
    Real Procedure "accumulation" "getAccumulation"
End
```

Compilation is done with:

```
$ elmerf90 accumulation.f90 -o accumulation.so
```