Semi-Lagrangian Advection in Elmer

Applications

Cyrille Mosbeux and P. Raback

In collaboration with

O. Gagliardini & N.Jourdain, A. Gilbert & F.Gillet-Chaulet

27 April 2023 - EGU
ElmerIce User’s Meeting

Université Grenoble Alpes

CRYOSPHERE & SEA LEVEL

Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage) be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u}\nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage) be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

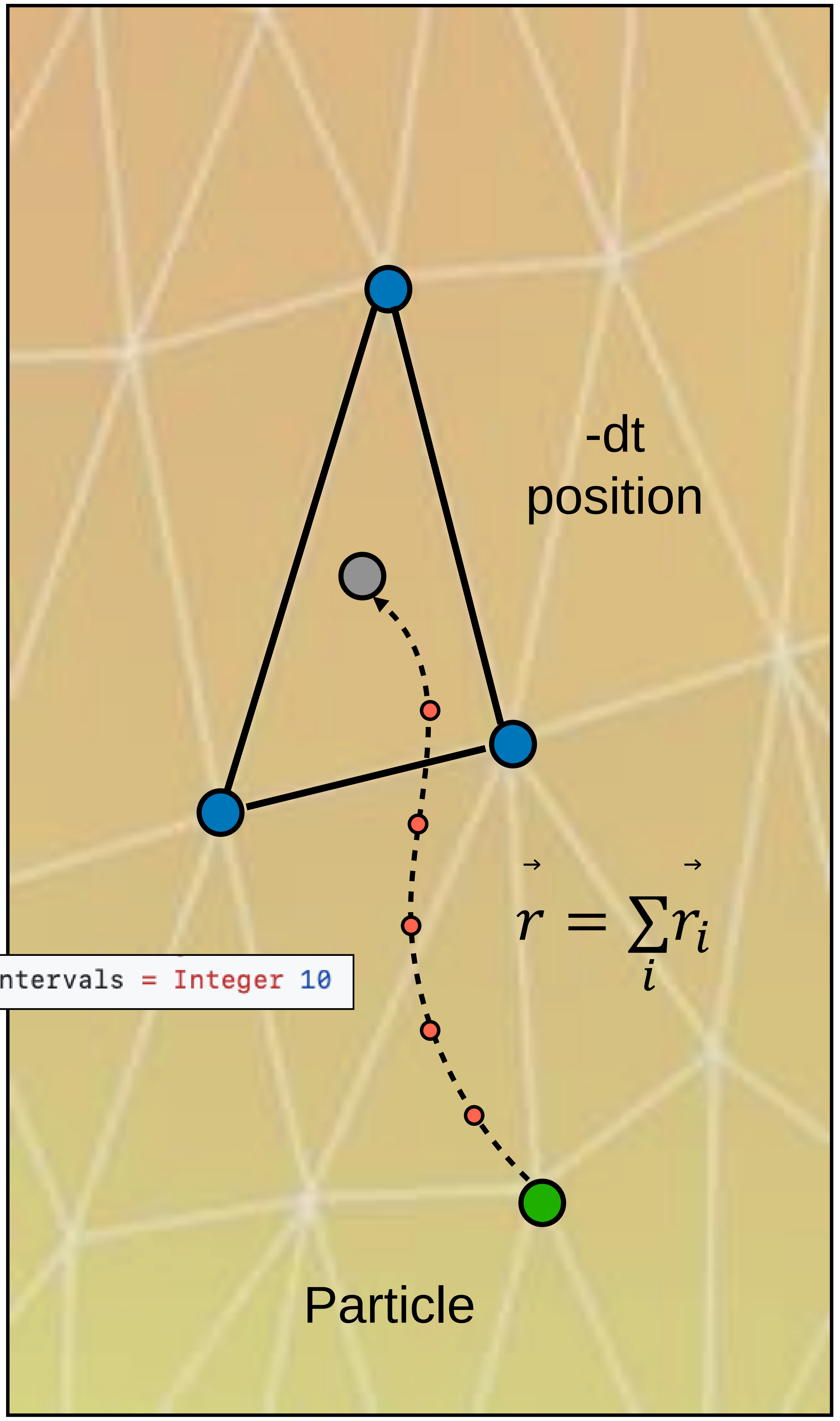
$$\frac{\partial D}{\partial t} + \boldsymbol{u}\nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

- Another solution is to use a Semi-Lagrangian Framework, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle



Max Timestep Intervals = Integer 10

Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage)be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u}\nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

- Another solution is to use a Semi-Lagrangian Framework, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle

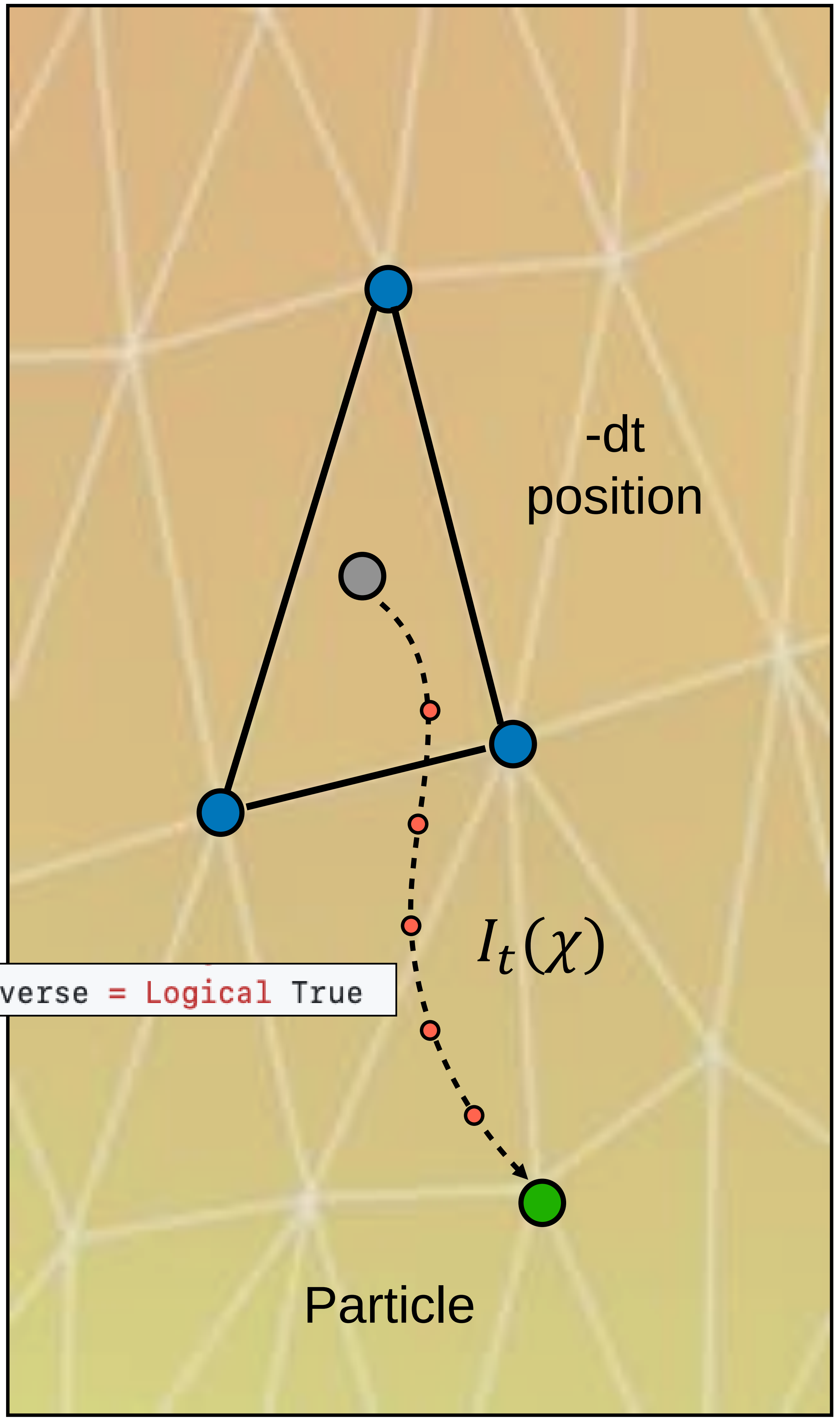
Here the source term can be expressed as

$$I_t(\chi) = \int_0^{-\delta t} f(\chi)\, dt$$

To track the evolution of the particle given $f(\chi)$

The evolution of damage over time and space can finally be calculated following

$$D = D(\vec{r_0}, t) + I_t(\chi)$$



Particle time reverse = Logical True

Principle of the semi-Lagrangian Method

- Any quantity can (e.g. damage)be advected in an Eulerian Framework following:

Source term : $\chi(\sigma, D)$

$$\frac{\partial D}{\partial t} + \boldsymbol{u}\nabla D = f(\chi)$$

But it comes with substantial numerical diffusion…

- Another solution is to use a Semi-Lagrangian Framework, following a particle (damage defined at a node) trajectory ($\vec{r}$) over time

$$\vec{r} = \vec{r_0} + \int_0^{-\delta t} \boldsymbol{u}\, dt$$

Track the transport of the particle

Here the source term can be expressed as

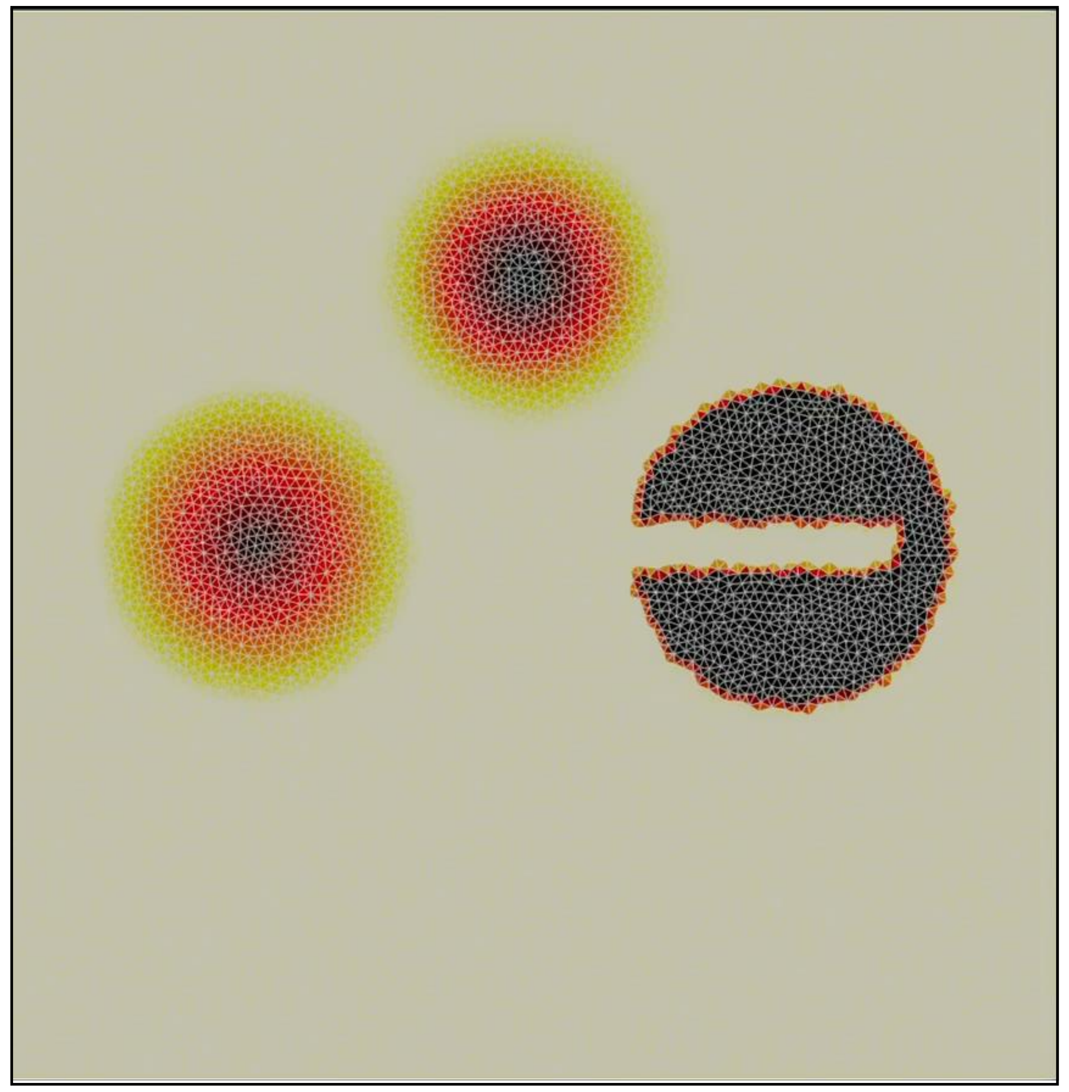
$$I_t(\chi) = \int_0^{-\delta t} f(\chi)\, dt$$

To track the evolution of the particle given $f(\chi)$

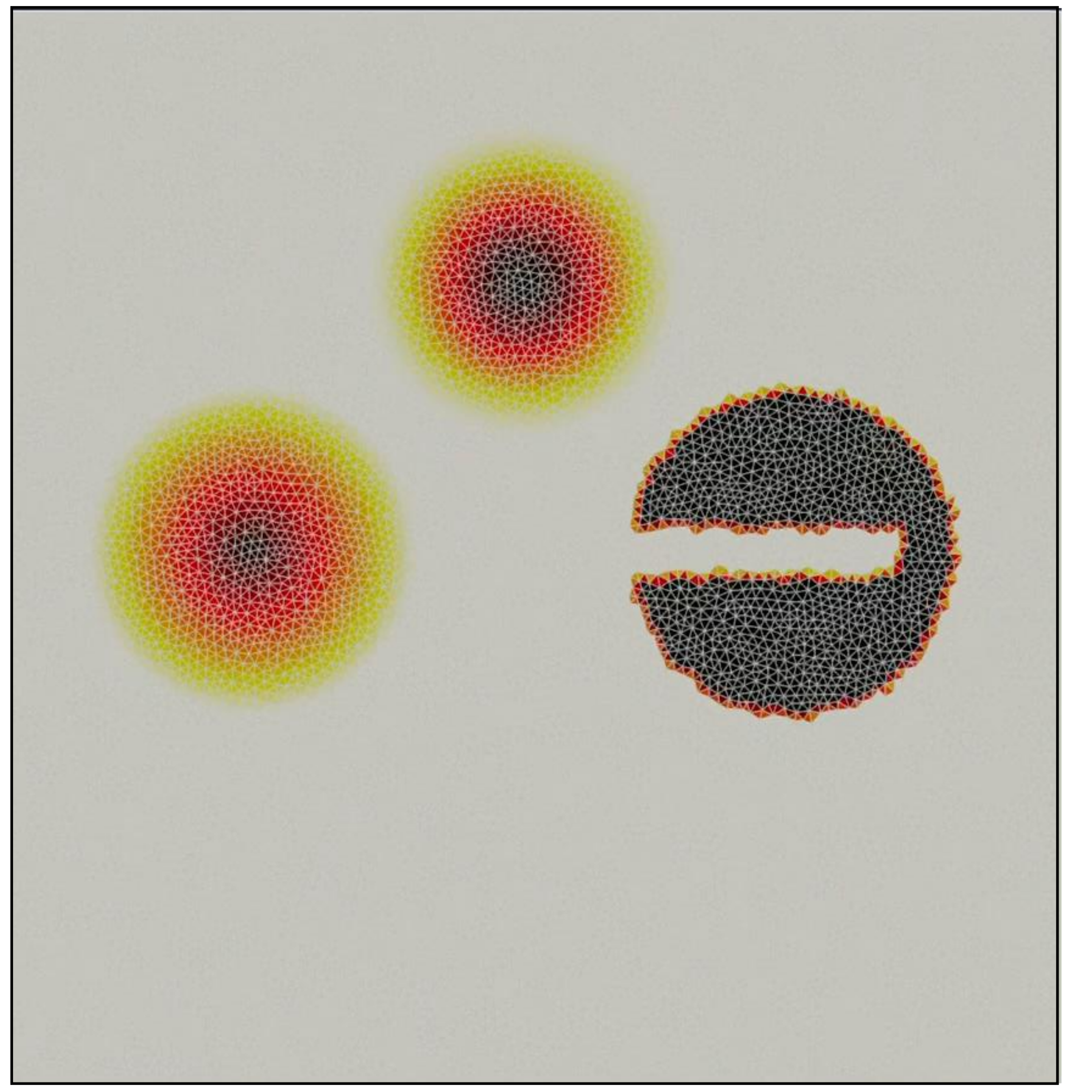
The evolution of damage over time and space can finally be calculated following

$$D = D(\vec{r_0}, t) + I_t(\chi)$$

Advection - Diffusion

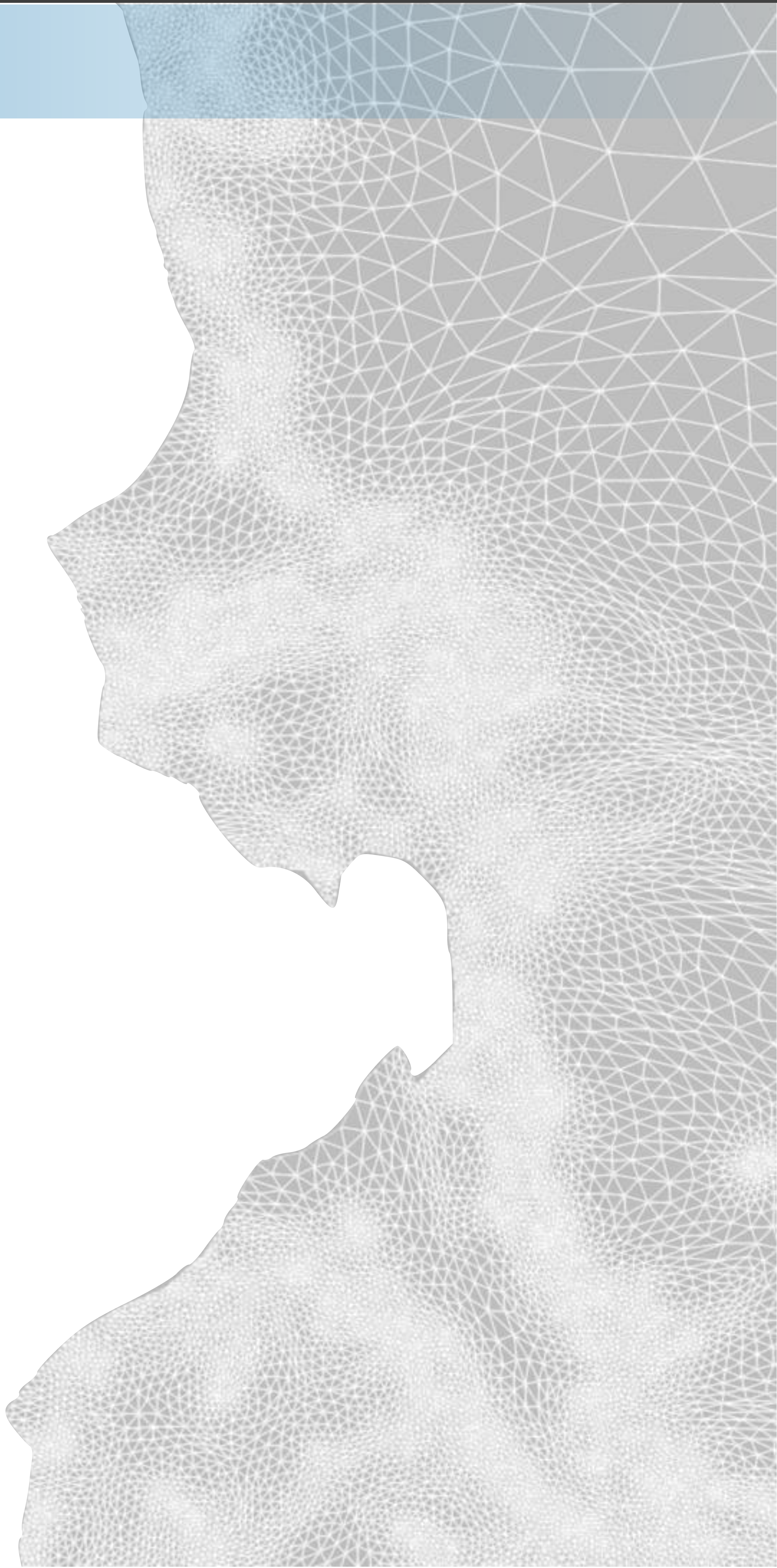


Semi-Lagrangian



The Semi-Lagrangian Solver in Elmer

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```



The Semi-Lagrangian Solver in Elmer

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinit
  !Parti

! Timeste
  Simula

  Partic
  Max Ti

! Time i
  Timeste
  Max Ti

! Give u
  Max In

! Integr
  Runge

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

```
! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values
```

Particles can be defined at:

1. Node
2. Element
3. Integration point
4. Discontinuous Galerkin nodes

For transient simulations, particles should be reinitialised every timestep… longer the timestep lesser the interpolation error

Timestep (external) can be fixed to the simulation timestep

More internal timestep intervals give an overall better results but is also more expensive (5-10 has been a good number for test cases)

The Semi-Lagrangian Solver in Elmer

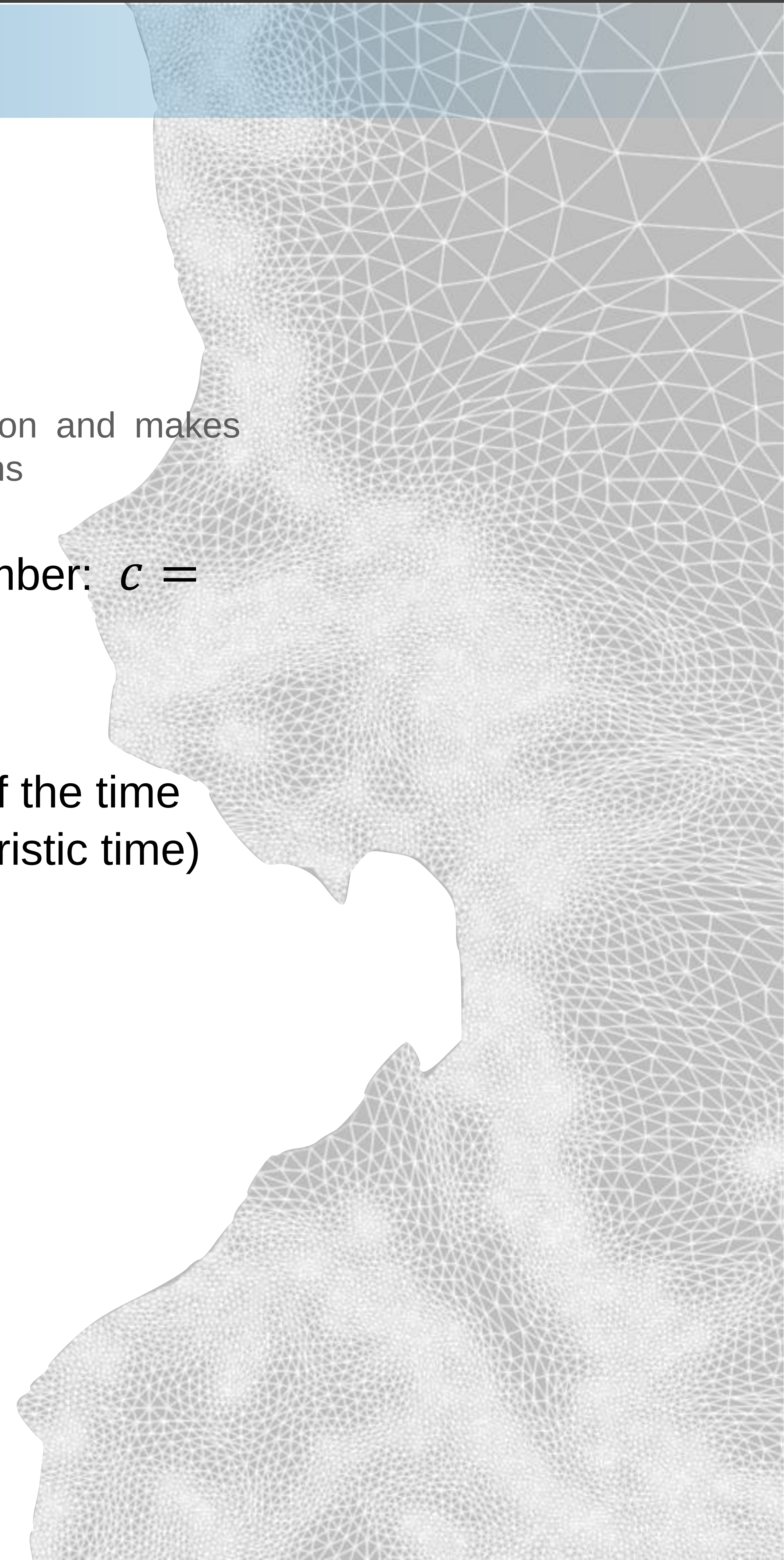
```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Tim

! Time in
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

```
! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
```

computes it for x,y,z direction and makes different time different directions

Timestep can be fixed to unisotropic current number: $c = \frac{u\Delta t}{\Delta x} \Rightarrow \Delta t = c\Delta x/u$

The internal timestep is defined as a portion c of the time needed to cruise over the element (i.e. characteristic time)



The Semi-Lagrangian Solver in Elmer

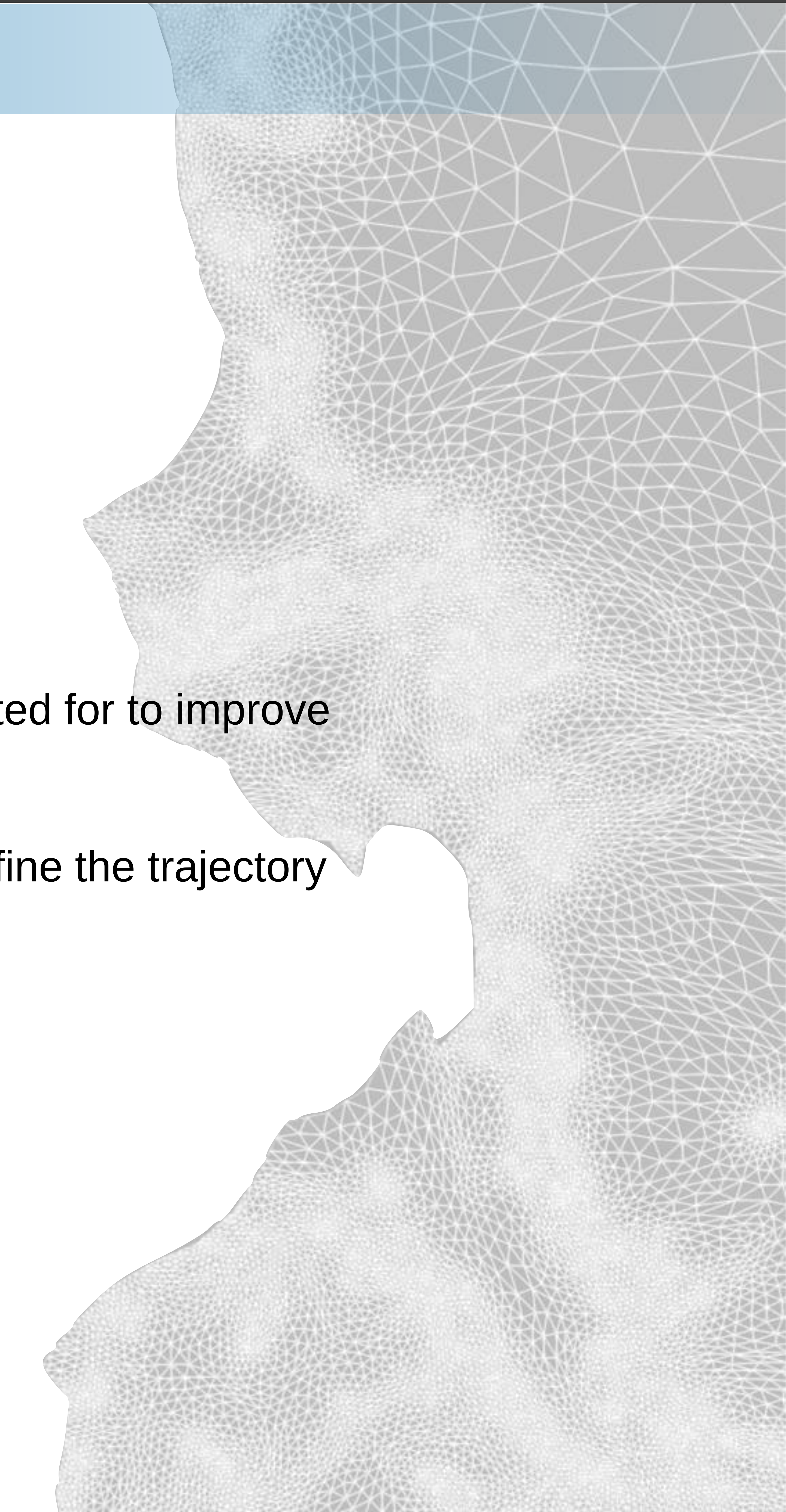
```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4
```

```
! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"
```

```
! Show some info in the end
  Particle Info = Logical True
  Particle Time = Logical True

! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"

 Particle Integral Dummy Argument = Logical True
End
```

The **gradient of the velocity** can be accounted for to improve the internal particle trajectory

The velocity variable (or flow solution) will define the trajectory of the particles



The Semi-Lagrangian Solver in Elmer

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```


```
! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```


```
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"
```

```
 Particle Integral Dummy Argument = Logical True
End
```

```
Body Force 1
  HDG Source = Real 0.0

  Particle Time Integral Source = Variable "dummy"
    Real lua "Damage(tx[0])"
```

The **source** term that allows the particle value to evolve over time or path.

The source can be computed in a USF and depends on the *"Particle Time Integral"* itself, defined as an internal *"dummy"* variable

```
! Path integral over time
IF( TimeInteg ) THEN
  IF( UseDummy ) THEN
    DummyVals(1) = TimeIntegVar % Values(No)
    Source = ListGetElementReal( TimeSource_h, Basis, Element, Found, DummyVals = DummyVals )
  ELSE
    Source = ListGetElementReal( TimeSource_h, Basis, Element, Found )
  END IF
  IF( Found ) THEN
    IF ( UseGradSource ) THEN
      GradSource = ListGetElementRealGrad( TimeSource_h,dBasisdx,Element)
      Source = Source + 0.5*SUM( GradSource(1:dim) * (PrevCoord(1:dim) - Coord(1:dim)) )
    END IF

    IF( TimeDepFields ) THEN
      IF( UseDummy ) THEN
        PrevSource = ListGetElementReal( TimeSource_h, Basis, Element, Found,tstep=-1, DummyVals = DummyVals  )
      ELSE
        PrevSource = ListGetElementReal( TimeSource_h, Basis, Element, Found,tstep=-1 )
      END IF
      IF ( UseGradSource ) THEN
        GradSource = ListGetElementRealGrad( TimeSource_h,dBasisdx,Element,tstep=-1)
        PrevSource = PrevSource + 0.5*SUM( GradSource(1:dim) * (PrevCoord(1:dim) - Coord(1:dim)) )
      END IF
      TimeIntegVar % Values(No) = TimeIntegVar % Values(No) + dtime * &
          ( (1-dtrat)*Source + dtrat * PrevSource )
    ELSE
      TimeIntegVar % Values(No) = TimeIntegVar % Values(No) + dtime * Source
    END IF
  END IF
END IF
```

The Semi-Lagrangian Solver in Elmer

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True

! Show some info in the end
```

```
! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```

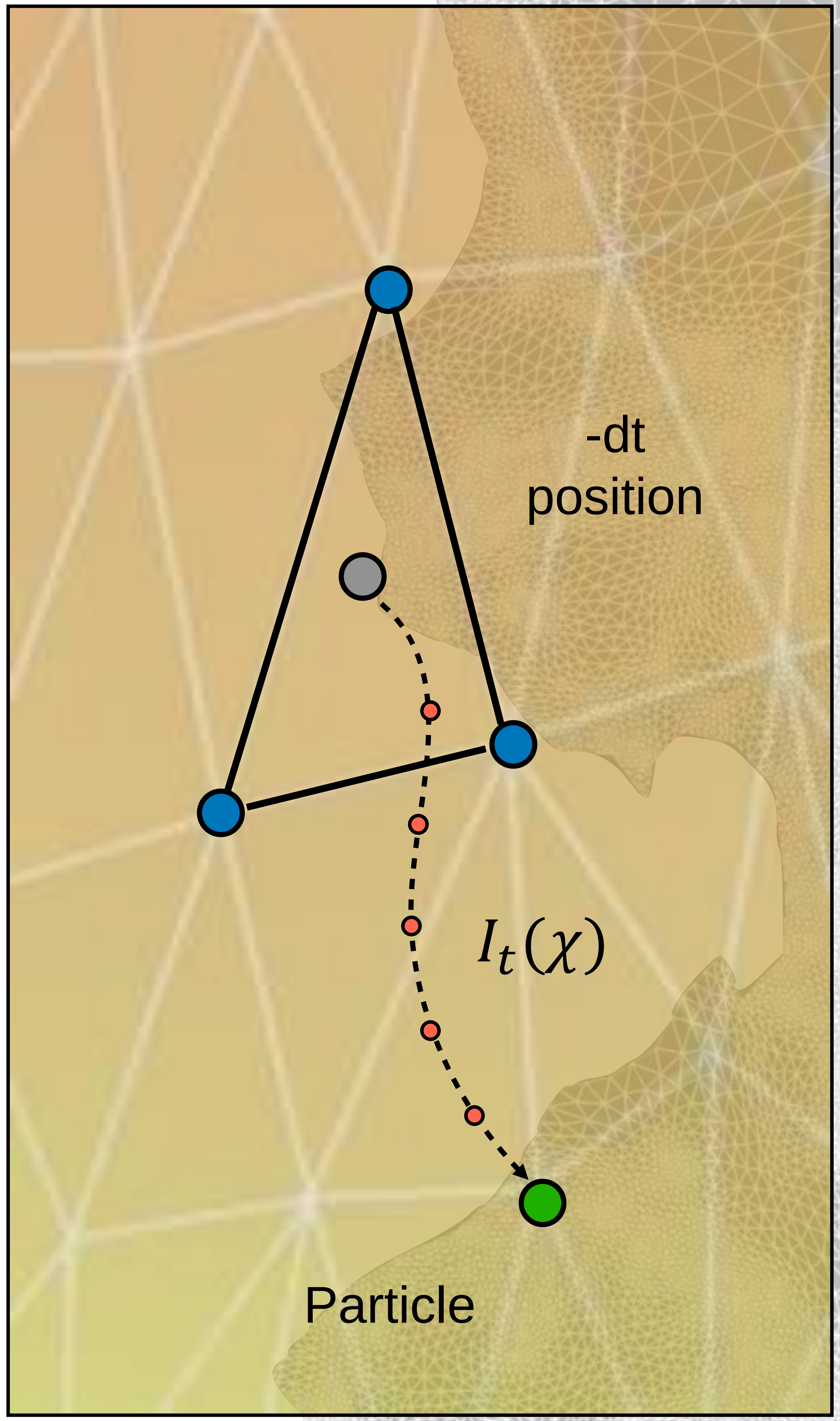
```
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"

  Result Variable 1 = String "Hadv"
  Result Variable 4 = String "Damage"
```

```
  Particle Integral Dummy Argument = Logical True
```

```
End
```

Particle time reverse is the keyword allowing the solver to go back in time and then reverse time to compute the source forward in time

The **dummy** value is the value of the Particle Time Integrator (evolving) over the integration



The Semi-Lagrangian Solver in Elmer

```
Solver 2
  Equation = ParticleAdvector
  Procedure = "ParticleAdvector" "ParticleAdvector"

!Relative Mesh Level = Integer -1

! Initialize particles at center of elements (as opposed to nodes)
  Advect Elemental = Logical False

  Reinitialize Particles = Logical True
  !Particle Accurate At Face = Logical False

! Timestepping strategy
  Simulation Timestep Sizes = Logical True

  Particle Dt Constant = Logical False
  Max Timestep Intervals = Integer 10 !Accuracy largely decrease for small values

! Time in average 4 steps in each element
  Timestep Unisotropic Courant Number = Real 0.25
  Max Timestep Size = Real 1.0e3

! Give up integration if particles are tool old
  Max Integration Time = Real 1.0e4

! Integration forward in time
  Runge Kutta = Logical False

  Velocity Gradient Correction = Logical True
  Velocity Variable Name = String "Velocity"

! Keywords for sourcing the particle
  Source Time Correction = Logical True
  Particle time reverse = Logical True
```

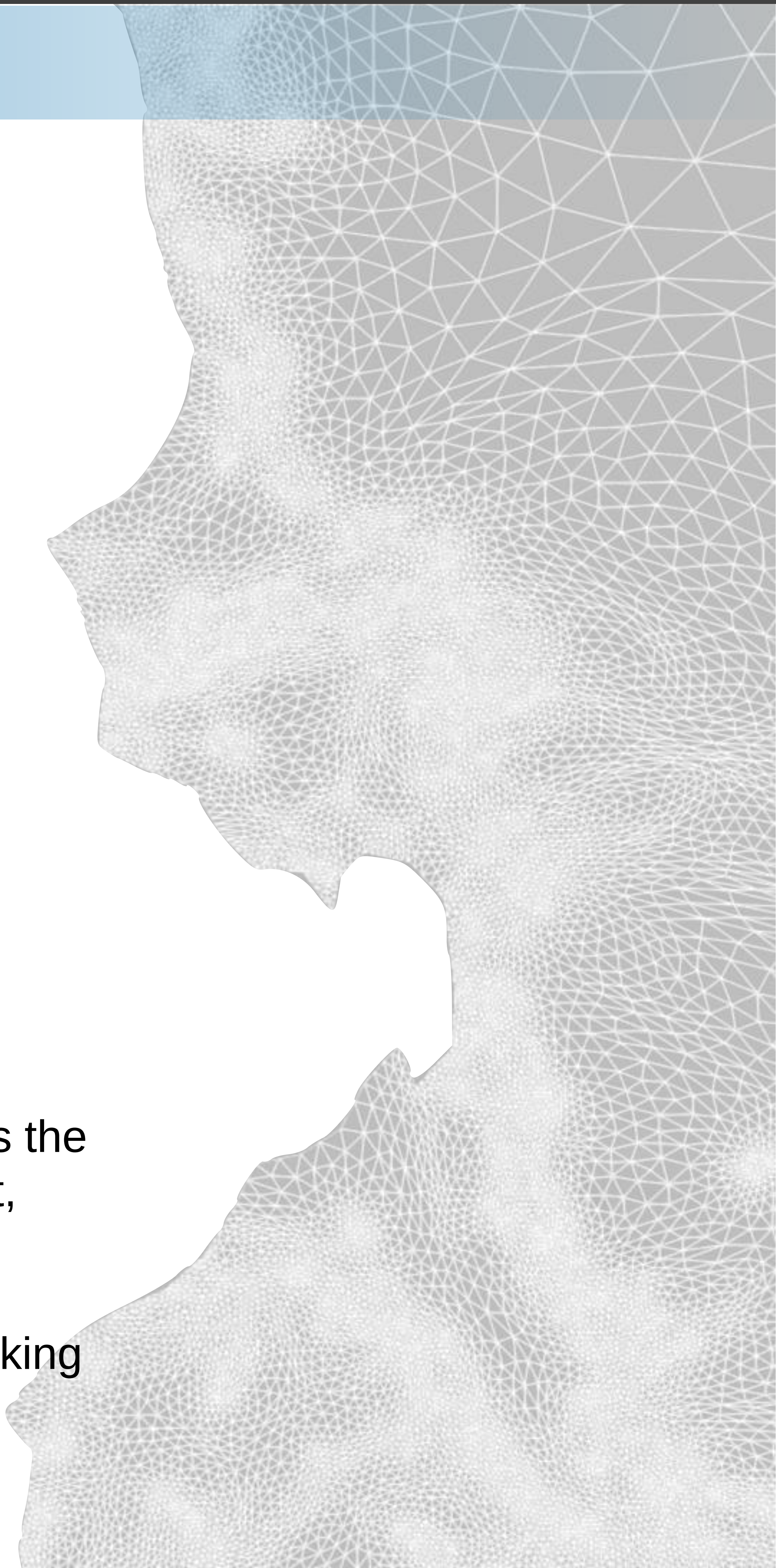
```
! The internal variables for this solver
  Variable 1 = String "Hpart"
  Variable 2 = String "Particle distance"
  Variable 3 = String "Particle time"
  Variable 4 = String "Particle time integral"

! Absolute displacement when going back-and-forth in time.
! For exact this should be zero (computation bugged in parallel)
  Variable 5 = String "Particle disp"

 ! Distance and velocity of the particle can be computed
  Variable 5 = String "Particle distance integral"
  Variable 6 = String "Particle velocity abs"
```

```
 Particle Integral Dummy Argument = Logical True
End
```

External variables can be defined.

Some **internal variables** can be called, such as the particle time/path integral, particle displacement, particle velocity, particle status,…

Some are useful to check that everything is working fine!



A 2D case and comparison with the Advection-Reaction Algorithm

Conditions

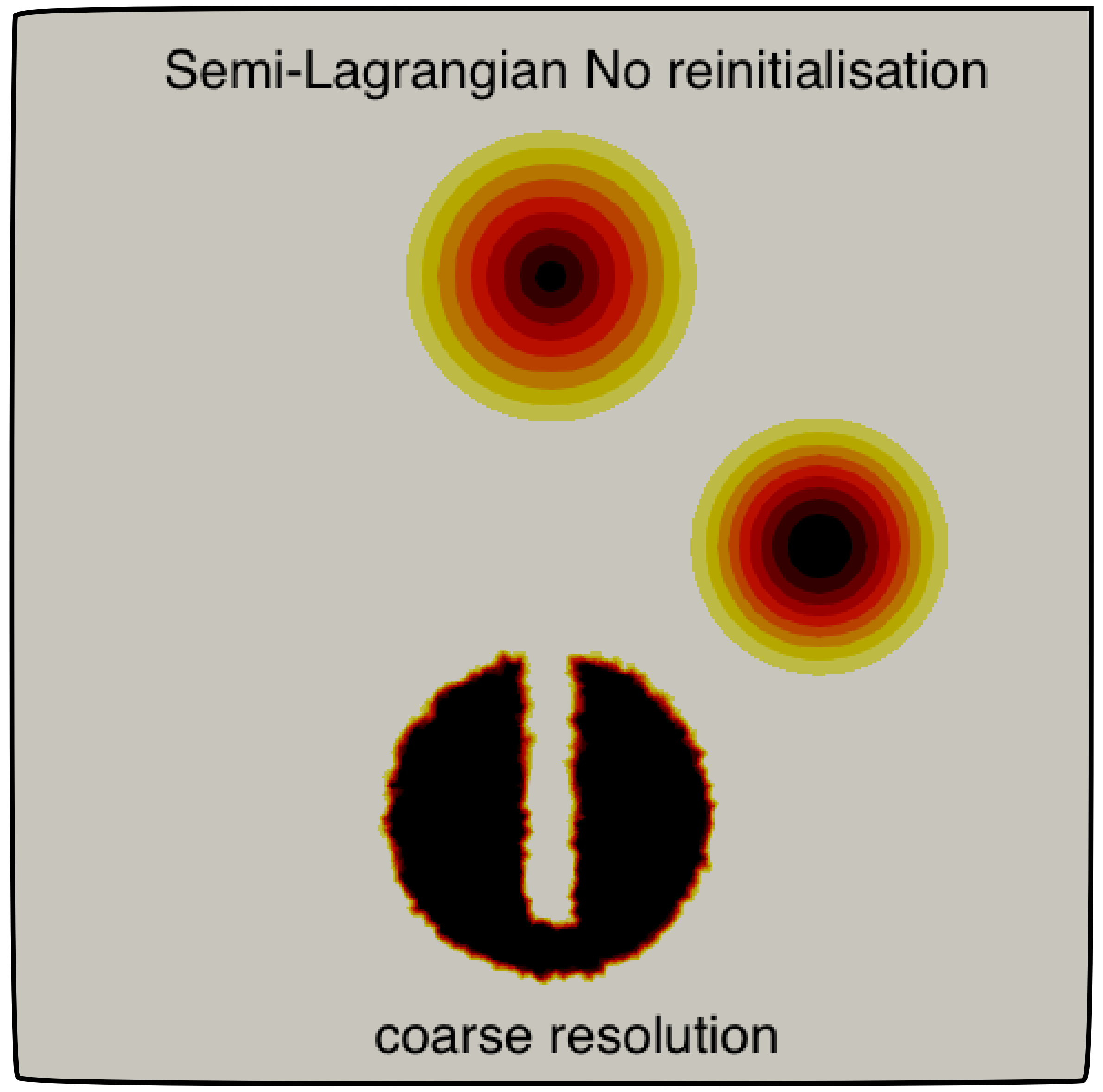
- Coarse: 100 x 100 celles (0.01 m x 0.01 m)
- Velocity = 0 - 0.7 m/a

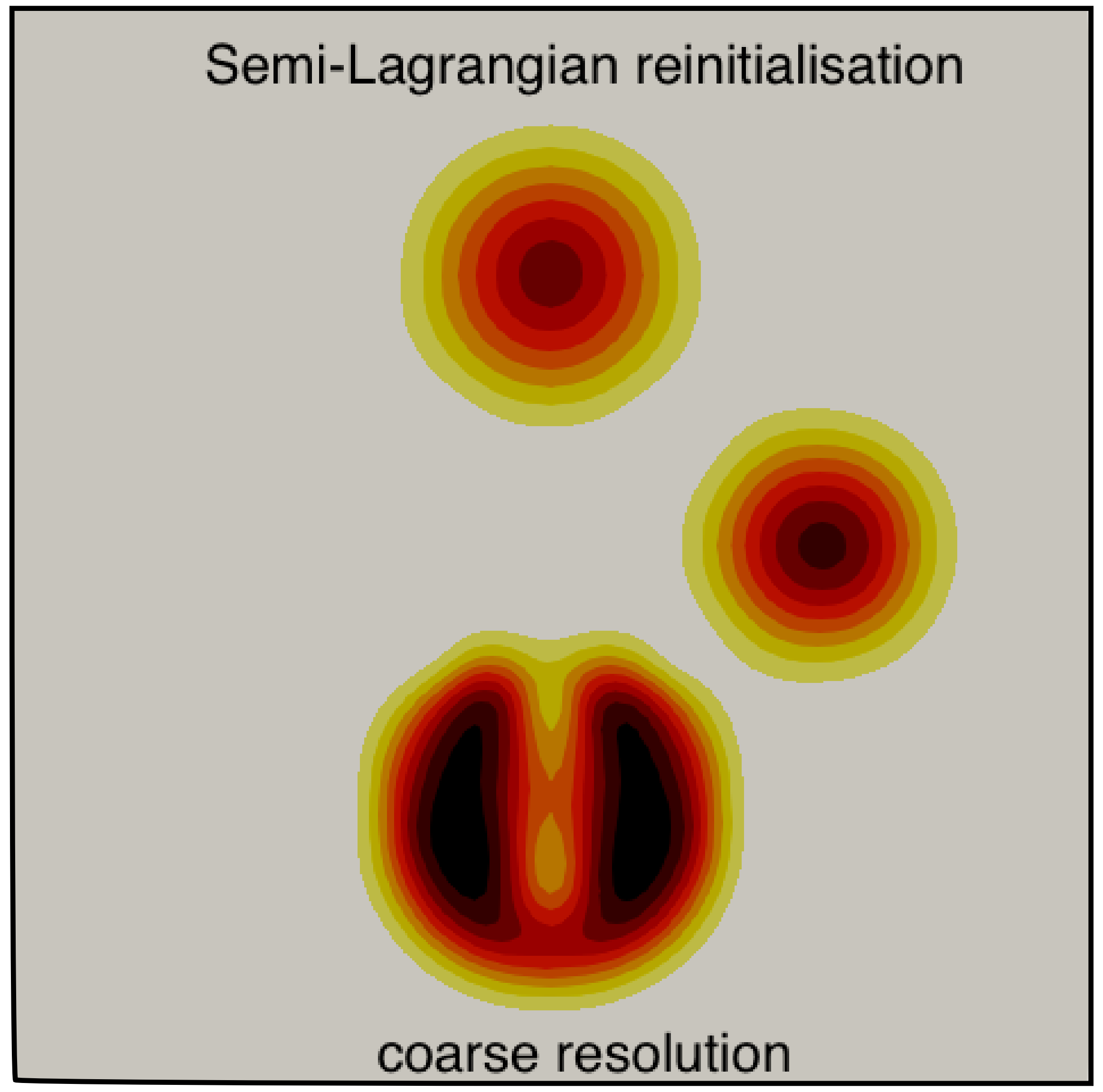
- $c = \frac{u\Delta t}{\Delta x} \Rightarrow c = \frac{[0-0.7]\times 2\pi/90}{10^{-2}} \sim [0-5]$

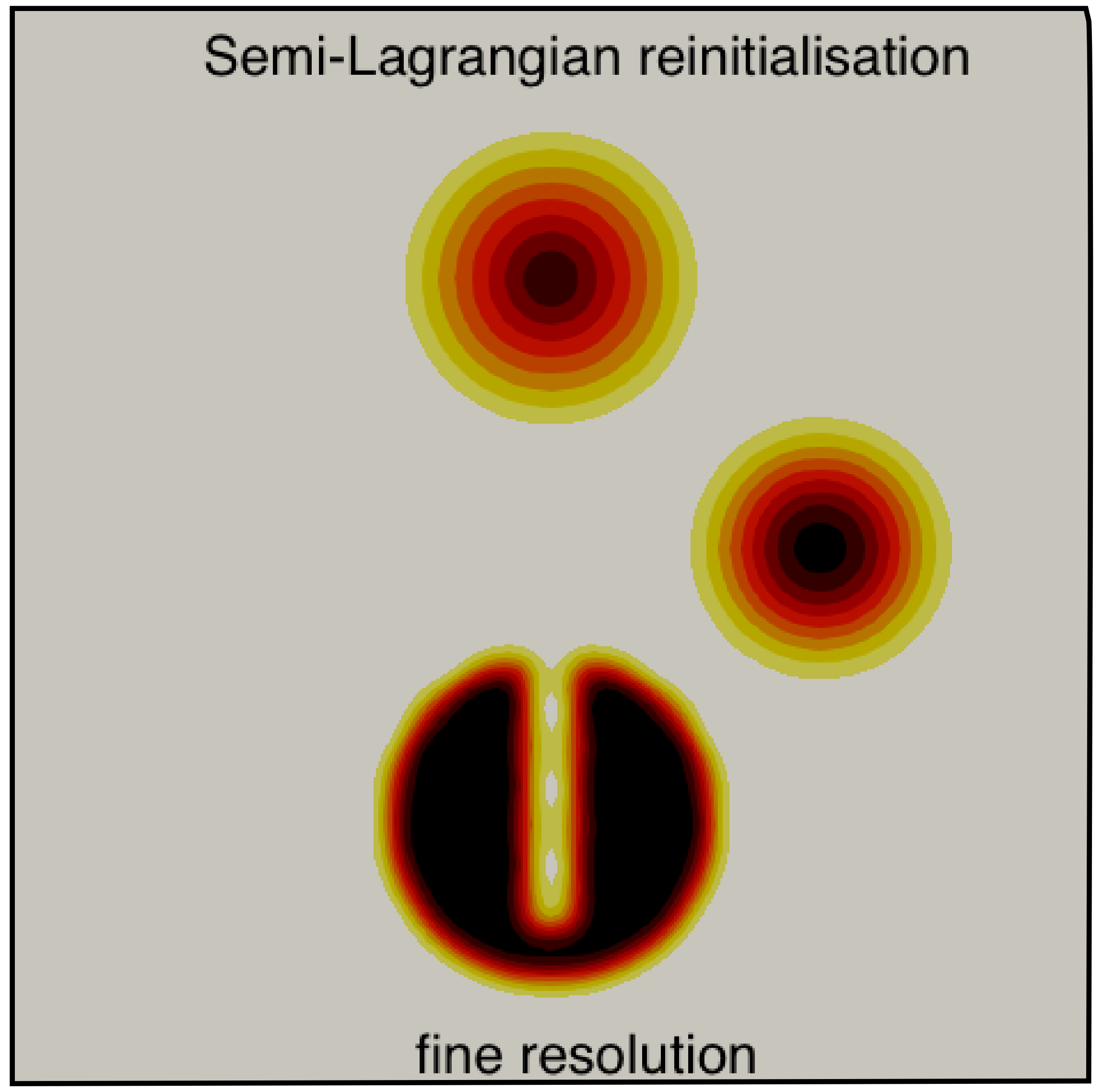
Conditions

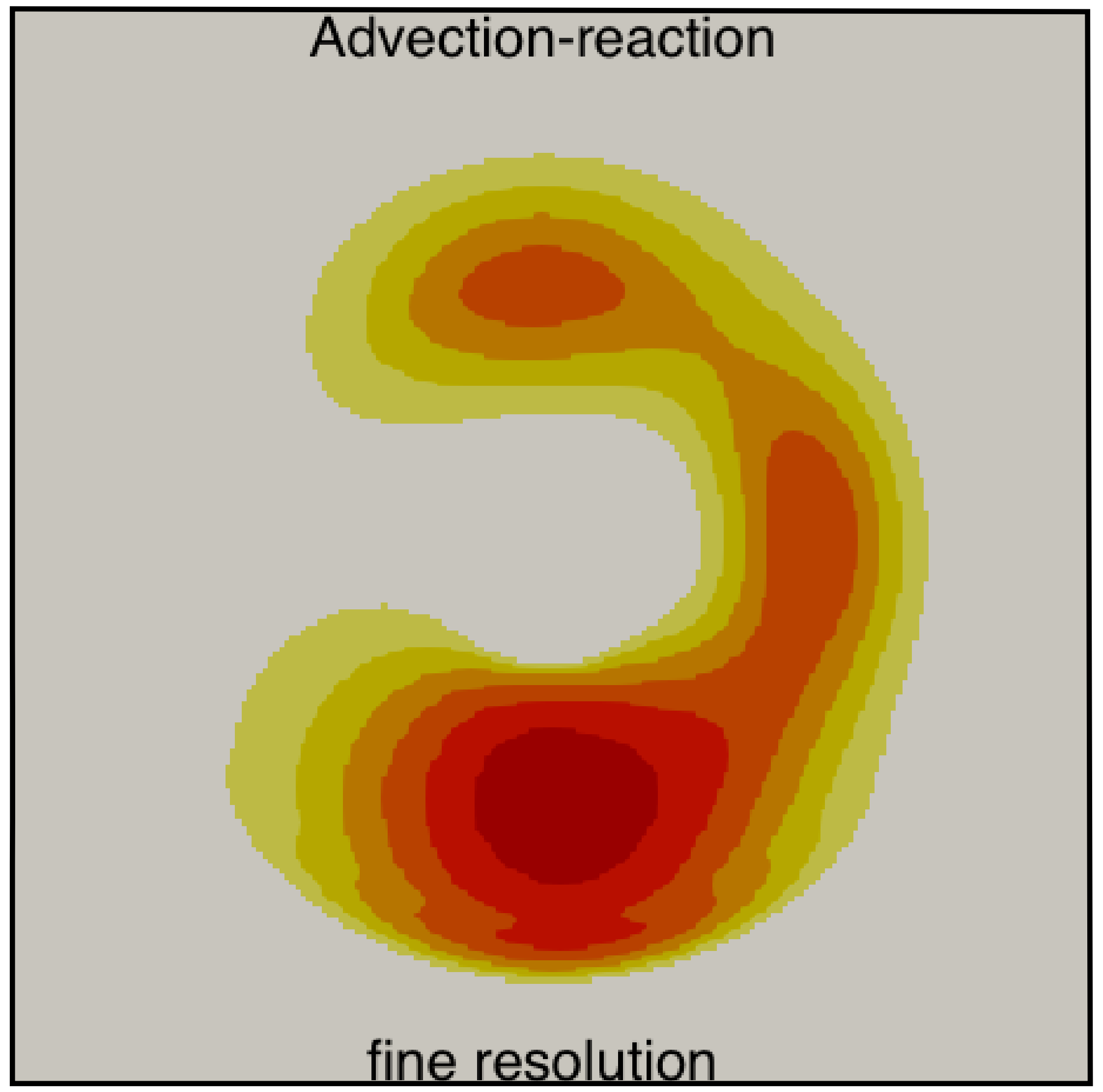
- Coarse: 200 x 200 celles (0.005 m x 0.005 m)
- Velocity = 0 - 0.7 m/a

- $c = \frac{u\Delta t}{\Delta x} \Rightarrow c \sim [0-10]$





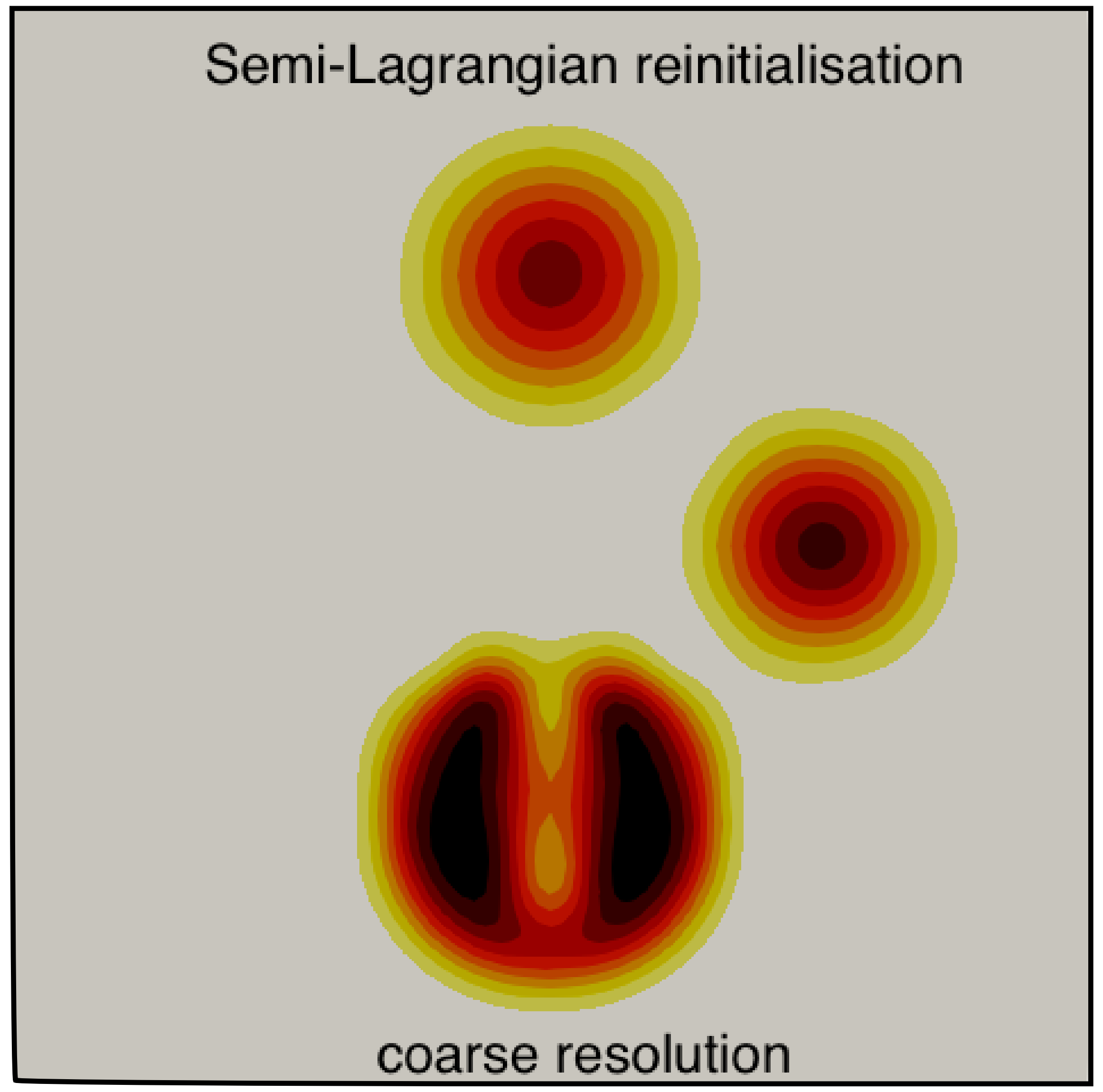




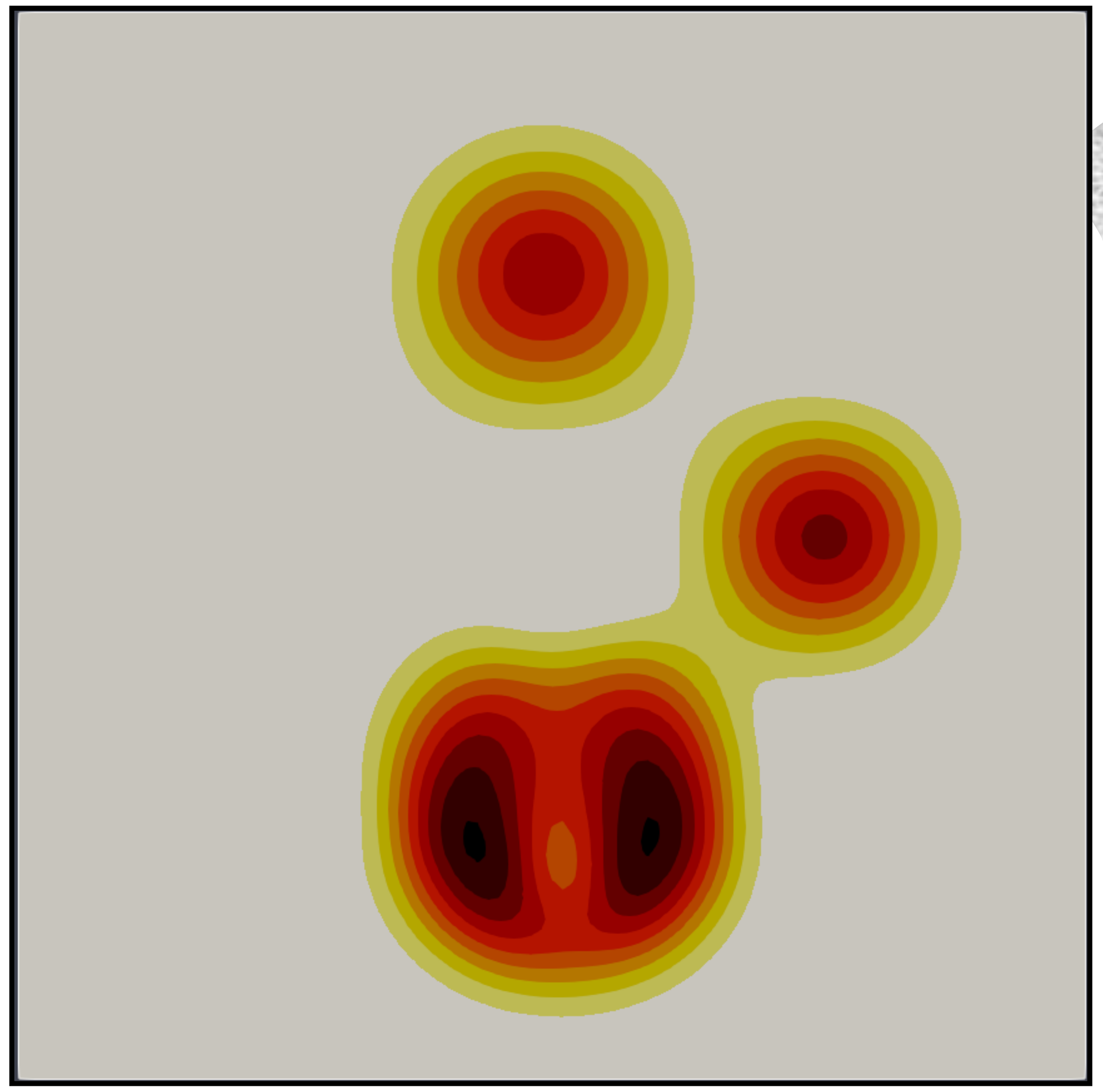
After a 180 degree rotation, 90 time steps for a full revolution

A 2D case and comparison with the Advection-Reaction Algorithm

180 degree rotation



90 time steps for a full revolution



180 time steps for a full revolution

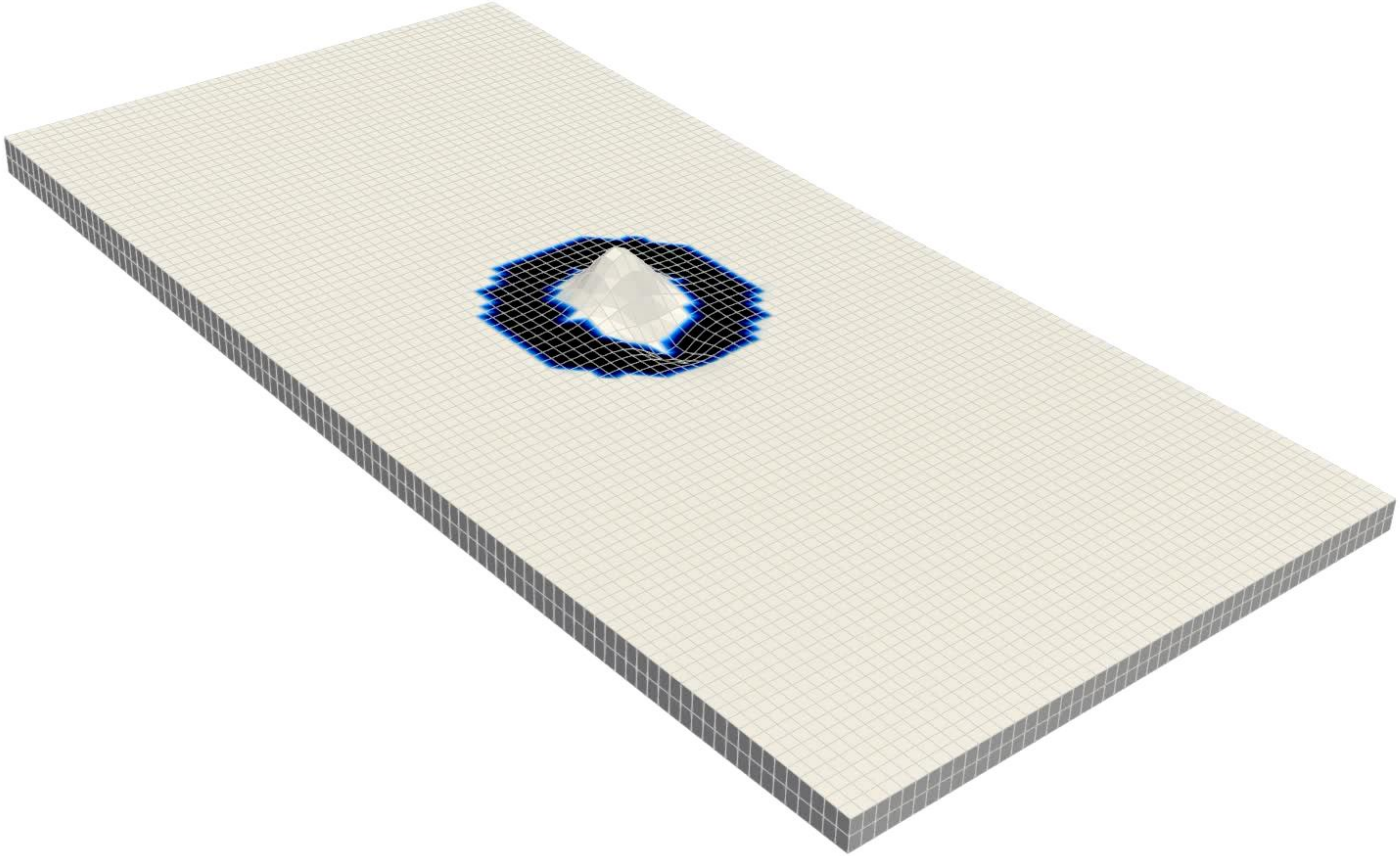
<u>More reinitialisation > More reinterpolation > Less accuracy</u>

A 3D application

Conditions

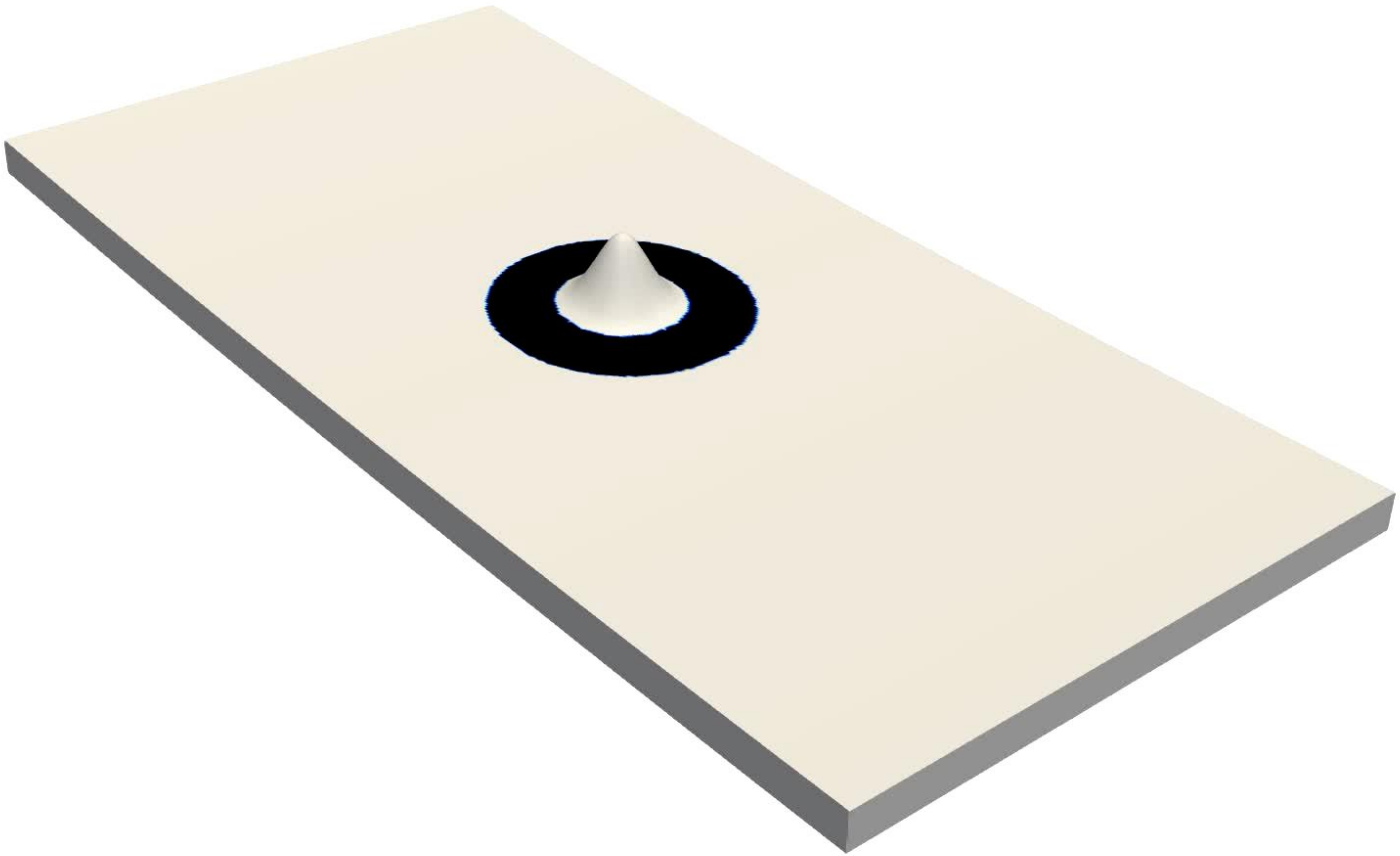
- Velocity field >> Stokes
  - ~ 1000 m/a
- 1x1 km 2D elements
- 4 partitions
- Vertical extrusion
- Mountain in the middle
  - almost steady state…

Advection of a donut tracer with reinitialization



A 3D application

Conditions
- Velocity field >> Stokes
  - ~ 1000 m/a
- 1x1 km 2D elements
- 4 partitions
- Vertical extrusion
- Mountain in the middle
  - almost steady state…



Advection of a donut tracer with reinitialization

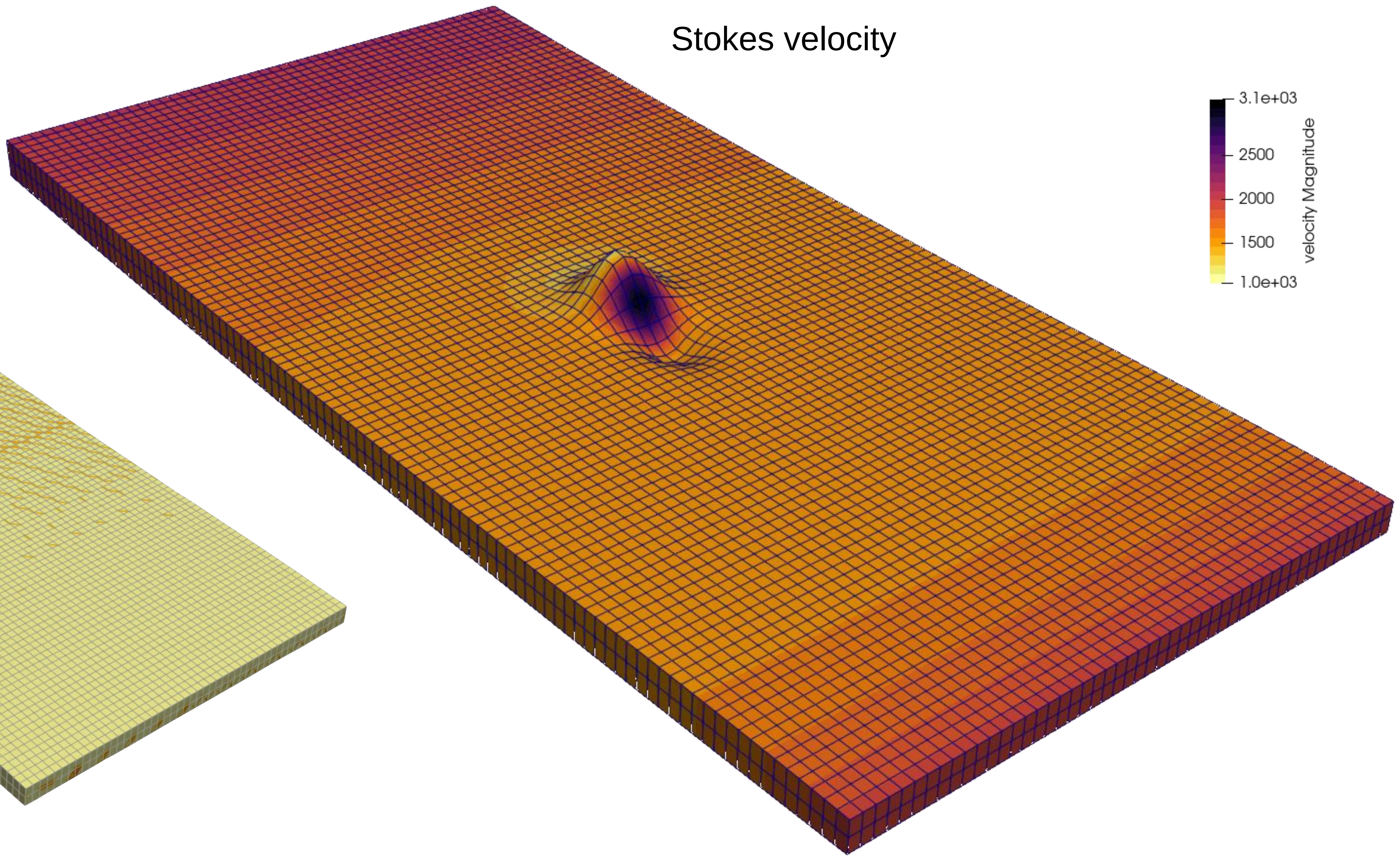
Increase resolution to 200 x 200 m
100 partitions

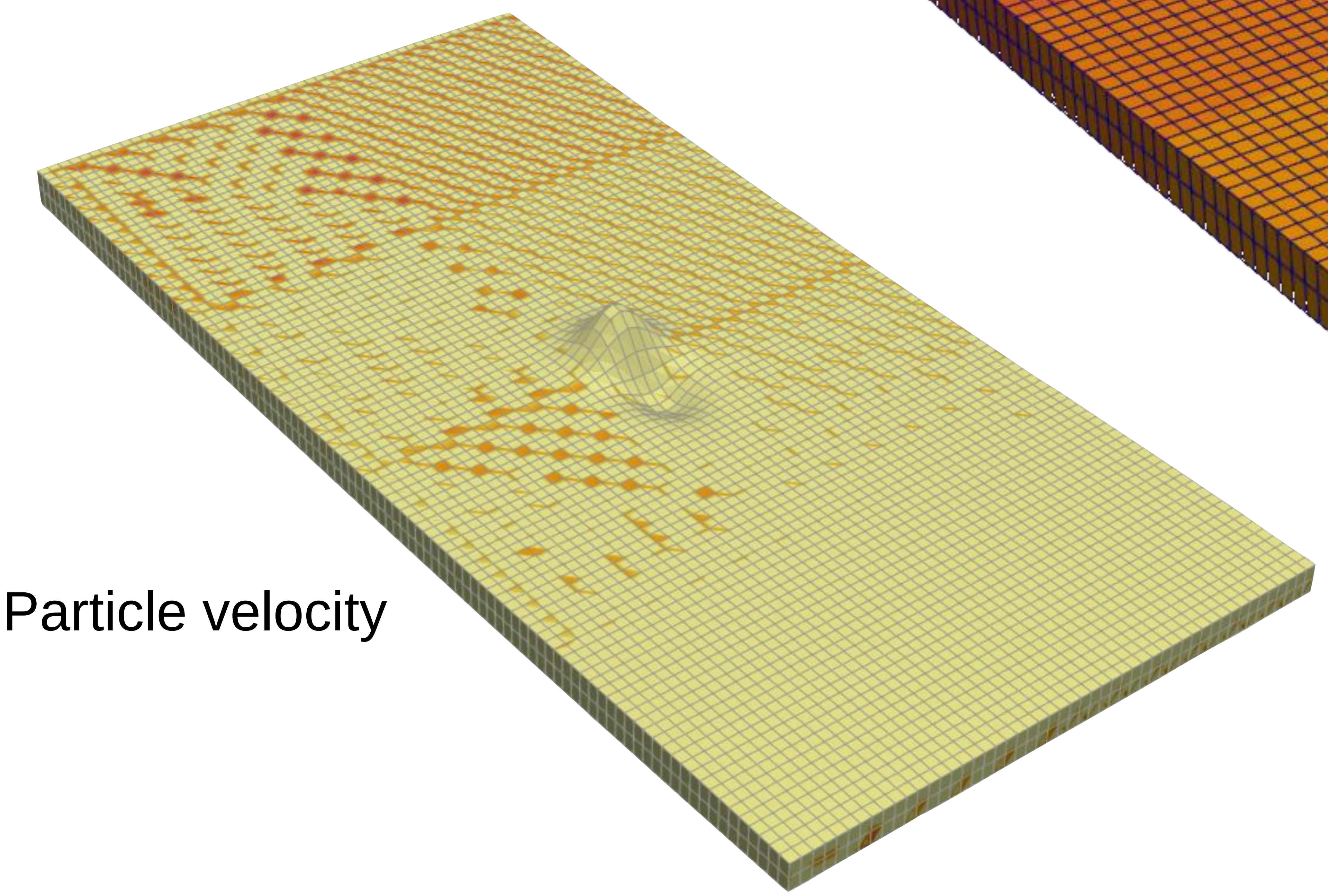
A 3D application

Conditions

- Velocity field >> Stokes
  - ~ 1000 m/a
- Squared 2D elements
- Vertical extrusion
- Mountain in the middle
  - almost steady state…

Stokes velocity





Particle velocity

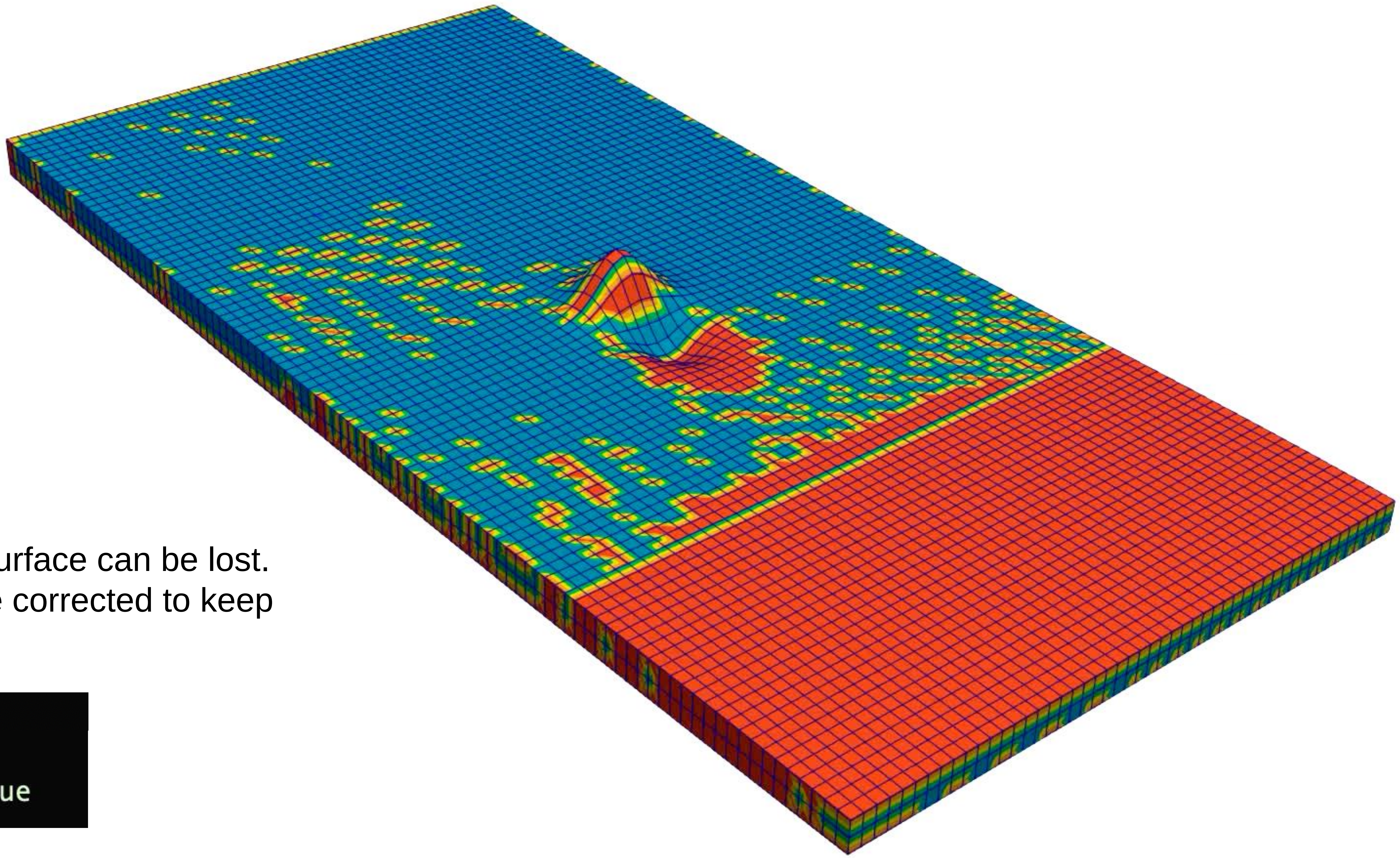
A 3D application

Conditions

- Velocity field >> Stokes
  - ~ 1000 m/a
- 1x1 km 2D elements
- 4 partitions
- Vertical extrusion
- Mountain in the middle
  - almost steady state…

Particle status

- PARITCLE_HIT
- WALLBOUNDARY



In 3D, velocity pointing outwards the surface can be lost. To avoid this, particle velocities can be corrected to keep them in the domain

```
Boundary Condition 5
  Particle Wall = Logical True
  Particle Tangent = Logical True
```

Application to damage

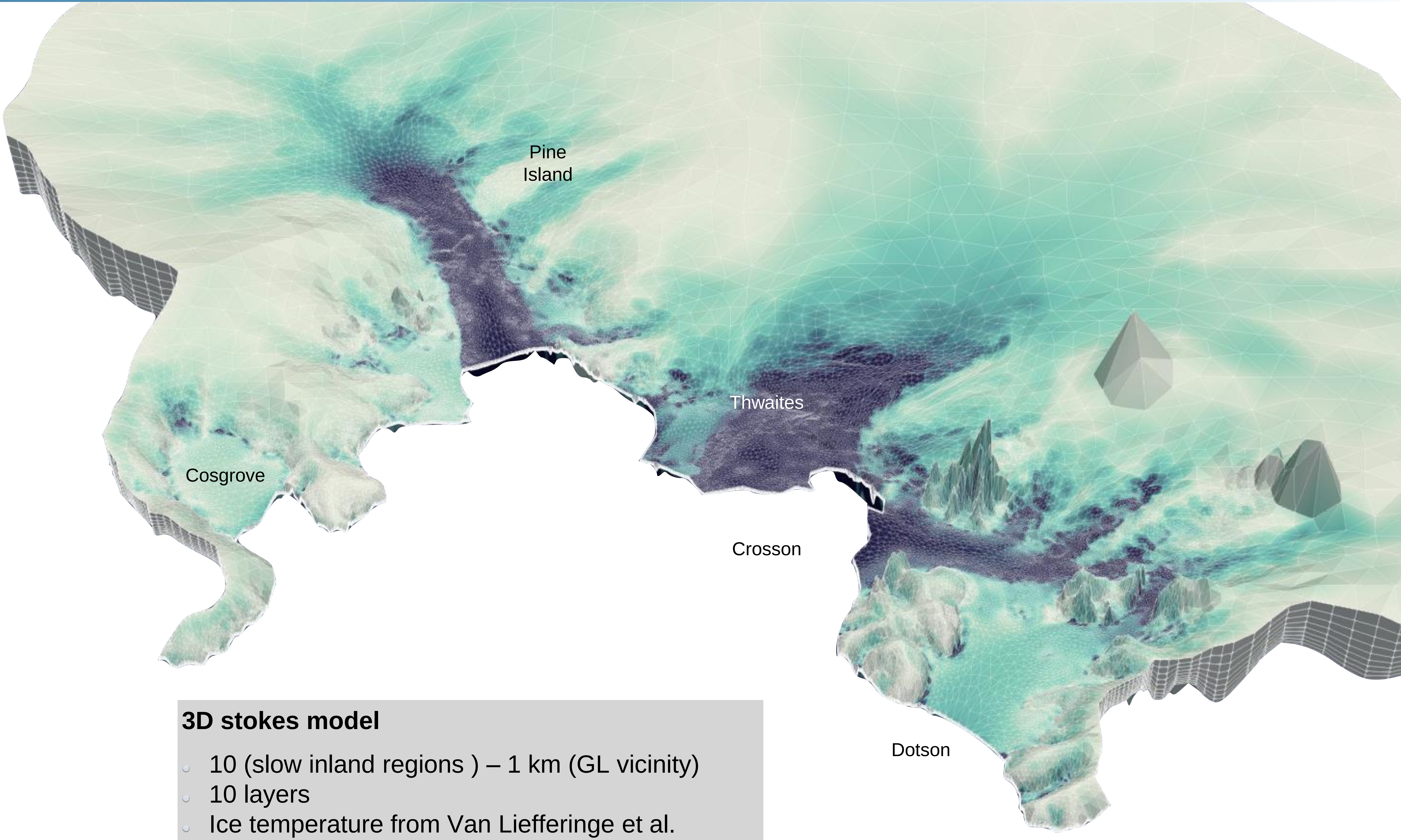
Reconstruct the past ice sheet

i.e. build a 1996 ice sheet state
(Original idea from de Rydt et al., 2020)

- Bedmachine correction with observed dhdt (IceSat1-2 gives us the trend over 2003-2019)
- Surface velocity observations are very good for the Amundsen sea back in 1996
- We can compare the reconstructed grounding line to observations.

Simulate the ice sheet with damage

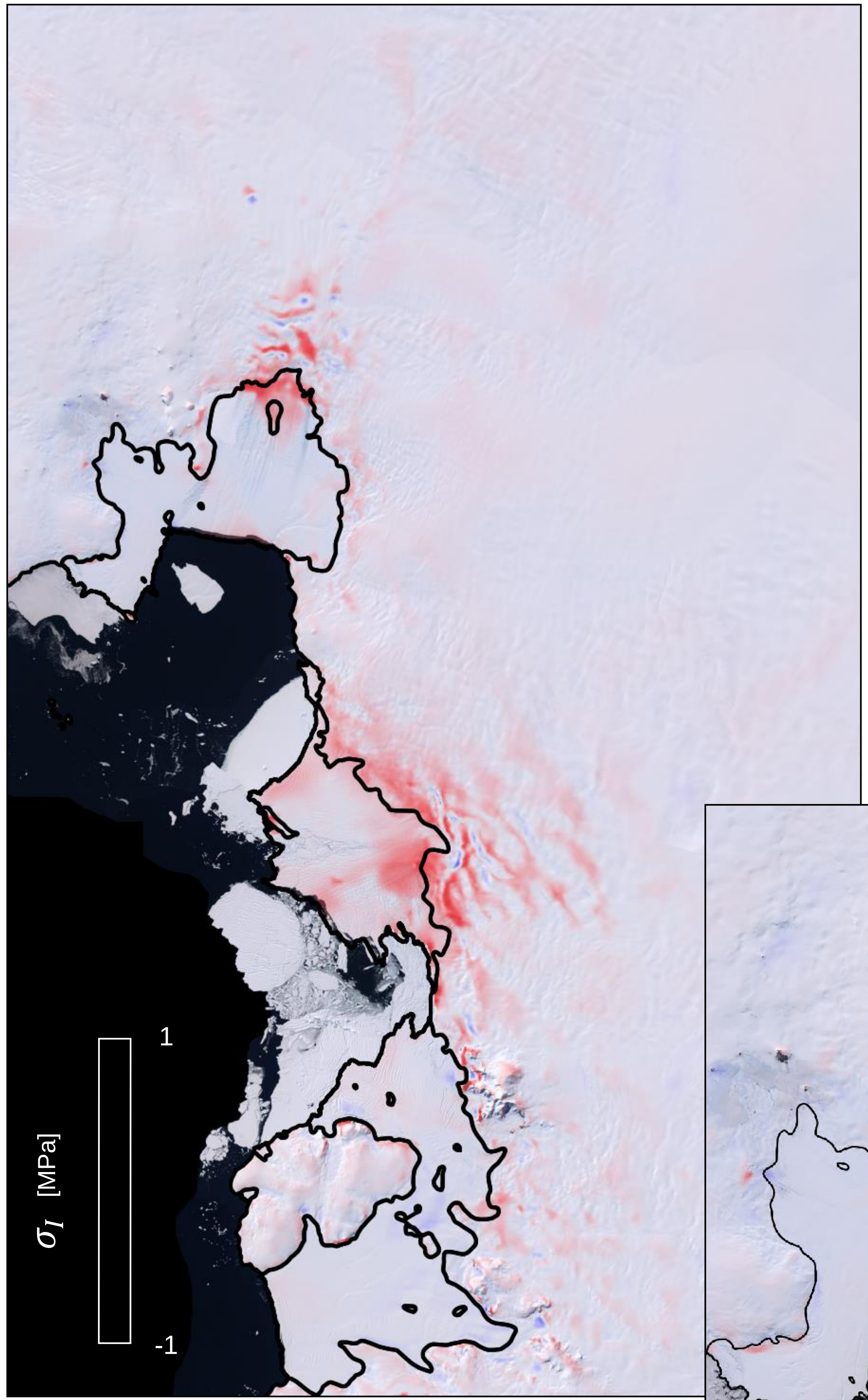
- Trend of dhdt, surface velocity and grounding line migration
- Comparison with :
  - damage observation
  - Inverted damage (data assimilation)

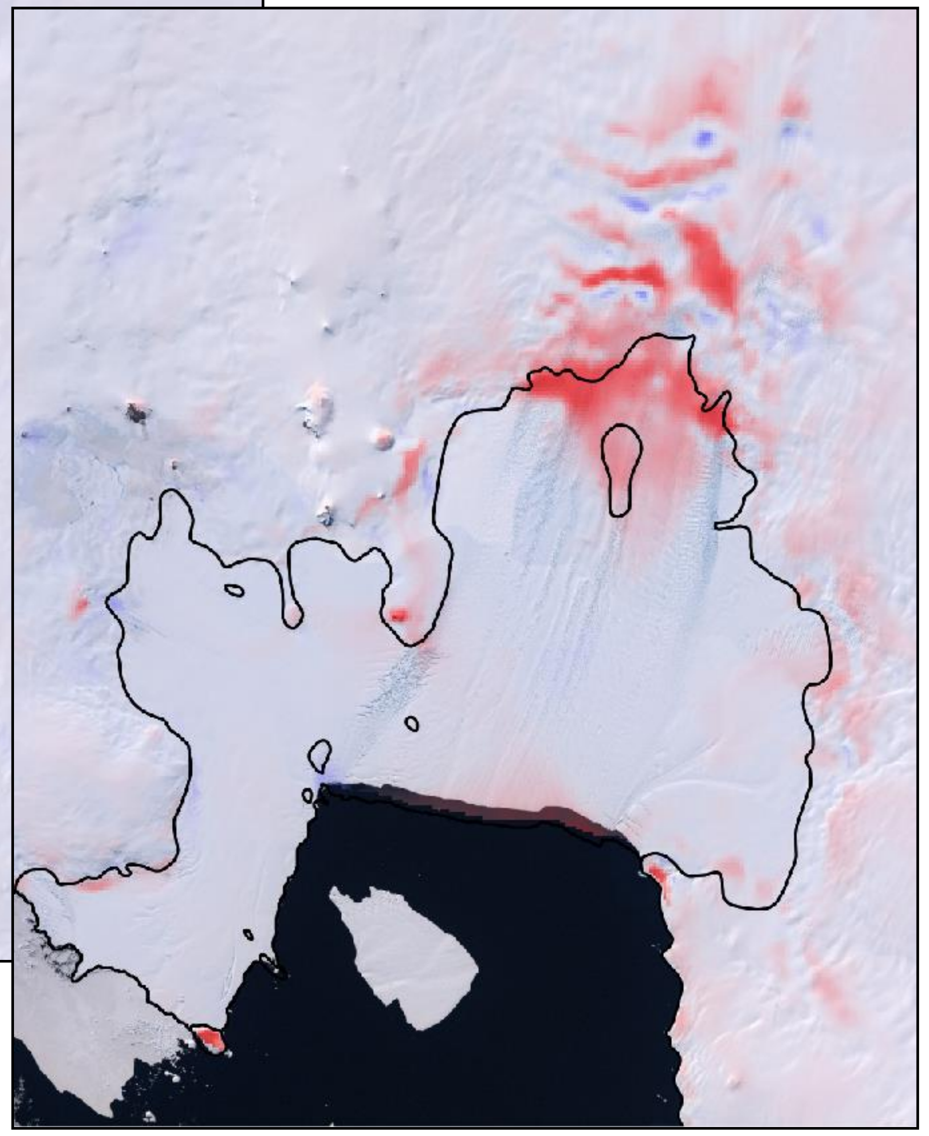


3D stokes model

- 10 (slow inland regions) – 1 km (GL vicinity)
- 10 layers
- Ice temperature from Van Liefferinge et al. (2013), updated in 2019
- Surface mass balance from MAR
- Ocean melt parametrized to have a steady and undamaged ice sheet in 1996

Application to damage



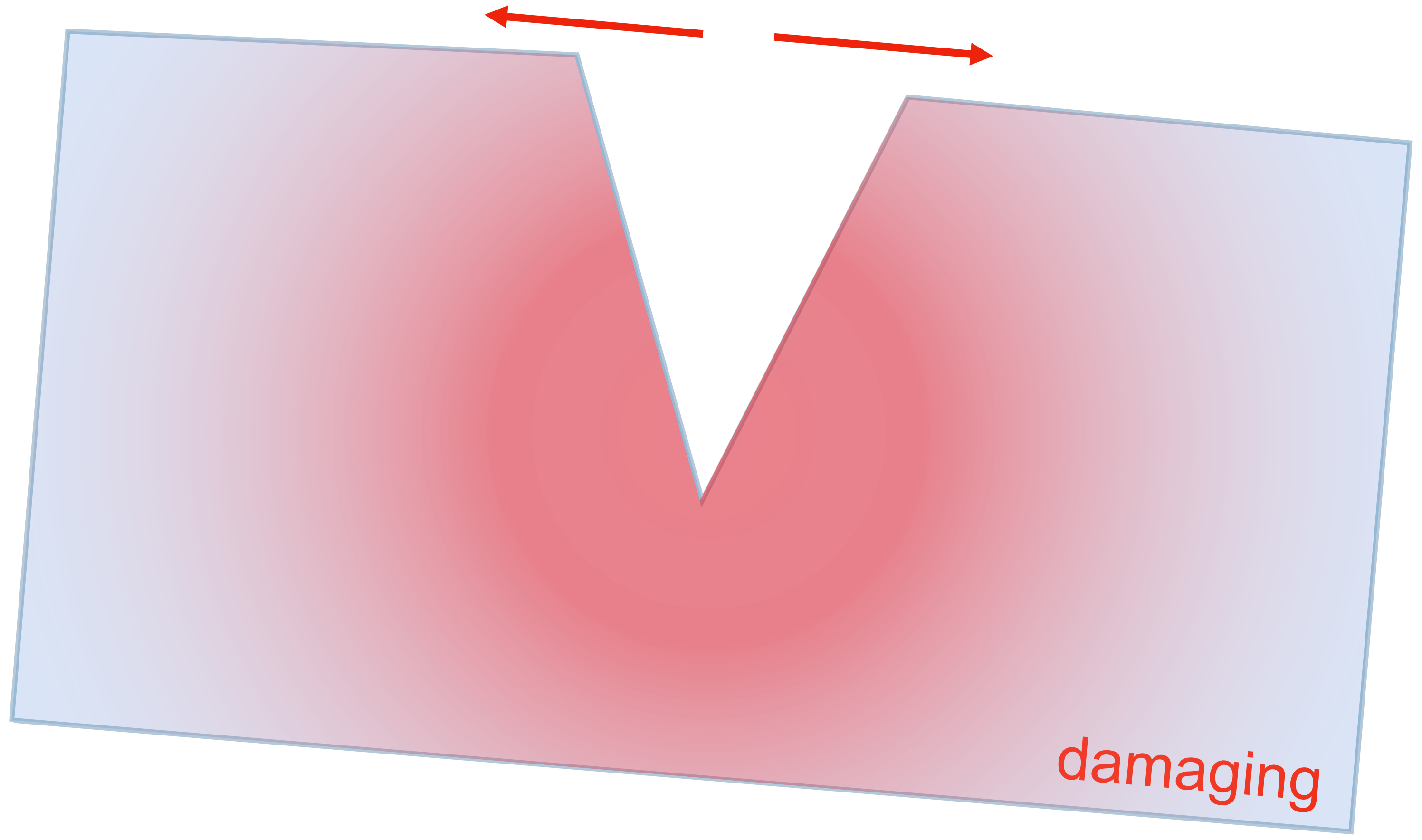


Principal Stress I

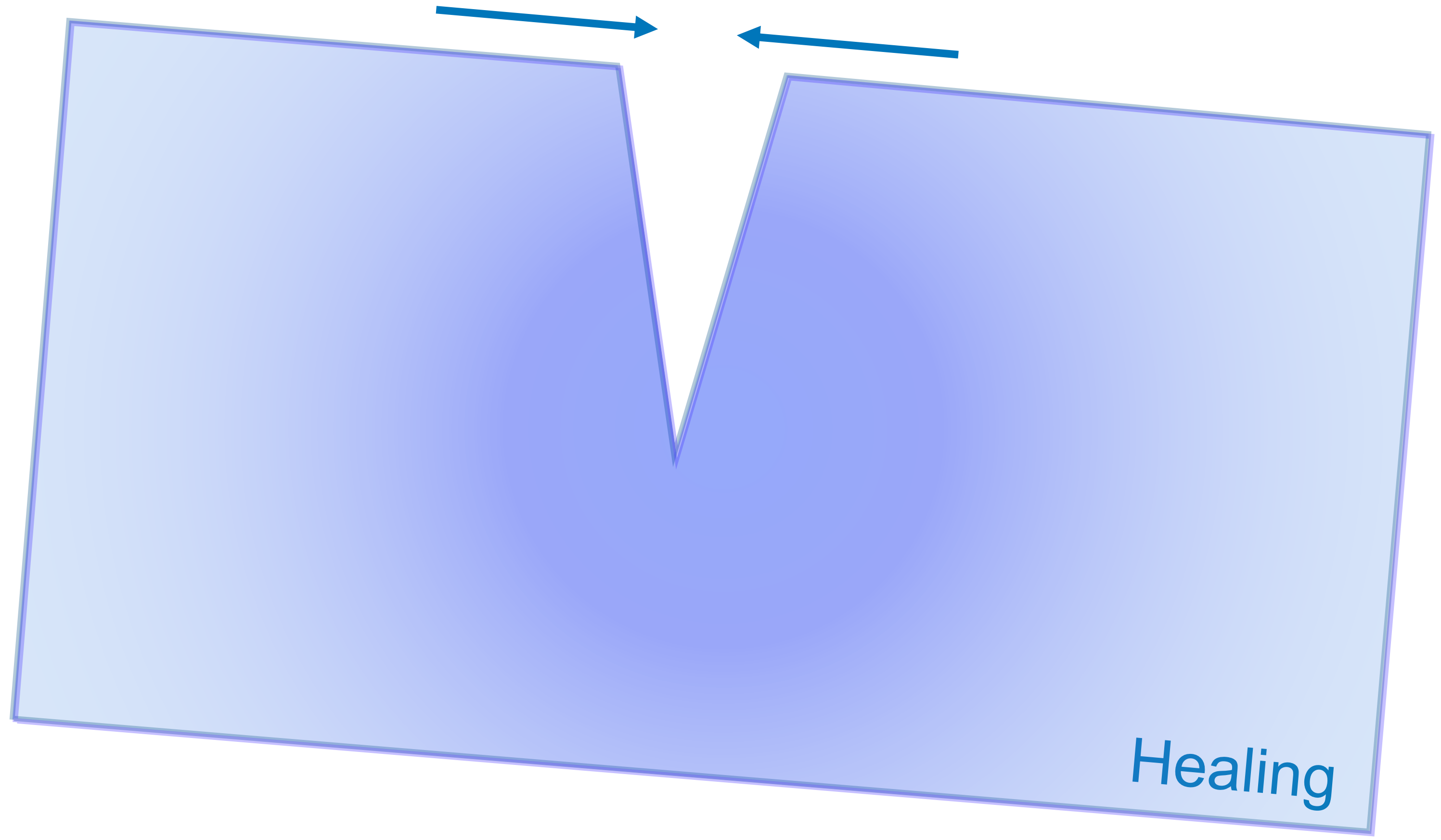
Damage (crevasses, rifts,…) creates where the extensional stress is large enough:

$$\chi(\sigma_I, \sigma_{th}, D) = \frac{\sigma_I}{1-D} - \sigma_{th} + p_w$$

with $D \in [0,1[$ and σ_{th} a minimal threshold.



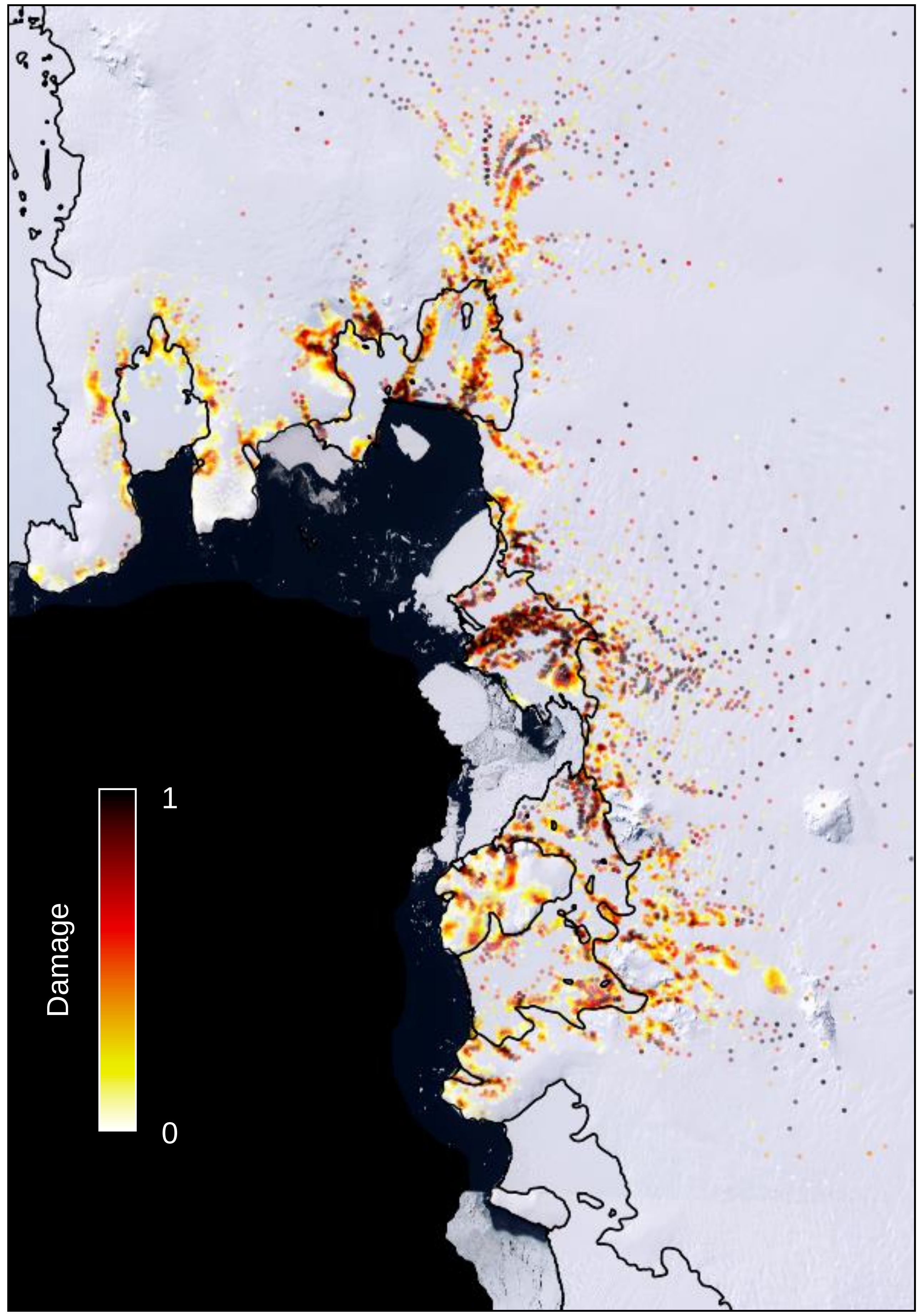
$\chi > 0$



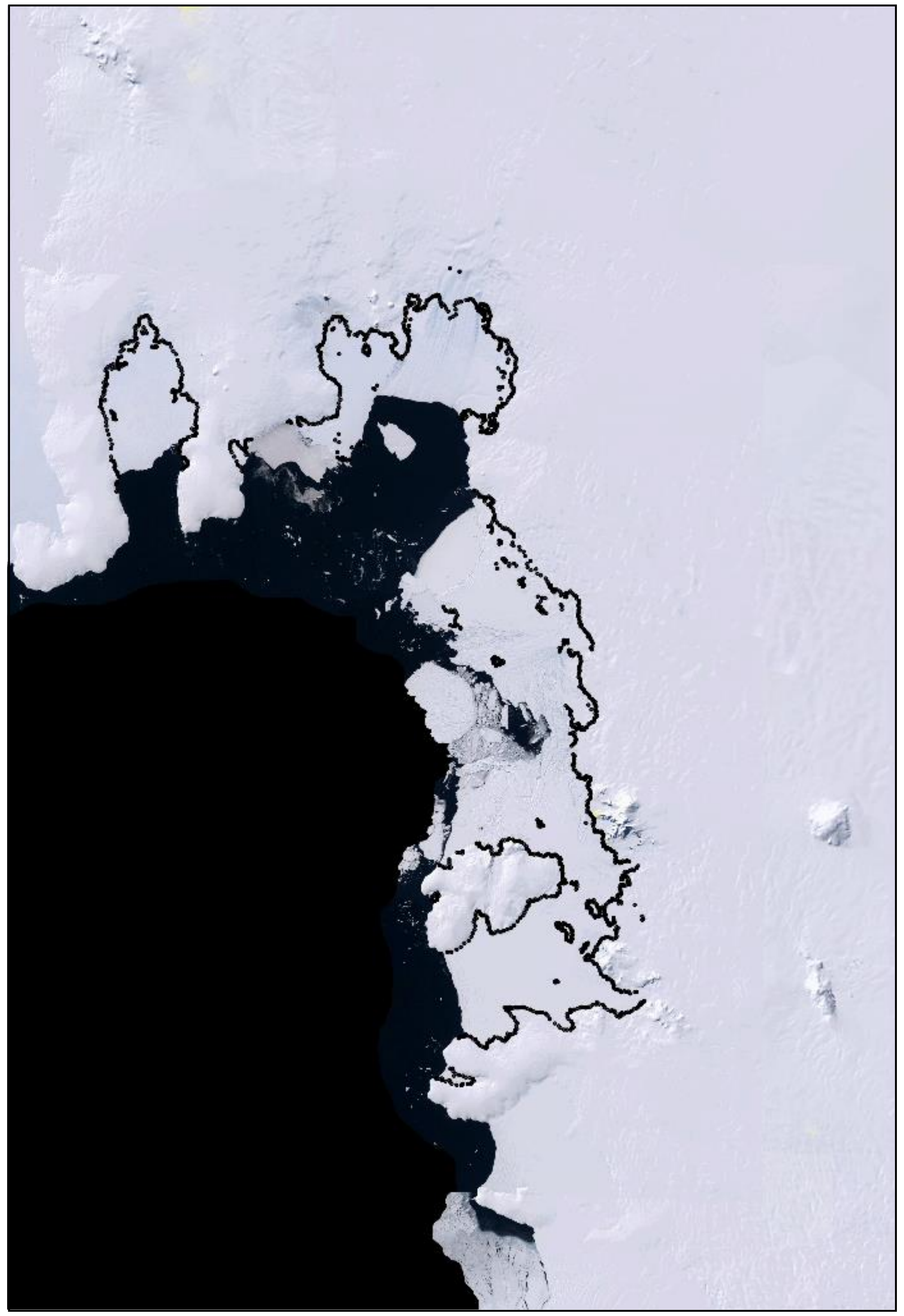
$\chi < 0$

(e.g. Krug et al., 2014, Sun et al. 2017)

Application to damage

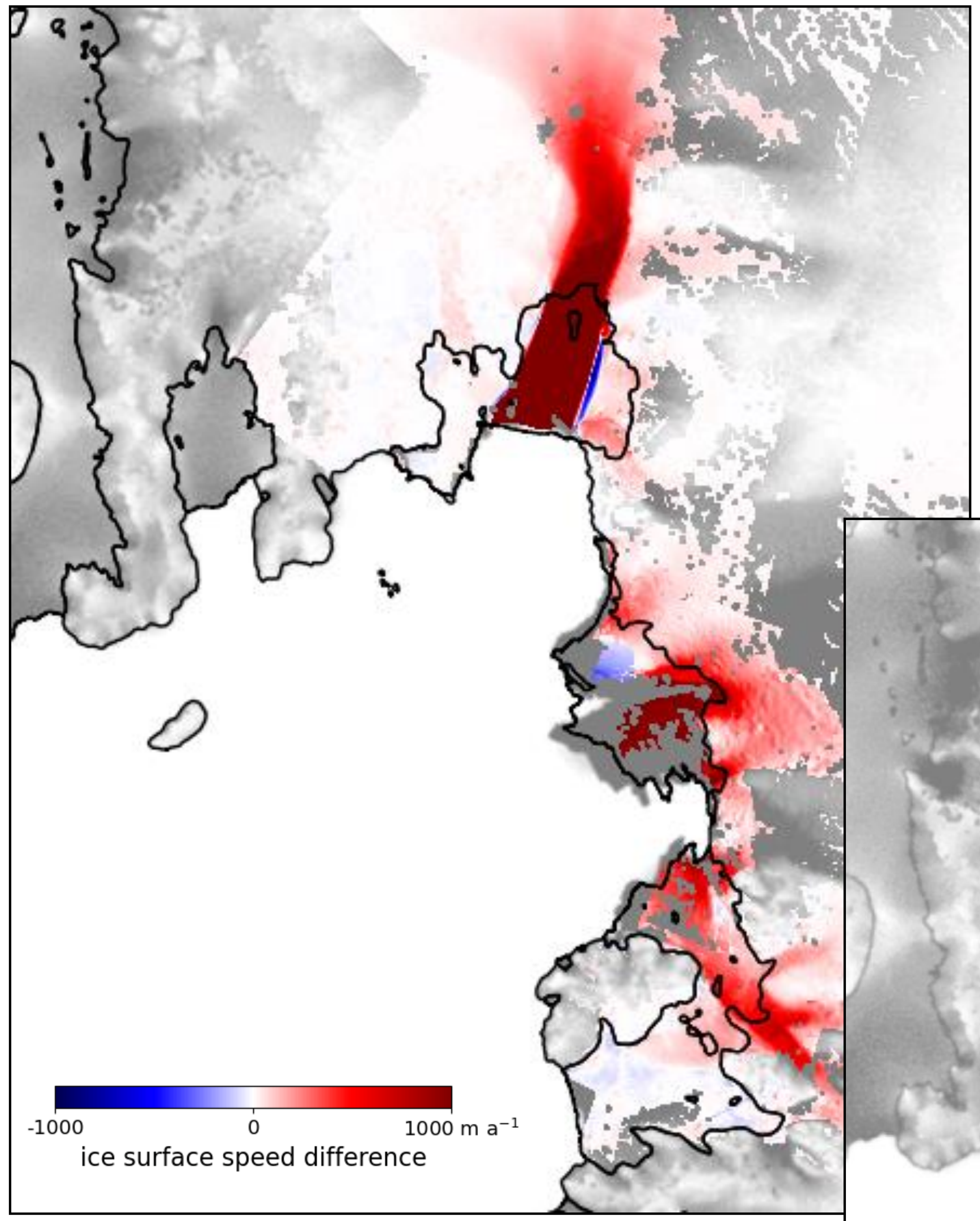


Damage reconstruction by data assimilation for the year 2018

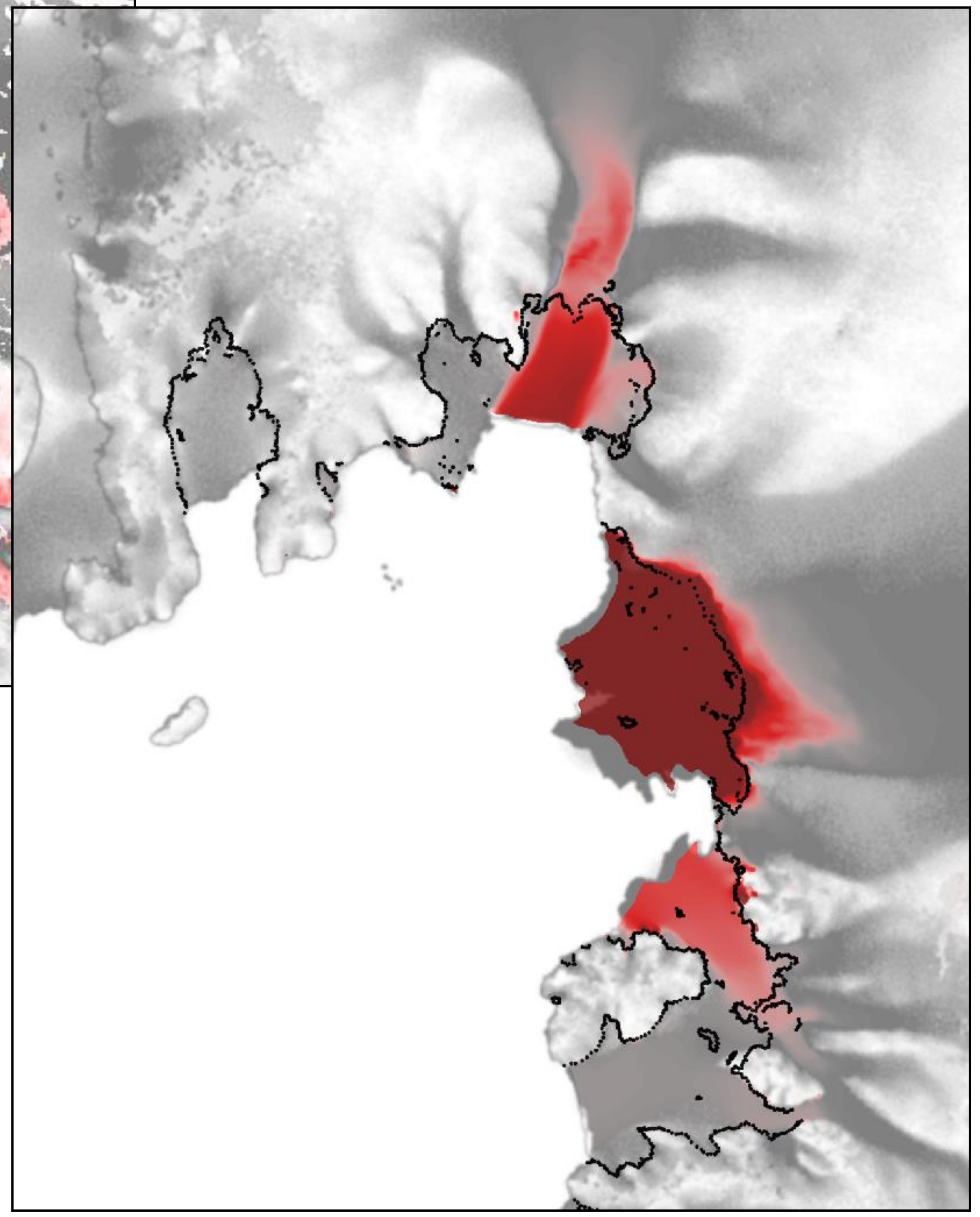


Damage evolution over 1996-2004 assuming no damage

(Set of parameters : $\sigma_{th} = 0.20, \;\; B = 0.5$ and no healing)



Observed speed change from 1996 to 2018



Model speed change from 1996 to 2004

Conclusions

- The solver works as expected in 2D serial simulations
- Some bugs remain to fix for 3D parallel simulations
- There has been a lot of fixes already (thanks to Peter) over the last months
  - check the commits in the dev branch if you already use the solver
  - more is coming …

- We will make clean test cases available to everyone (some are already present on my personal GitHub: https://github.com/cmosbeux/Damage)
- The "precision" of the semi-Lagrangian method increases with the timestep (opposite to classical advection method)… See how we can combine this with recent work on "increasing the larger possible timestep"