GeMA – one-way HM coupling in 2D involving upscaling process and use of external simulator

Version 1.0

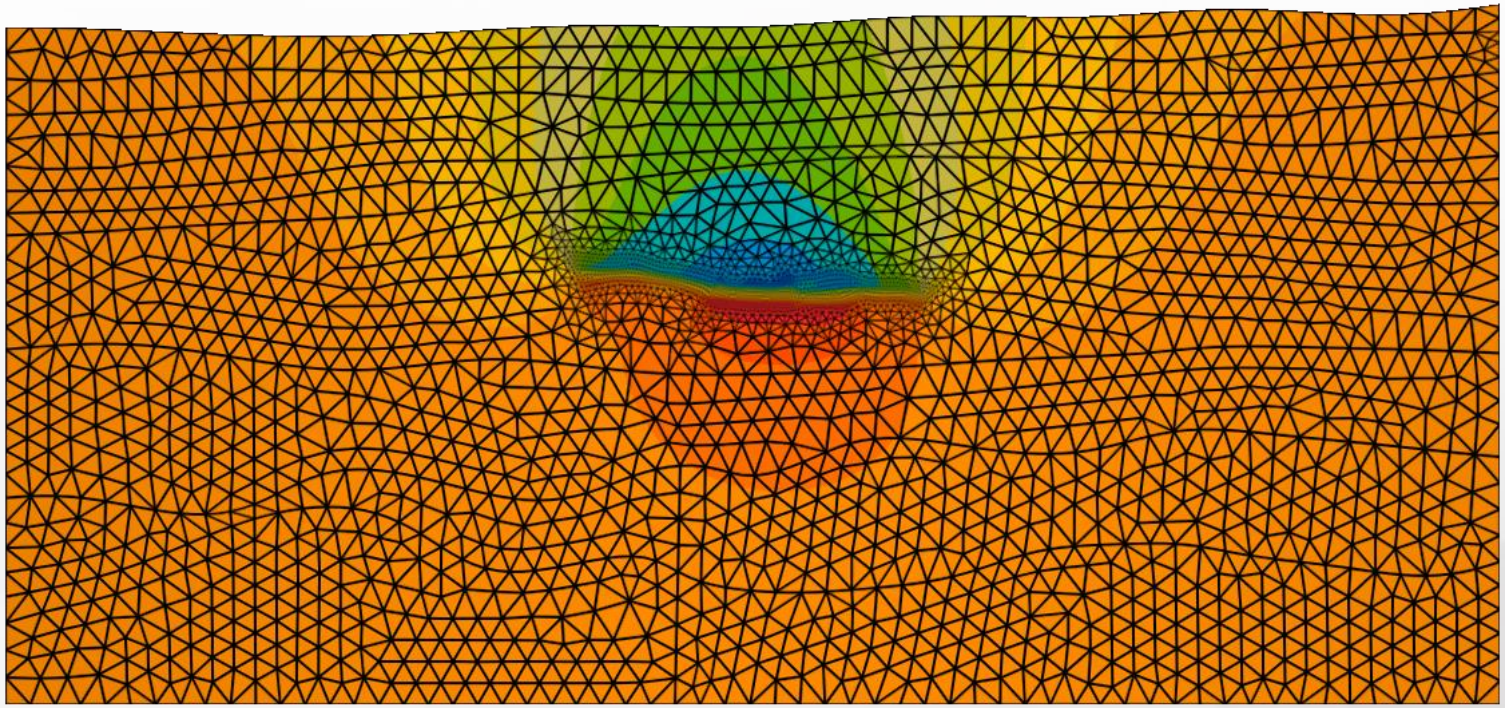
Example set purpose

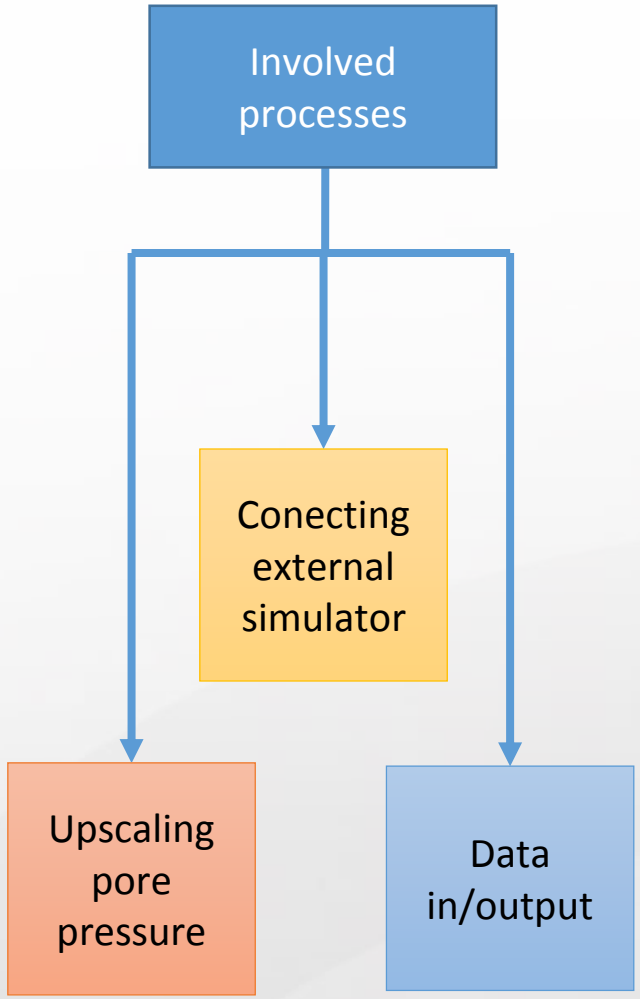
This example shows:

- How to deal with external simulators during the orchestration
- How to load a mesh objet from a mesh file / to write a mesh objet to a mesh file
- How to prepare a destination mesh to receive data from a source mesh
- How to orchestrate an upscaling process

The example

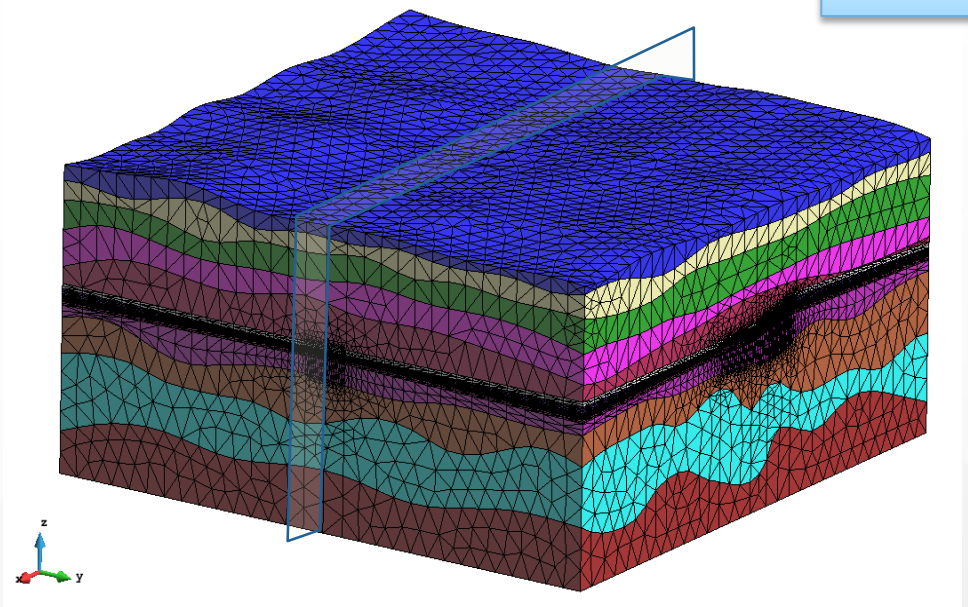
The example presents a one-way HM coupling simulation in 2D involving upscaling process and use of external simulator. Pore pressures resulting from the hydraulic simulation will impact the displacements computed during the mechanical process.

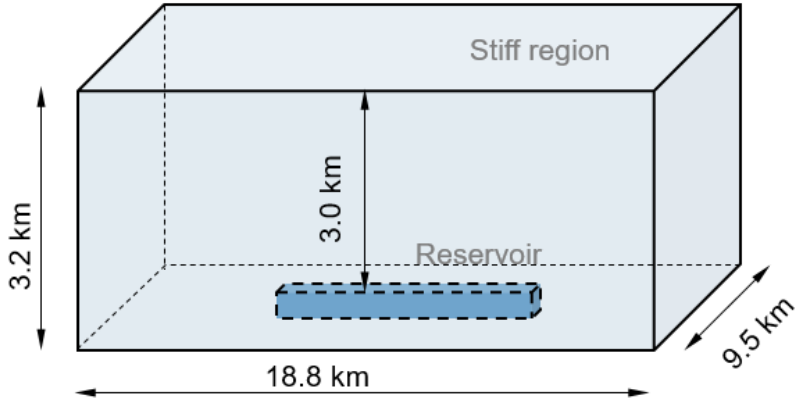


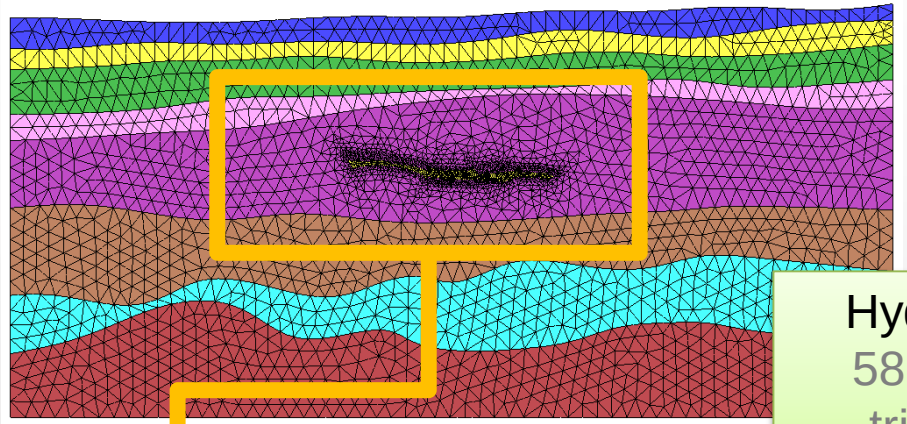


The problem

Simulation time 4000 days

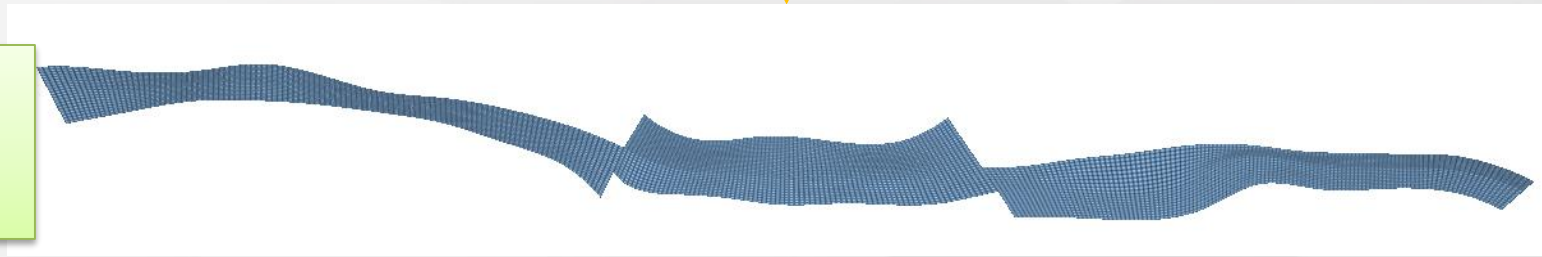






Hydraulic Mesh
5898 quadratic triangles (T6)

Mechanical Mesh
7020 linear quadrilaterals (Q4)



Model file: Meshes

Multi-physics and multi-scale simulation often requires a specific discretization for each physic domain. Here we declare empty hydraulic and mechanical meshes to be filled during the orchestration.

```
-- State variables
StateVar{id = 'POREP', description = 'pore pressure'}
StateVar{id = 'u', dim = 2, description = 'Displacements in the X and Y directions'}

-- Mesh definition
Mesh
{
  id          = 'mesh_H',
  typeName    = 'GemaMesh.elem',
  description = '2D mesh for hydraulic problem',
  coordinateDim  = 2,
}

Mesh
{
  id          = 'mesh_M',
  typeName    = 'GemaMesh.elem',
  description = '2D mesh for mechanical problem',
  coordinateDim  = 2,
}
```

Hydraulic Mesh

Mechanical Mesh

Solution file: External simulator

Here we want to build an automatic way to run the external simulador (geoflux3D) through the orquestration *Lua* script and automatically reproduce its working environment.
Important Note: it requires to be able to run the simulator in command line.

```
function ProcessScript()

  dofile('geoflux.lua')

  -- Run external hydraulic simulation
  geoflux.runHydraulic(...)

  -- Run upscaling process
  ...

  -- Run external mechanical simulation
  geoflux.datFile(...)
  geoflux.bcoFile(...)
  geoflux.runMechanical(...)

end
```

allow access to the *Lua* functions of our external simulator (geoflux3D)

function implemented in geoflux.lua responsible for running hydraulic simulation

generate geoflux3D specific configuration files

function implemented in geoflux.lua responsible for running mechanical simulation

Example of tasks in geoflux.lua:

- main function responsible for running geoflux3D.exe
- specific calls to GeMA mesh
- write/read/modify configuration files
- convert input/output files
- auxiliar routines ...

Solution file: Load/Save mesh

During the process, the hydraulic and mechanical meshes objects are filled. The pore pressure data is stored on the meshes nodes and can be written on output files in for visualization (NF or VTK format).

```
-- Load hydraulic mesh with porep values
io.loadMeshFromFile('mesh_H', posFile_H, 'nf', 'POREP', nil)

-- Save hydraulic mesh in vtk file
vtkLib.saveMeshFile('mesh_H', {'POREP'}, nil, outName_H, {allStates = true})

...

-- Load mechanical mesh with no values
io.loadMeshFromFile('mesh_M', ntrFile_M, 'nf')

...

-- Save mechanical mesh in pos and vtk files
vtkLib.saveMeshFile('mesh_M', {'POREP'}, nil, outName_M..'_mm', {allStates = true})
io.saveMeshFile('mesh_M', outName_M..'_mm', 'nf', 'POREP', nil, {allStates = true})
```

Hydraulic Mesh

Mechanical Mesh

No data, only nodes coordinates and cells topology are filled

data of all simulation states is saved

Solution file: Prepare meshes for upscaling

Important Note: it is the responsibility of the user to prepare the mechanical mesh to receive data. After loading the mesh geometry (see previous slide), in our case we have to define a new data value on nodes to store the pore pressure data. As we already know the number of steps from the hydraulic simulation we can allocate the required space to store the data and set the corresponding time for each state.

```
-- Add porep value in mechanical mesh
local mesh_H = modelData:mesh('mesh_H')
local mesh_M = modelData:mesh('mesh_H')
local nstate = mesh_H:numNodeValueStates('POREP')
mesh_M:addNodeValueSet(ValueInfo('node attribute', {id = 'POREP', history = nstate}))
```

data type

data name

allocation of fixed number of states

```
-- Set porep value times
for i=1, nstate do
  local t = mesh_H:nodeValueStateTime('POREP', i-1)
  mesh_M:setNodeValueStateTime('POREP', i-1, t)
end
```

set the mechanical pore pressure times as the same as in the hydraulic case

Solution file: Upscaling process (1)

Creating a bucket index is the first step of the upscaling process. It allows us to reduce the locating algorithms complexity during the key operation of mesh mapping (data transfer) between two different meshes. In our case such operation requires to create a bucket index that stores the elements of the hydraulic mesh.

```
-- Build bucket index for hydraulic mesh
local bucketOptions = {
  capacity         = 100,    -- Each bucket will have a maximum of 100 references stored in
  gridStartingSize = -1,     -- The grid size will be automatically initialized
  activeOnly       = false,  -- All mesh elements will be stored, active or not
  info             = true,   -- Will print informations about the mesh mapping
}

local bucketIndex = mm.buildBucketIndex('mesh_H', 1, bucketOptions)
```

bucket index will store references to the mesh elements

Solution file: Upscaling process (2)

The mesh mapping is the key operation of the upscaling process. It first locates the mechanical nodes inside the hydraulic mesh (identifying its containing cell or closest node). Then it computes the corresponding pore pressure value using the specified interpolation method.

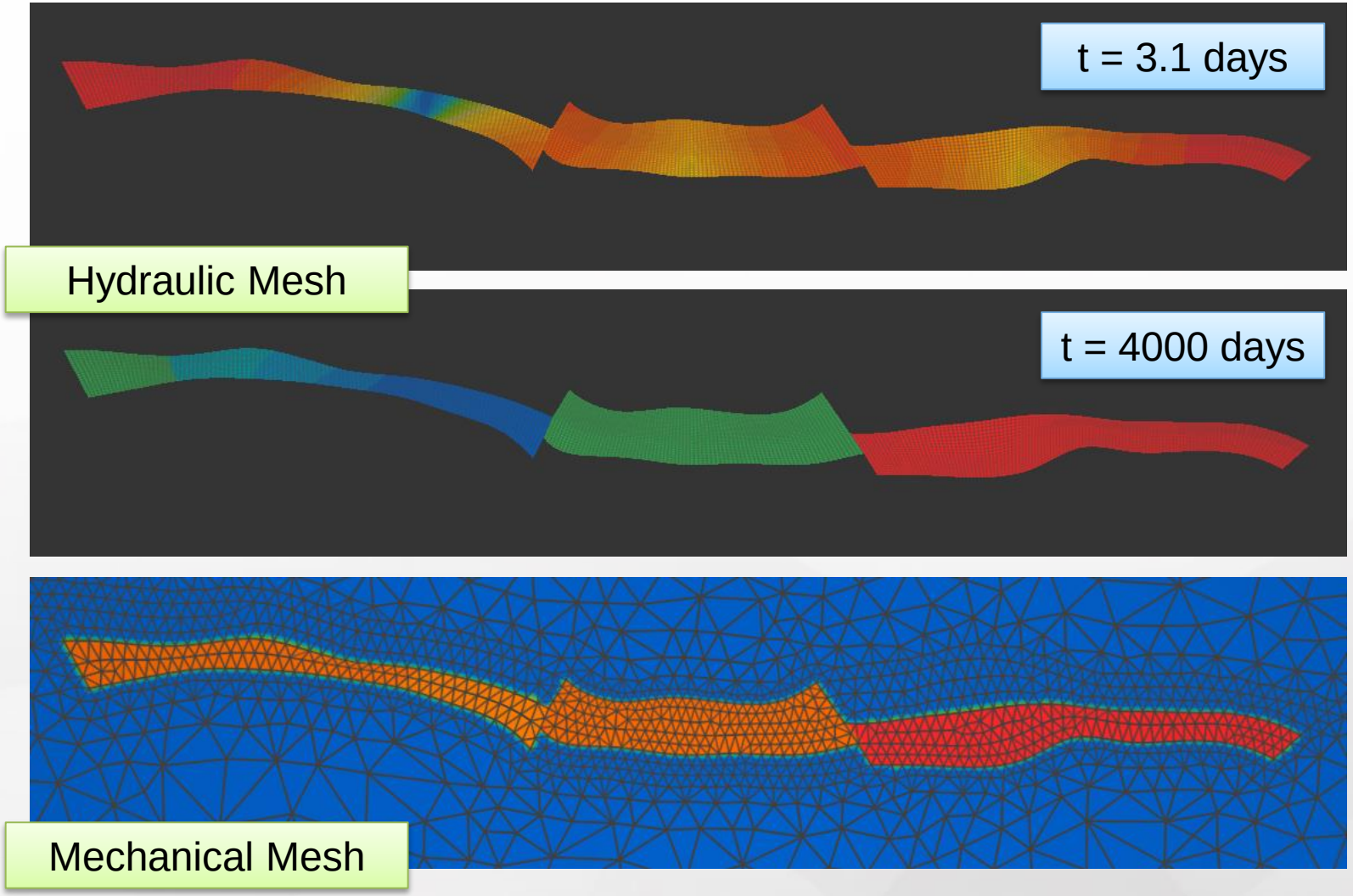
```
-- Transfer porep data to mechanical mesh
local eps = 0.5 * mesh_M:edgeMinLength()
local nodata = -9999
local mappingOptions = {
  noDataMode  = true,     -- Will activate the `noData` mode
  noDataValue = nodata,   -- Will set the `noData` value to nodata
  noDataEps   = eps,      -- Will set the tolerency value to eps
  state       = -1,       -- All states will be mapped
  interpType  = 'idw',    -- IDW interpolation will be used
  interpParam = 2.0,      -- IDW parameter (inverse distance squared in this case)
  activeOnly  = false,    -- Gauss points of all elements of the destination mesh will receive data
  info        = true,     -- Will print informations during the mesh mapping
}

mm.meshMapping('mesh_M', bucketIndex, 'POREP', nil, false, mappingOptions)
```

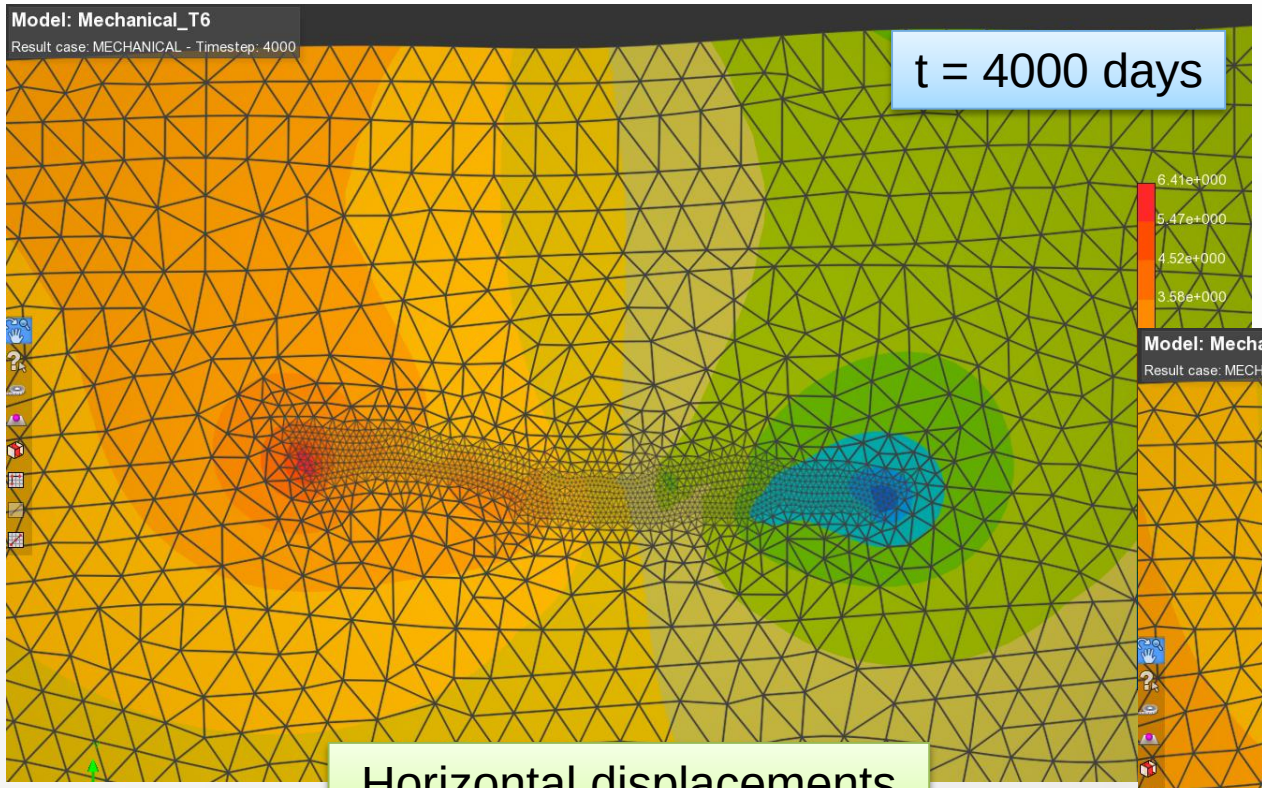
distance criterion above which we consider that mechanical nodes are out of the hydraulic mesh domain

set the interpolation method and its parameters (see interpolation section for details and options)

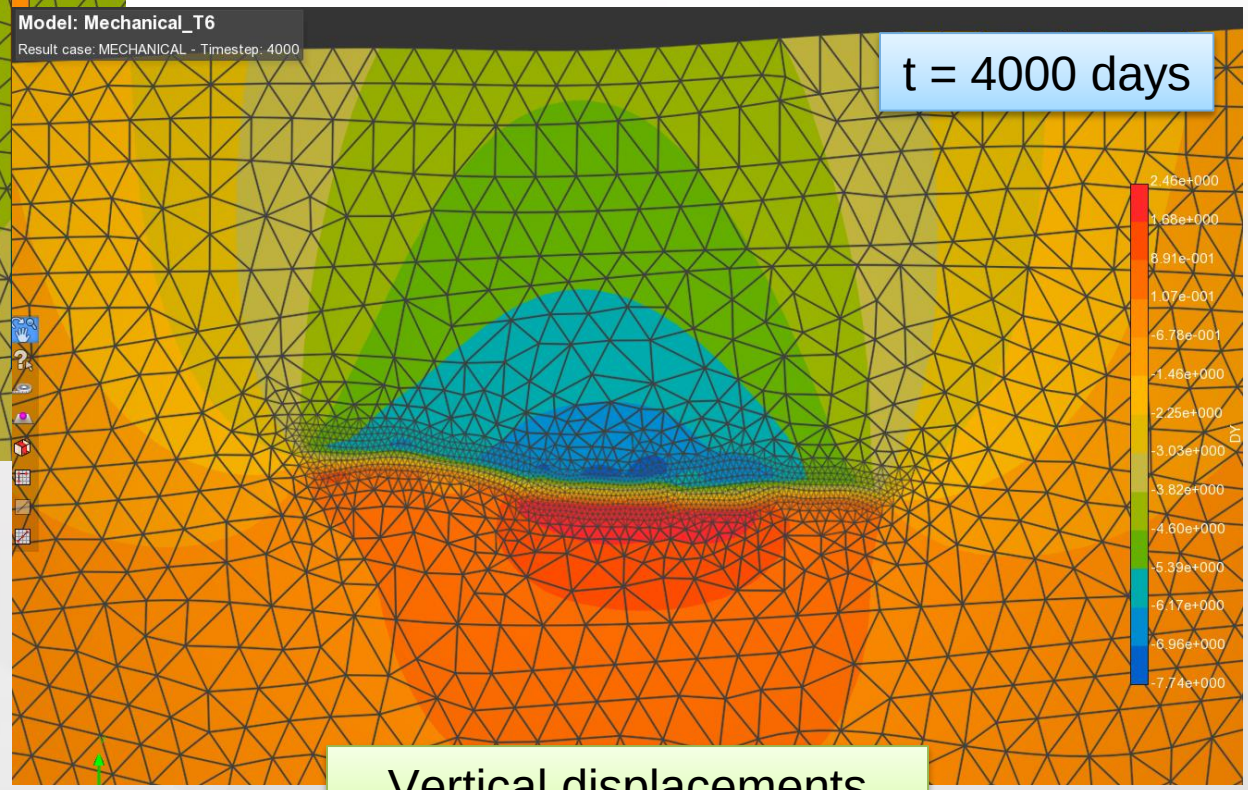
Results: Evolution of the hydraulic pore pressure



Results: Mechanical displacements



Horizontal displacements



Vertical displacements