Internal extrusion and working with structured meshes

Peter Råback
ElmerTeam
CSC – IT Center for Science

Elmer/Ice advanced course
CSC, 4-6.11.2013

Outline

- About structured meshes in Elmer
- Extrusion of meshes
- Utilizing extruded structures

Structured meshes for computational glaciology

- Generally Elmer treats all meshes in Elmer as unstructured
- In computational glaciology the footprint is always of irregular shape
- For optimal accuracy it makes sense that the number of elements in depth direction does not vary
  - Solution: 2D meshes + extrusion
- Extrusion strategies written mainly for computational glaciology but may also have other used

Creating extruded meshes

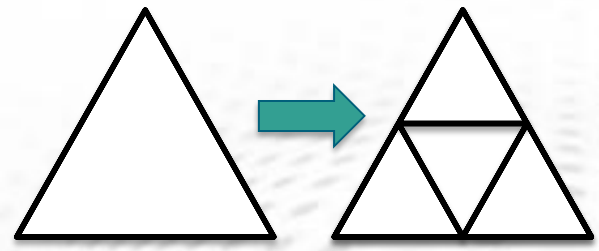
- ElmerGrid
  - 2D Elmer mesh format -> extruded mesh
- Stand-alone program
  - Written by Thomas
- Internal extrusion
  - Performed on the parallel level
  - Minimizes disk I/O
  - Removes memory bottle-necks

Bottle-necks in pre-processing

- After the solution pre-processing is typically the 2nd most time- and memory intensive task
- Mesh partitioning is typically less laborious than mesh generation
  - In Elmer we haven't utilized parallel graph partitioning libraries (e.g. ParMetis)
- Serial mesh generation limited to around ~10 M elements

Finalizing the mesh in parallel level

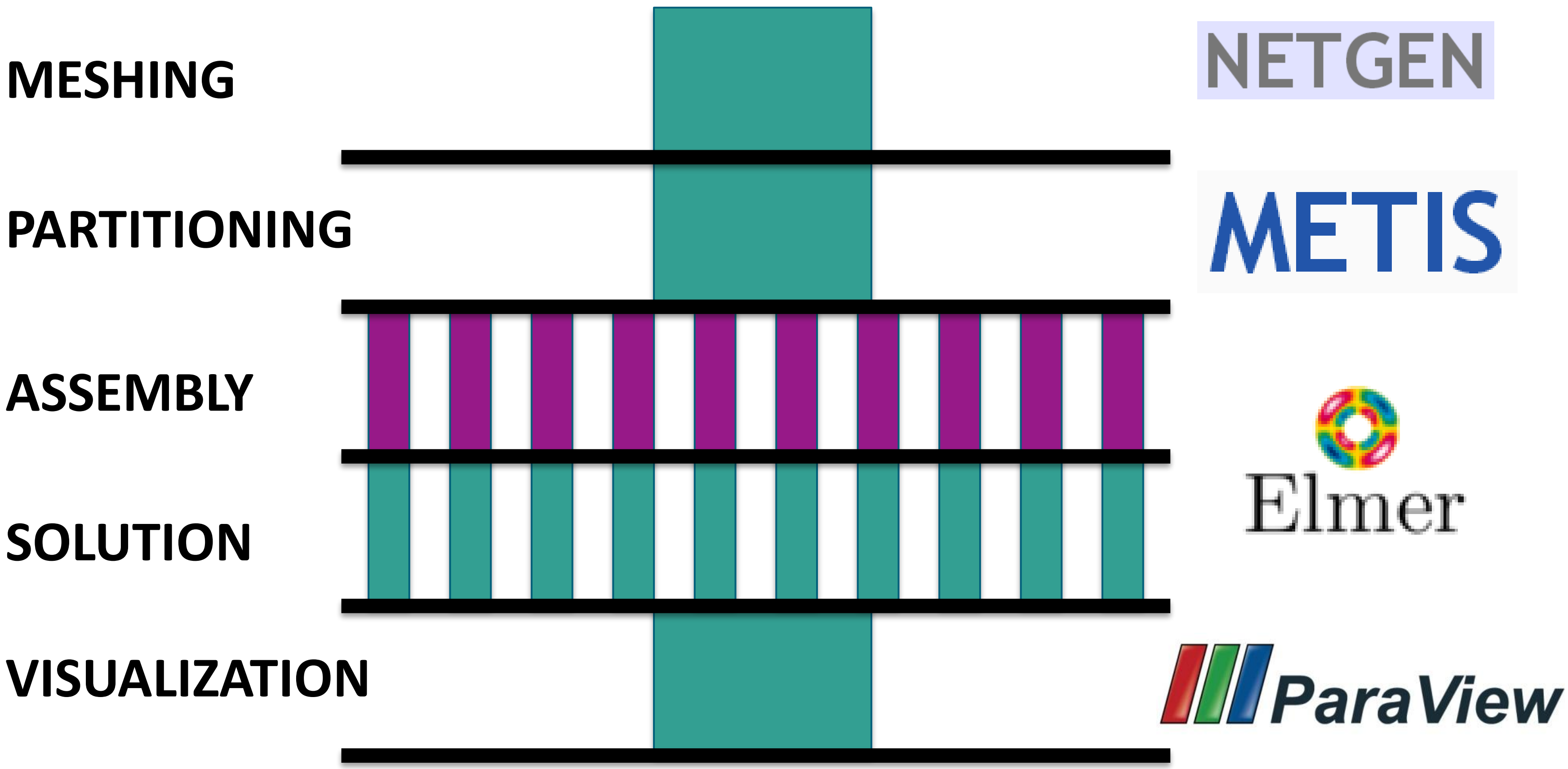
- First make a coarse mesh and partition it
- Bisection of existing elements in each direction
  - *2^DIM^n* -fold problem-size
  - Known as ”Mesh Multiplication”
  - Simple inheritance of mesh grading



- Increase of element order (p-elements)
  - p-hierarchy enables the use of p-multigrid
- Extrusion of 2D layer into 3D for special cases
  - Example: Greenland Ice-sheet

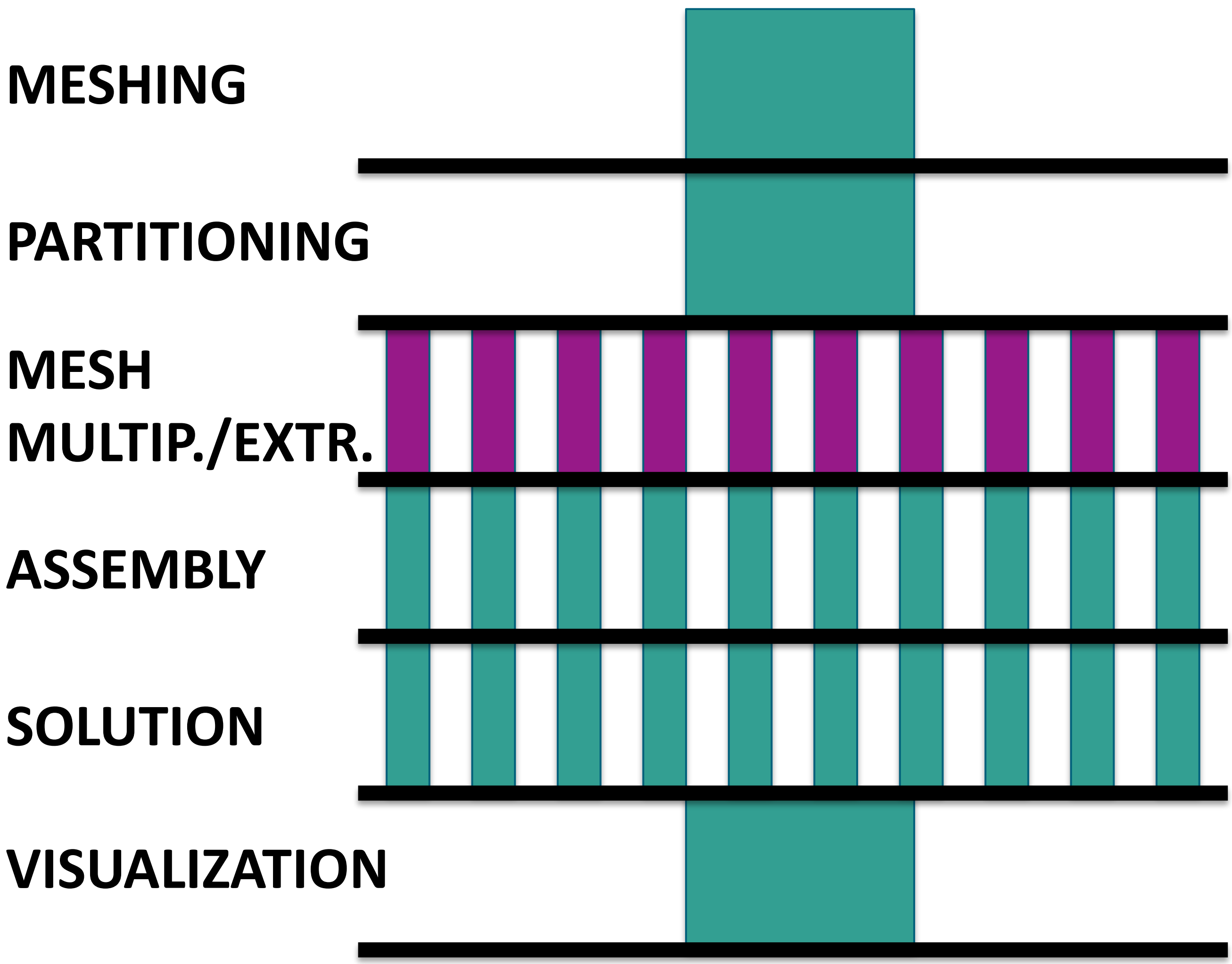
Standard parallel workflow

- Both assembly and solution is done in parallel using MPI
- Assembly is trivially parallel
- This is the basic parallel workflow used for Elmer



Parallel workflow

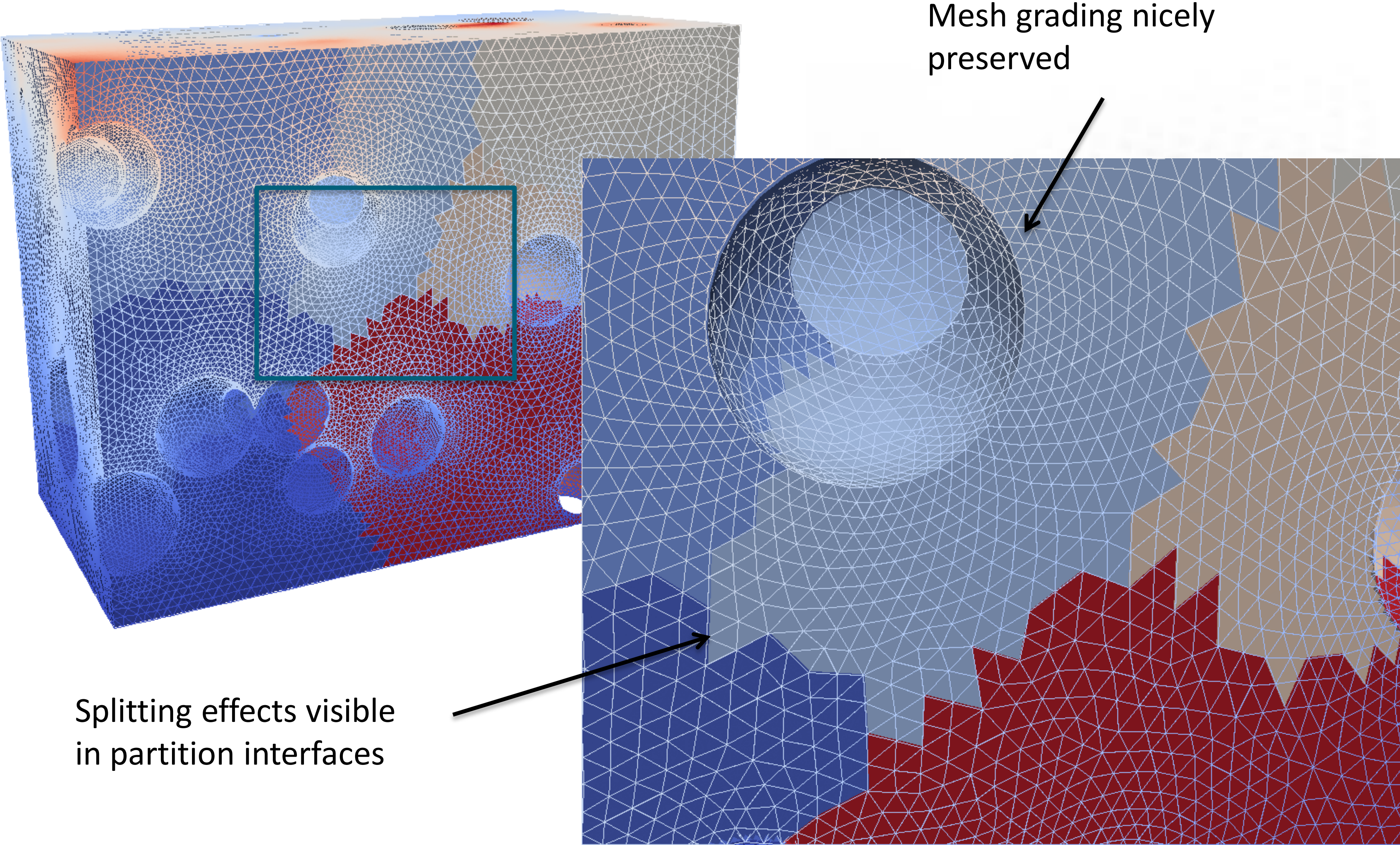
- Large meshes may be finilized at the parallel level



Mesh multiplication, example

`Mesh Levels = 2`

Mesh grading nicely preserved



Splitting effects visible in partition interfaces

Mesh Multiplication, example

- Implemented in Elmer as internal strategy ~2005
- Mesh multiplication was applied to two meshes
  - Mesh A: structured, 62500 hexahedrons
  - Mesh B: unstructured, 65689 tetrahedrons
- The CPU time used is negligible

Mesh	#splits	#elems	#procs	T_center (s)	T_graded (s)
A	2	4 M	12	0.469	0.769
	2	4 M	128	0.039	0.069
	3	32 M	128	0.310	0.549
B	2	4.20 M	12	0.369	
	2	4.20 M	128	0.019	
	3	33.63 M	128	0.201	

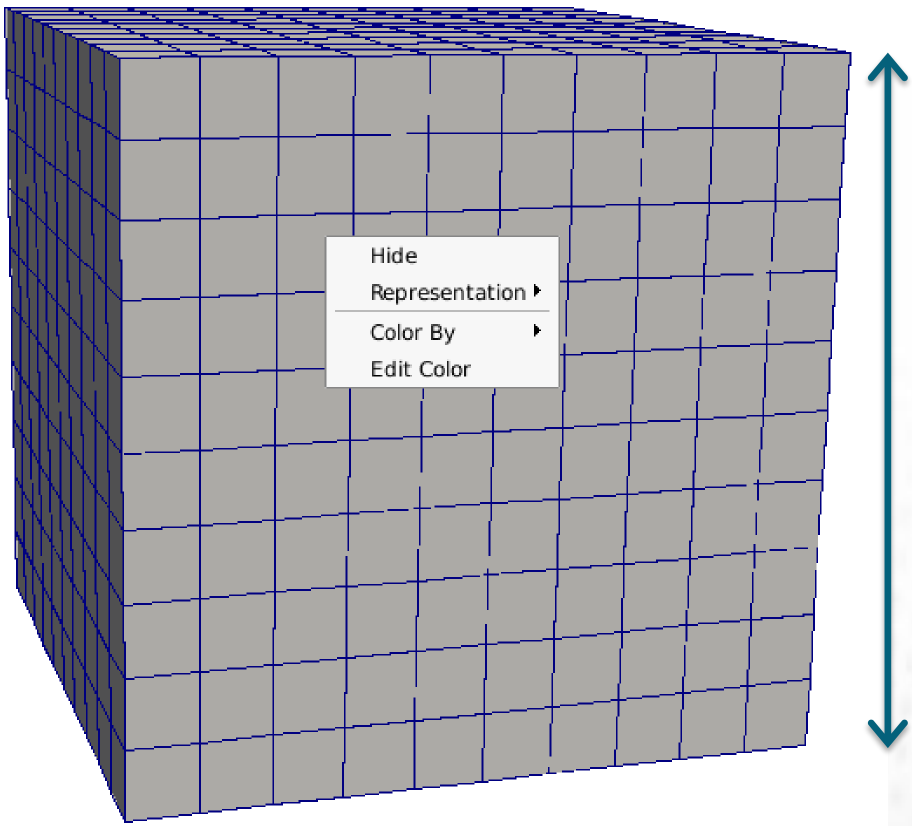
Limitations of mesh multiplication

- Standard mesh multiplication does not increase geometric accuracy
  - Polygons retain their shape
  - Mesh multiplication could be made to honor boundary shapes (done in Alya by BSC, Spain)
- Optimal mesh grading difficult to achieve
  - The coarsest mesh level does not usually have sufficient information to implement fine level grading

Extrusion of partitioned meshes

- Implemented as an internal strategy in Elmer (2013)
  - Juha, Peter & Rupert
- First partition a 2D mesh, then extrude into 3D
- Implemented also for partitioned meshes
  - Extruded lines belong to the same partition by construction!
- Deterministic, i.e. element and node numbering determined by the 2D mesh
  - Complexity: O(N)
- There are many problems of practical problems where the mesh extrusion of a initial 2D mesh provides a good solution
  - One such field is glasiology where glaciers are thin, yet the 2D approach is not always sufficient in accurary

Internal extrusion

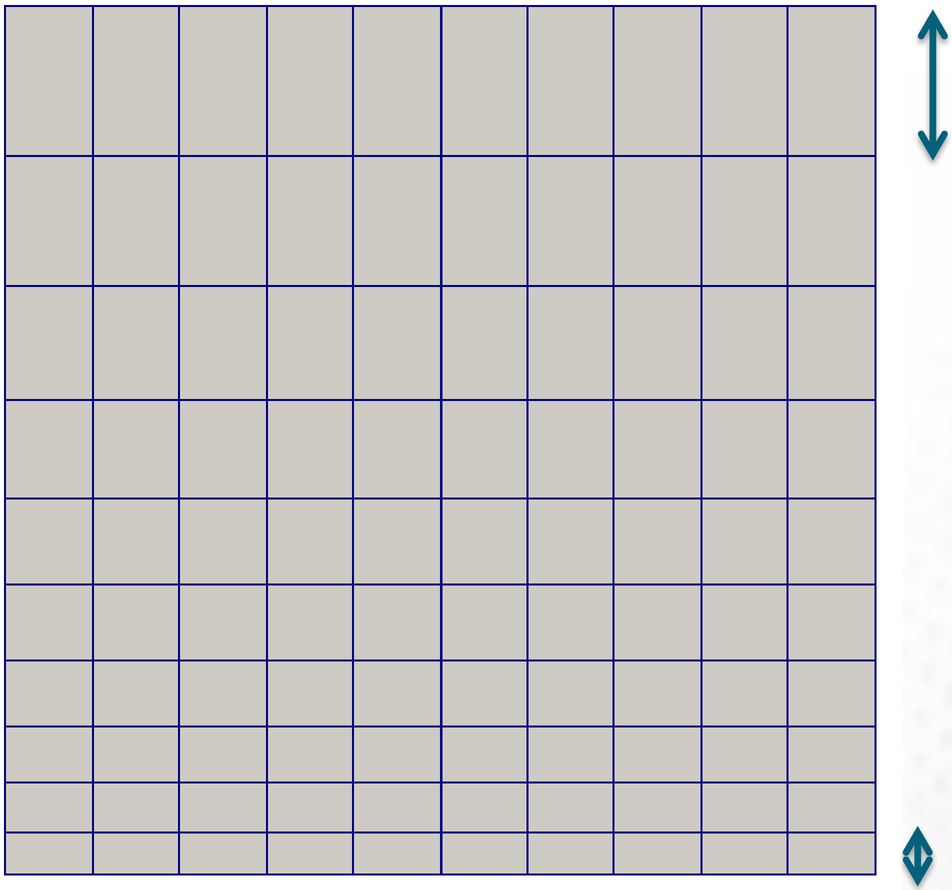


`Extruded Mesh Levels = 11`

The number of levels is a bit unintuitive as it refers to node layers.

By default z in [0,1]

Internal extrusion



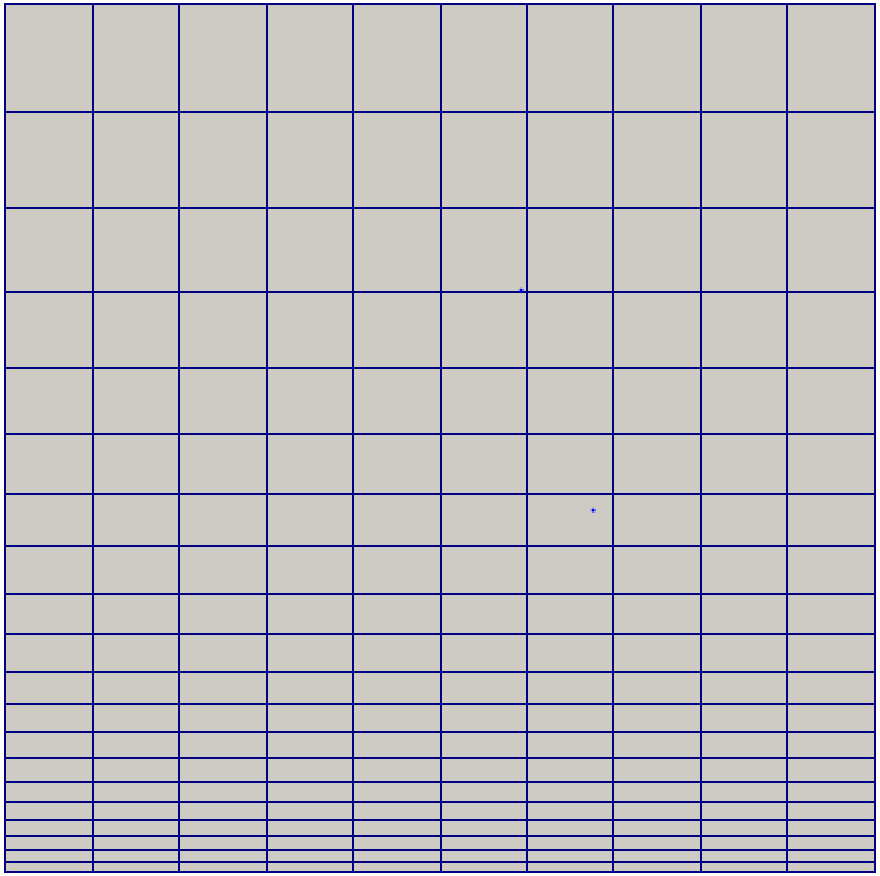
```
Extruded Mesh Levels = 11
Extruded Mesh Ratio = 4.0
```

UnitSegmentDivision: Mesh division ready
UnitSegmentDivision: w(0) : 0.0000E+00
UnitSegmentDivision: w(1) : 4.9566E-02
UnitSegmentDivision: w(2) : 1.0650E-01
UnitSegmentDivision: w(3) : 1.7191E-01
UnitSegmentDivision: w(4) : 2.4703E-01
UnitSegmentDivision: w(5) : 3.3333E-01
UnitSegmentDivision: w(6) : 4.3247E-01
UnitSegmentDivision: w(7) : 5.4634E-01
UnitSegmentDivision: w(8) : 6.7714E-01
UnitSegmentDivision: w(9) : 8.2740E-01
UnitSegmentDivision: w(10) : 1.0000E+00

Internal extrusion

Just a dummy, refers to z in [0,1]

```
Extruded Mesh Levels = 21
Extruded Mesh Density = Variable Coordinate 1
  Real MATC "1+10*tx"
```



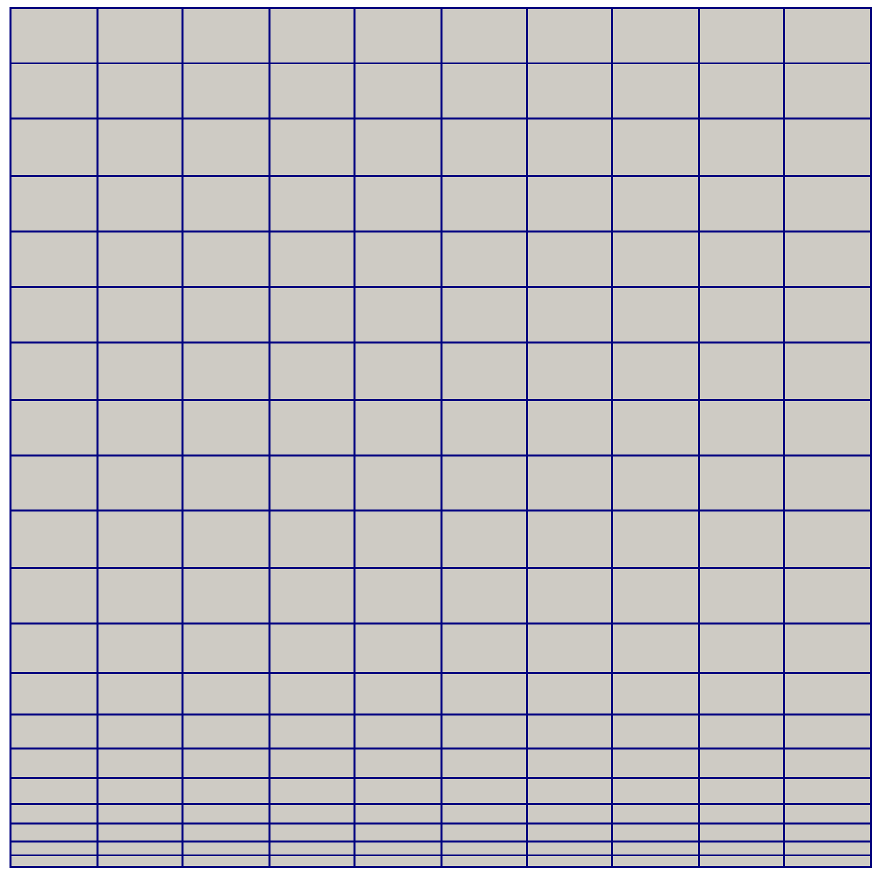
Internal extrusion

Just a dummy, refers to z in [0,1]

```
Extruded Mesh Levels = 21
Extruded Mesh Density = Variable Coordinate 1
  Real
    0.0 1.0
    0.3 5.0
    1.0 5.0
  End
```

Density characterized by a mesh parameter *h*

Always the requested number of layers generated!

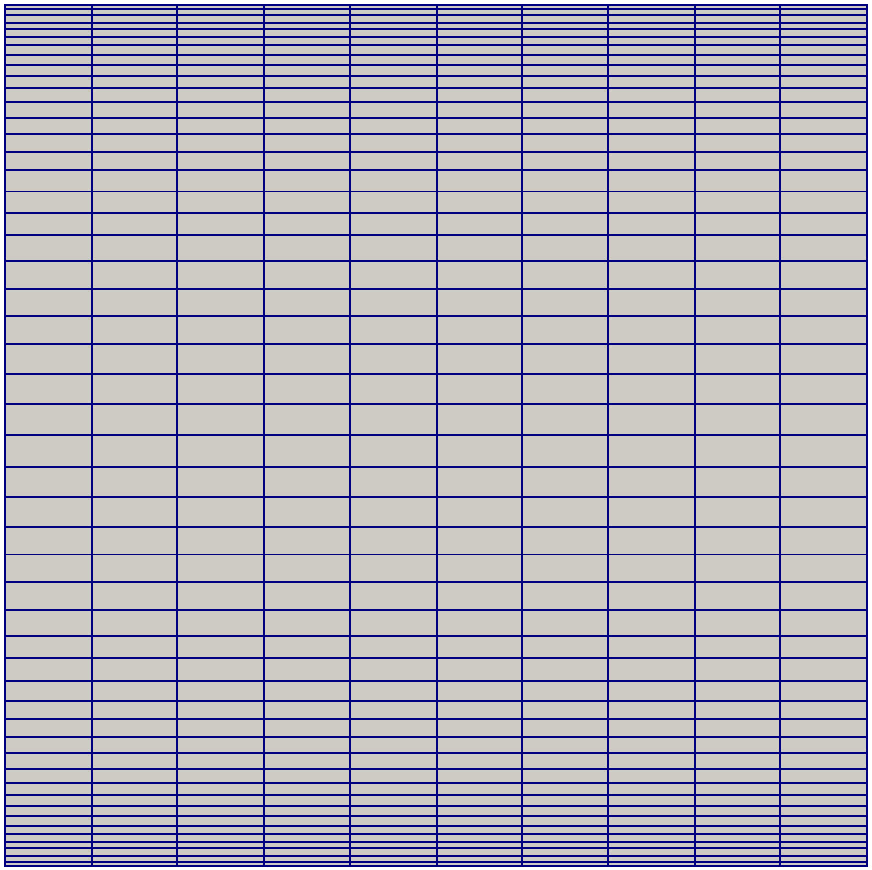


Internal extrusion

```
Extruded Mesh Levels = 11
Extruded Mesh Density = Variable Coordinate 1
  Real MATC "0.2+sin(pi*tx)"
```

Any functional dependence is ok as long as it is positive!

The optimal division is found iteratively using Gauss-Seidel type of iteration and large variations make the iterations converge slowly.



MeshExtrude subroutine in MeshUtils.src

```
!------------------------------------------------------------------------------
!> Given a 2D mesh extrude it to be 3D. The 3rd coordinate will always
!> be at the interval [0,1]. Therefore the adaptation for different shapes
!> must be done with StructuredMeshMapper, or some similar utility.
!> The top and bottom surface will be assigned Boundary Condition tags
!> with indexes one larger than the maximum used on by the 2D mesh.
!------------------------------------------------------------------------------
 FUNCTION MeshExtrude(Mesh_in, in_levels) RESULT(Mesh_out)
!------------------------------------------------------------------------------
   TYPE(Mesh_t), POINTER :: Mesh_in, Mesh_out
   INTEGER :: in_levels
!------------------------------------------------------------------------------
 .....
```

UnitSegmentDivision in MeshUtils.src

```
!------------------------------------------------------------------------------
!> Create node distribution for a unit segment x \in [0,1] with n elements
!> i.e. n+1 nodes. There are different options for the type of distribution.
!> 1) Even distribution
!> 2) Geometric distribution
!> 3) Arbitrary distribution determined by a functional dependence
!> Note that the 3rd algorithm involves iterative solution of the nodal
!> positions and is therefore not bullet-proof.
!------------------------------------------------------------------------------
 SUBROUTINE UnitSegmentDivision( w, n )
   REAL(KIND=dp), ALLOCATABLE :: w(:)
   INTEGER :: n
   !---------------------------------------------------------------
```

```
! Compute the point in the local mesh xn \in [0,1]
! and get the mesh parameter for that element from
! external function.
!---------------------------------------------------
DO i=1,n
  xn = (w(i)+w(i-1))/2.0_dp
  h(i) = ListGetFun( CurrentModel %
       Simulation,'Extruded Mesh Density', xn )
END DO

! Utilize symmetric Gauss-Seidel to compute the new
! positions, w(i) from a weighted mean of the desired
! elemental densities, h(i).
!---------------------------------------------------
DO i=1,n-1
  w(i) = (w(i-1)*h(i+1)+w(i+1)*h(i))/(h(i)+h(i+1))
END DO
DO i=n-1,1,-1
  w(i) = (w(i-1)*h(i+1)+w(i+1)*h(i))/(h(i)+h(i+1))
END DO
```

$$dw \propto h$$

Internal extrusion

Other keywords:

```
Extruded Coordinate Index = Integer ! 1,2,3
```

What coordinate to extrude

```
Extruded Min Coordinate = Real
Extruded Max Coordinate = Real
```

Override the default interval [0,1]

```
Preserve Baseline = Logical
```

Preserve the 1D boundary of the baseline

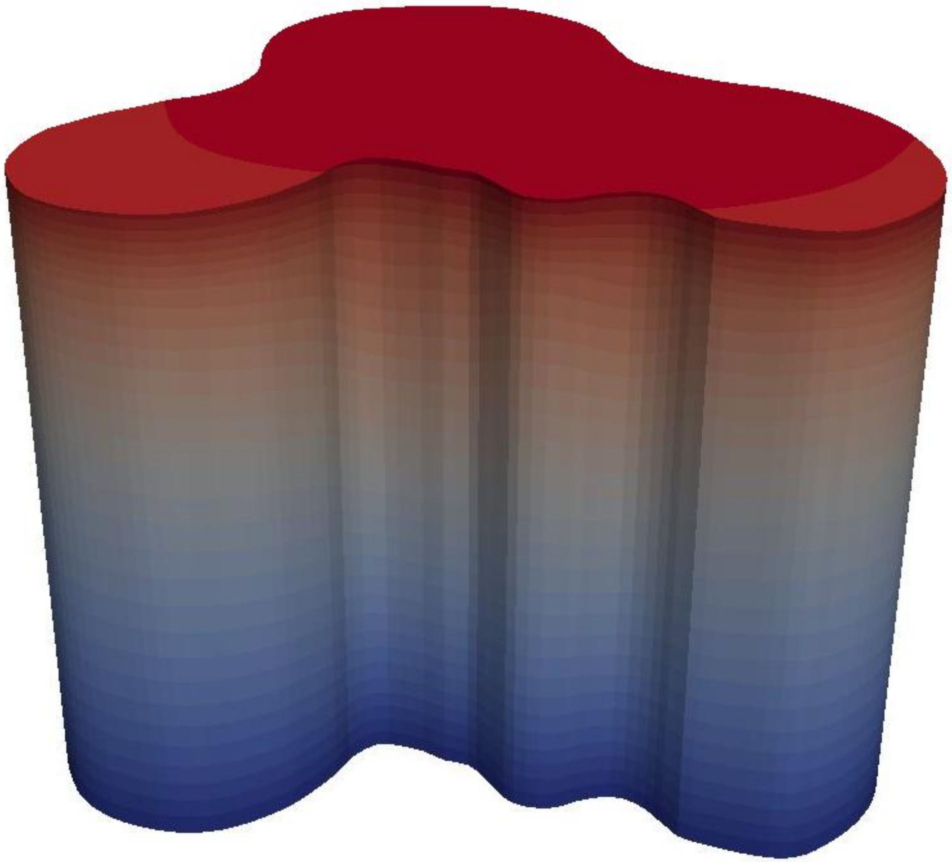
Internal extrusion – numering of BCs

- Side boundaries get a BC constraint so that
  - 2D constraint BC = 1D contraint BC + offset
  - offset is set if the baseline BCs are preserved
- Top and bottom boundaries get the next free BC constraint indexes
- Note thet the BCs refer directly to the ”Boundary Condition”
  - ”Target Boundaries” is used only when reading in the mesh in the 1st place and they are not available any more at this stage

Internals extrusion – real shapes

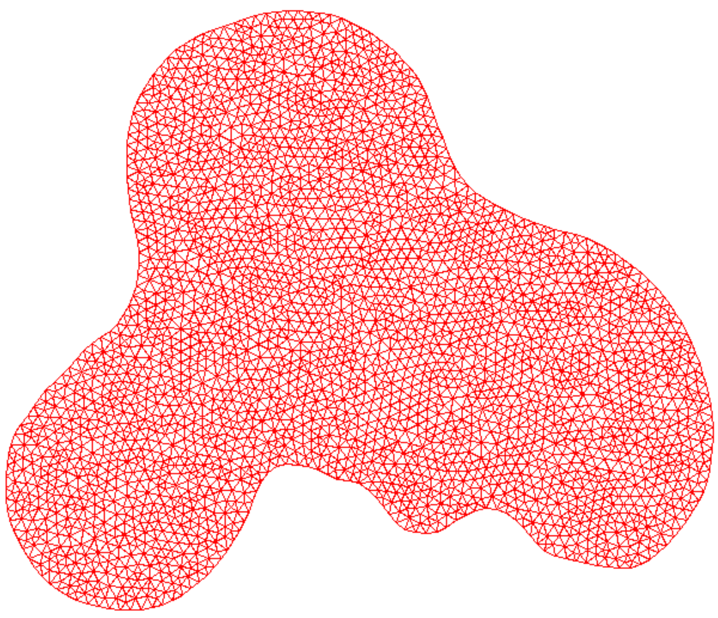
- The mesh division is only set along the 1D extruded line
- For true geometries some additional strategy is needed to map the mesh between the real top and bottom surfaces
  - StructuredMeshMapper
  - MeshUpdate solver

Internal extrusion: Example, AaltoVase

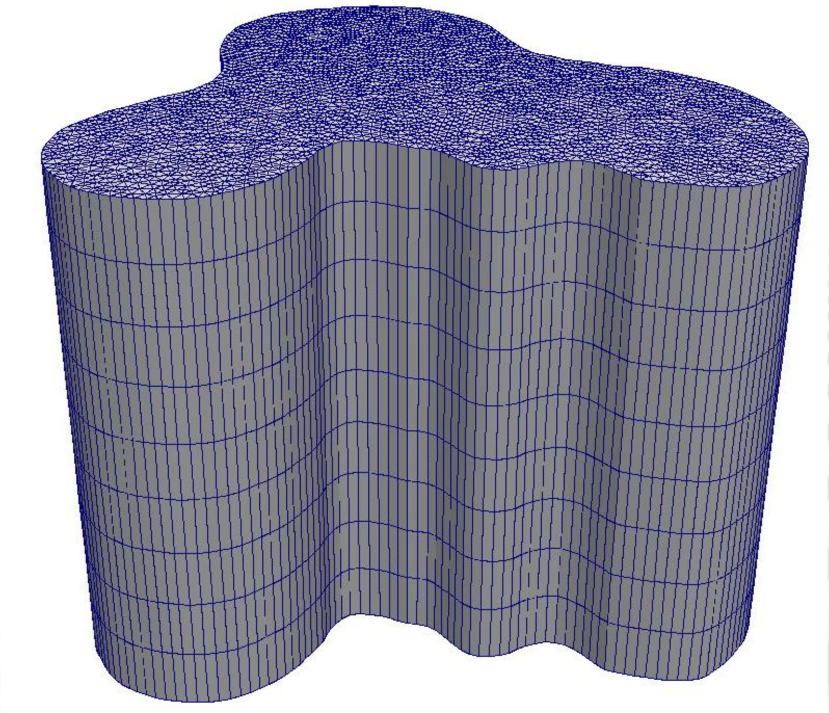


Design Alvar Aalto, 1936

Internal extrusion: Example, extrude.sif



2D mesh by Gmsh



3D internally extruded mesh

Play around with different options to see how your vase is meshed.

Utilizing extruded structures

- If the mesh is extruded it makes sense to utilize this fact also in later steps
  - Operators in the extruded directions
  - Combination of full 3D and 2D higher order models
- Tailored solvers that assume extruded structure
  - **`StructuredMeshMapper`**
  - **`StructuredProjectToPlane`**
  - **`StructuredFlowLine`**
- No assumptions on the numbering of the nodes is needed

DetectExtrudedStructure in MeshUtils.src

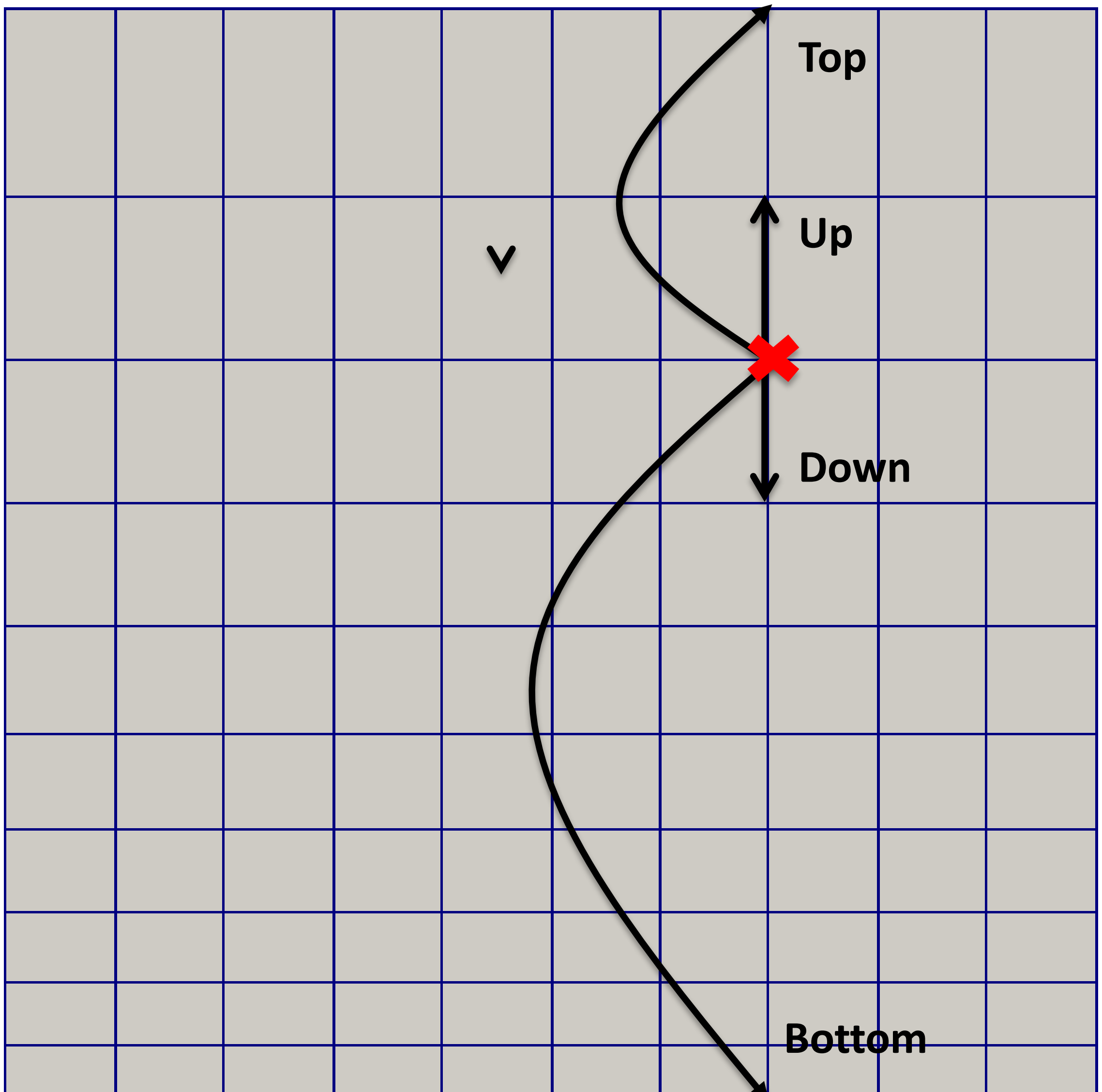
```
!------------------------------------------------------------------------------
!> This subroutine finds the structure of an extruded mesh even though it is
!> given in an unstructured format. The routine may be used by some special
!> solvers that employ the special character of the mesh.
!> The extrusion is found for a given direction and for each node the corresponding
!> up and down, and thereafter top and bottom node is computed.
!------------------------------------------------------------------------------
SUBROUTINE DetectExtrudedStructure( Mesh, Solver, ExtVar, &
    TopNodePointer, BotNodePointer,  &
    UpNodePointer, DownNodePointer, &
    NumberOfLayers, NodeLayer )
```

DetectExtrudedStructure

- Go through each element
  - If in the element vector spanned by two nodes **(i,j)** is directed as extruded direction set **UpNodePointer(i)=j** or **DownNodePointer(i)=j**
  - Complexity O(N)
- Go through each element until no change
  - **TopNodePointer(i)=UpNodePointer(TopNodePointer(i))** **BotNodePoiner(i)=DownNodePointer(BotNodePointer(i))**
  - Complexity O(N*N_z)

- As a result we have for each node pointers to **up** and **down**, and **top** and **bottom** nodes at the extruded line.

DetectExtrudedStructrure – Up, Down, Top, Bottom



StructuredMeshMapper

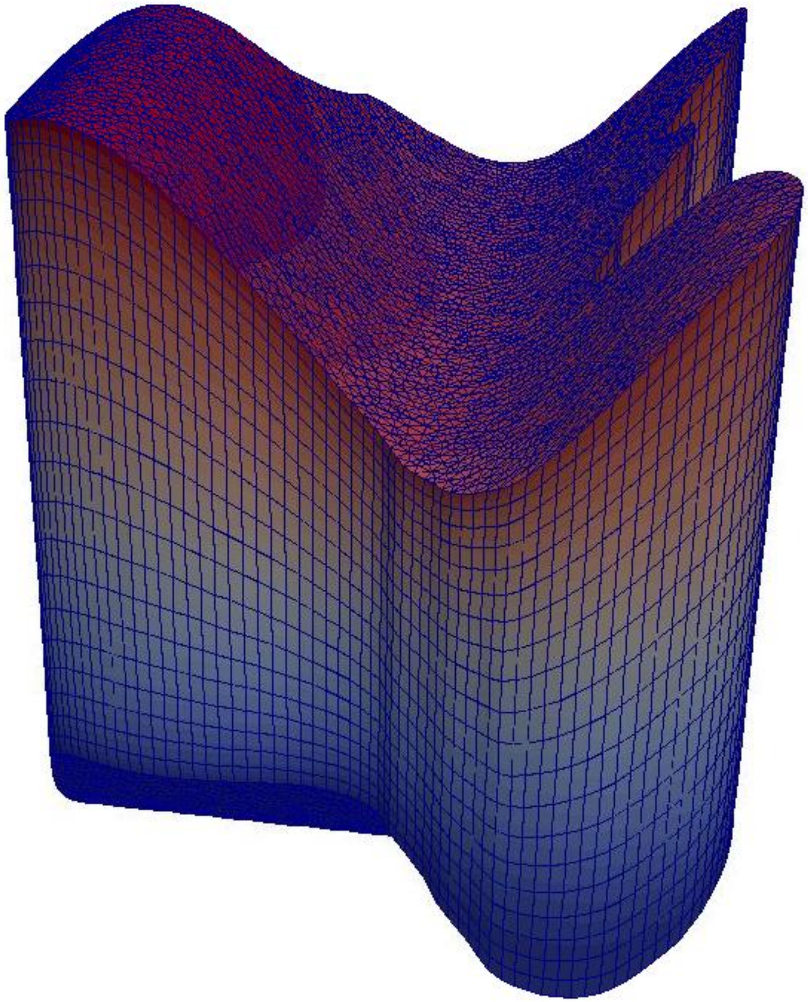
- Takes a mesh with an extruded structure
- Maps the mesh between its bottom and top surfaces
  - Original relative element division is maintained
- Various ways to define the displacement at the top and bottom
  - Constant
  - Given field
  - Variable for GetReal in boundary condition
- For documentation and explanation of keywords see Ch. 61 in Elmer Models Manual

StucturedMeshMapper vs. MeshSolve

- Pros
  - Much faster: complexity O(N)
  - No convergence issues
  - Retains the extruded form of the mesh
- Cons
  - Applicable only to extruded meshes
  - Currently does not compute mesh velocity
    - Is this needed?

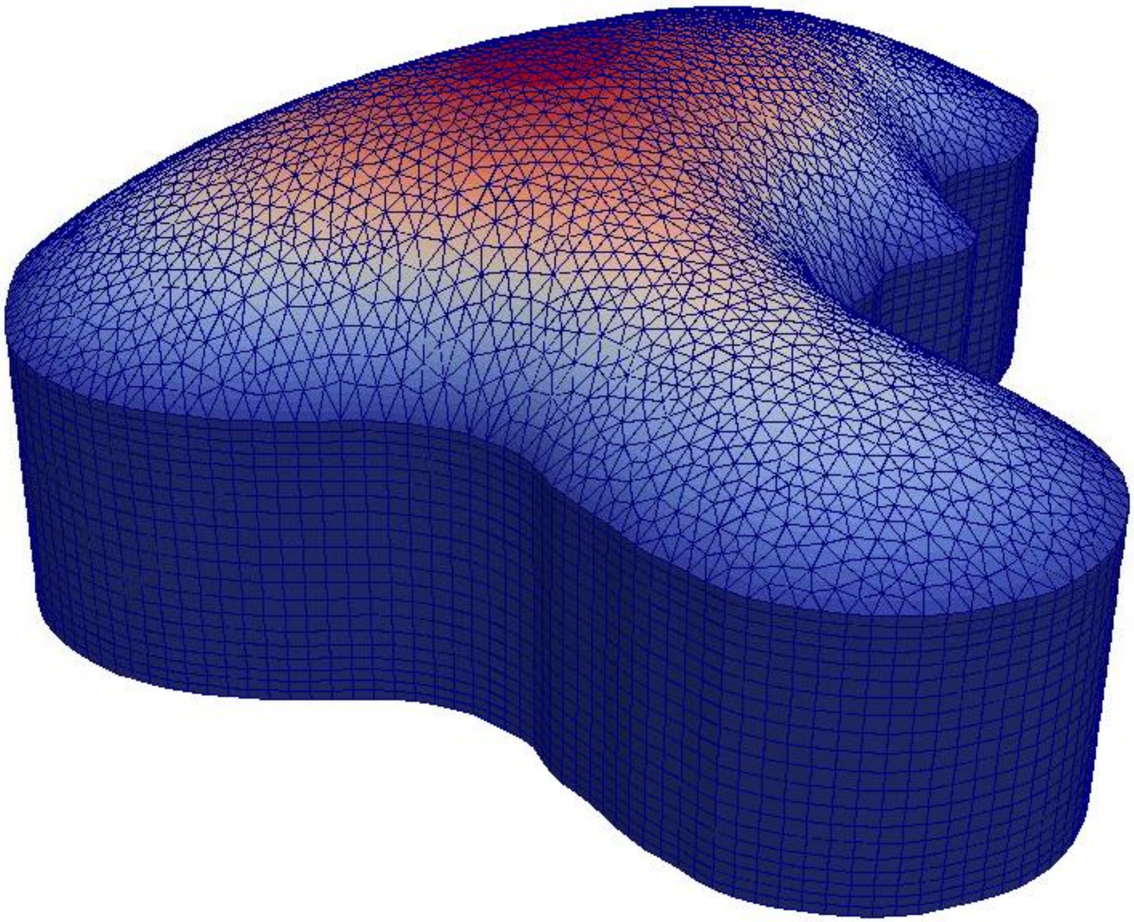
StructuredMeshMapper: Example map.sif

- Mesh mapped using analytical functions



StructuredMeshMapper: Example map_temp.sif

- Mesh mapped using a given temperature field



StructuredProjectToPlane

- Takes a mesh with an extruded structure
- Maps data to top or bottom surface, but also to whole mesh depending on the operator
  - Complexity O(N) or O(N*N_z)
- Works in parallel if each extruded line is in the same partition
  - No communication
- For documentation and explanation of keywords see Ch. 60 in Elmer Models Manual
  - Documentation is not complete!

StructuredProjectToPlane – Operators on geometry

Options for ”Operator i = String”

- height – Calculate height from bottom
- depth – Calculate depth from top
- index – index of layer starting from top
- thickness – Calculate the thickness of the mesh
- distance – Calculate the minimum distance to surface

StructuredProjectToPlane – Operators on variables

Options for ”Operator i = String”

- sum – take the sum on all nodes on the extruded line
- int – take the integral over the extruded line
- min – take the minimum value
- max – take the maximum value
- Isosurface – take the value on the isosurface
  - Additional required keywords:
    Isosurface Variable i = String
    Isosurface Value i = Real

StructuredProjectToPlane – Operators on vars…

Options for ”Operator i = String”

- layer below top – Value at the given layer
- layer above surface – Value at the given layer
  - Additional required keyword: Layer Index i = Integer

StructureProjectToPlane – operator 'int' (simplified)

- After the structured mesh is found line integrals become really simple
- Limitation: Higher order elements not used optimally

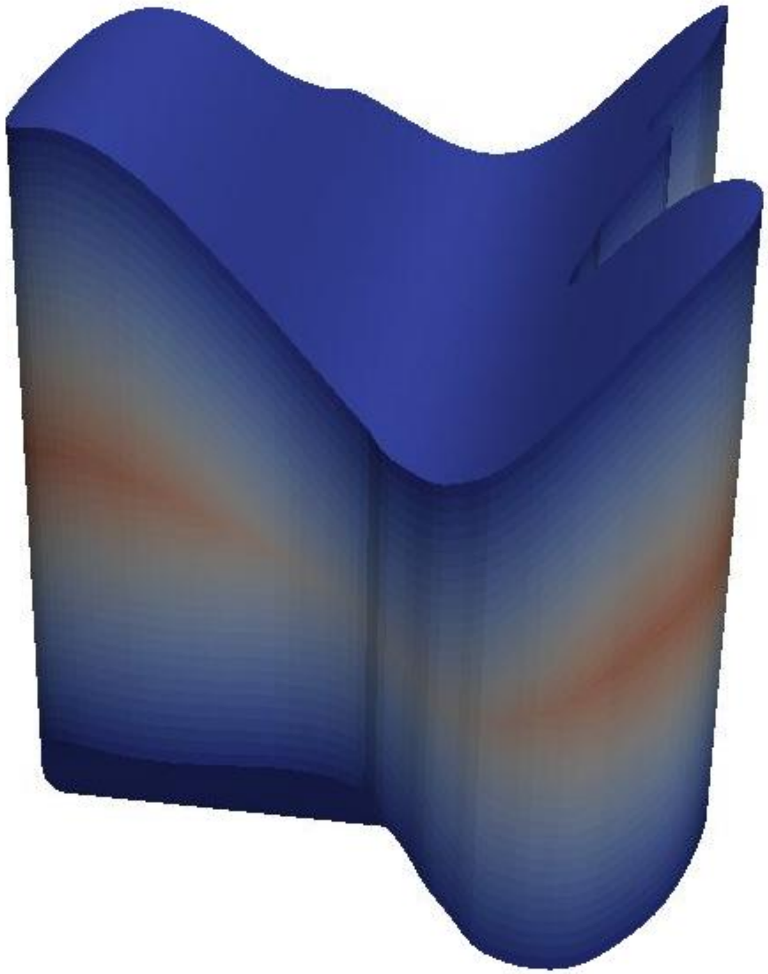
```
CASE ('int')
    TopField = 0.0_dp
    DO i=1,nsize
      itop = TopPointer(i)

      dx = 0.5*(Coord(UpPointer(i)) - Coord(DownPointer(i)))

      TopField(TopPerm(itop)) = TopField(TopPerm(itop)) + dx * FieldIn(i)
    END DO
```

StructuredProjectToPlane: Example project.sif

- Total of 12 different mapping operations on geometry and temperature



Operator 1 = depth
Operator 2 = height
Operator 3 = thickness
Operator 4 = distance
Operator 5 = index

Geometric operators

Variable 6 = Temperature
Operator 6 = min
Operator 7 = max
Operator 8 = sum
Operator 9 = int

Operator 10 = isosurface
Isosurface Variable 10 = String Coordinate 3
Isosurface Value 10 = Real 0.5

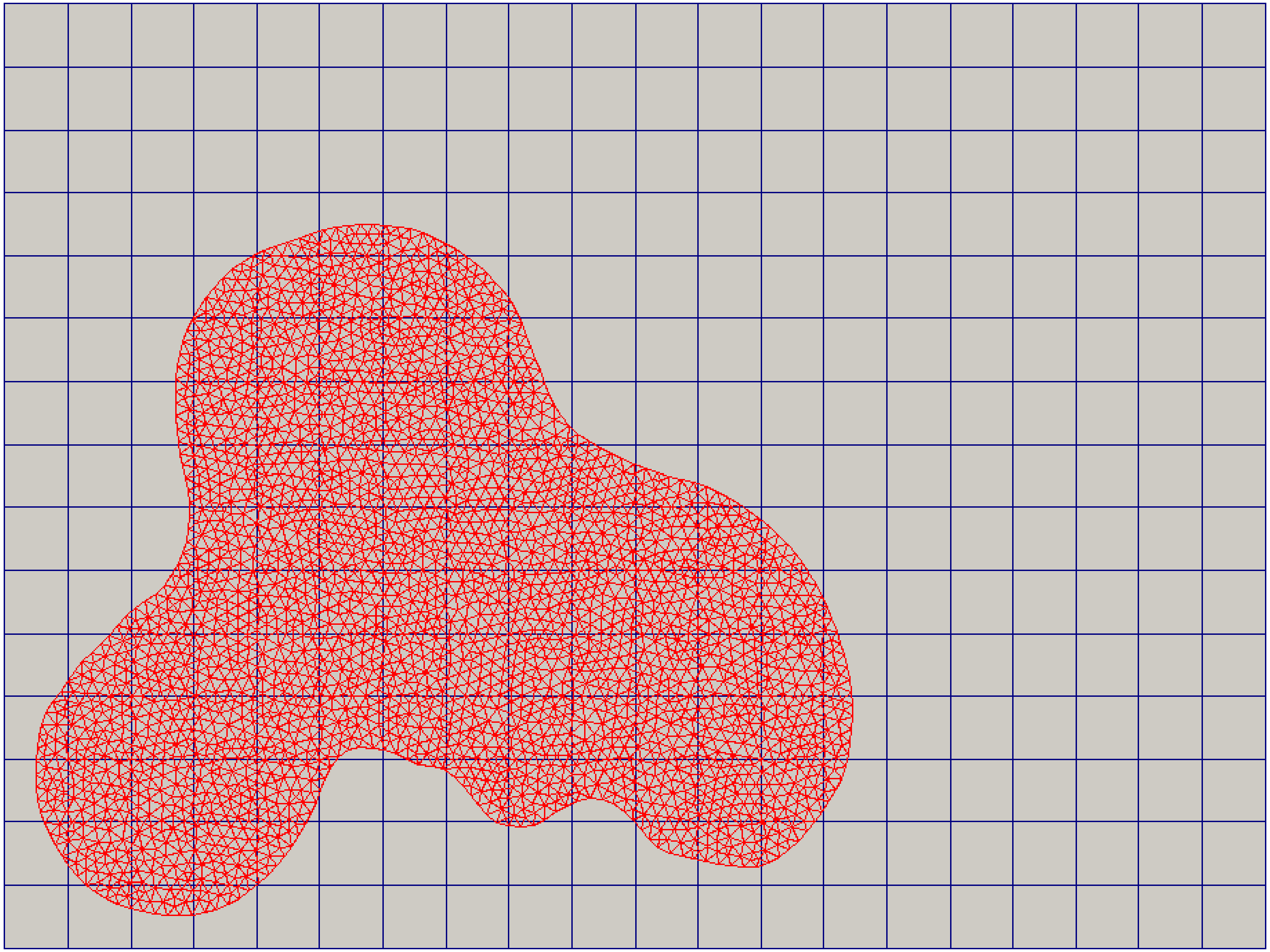
Operator 11 = Layer Below Top
Layer Index 11 = Integer 3

Operator 12 = Layer Above Bottom
Layer Index 12 = Integer 3

GridDataReader - Getting the real data in

- Typically elevations, temperature forcing etc. data is available in a uniform (x,y) mesh in NetCDF files
  - You may utilize GridDataReader to read this data
- For documentation and explanation of keywords see Ch. 57 in Elmer Models Manual
- Most important features
  - Reading multiple fields
  - Scaling, constant offsets, linear combination of fields
  - Steady state & transient operation
  - Linear interpolation for time and space
  - Applicable to 2D and 3D cases

GridDataReader – Problem illustration



GridDataReader

- Define the grid parameters of the NetCDF file
  - h_x, h_y, x_0, y_0...
- Go through each node (x,y) in the FE mesh
  - For each node the correct cell is found easily
    i=floor$((x-x_0)/h_x)$ and j=floor$((y-y_0)/h_y)$
  - Field value then interpolated using bilinear interpolation
    $f(x,y)=pq\, f(i,j)+p(1-q)\, f(i,j+1)+$
    $(1-p)q\, f(i+1,j)+(1-p)(1-q)\, f(i+1,j+1)$
- Complexity O(N) & small memory consumption
  - only one cell read at a time
- If grid is not uniform finding of the correct cell is not as easy
  - Hack by Rupert, uses more memory & CPU-time

Summary

- If you can utilize extruded meshes do so
- Internal mesh extrusion
  - Removes efficiently many meshing bottle-necks
- Most importat solvers utilizing extruded structures
  - StructuredMeshMapper
  - StructuredProjectToPlane
- Mesh mapping typically requires data
  - GridDataReader for NetCDF input
  - Internally solved field
- Complexity of all operations is almost O(N)
  - Optimal scalability for larger problems