Hanging nodes in FSDM

Margrit Klitz & Daniel Vollmer
Simulation and Software Technology, Cologne
Institute for Aerodynamics and Flow Technology, Brunswick

DLR

Content

1. Where hanging nodes come from and why we like them

2. Adaptions in FSDM

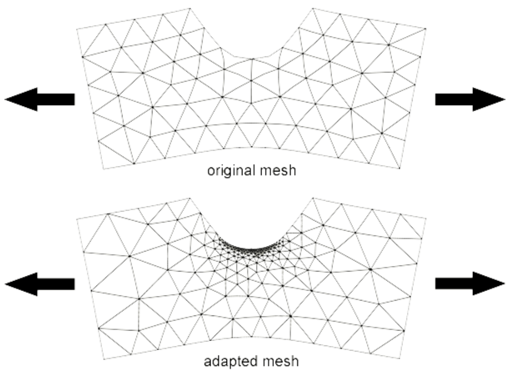
3. Testing in FSDM

4. Open questions to discuss in this group

Motiviation: Hanging Nodes

- DLR goal: virtual design of an aircraft.
  - Flight characteristics determined by numerical simulation
  - Key element: numerical flow simulation
  - → CFD software Flucs
- Complex 3d transient flows
  - highly time-consuming
  - → use mesh adaptivity
- → Creation of hanging nodes along non-conforming interfaces

- Multidisciplinary optimization of a transport aircraft configuration.



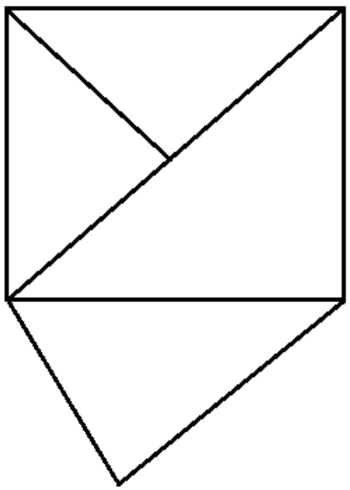
Why we like them

Normally:

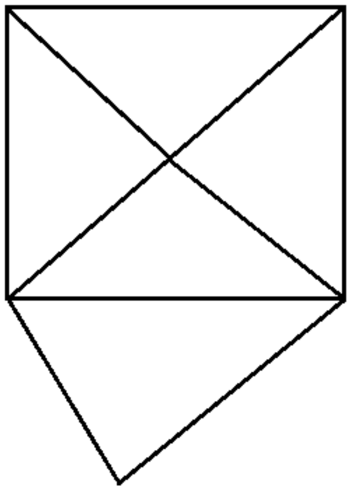
- Disturb continuity of finite element space
- Much effort required to remove hanging nodes

However:

- Hanging nodes allow for very flexible grid structures and adaptivity
- Discontinuous Galerkin methods & Finite-Volume methods: very general non-matching grids containing hanging nodes allowed
- Go well with the next Generation flow solver Flucs in the DLR



A non-conforming mesh with 1 hanging node

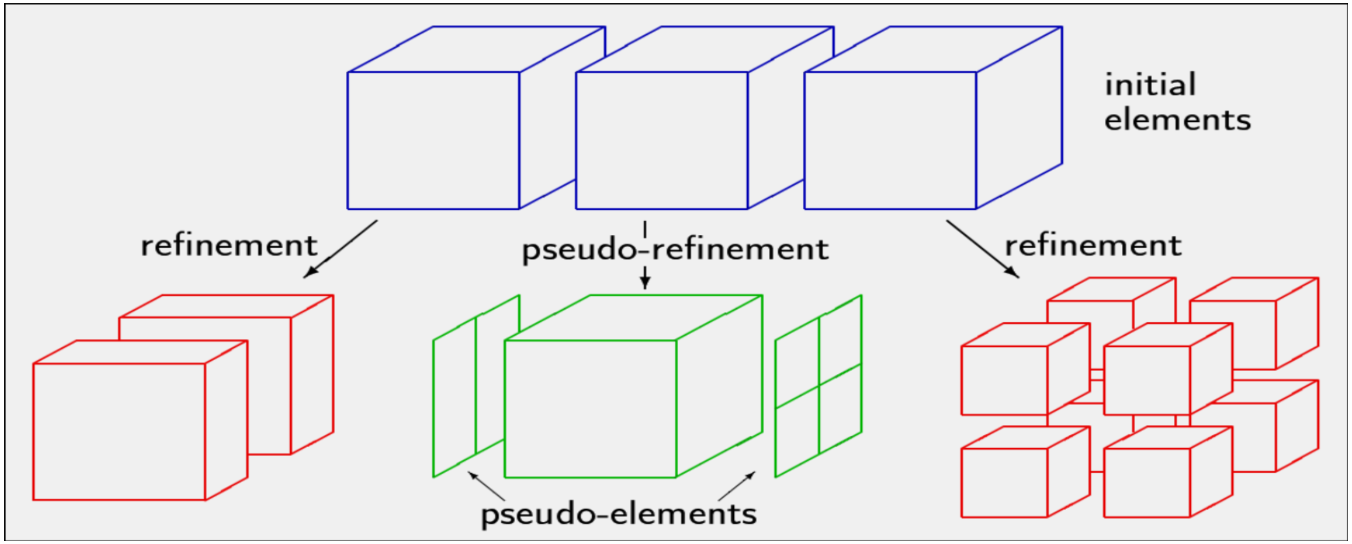


A conforming mesh

Note: One of the main use cases for is a new mesh adaptation that is currently being developed in DLR Project VicTtoria.

→ So far not accounted for in FSDM

Main Idea: Hanging Nodes via Pseudo Elements



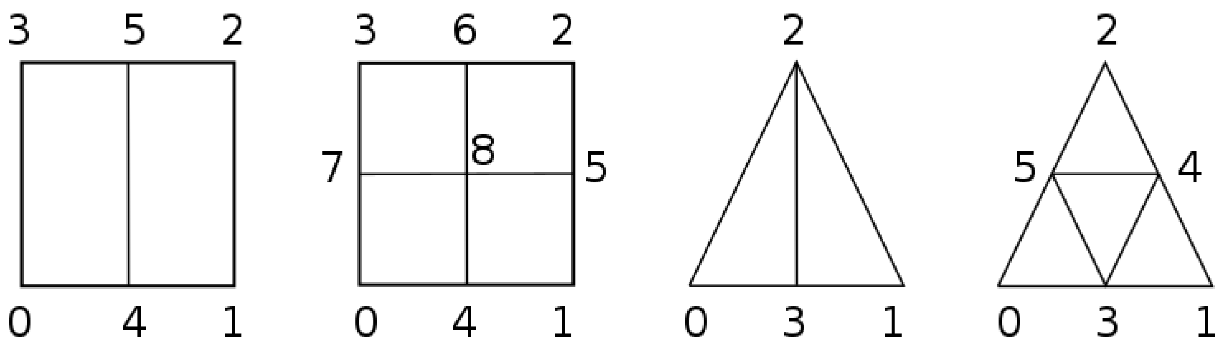
- In-between the real elements with hanging faces
- “Pseudo-conform” grid
- Pseudo elements can be treated like normal elements to a certain extent

Implemented types of pseudo elements

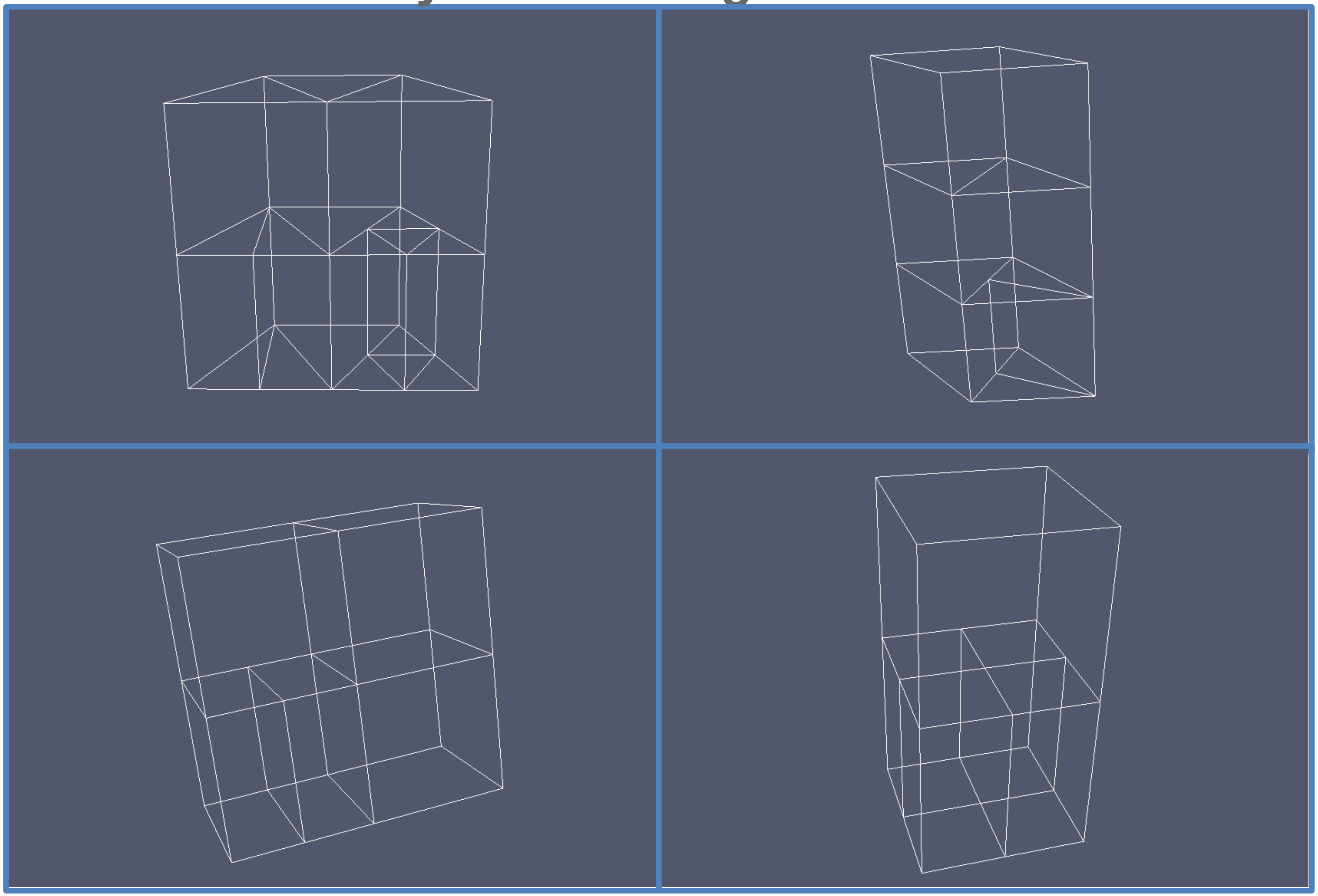
1. PCT_Quad2Quad with 3 faces
2. PCT_Quad4Quad with 5 faces
3. PCT_Tri2Tri with 3 faces
4. PCT_Tri4Tri with 5 faces
5. PCT_Node1Node: 1d-element that is equivalent to an edge and simply connects 2 nodes (Kolja’s “virtual edge”).

→ Belong to the unstructured cell types in FSDM, but neither to the volume nor the surface cell types

→ Definition of cell types and test cases in FSDM (Verena Muckhoff)



Addition of many small test grids…



Face Extraction Algorithm

- Flucs requires the connectivity information and the node-coordinates of the faces of the mesh
- `class FSMeshFaceExtractor:` Extracts and matches all unstructured faces and writes them in a list → `GetFaceConnectivity()`
- Pseudo elements are handled in a natural way by the existing face extraction algorithm

Adaptions

- The tricky part is the step that removes the pseudo elements from this list → `PrepareFaceConnectivity()`
- At process borders, additional communication is required (parts of the cells connected to pseudo faces may be distributed among the processes)
- Also holds for higher order cells in FSDM: only corners are relevant

Additions (moved from Flucs to FSDM → `GetFaceNodeCoordinates()`)

- Computation of the pre-defined node ordering for all face types
- Computing the node coordinates

Interface

- Prepare calls to faces of the mesh
  `fex = FSMeshFaceExtractor()`

- Don't match remote faces*, keep pseudo cells
  `fex.PrepareFaceConnectivity(mesh, False, True)`
- Or match remote faces, keep pseudo cells
  `fex.PrepareFaceConnectivity(mesh, True, True)`
- Default: match remote faces, remove pseudo-cells
  `fex.PrepareFaceConnectivity(mesh)`

- After: if desired we can also ask for the coordinates of the faces
  `fex.PrepareFaceNodeCoordinates(FSQuantityDescArray())`

- Wrapped to Python primarily for testing: `example/Advanced/howToInitPseudoCellMeshAndExtractFaces.py`

* whether to communicate and fill unmatched faces between processes.

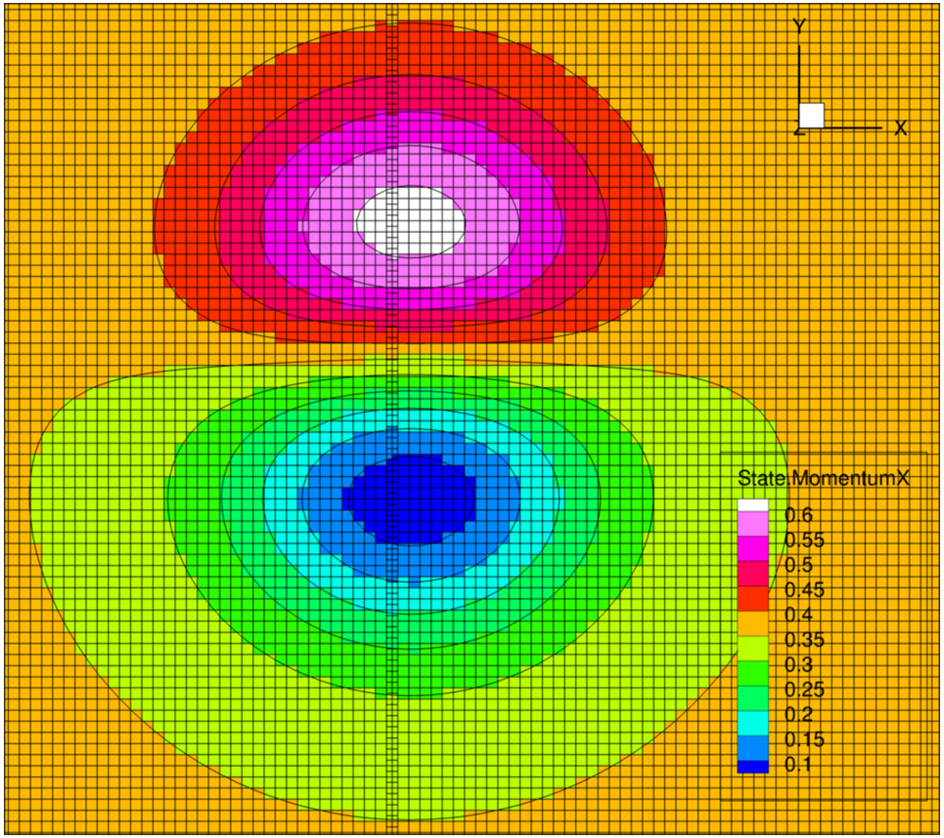
A Numerical Example

- Demonstration for a simple convection problem: Kok-vortex transport
- Mesh contains regular hexahedra, one column of hexahedra was refined with hanging nodes in a 1:2 fashion
- Mesh generation by FSDM Python script
- Results are nearly identical for both Finite-Volume and Discontinuous-Galerkin discretization in Flucs.
- Minuscule differences probably due to the temporarily higher resolution while convecting across the refined region

Instantaneous snapshot of the simulation

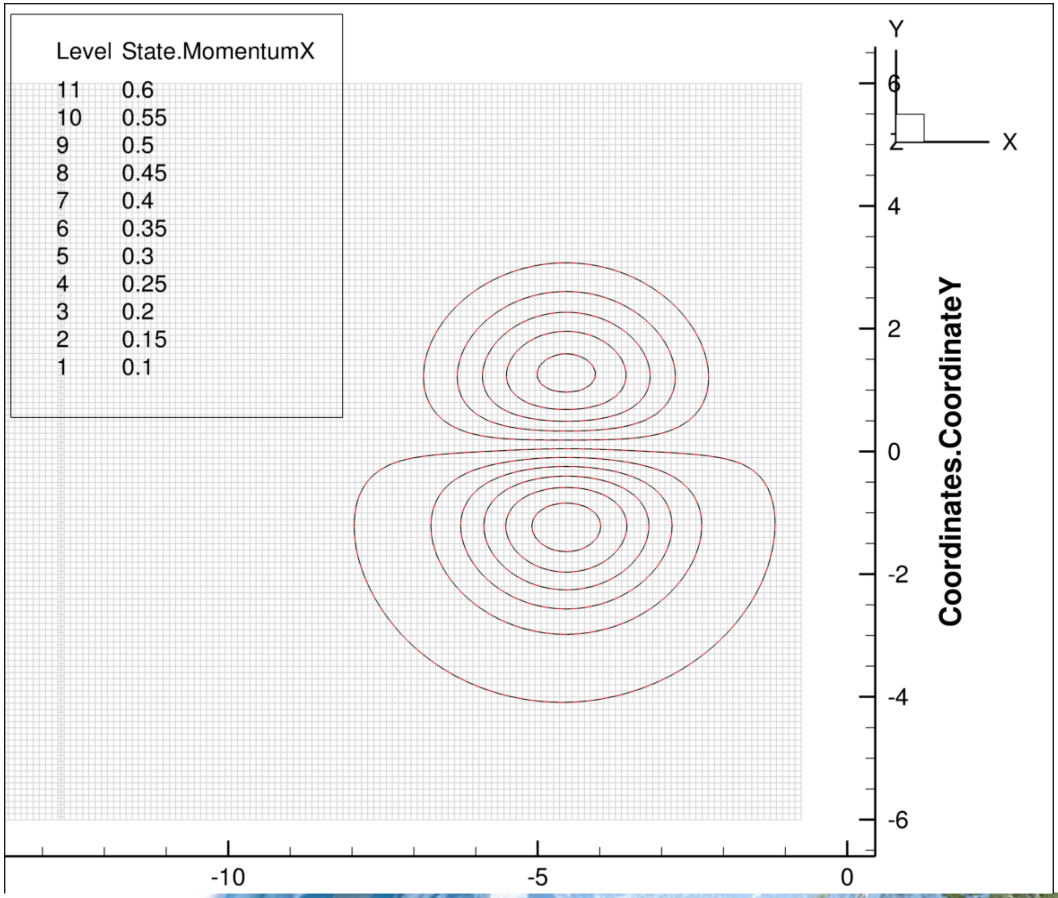
- 2nd order Finite-Volume discretization of the Euler equations
- Mach=0.3, time-step = 0.025 using RK4 time-integration
- Vortex is convecting from left to right

- Contour *lines* show the x-component of the momentum on the regular mesh while the *flood colors* show the x-momentum on the mesh with hanging nodes.
- Other quantities (and discretizations) are similarly accurate.

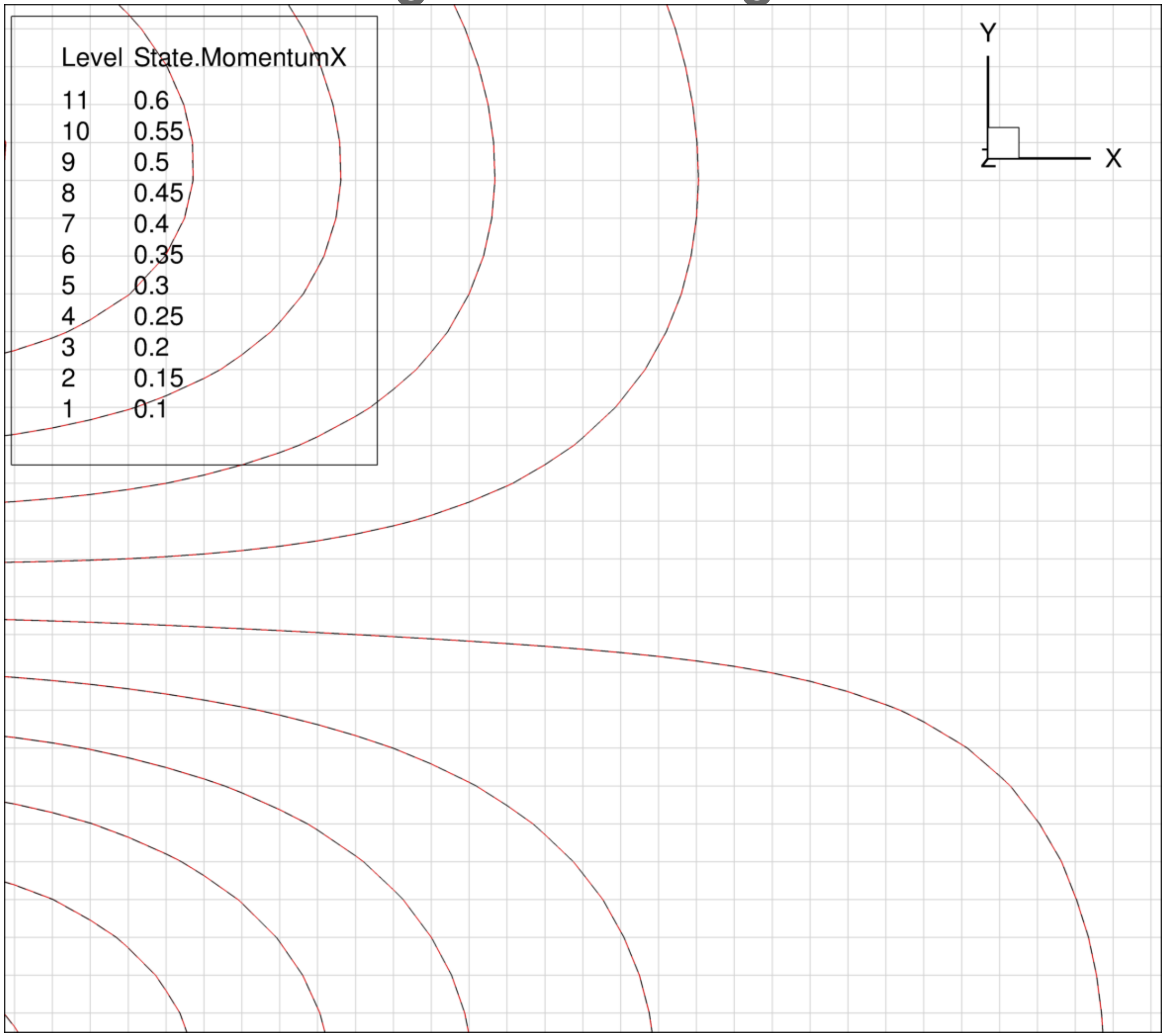


After vortex convection

- Result after vortex convection across the mesh
- The black line represents the x-momentum on the regular mesh
- Dashed red lines shows the x-momentum on the mesh with hanging nodes



Zoom in on the right-hand edge of the vortex



Testing (with Hanging Nodes)

- **Multiple** different ways that FSDM functionality is tested.
- Python scripts, tests in Flucs itself and a Google Test framework for FSDM (which is so far only used by DLR-SC)
- Google test framework: large number of small test meshes containing all types of pseudo cell faces were created, continuous testing with new face extractor
- Meshes also exist as HDF5-files to be used by Python scripts (test the correct ordering of the nodes)
- Since some functionality was moved from Flucs to FSDM, also tests there

Testing (all in all)

- **Multiple** different ways that FSDM functionality is tested.
- Python scripts
- C++-tests hard-coded in FSDM
- Google test framework (used by SC and by Sogeti)

→ Maybe find a unified way to do this?

DLR

Open Questions

- So far the hanging nodes are in a separate branch
- In this branch the `#ifdefs` for higher-order cells are removed
- Can we merge into the trunk?
- Can we find a common ground for testing FSDM functionality?