THE POWER OF MODULAR TREE-BASED AMR

RESOLVING HANGING NODES AND CUTTING HOLES

Johannes Holke, SIAM CSE23 02.03.2023
DLR Institute for Software Technology (SC)
High-performance Computing | Scalable adaptive mesh refinement (AMR)

Knapp, David; Dreyer, Lukas; Elsweijer, Sandro; Ünlue, Veli; Burstedde, Carsten; Markert, Johannes; **Lilikakis, Ioannis**; Boeing, Niklas; **Becker, Florian**; Gassner, Gregor

DLR

Disclaimer

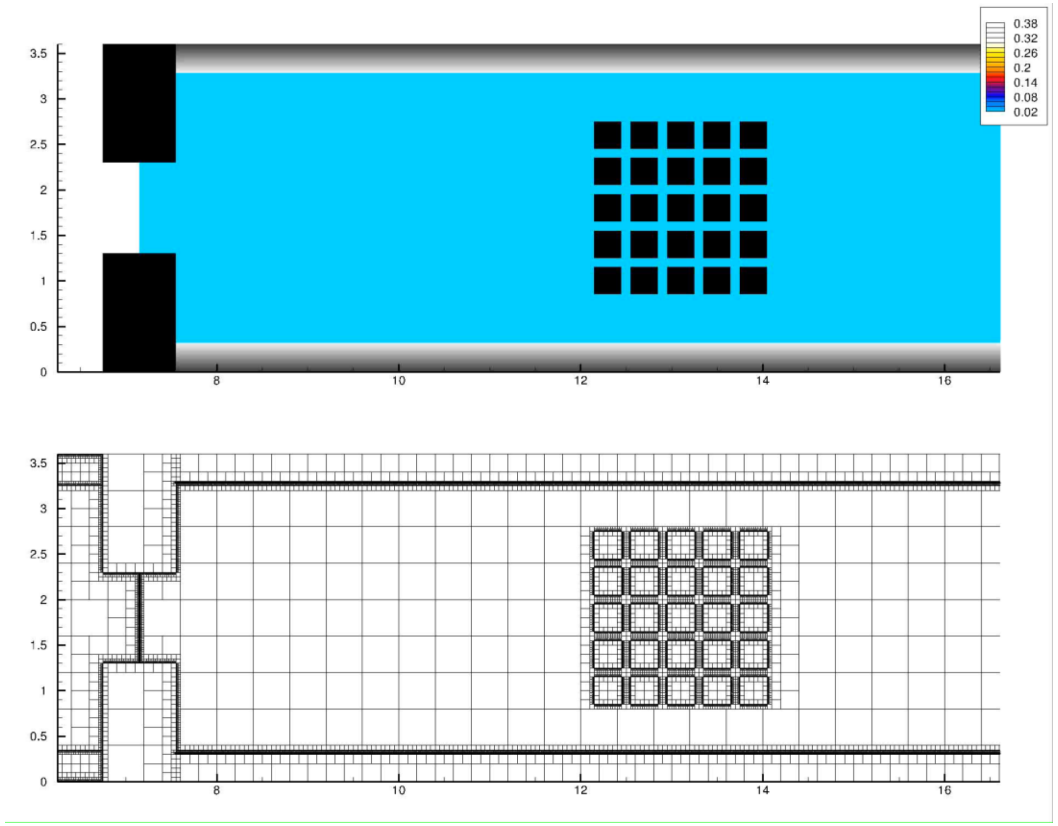
This talk has

99% mesh handling

1% PDEs

Modular tree-based AMR

We have seen a lot of AMR so far,
much was tree-based
using space-filling curves:



Caviedes-Voullieme, Gerhard, Sikstel, Müller

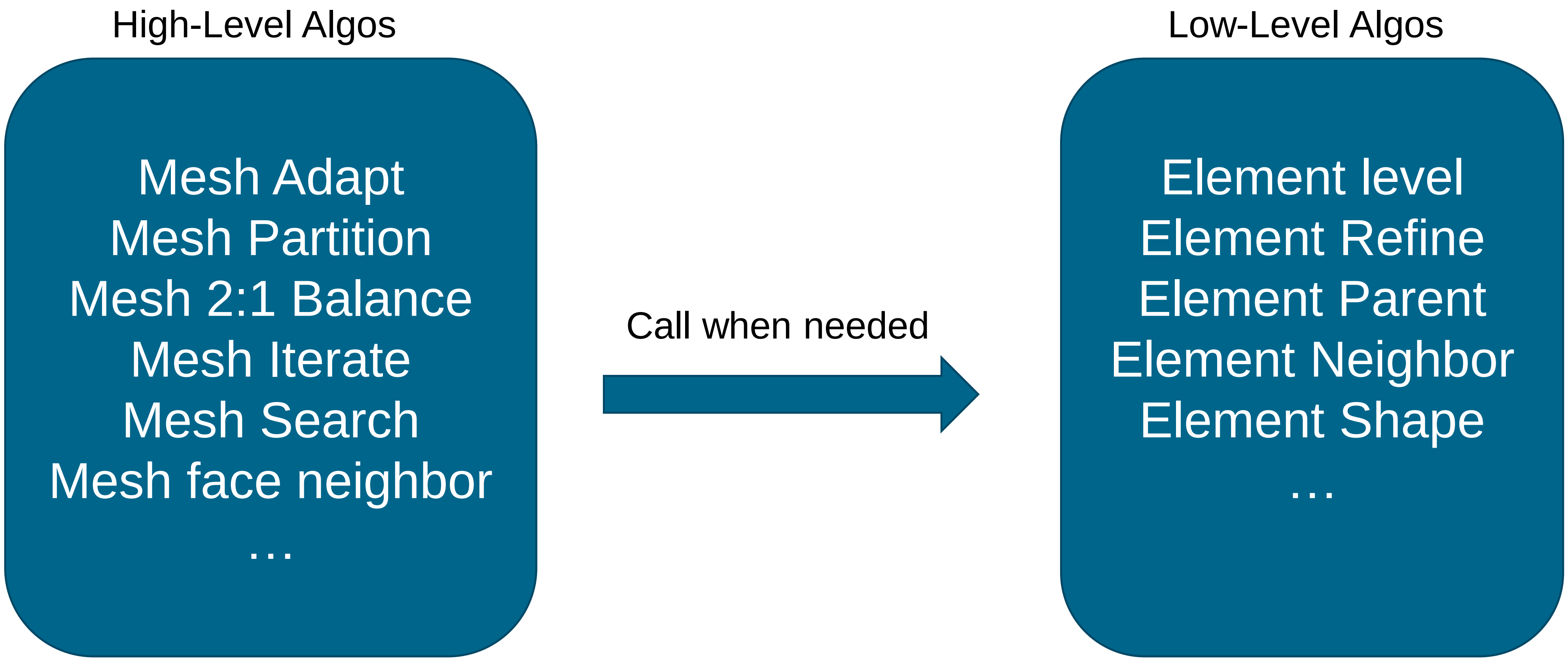
- Memory efficient
- Fast
- p4est standard: All AMR algorithms in <1 Second

Modular tree-based AMR

Historically these were limited to quads/cubes (with some notable exceptions)

We extend tree-based AMR to all* element shapes.

Modular tree-based AMR



Implement these once

Implement these for each
- Shape (tri, tet, quad, hex, prism, …)
- Refinement pattern/SFC (Morton, Peano, …)

Modular tree-based AMR

Example: refining the mesh.

Instead of:

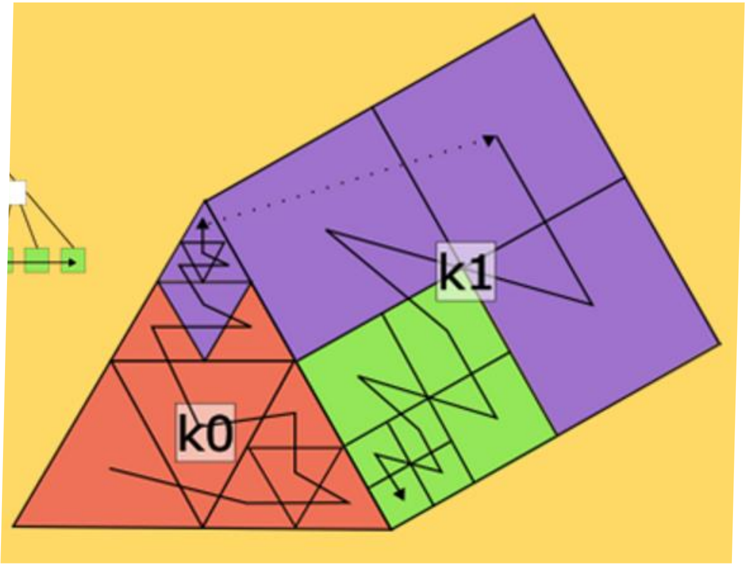
```
if (refine (quad)) {
    Allocate (new_quads, 4);
    // Fill with children
}
```

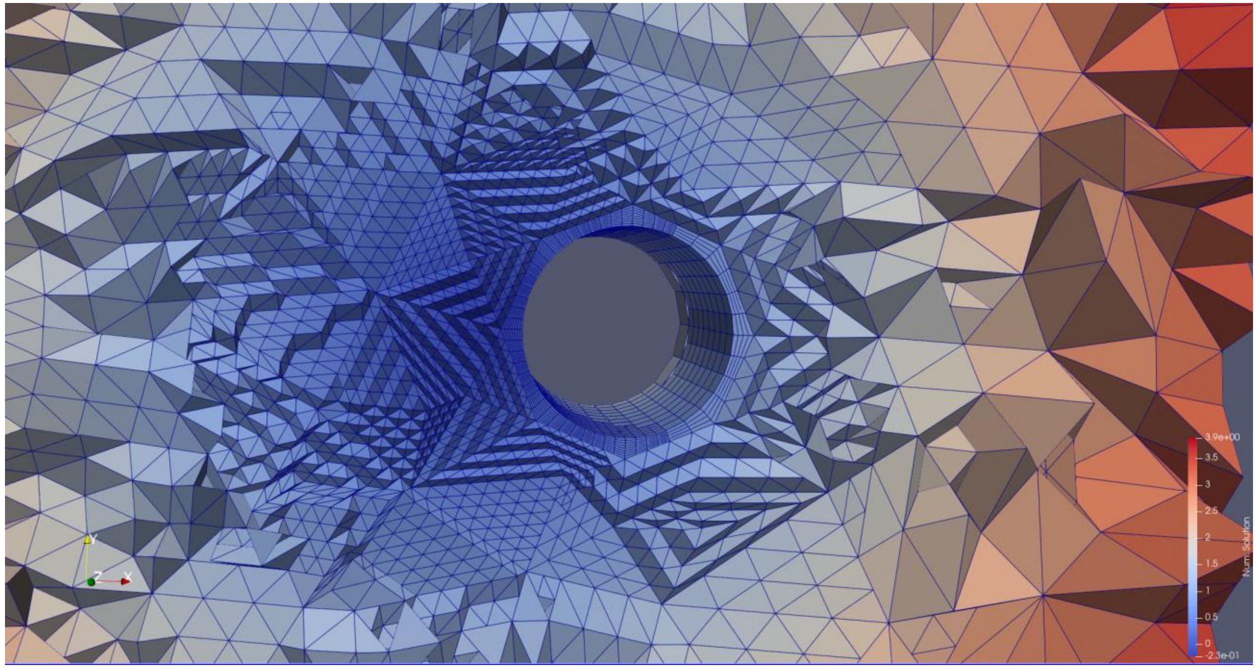
We do:

```
if (refine (element)) {
    int num_children = element->num_children();
    Allocate (new_elements, num_children);
    // Fill with children
}
```

Modular tree-based AMR

Thus, we can take the same algorithms, and operate on any element shape and also mix element shapes in the same mesh.





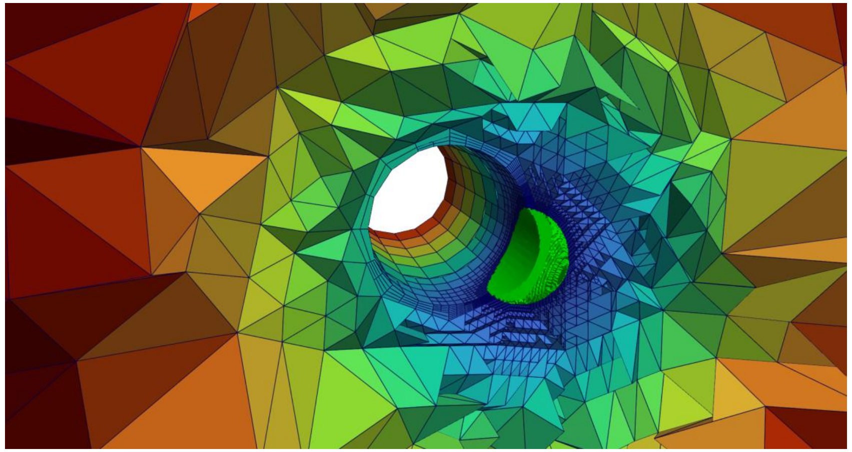
All with the performance and scalability of tree-based AMR.

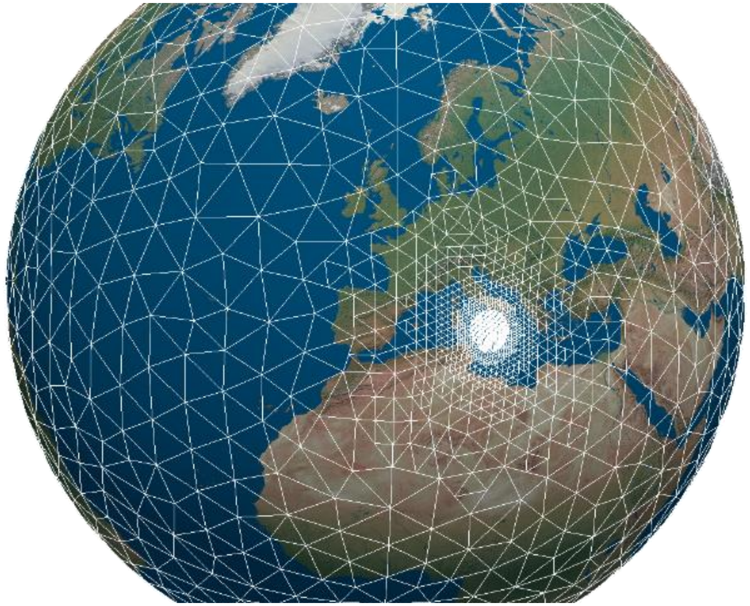
t8code („tetcode“)

DLR-AMR / t8code Public

DLR

- Parallel management of **adaptive meshes** and **data**
- C/C++ and MPI
- Tree-based/semi-structured with **space-filling curves**
- Vertex, Line, Quad, Tri, Hex, Tet, Prism, Pyramid
- Modularly extandable
- Scales up to **1 mio. MPI** ranks (with >90% efficiency),
- >**1 Trillion** elements
- Complex geometries (comparable to unstructured meshes)
- Curved meshes





And now, some cool stuff

We were forced to make the high-level algorithms more flexible and robust (changing number of children, changing shape of elements, etc.).

This allows us now to implement „non-standard“ features.

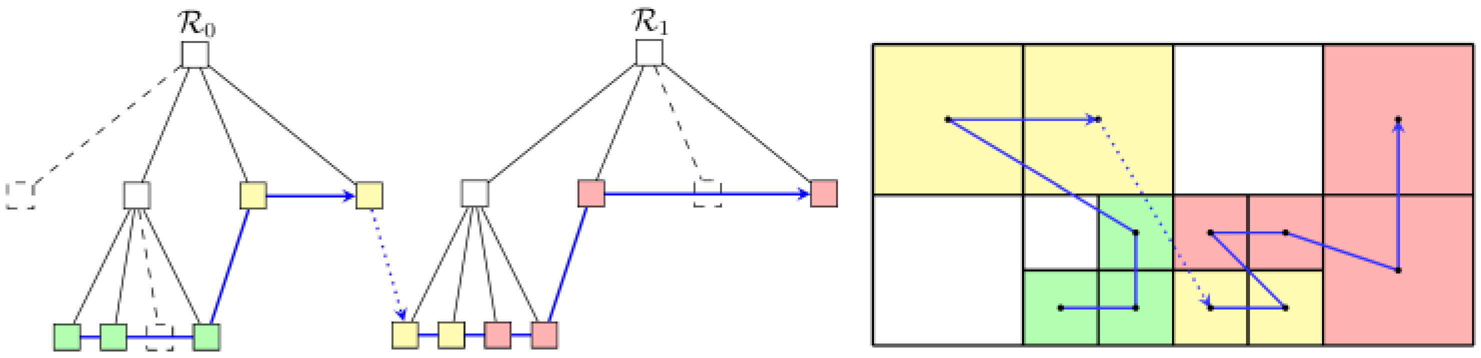
Cutting holes

- Embedding obstacles in the mesh
- Rectangular domain with single tree
- Coarsening arbitrary data (for visualizing or compressing)

Basically we are doing:

```
if (refine (element) == -2) {
    int num_children = 0;
    Allocate (new_elements, num_children);
}
```

Cutting holes



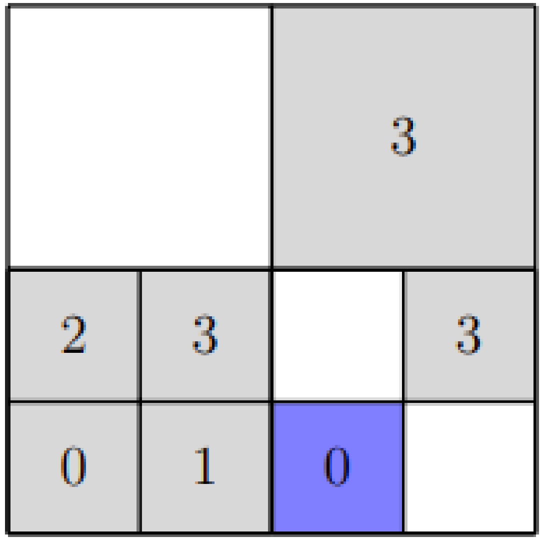
(a) (b)

No „virtual elements“ of weight 0 or similar constructs.

No memory needed for unused elements.

Cutting holes

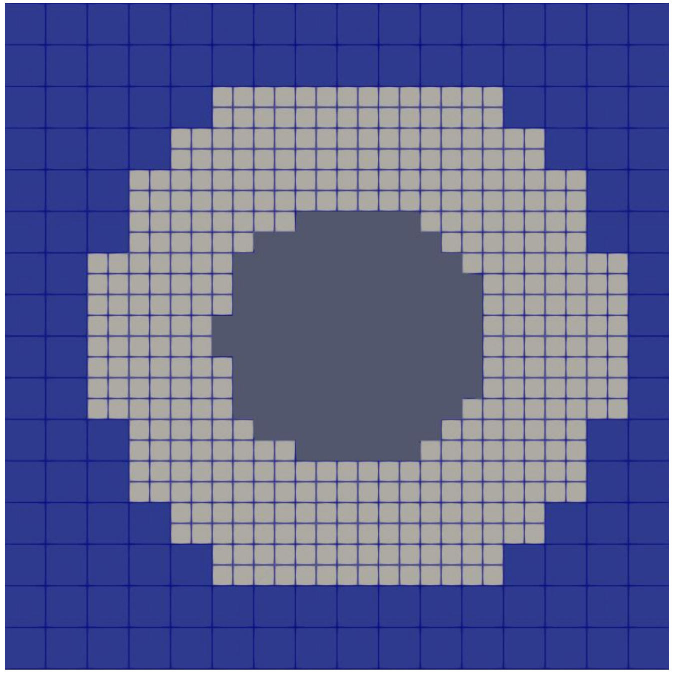
Challenge: How to coarsen a mesh with holes?

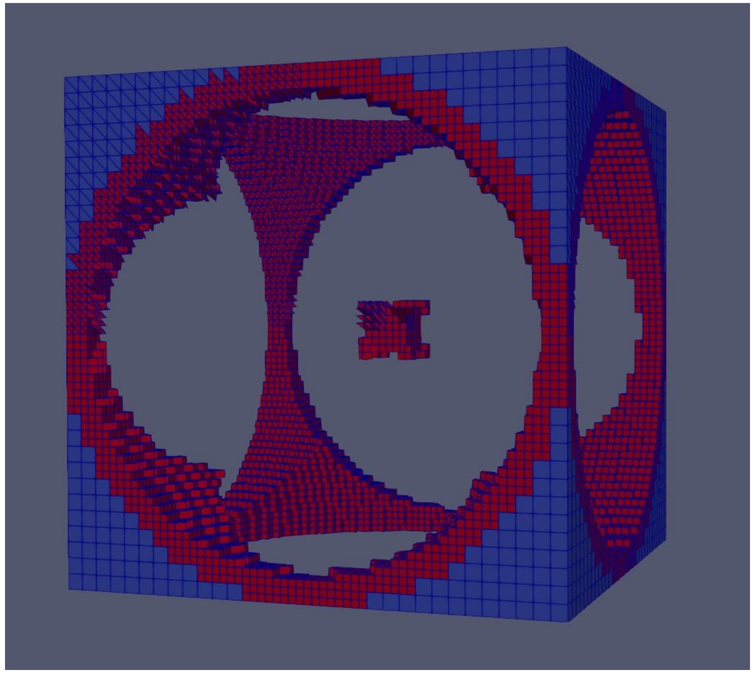


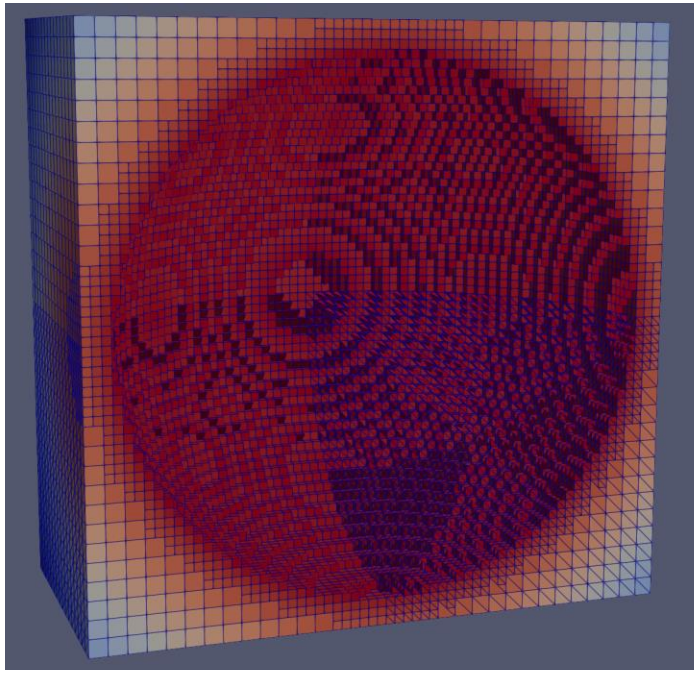
New is_incomplete_family Check

Cutting holes

- The mesh with holes is just a normal AMR mesh now
- Can refine/coarsen/load-balance it etc.
- No need to: fill the holes, coarsen, redo the holes



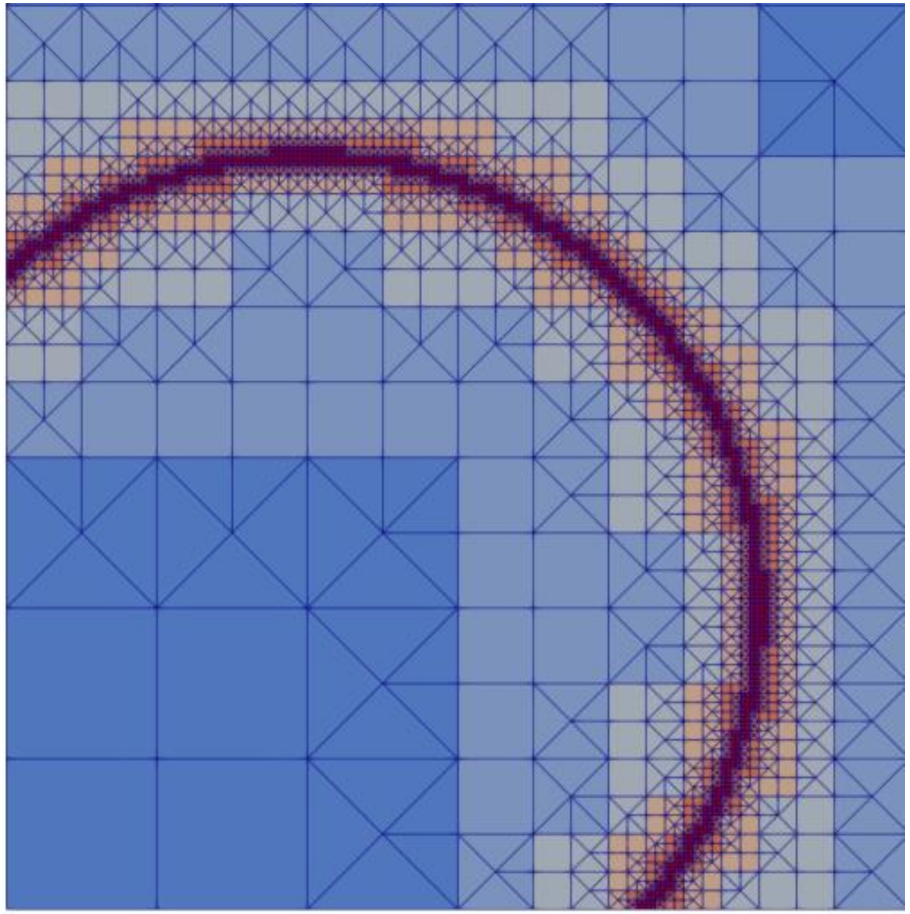




Multi tree, hybrid mesh

Even cooler stuff - subelements

- One application of subelements is resolving hanging nodes:

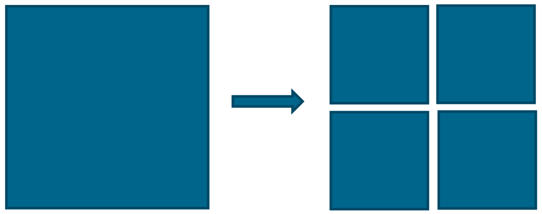


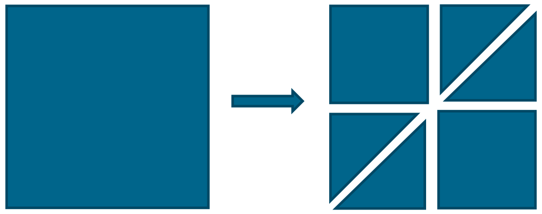
This is a **tree-based mesh with a space-filling curve**.
We see **one single tree**.

Subelements

With standard elements

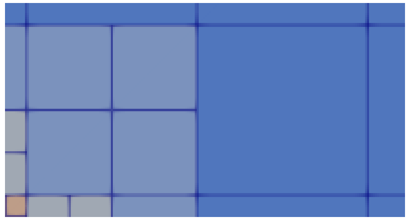
- We could do different refinement patterns, but…

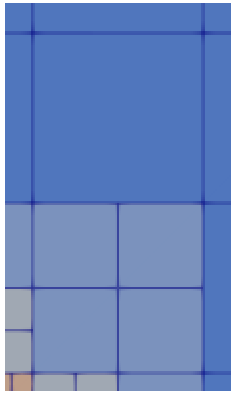




- We cannot change behavior at will

„A level X element with Index Y allways has to refine the same way“

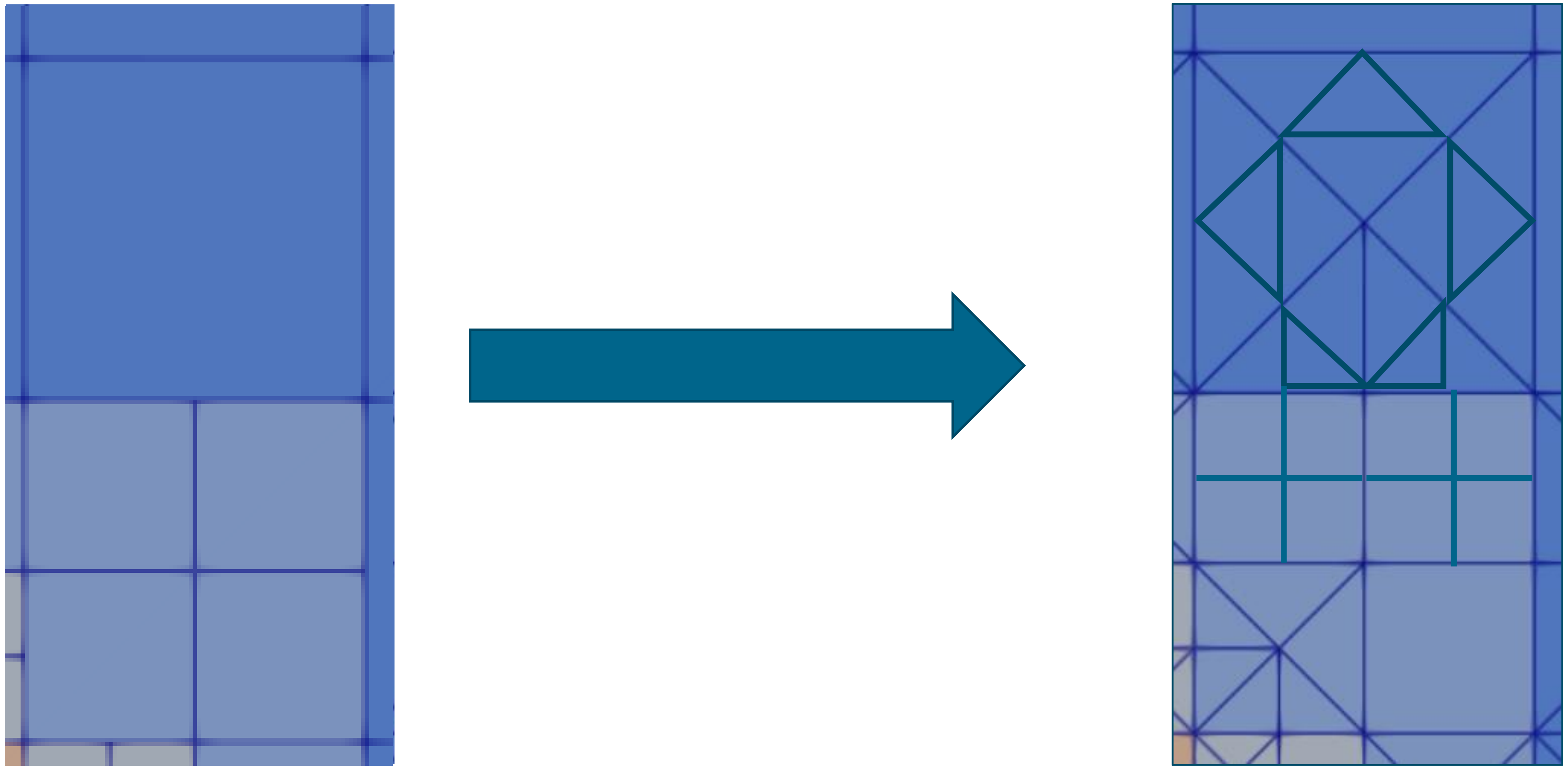




Subelements

With standard elements

- We must continue refinement

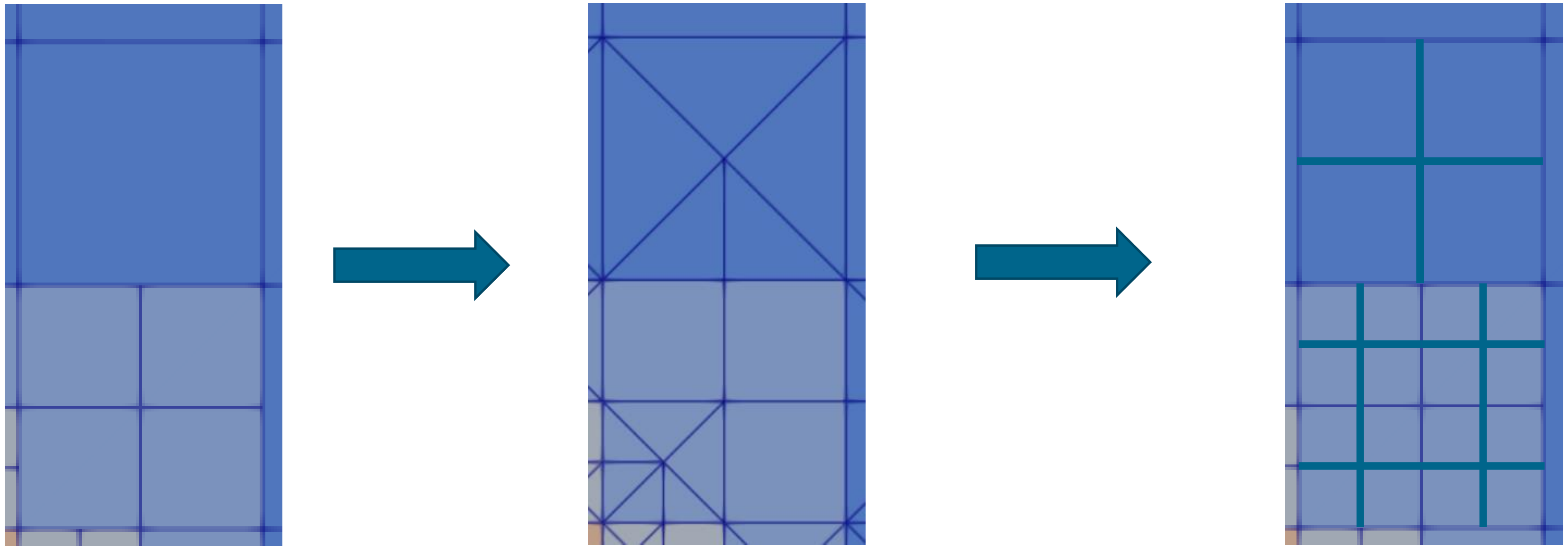


Subelements

Idea of Subelements:

- For **one level** you can do **whatever you want**

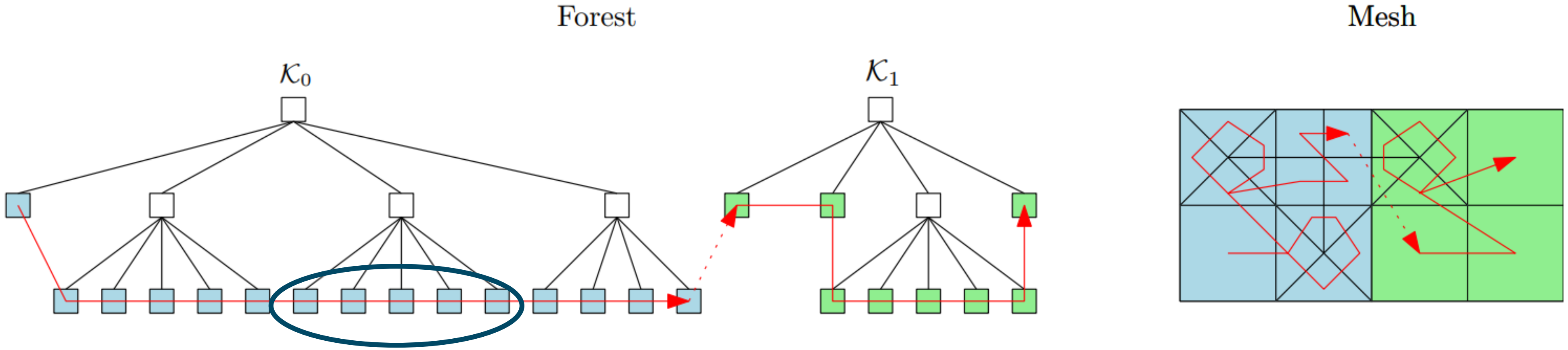
- Before you refine, remove subelements



Subelements

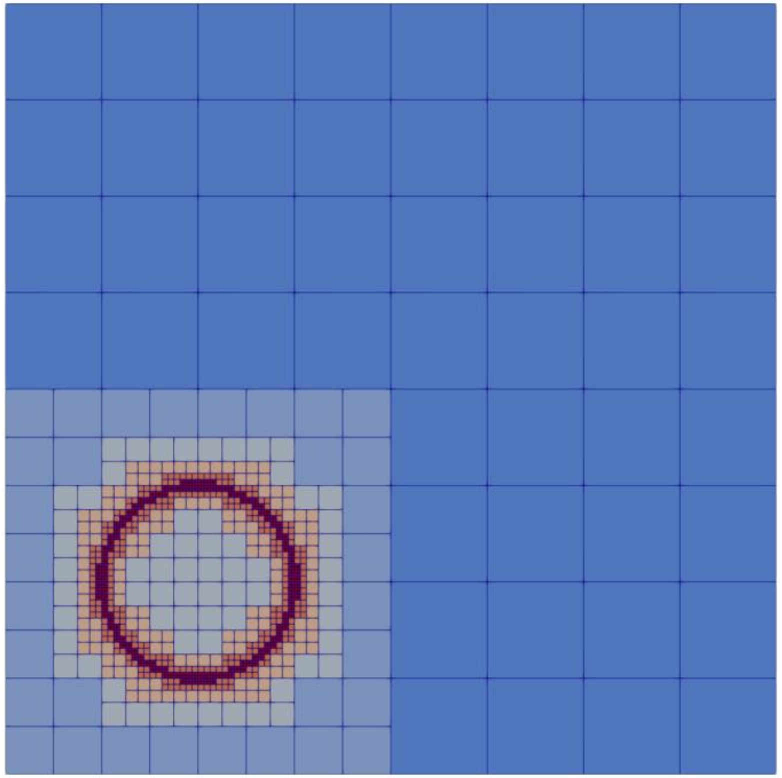
- Subelements have same SFC index as their „parent“ element plus an additional subelement ID

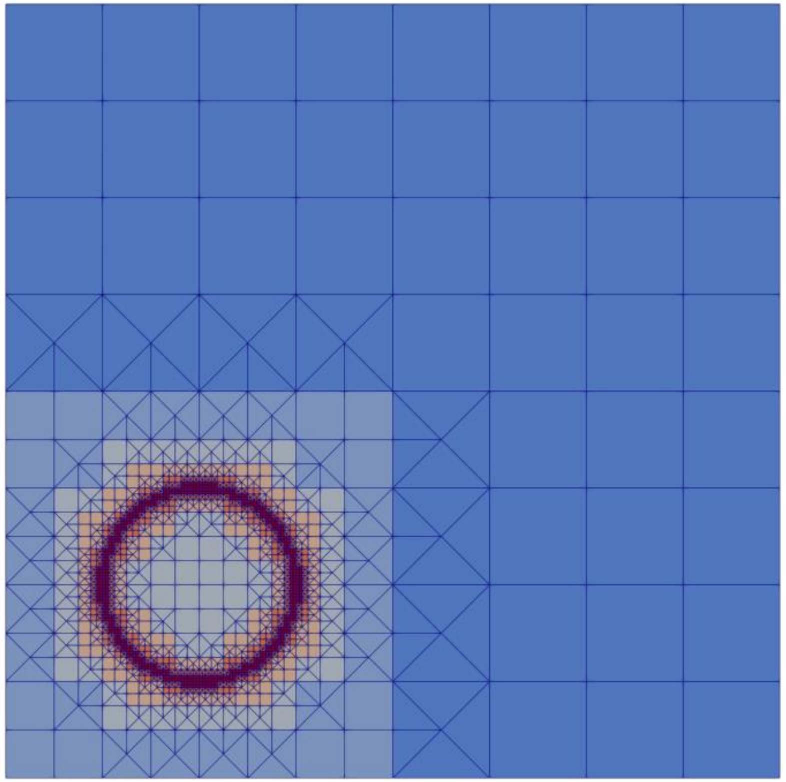
- Subelements look like elements to the outer world
  - They implement a subset of low-level algorithms
  - Iteration, ghost elements, etc.



Subelements – Resolve hanging nodes

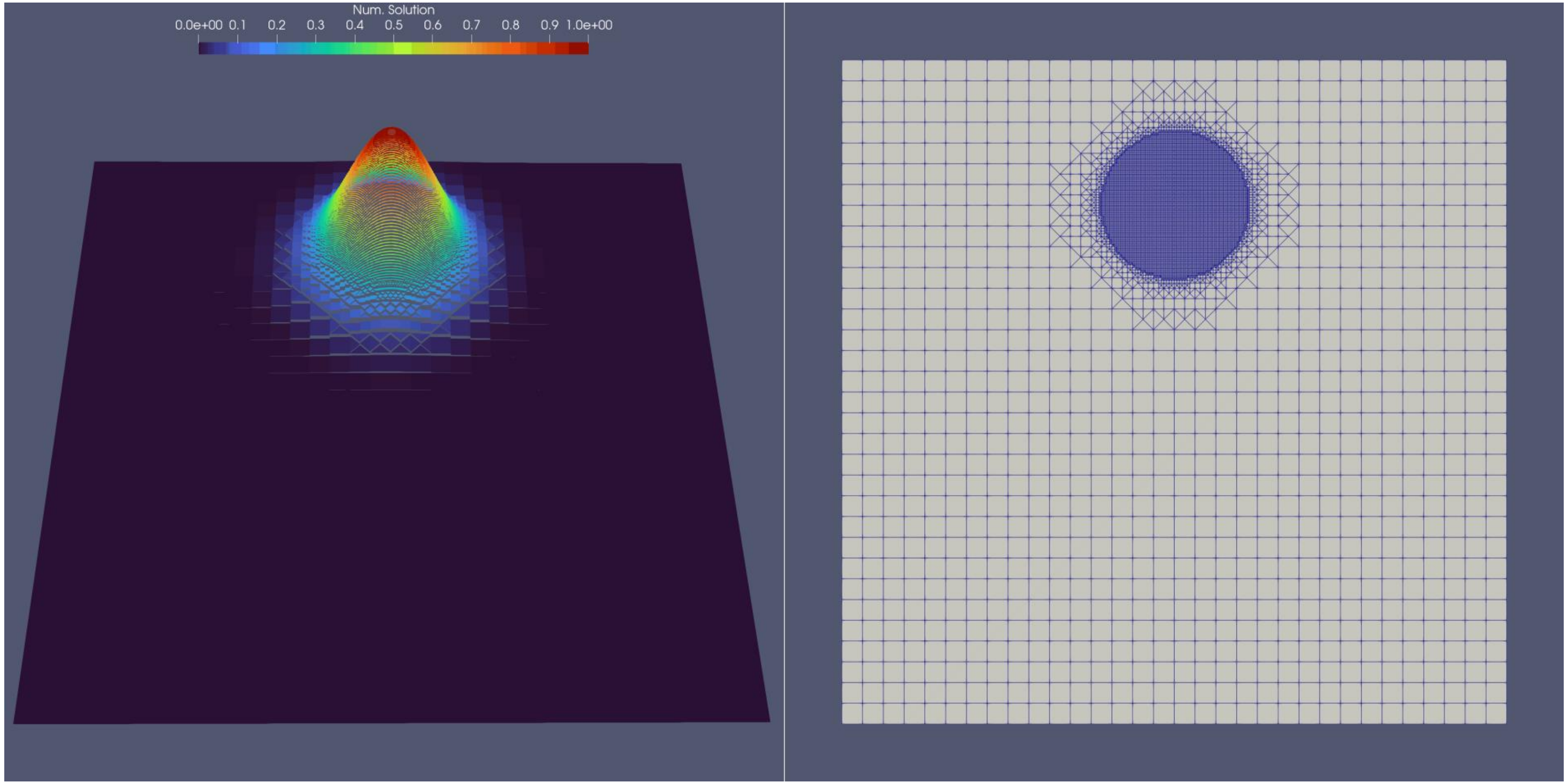
- 2:1 balance your mesh
- For each element with a hanging face use one of 15 subelement patterns:





Subelements

We implemented full hanging node resolution for 2D quads with it:



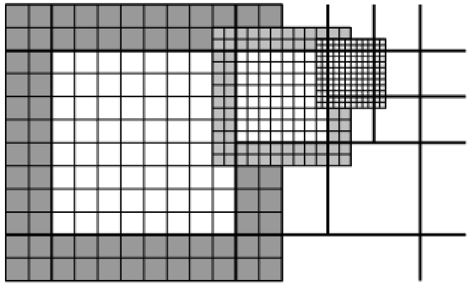
3D hexes and other element shapes currently work in progress

Subelements – What next?

Your imagination is the limit!

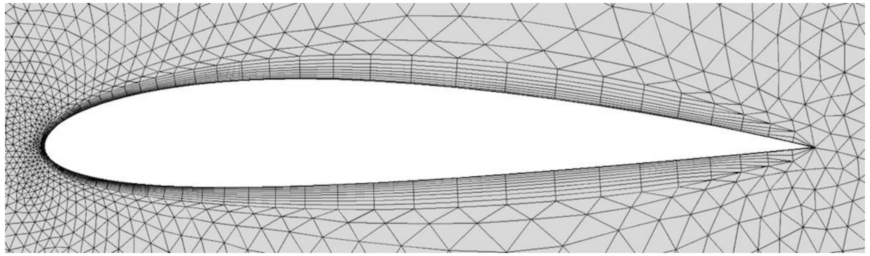
- Anisotropic refinement

- Uniform subgrids for GPUs

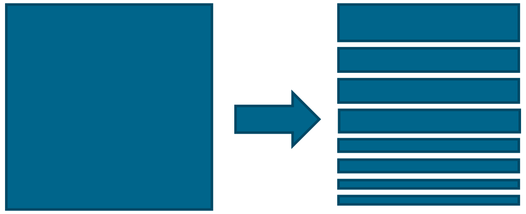


Donna Calhoun et. Al.

- Boundary layers



https://www.comsol.fr/blogs/your-guide-to-meshing-techniques-for-efficient-cfd-modeling/



- Your ideas?

t8code – www.github.com/dlr-amr/t8code
Holke, Johannes, Burstedde, Carsten, Knapp, David, Dreyer, Lukas, Elsweijer, Sandro, Uenlue, Veli, Markert, Johannes, Lilikakis, Ioannis, Boeing, Niklas, & Becker, Florian. (2023). t8code (v1.1.0). Zenodo. https://doi.org/10.5281/zenodo.7681843

Becker, Florian (2021) ***Removing hanging faces from tree-based adaptive meshes for numerical simulations***. Master‘s Thesis, Universität zu Köln.

Lilikakis, Ioannis (2022) ***Algorithms for tree-based adaptive meshes with incomplete trees.*** Master‘s Thesis, Universität zu Köln.

More on t8code at IMR23!