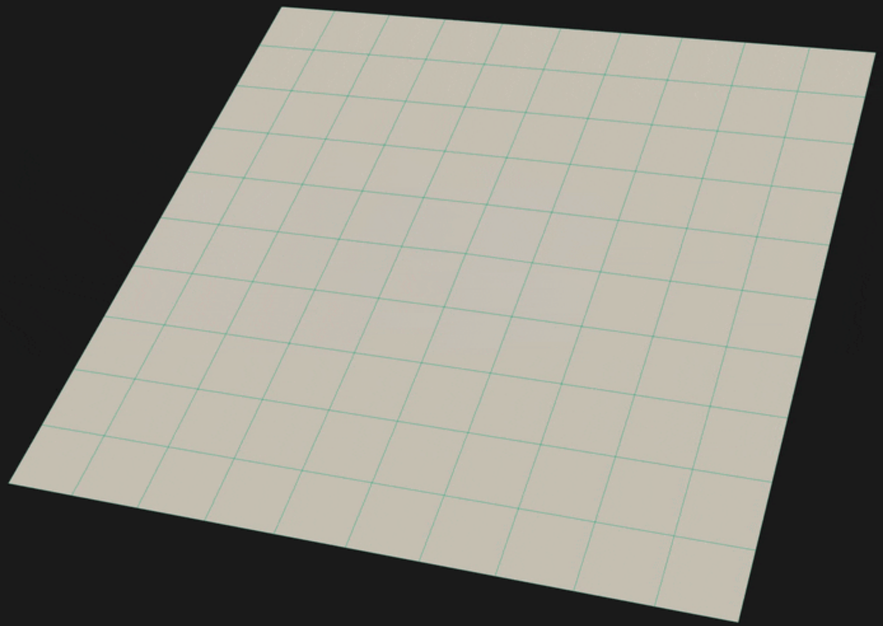
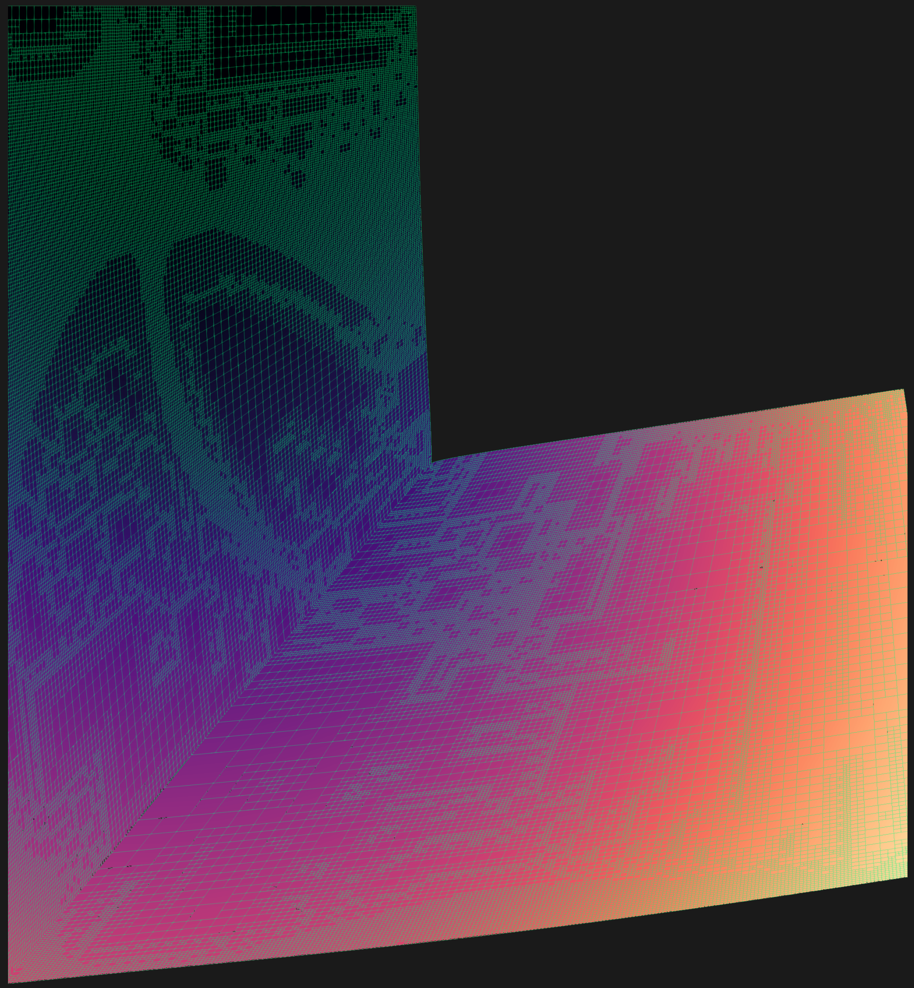
P4est Datastructures for Ferrite.jl





A Short History of Frustration

Commits on Aug 2, 2021

add lookup tables and helper functions
koehlerson committed on Aug 2, 2021

clean up and tests for lookup
koehlerson committed on Aug 2, 2021

Commits on Aug 14, 2021

corner neighborhood for 2D
koehlerson committed on Aug 14, 2021

face neighbor for 2D
koehlerson committed on Aug 14, 2021

topology for 3D
koehlerson committed on Aug 14, 2021

Commits on Oct 1, 2022

Merge branch 'master' into mk/adaptivehex
koehlerson committed on Oct 1, 2022

cleanup topology from the past
koehlerson committed on Oct 1, 2022

manufactured solution 2d adaptivity done
koehlerson committed on Aug 8

What is P4est?

- a library/concept that describes a forest with linear octrees
- each octree introduces a new octree coordinate system with an **a priori fixed size** 2^b, where b is the maximum level

```
+--+--+------+
|  |  |      |
+--+--+      |
|  |  |      |
+--+--+------+
|     |      |
|     |      |
|     |      |
+-----+------+
```

```
        --■--
    ---/-/ \-\---
 --/   /     \   \--
■     ■      /■\    ■
            /| |\
          /- / \ -\
         /  |   |  \
        /   |   |   \
       ■    ■   ■    ■
```

Octant and Octree

```julia
struct OctantBWG{dim, N, M, T} <: AbstractCell{dim,N,M}
    #Refinement level
    l::T
    #x,y,z \in {0,...,2^b} where (0 ≤ l ≤ b)}
    xyz::NTuple{dim,T}
end
```

Leave array realized as `Vector`, storing physical corner nodes of root octant in `nodes`

```
   +---------+
  /         /|
 /         / |
+---------+  |
|         |  |
|         |  +
|         | /
|xyz      |/
⊗---------+
```

```julia
struct OctreeBWG{dim,N,M,T} <: AbstractAdaptiveCell{RefHypercube{dim}}
    leaves::Vector{OctantBWG{dim,N,M,T}}
    #maximum refinement level
    b::T
    nodes::NTuple{N,Int}
end
```

Leave Index vs. Morton Id

```
x-----------x-----------x
|           |           |
|           |           |
|     9     |    10     |
|           |           |
|           |           |
x-----x--x--x-----------x
|     |6 |7 |           |
|  3  x--x--x           |
|     |4 |5 |           |
x-----x--x--x     8     |
|     |     |           |
|  1  |  2  |           |
x-----x-----x-----------x
```

```
x-----------x-----------x
|           |           |
|           |           |
|     33    |    49     |
|           |           |
|           |           |
x-----x--x--x-----------x
|     |15|16|           |
|  9  x--x--x           |
|     |13|14|           |
x-----x--x--x      17   |
|     |     |           |
|  1  |  5  |           |
x-----x-----x-----------x
```

Intraoctree Operation `*_neighbor`

```
x-------x-------x
|       |       |
|   3   |   4   |
|       |       |
x-------x-------x
|       |       |
o   1   *   2   |
|       |       |
x-------x-------x
```

- octant 1 at `xyz=(0,0)`, maximum refinement level of 1 and faceindex 2 (marked as `*`)
- face neighbor will be octant 2 with `xyz=(1,0)`
- leaving the octree boundaries possible by e.g. query with face index 1 (marked as `o`), returns octant at `xyz=(-1,0)`.
- same for `edge` and `corner`

Interoctree Operation `transform_*`

```
x-------x-------x
|       |       |
|  o3   |  o4   |
|       |       |
x-------x-------x
|       |       |
|  o1   *  o2   |
|       |       |
x-------x-------x
```

- 4 octrees with a single leaf each and maximum refinement level of $b = 1$

- transform octant 1 into coordinate system of octant 2
- Octant of octree 1 at `xyz=(0,0)` from octree 1 coordinate system, in octree 2 coordinate system at `xyz=(-2,0)`
- same for `edge` and `corner`

Collecting Nodes of the Forest

- Big problem with the lazy forest approach: collecting nodes without duplication

1. Distribute intraoctree unique nodes based on octree index k and octant corner xyz
2. Remove interoctree duplication based on vertex, edge and face neighborhood

Two sweeps through the forest.

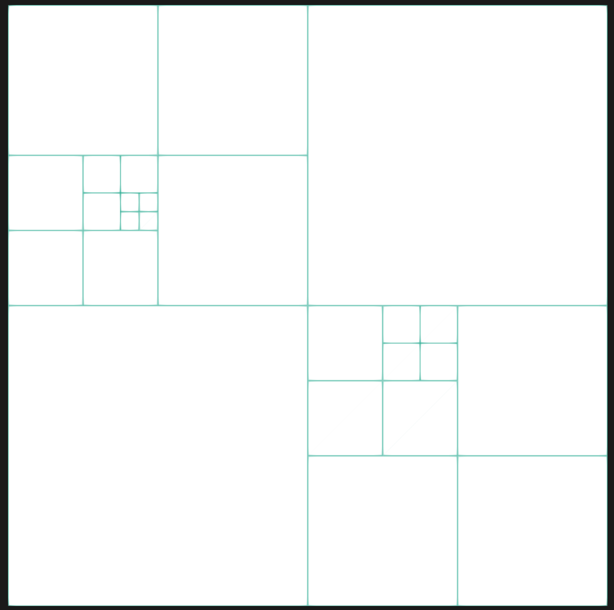
Collecting Hanging Nodes of the Forest

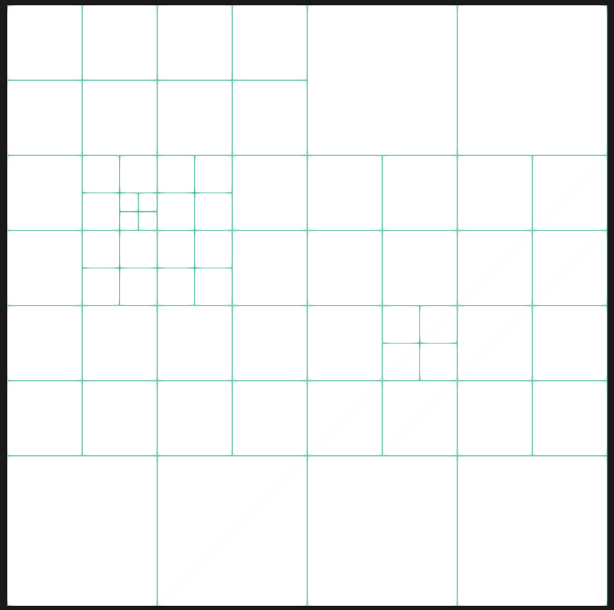
- In order to constraint hanging nodes, we need to detect them and associated "master" nodes

1. Loop over all octrees and octants
2. Loop over all corners
3. If pivot corner center of parent face hanging node candidate
4. Check inter and intraoctree

Balancing

enforce 2:1 balance





Reconstruct Boundary Facesets

After adaptation the boundarysets changed, so they need to be updated

1. Loop over all facesets
2. Loop over all faceindices in the faceset
3. Loop over all descendants of the faceindex and detect if any descendant face is contained in root face

A Story About AbstractGrid Interface 1/2

```
julia> grid = generate_grid(Quadrilateral,(1,1));
julia> adaptive_grid = ForestBWG(grid,3);
julia> dh = DofHandler(adaptive_grid);
julia> add!(dh,:u,Lagrange{RefQuadrilateral,1}());

julia> close!(dh)
ERROR: type OctantBWG has no field nodes
Stacktrace:
 [1] getproperty
   @ ./Base.jl:37 [inlined]
 [2] get_node_ids(c::OctantBWG{2, 4, 4, Int32})
   @ Ferrite ~/repos/Ferrite.jl/src/Grid/grid.jl:119
```

A Story About AbstractGrid Interface 2/2

The cell is in this case an octant, which will never know without the tree its vertices.

```
function add_vertex_dofs(cell_dofs::Vector{Int}, cell::AbstractCell, vertexdict, nvertexdofs::Vector{Int}, nextdof::Int, n_copies::Int)
    for (vi, vertex) in pairs(vertices(cell))
        ...
    end
end
```

How to handle this? Use special dispatches for the DofHandler?

Open ToDos

- finalize elasticity example with Zienkiwicz Zhu error estimator based on L2 Projection
  - L2 Projector seems to dislike the hanging nodes
- reconstruct neighborhood information after adaptation of the grid
- iterator over (half) faces, edges and corners
- Directly iterate over the lazy datastructure instead of materializing to a `Ferrite.Grid` for each solve