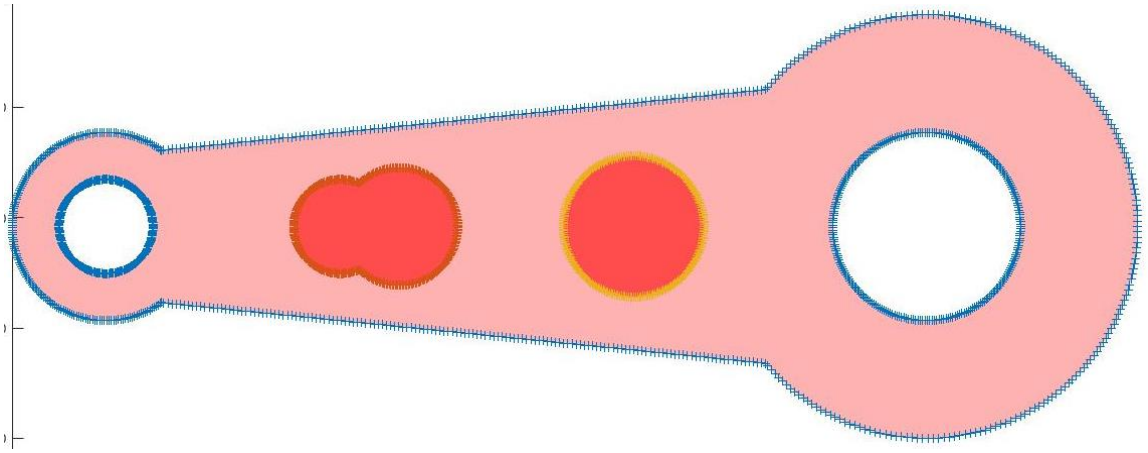
qtSBFEM underpinnings for crack localization

Aim

The aim is to analyse a wrench-like structure using my developed qtSBFEM, inspired by UNSW's crack propagation algorithms.

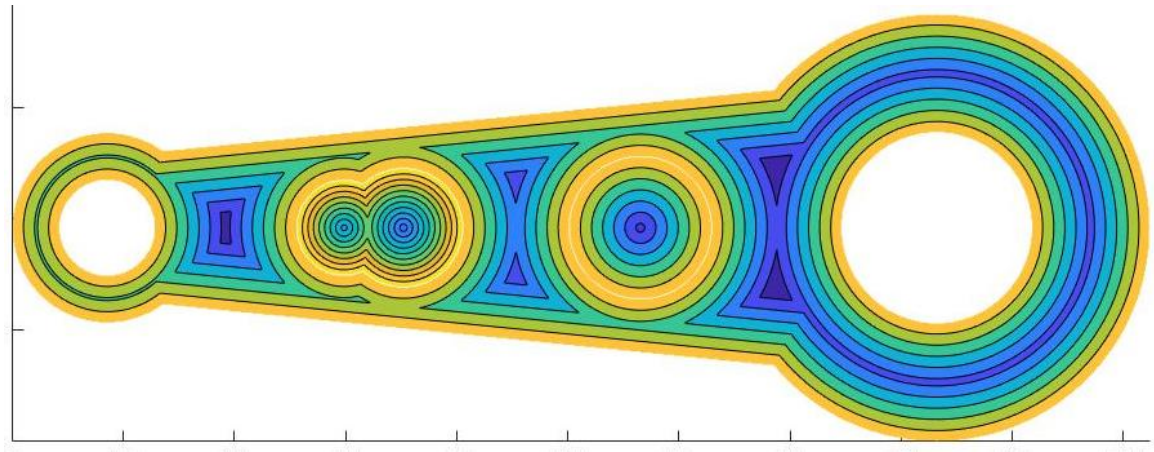
Implementation

The **explicit geometry** of the polygons and associated points, which form the basis of the quadtree analysis are presented below:

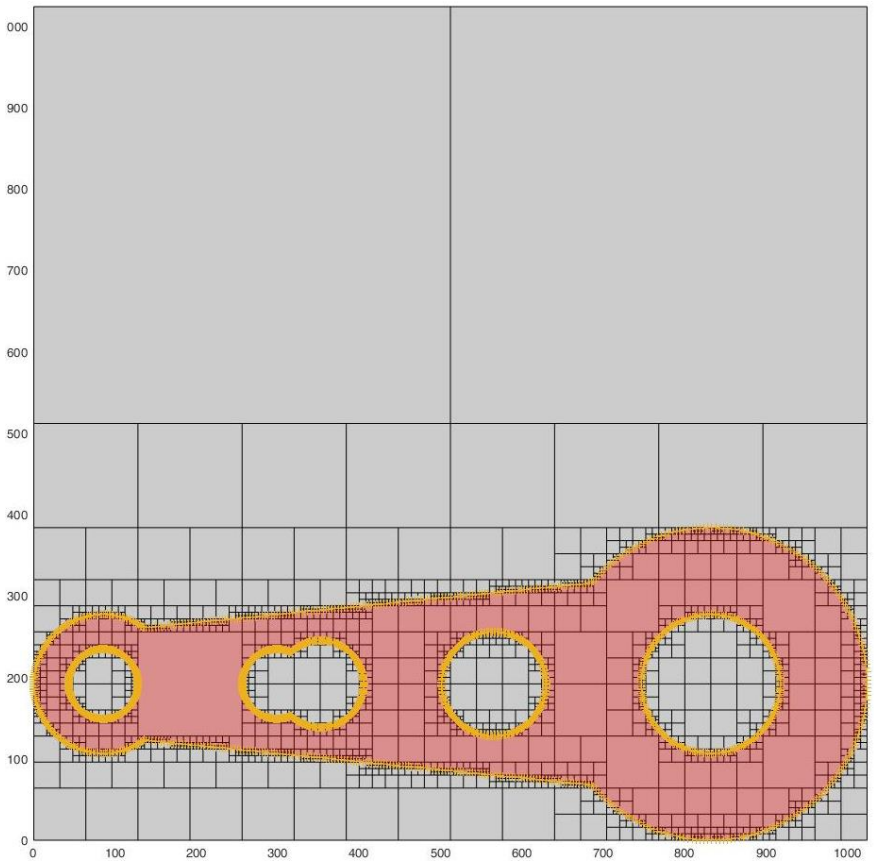


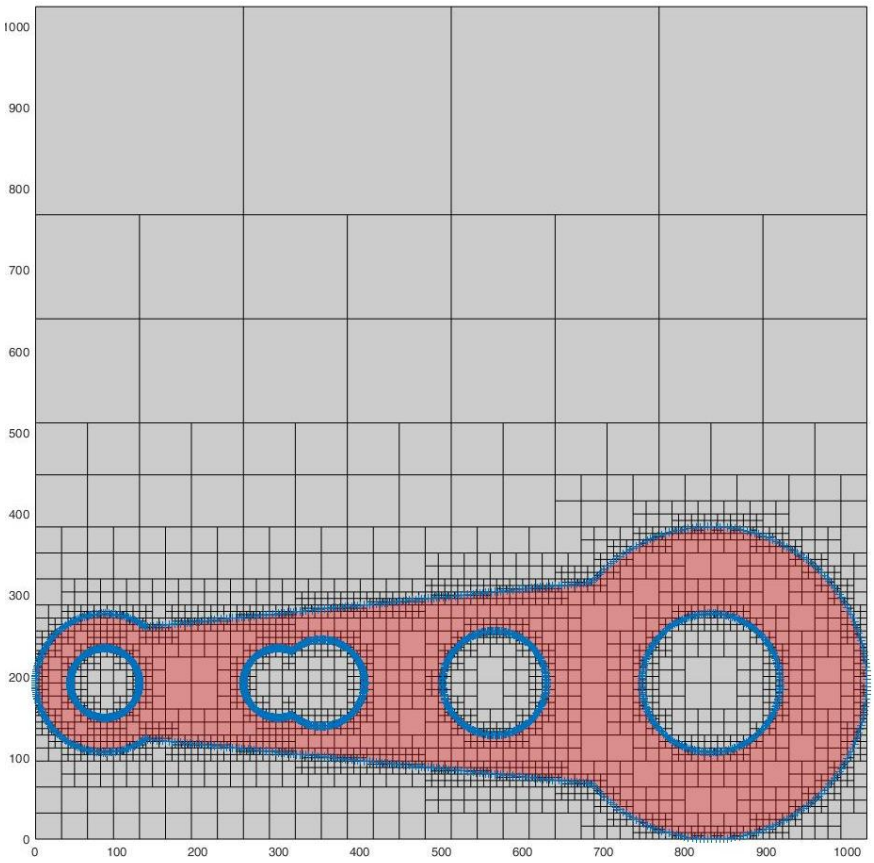
The individual shadings represent regions with different material properties. The scatter plots (+) represent explicit points on the boundary.

The **implicit geometry** is constructed via signed distance functions. These form the basis of determining if a standard rectangular quadtree block must be split into arbitrary polygons, as encountered on the domain boundary or transition between different materials for example.

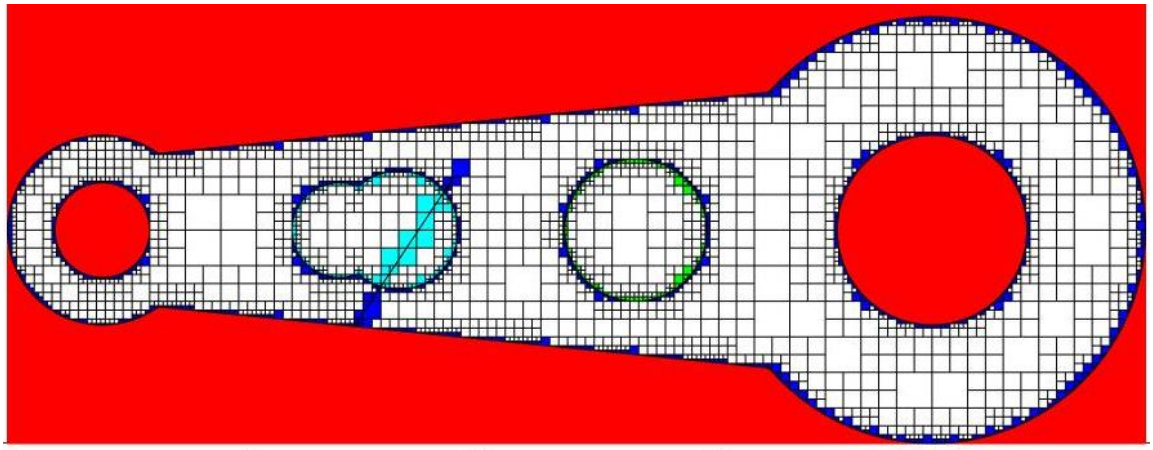


Now, a **standard quadtree decomposition** is performed, with the caveat that not more than 1 explicit point (+) may be in a block. It is then **balanced** to exploit precomputation:

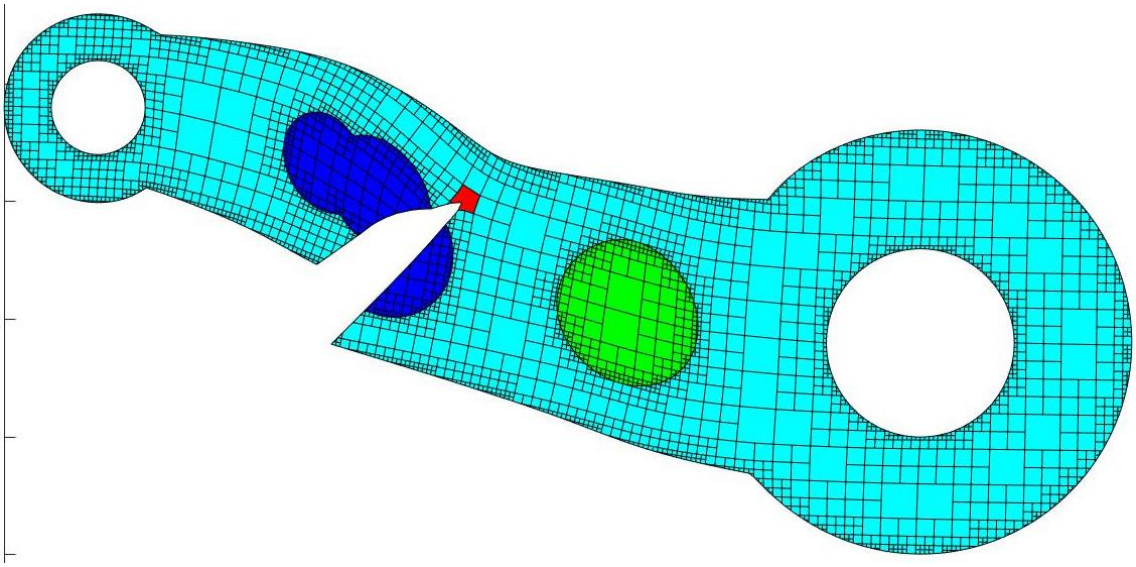




Performing trimming and clipping of the unnecessary blocks for analysis by leveraging the signed distance functions the final pre-processor mesh is obtained: a **hybrid quadtree**. The **crack** is inserted as well and the traversed blocks are clipped similarly. Trimmed blocks are coloured and grouped according to material properties. These are the blocks for which one must individually compute the stiffness matrix. All blocks in white (16 possible orientations in balanced quadtrees) have precomputable properties. The crack location is chosen arbitrarily, not based on physical considerations, but rather so to provoke the worst case to program for.

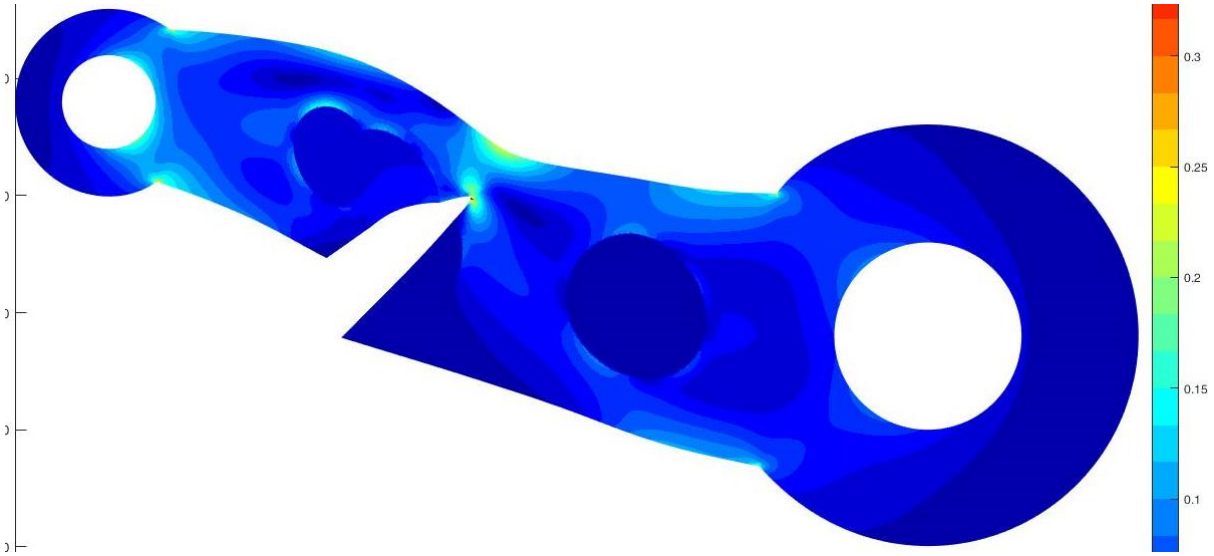


Next a standard FEM-type analysis is performed. The **displacement field** is plotted. The inner circle on the left is held in (x,y)-direction. On the right inner circle, I enforce a unit displacement in negative y-direction. Plotting with a **scaling factor of 100x** results in:



Colouring is according to material properties. The red region is reserved for the crack tip element, which requires special treatment in my code.

Having performed post processing, I plot the **von Mises stresses**. To accelerate I compute the stresses at the nodes and interpolate linearly over elements. In this analysis the E-mod of the inclusions are assumed to be an order of magnitude smaller than the rest of the domain, so that one can see a difference in the plots (visual inspection for debugging and testing).



Timings

The above discretization as shown is termed the coarse mesh. To understand how the code scales, I have also forced a finer discretization, the fine mesh, via changing two lines in the input file.

Coarse Mesh
DOF: ~8600

```
>> qtSBFEM
preprocessor
Elapsed time is 1.408933 seconds.
precompute K
Elapsed time is 0.186396 seconds.
polygon K
Elapsed time is 1.311940 seconds.
assemble K
Elapsed time is 0.164363 seconds.
enforce BC
Elapsed time is 0.161723 seconds.
Solve U = K\F
Elapsed time is 0.072807 seconds.
post process strains and stress
Elapsed time is 1.202903 seconds.
```

Fine Mesh
DOF: ~36000

```
>> qtSBFEM
preprocessor
Elapsed time is 5.123357 seconds.
precompute K
Elapsed time is 0.067878 seconds.
polygon K
Elapsed time is 2.413592 seconds.
assemble K
Elapsed time is 0.628066 seconds.
enforce BC
Elapsed time is 0.232572 seconds.
Solve U = K\F
Elapsed time is 0.368822 seconds.
post process strains and stress
Elapsed time is 4.725149 seconds.
```

Remarks

The steps associated with most of the computational cost, are not applicable to the crack ID problem.

Step	**Anticipated Reduction in computational Cost**
prepocessor	Baseline (ucracked) qtDecomp can be performed ahead of analysis
Precompute K	Ahead of analysis
Polygon K	Clear majority of polygons computed ahead of analysis
Assemble K	-
Enforce BC	-
Solve	-
Post process	Only compute those strains at sensor locations (at least 90% reduction)