Multi-scale modeling reports

Michał Policht

1. Simple grain growth CA

Task

The task is to create a visual program that evaluates cellular automata with Moore neighborhood. Requirements can be summarized as follows.

- Engine should simulate grain growth according to basic Moore neighborhood rules.
- Engine should ba also aware of boundary conditions. Implementation can pick between absorbing or periodic boundary conditions.
- The program should provide import/export facility. Text files and bitmap files must be supported.

Technicalities

- The program is written in C++ and QML languages. It uses Qt framework. User interface is implemented mostly in QML language, taking advantage of Qt Quick technology. Qt Quick applications can be deployed on numerous devices, including mobile phones and embedded systems. Computing engine is implemented in C++, which is a decent choice for computation intensive problems, such as evaluation of cellular automata.
- Program has been implemented around model/view architecture. Qt provides sophisticated *Model/View* framework. Qt model is implemented in *Model* class. Qt model delegates requests from the view to the core model described below.
- *Space*, *SpaceProperties* and *Cell* are core model classes. As program is going to be expanded, there is a concept of space processor, which is realized by *AbstractSpaceProcessor* abstract class. Thanks to this approach, almost any new problem, which is going to be introduced in the future, can be reduced to implementation of new space processor. Initially *MooreProcessor* class solves basic Moore neighborhood problem.
- Grains are distinguished by their color, which works as their id. Some colors have special meaning. For example empty cells are defined by black color. To keep things in order all special colors are stored in *Palette* singleton.

- Cellular automata space is double buffered. During simulation changes are applied to background buffer, while active buffer remains untouched. This is both: convenient (it is easy to analyze neighborhood) and fast (swapping buffers is fast operation).

User interface

User interface is controlled by *CAMPWindow* QML component. Figure (17) shows typical view of the user interface.

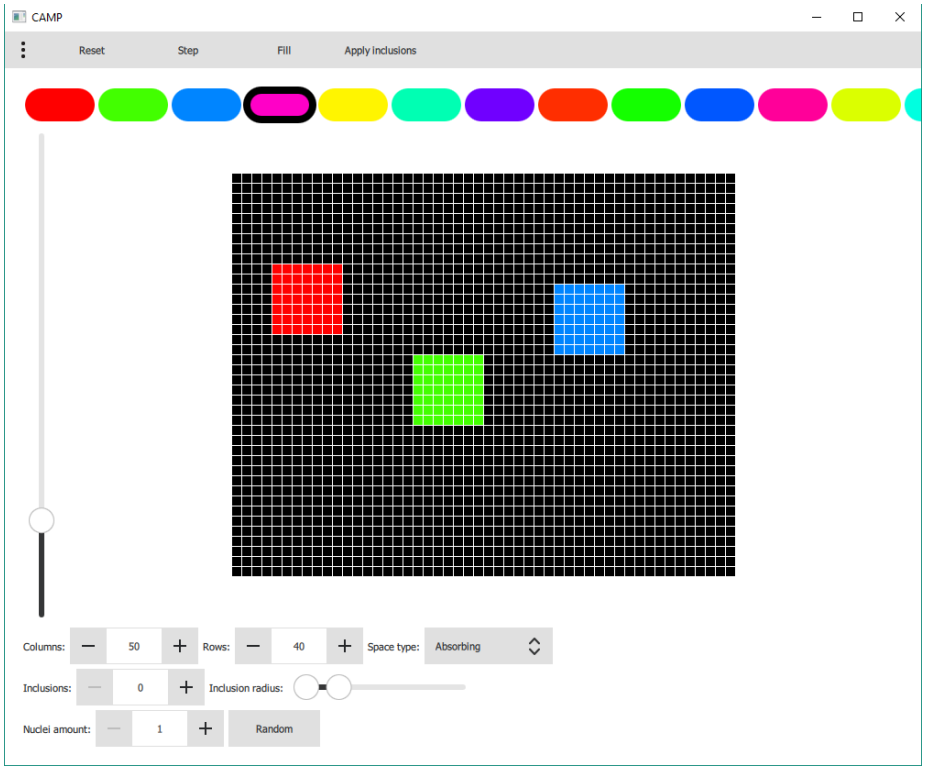


Figure 1: User interface.

User can zoom in/out the view with the slider on the left.

Over the top there are buttons: *Reset*, *Step*, *Fill*, *Apply Inclusions*. Button *Reset* clears the space. Button *Step* performs a single step of simulation. Button *Fill* runs whole simulation (until the space is completely filled with grains). Button *Apply inclusions* adds inclusions.

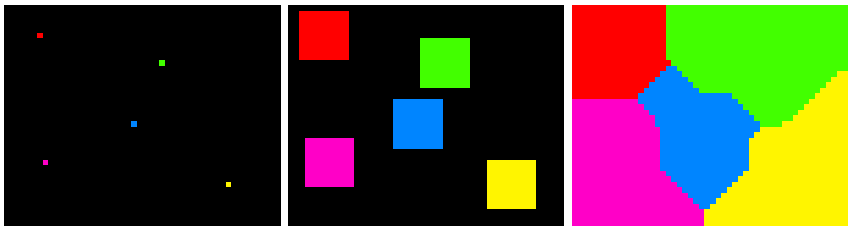
To place a nucleus user can pick a cell with left mouse click. Color bar on the top allows one to choose a color of a nucleus. Second click on the cell with the same color as selected on the bar clears the cell. Nuclei can be randomly placed with by clicking *Random* button on the bottom. The amount of randomly generated nuclei can be controlled by *Nuclei amount* spin box to the left.

Similarly user can control the number of inclusions using *Inclusions* spin box. They are randomly injected after user clicks *Apply inclusions* button. Range of their size can be controlled by *Inclusions radius* slider.

User can also control size of the space (*Columns* and *Rows* spin boxes). Space type can be absorbing or periodic.

To export or import data, or an image user clicks ⋮ (vertical ellipsis) button. Format is selected by an extension. This means that if user names a file with one of the recognized image format extensions, such as *.png* or *.bmp*, data will be exported to an image. Otherwise a text file is created. Format is recognized automatically during imports.

Analysis of results and conclusions



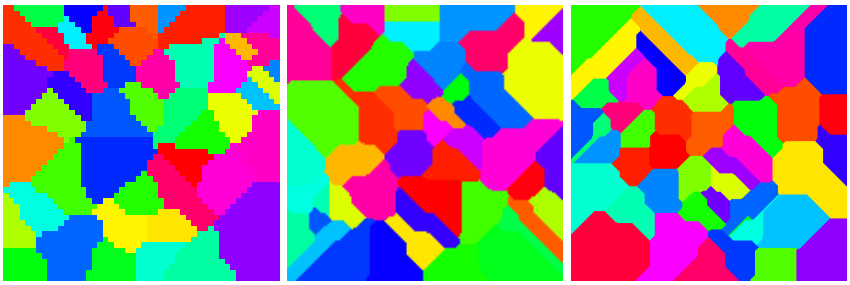
(a) Nuclei. (b) Grain growth. (c) Final structure.

Figure 2: Stages of grain growth.

Figure (2) shows stages of grain growth. As one can see plain Moore neighborhood creates square shapes during early stages of the growth. As cells make contact with each other (through neighborhood) the grains start to take irregular forms. At the end whole space is filled and grain boundaries are visible.

(a) 50 × 50. (b) 150 × 150. (c) 300 × 300.

Figure 3: Micro-structures with 10 nuclei.



(a) 50 × 50. (b) 150 × 150. (c) 300 × 300.

Figure 4: Micro-structures with 50 nuclei.

Figures (3) and (4) show generated micro-structures with respectively 10 and 50 nuclei using different sizes of CA space. It seems that changing the size of CA space does not affect the shape of grains too much. We get better quality picture at higher resolutions however. Above implies that amount of nuclei determines a scale factor at which one looks at a micro-structure.

Computer generated micro-structures can be compared with real micro-structures. Figure 5 shows aluminum alloys. It can be seen that real micro-structures briefly resemble computer generated images. Notably computer generated images are missing features such as inclusions. This will be fixed during further classes.

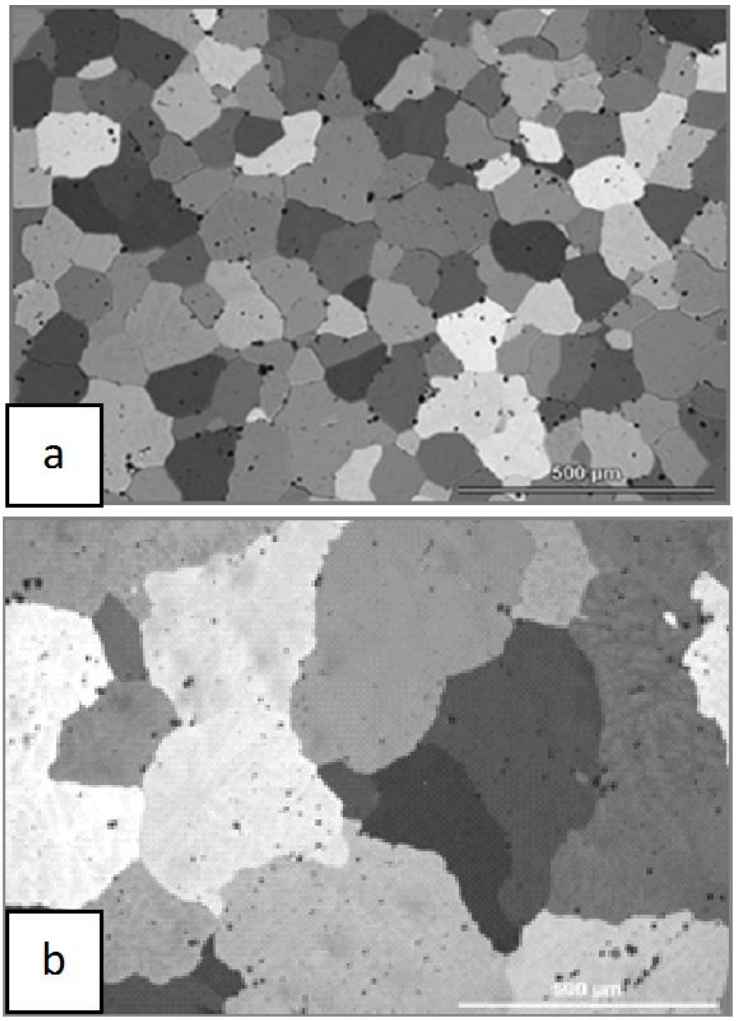


Figure 5: 7000 (a) and 7000Zr(b) alloys after solution heat treatment (490°C-2h). Source: *https://www.intechopen.com/books/aluminium-alloys-new-trends-in-fabrication-and-applications/pure-7000-alloys-microstructure-heat-treatments-and-hot-working*

Issues and possible improvements

- One notable issue is that program uses *TableView* component from *Qt Quick Controls 2* module to visually represent CA space. This component was in beta version at the time of writing the program. This can affect performance and produce glitches. *TableView* implementation is obviously immature and lacks optimizations.
- Code foundations are solid, but it lacks advanced optimizations. At least two optimizations could be implemented.
  - In cellular automata only evolution of boundaries matter. During first steps of simulation algorithm just enlarges squares formed from

nuclei. This means that we could initially arrange large squares, instead of nuclei that occupy single cells.

- In each step of simulation, basic Moore processing engine iterates over whole space, one cell after another. It is not necessary as only certain cells can be modified - those at boundaries of existing grains. It does not make sense to check Moore neighborhood inside a grain for example. It would be more efficient to store a list of "active" cells, which participate in grain growth.

Author of the program tends to stick the rule formulated by Donald Knuth, which states that "premature optimization is the root of all evil" anyways, so a primary focus was put to create clean code and solid foundations for further classes.

- To improve performance programming techniques that effectively utilize modern hardware such as multi-threading, CUDA or OpenCL could be used.

2. Inclusions

Task

The task is to modify program, so that inclusions can be injected into microstructure. It should be possible to add inclusions at the beginning and at the end of simulation. Those to be added at the end of simulation should be formed at grain boundaries.

Technicalities

Compared to previous version, three classes have been introduced.

- *AbstractInclusionsProcessor* - represents abstract inclusion processor.
- *InclusionsProcessor* - a class that processes a space and adds initial inclusions.
- *BoundaryInclusionsProcessor* - a class that injects inclusions at grain boundaries.

User interface

User interface has additional buttons: *Apply initial inclusions* and *Apply inclusions*. First button adds initial inclusions as name suggests. The second one adds boundary inclusions. Size and amount of the inclusions can be controlled by *Inclusions* spin box and *Inclusion radius* range slider. For each inclusion program chooses random radius from the given range.

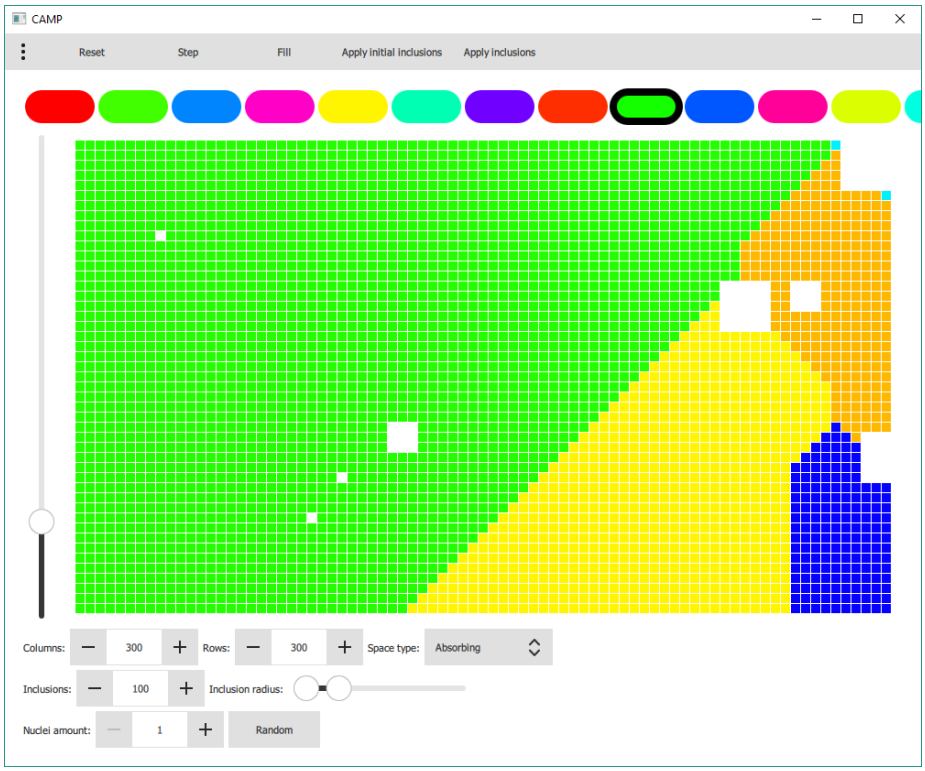
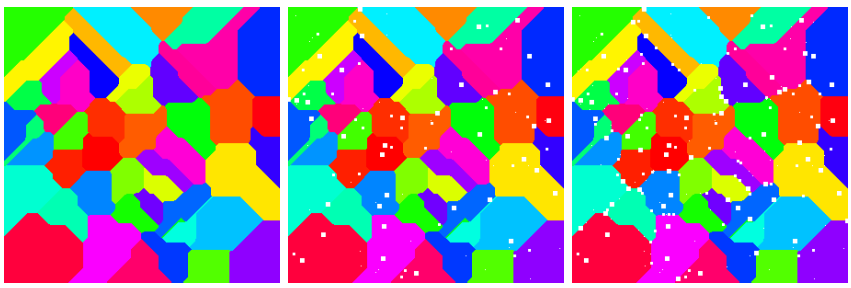


Figure 6: User interface.

Analysis of results and conclusions

Figure (7) shows the same micro-structure as in the first report. On (7b) 100 inclusions have been added at random spots. After that additional 100 inclusions have been added at grain boundaries, which can be seen on (7c).



(a) Initial micro-structure.

(b) Inclusions added at random cells.

(c) Additional inclusions at grain boundaries.

Figure 7: Micro-structures with 50 nuclei.

Once again we can compare computer generated image with reality. Let's bring back aluminum alloy micro-structure from the first report (figure 5). Missing inclusions are now in place. Inclusions consist of nonmetals and chemical

compounds, which are present in steel and alloys. With improved CA automata we can model these features.

Issues and possible improvements

- As usually, the code could be optimized. Especially algorithm, which adds inclusions at grain boundaries performs a simple linear search for the boundary, starting at random location. If we had indexed cells, which are located at grain boundaries, we could simply pick random cell from the set and put boundary inclusion there, without the need to look for the grain boundary every time we add inclusion. However, in our simple educational program basic algorithm was good enough to process inclusions in reasonable time.

3. Influence of grain curvature

Task

The task was to add some control on how grains are shaped. This had to been done in "automata" way, so a set of rules were introduced according to which each grains should grow.

1. The id of particular cell depends on its all neighbors. If five to eight of the cell neighbor ids are equal to S, then cell transforms to the state S.
2. The id of particular cell depends on its nearest neighbors. If at least three of the cell neighbor ids are equal to S, then cell transforms to the state S.
3. The id of particular cell depends on its further neighbors. If at least three of the cell neighbor ids are equal to S, then cell transforms to the state S.
4. The id of particular cell depends on its all neighbors and has X probability chance to change.

Technicalities

It was enough to implement *Lab4Processor* class, which processes space according to the rules.

User interface

User interface has *Threshold* spin box, which controls the probability of change (rule 4).

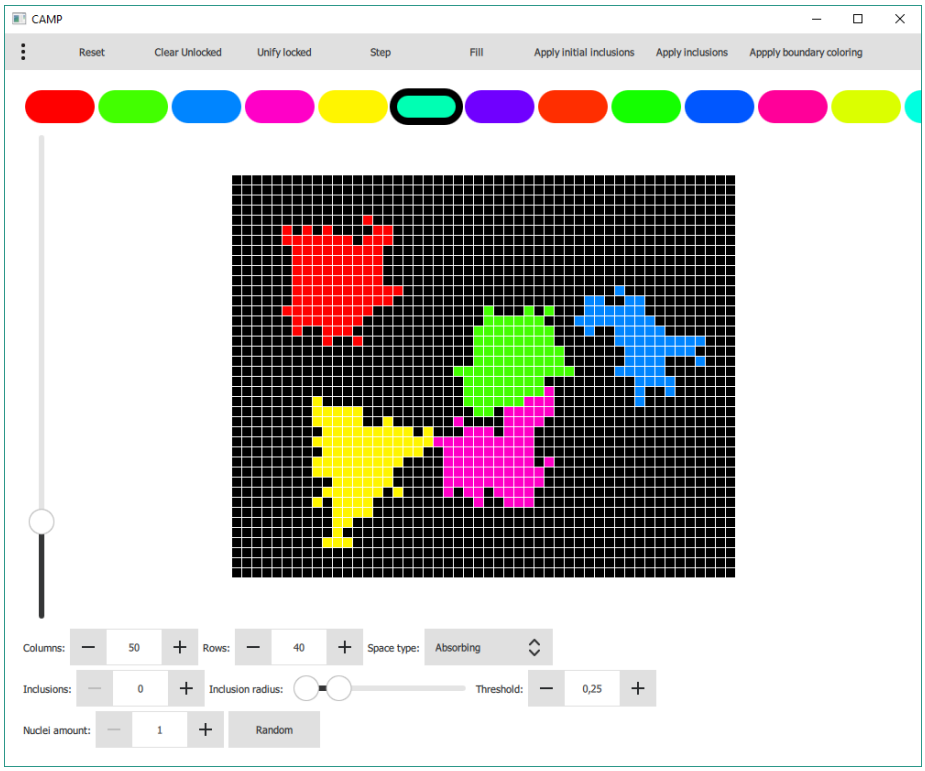


Figure 8: User interface.

Analysis of results and conclusions

Figures (9), (10), (11) and (12) show evolution of a space with various thresholds.

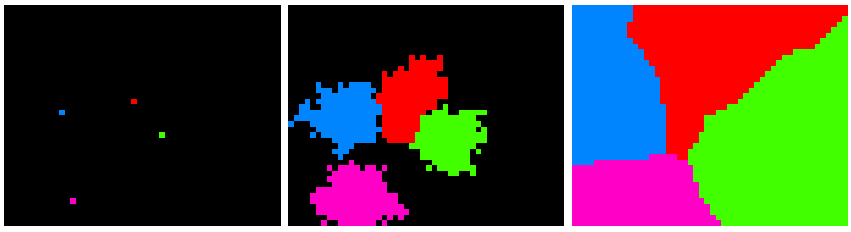


Figure 9: Evolution of CA space with threshold set to 25%.

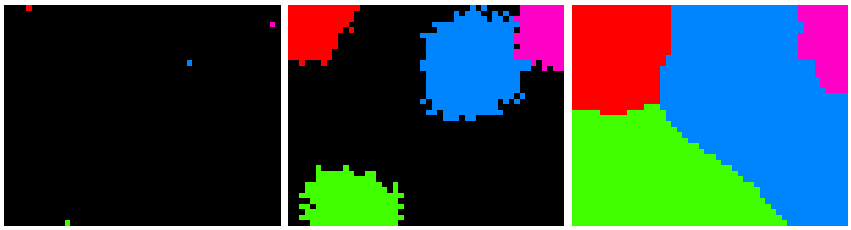


Figure 10: Evolution of CA space with threshold set to 50%.

Figure 11: Evolution of CA space with threshold set to 90%.

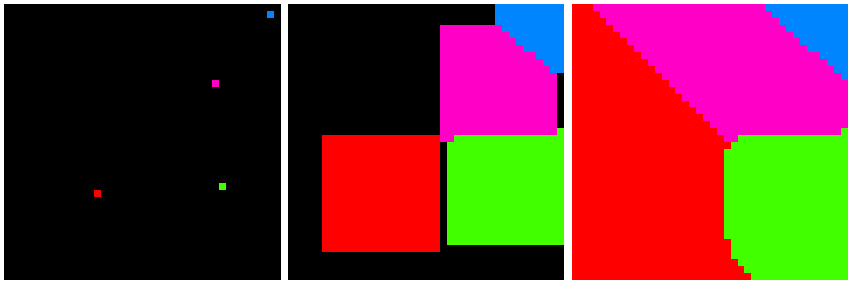
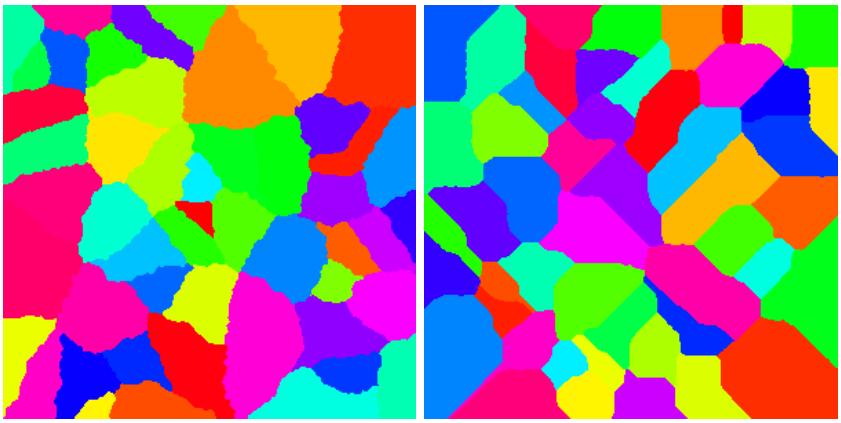


Figure 12: Evolution of CA space with threshold set to 100%.

As one can see, the threshold determines the shape of grains. With smaller thresholds we obtain less regular, less rectangular shapes. With threshold set to 90% we get almost square shapes. With edge case of 100% threshold we get the same results as with basic Moore algorithm.

Final results of CA space evolution performed with two different thresholds can be seen on figure (13). More advanced automata model allows us to simulate creation of micro-structures with less regular grain boundaries. Many real micro-structures show up this kind of irregularity (notably rocks and minerals). Figure (14) shows irregular micro-structures of quartz.



(a) Threshold set to 25%. (b) Threshold set to 90%.

Figure 13: Micro-structure with 50 nuclei.

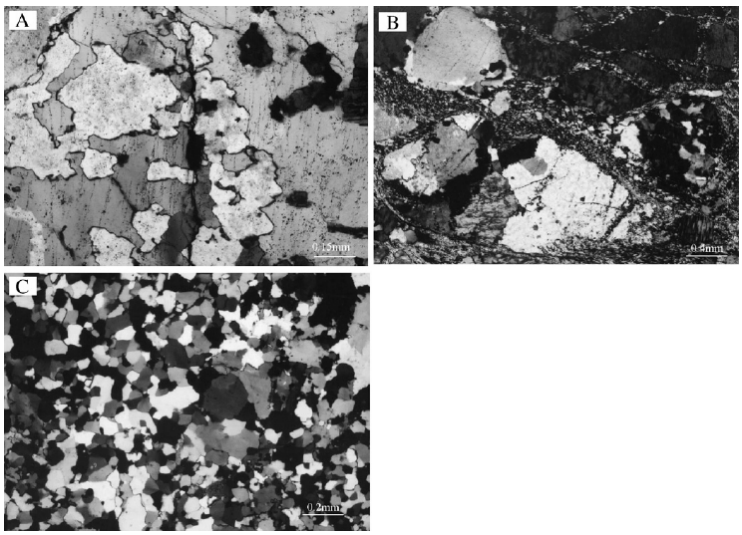


Figure 14: Quartz micro-structure. Source: *https://www.researchgate.net/figure/Photomicrographs-of-quartz-microstructure-with-crossed-polarized-light-A-Quartz-grains_fig3_222426536*

Issues and possible improvements

Not much could be changed as strict algorithm was defined in the task.

4. Substructures

Task

The task was to extend the program, so that it could emulate the formation of dual-phase micro-structures.

Technicalities

The idea to implement this feature relies on the fact that substructure behaves like inclusions; it is neither affected nor it does not affect main micro-structure. As it was stated in previous reports, colors have semantics. Color not only represents visual appearance and id, but it also affects the simulation (for example black cells are treated as empty space). Taking this into account, a second color has been added to *Cell* class, so that each cell now has two colors: one is visual and the second one is logical. This way a view can represent a cell with visual color, while space processors (classes that extend *AbstractSpaceProcessor* class) will use logical color. So a cell can appear to processor as an inclusion, but it can preserve its unique color id.

User interface

User locks grains by holding left mouse button over a particular grain. Locked grains won't be cleared, when user click *Clear unlocked* button. Locked grains can be unified by clicking *Unify locked* button.

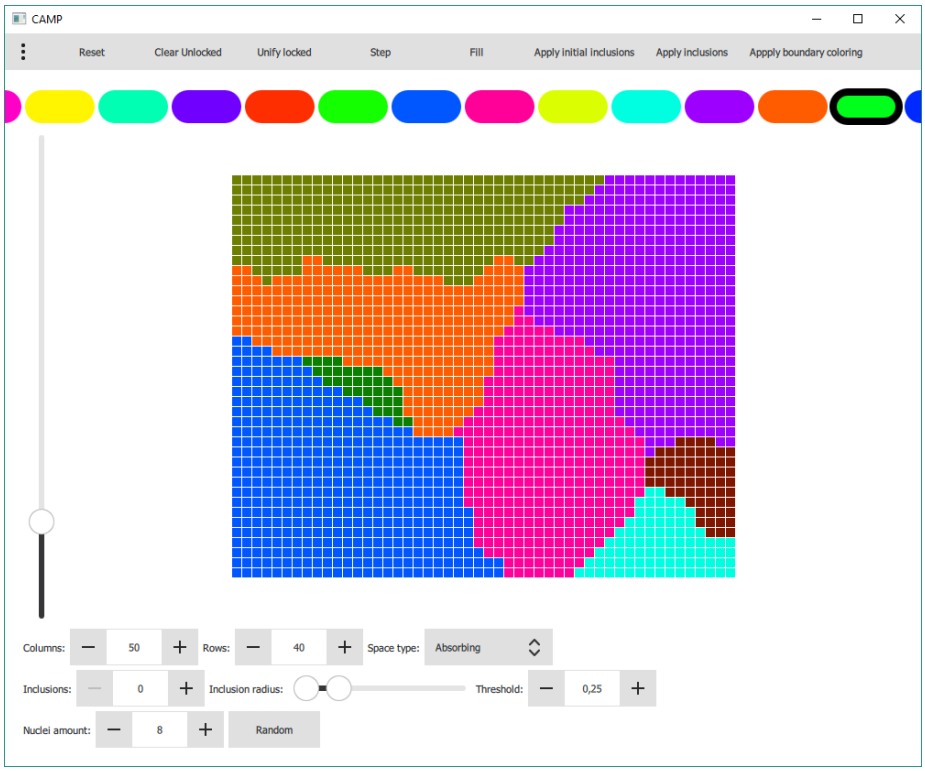
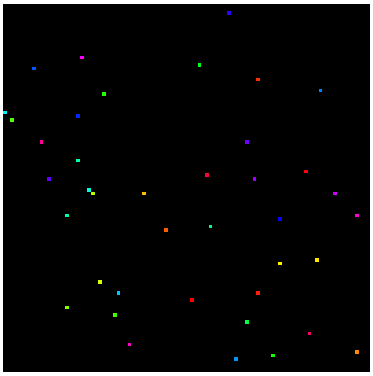


Figure 15: User interface.

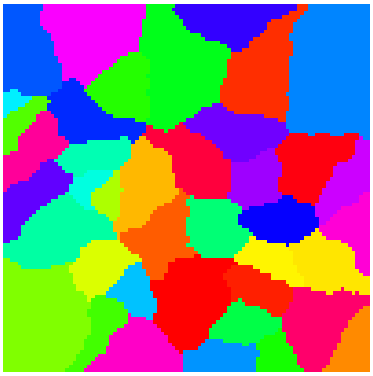
Analysis of results and conclusions

Let us illustrate generation of sample dual-phase micro-structure with figures.

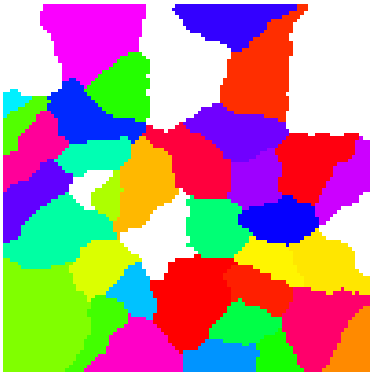
1. Figure below shows initial space with 40 nuclei.



2. Grains of substructure are formed.



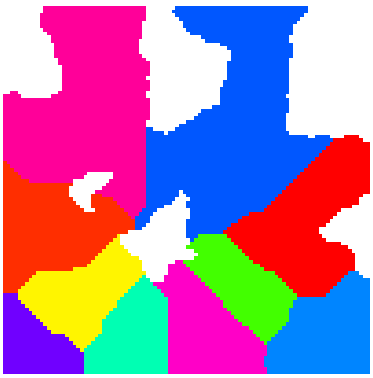
3. Some of the grains are locked.



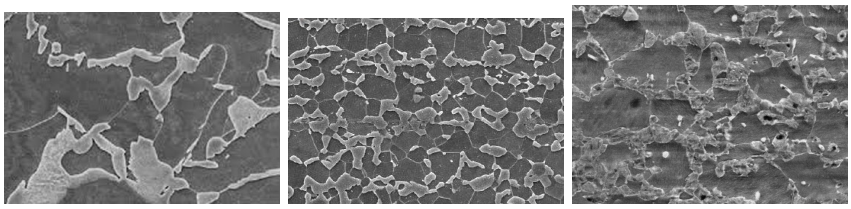
4. Locked grains are unified.

5. Additional 10 nuclei are added.

6. A final dual-phase micro-structure is formed .



As we can see this simple modification allowed us to emulate creation dual-phase micro-structures. Figure (16) shows real micro-structures of dual-phase steel, with various amounts of martensite.



(a) DP steel with 27.6% of martensite (b) DP steel with 29.7% of martensite (c) DP steel with 32.1% of martensite

Figure 16: Dual-phase steel. Source: Monika Pernach, AGH

Issues and possible improvements

Various aspects of the program could be improved. Import and export facility would require deep modifications to properly store additional information about substructure. It was out of the scope of the task, so it was left behind.

5. Grain boundaries

Task

The task was to extend the program, so that it can colorize grain contours. The program should report the percent of grain boundaries.

Technicalities

The problem was solved by implementing *BoundaryColoringProcessor*, which looks for boundaries and turns them into inclusions. User can then proceed with *Clear unlocked* button to clear everything except inclusions, which are created on grain boundaries.

User interface

No changes have been made to user interface, except the addition of *Apply boundary coloring* button.

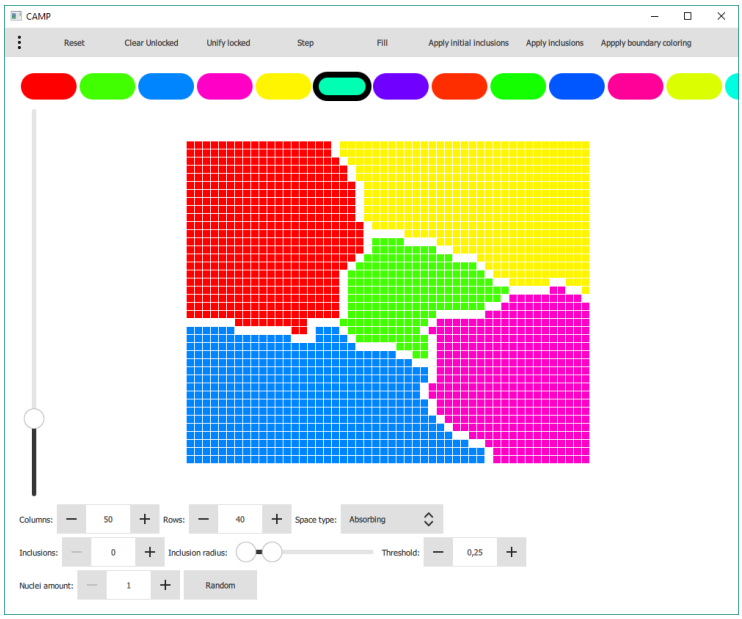
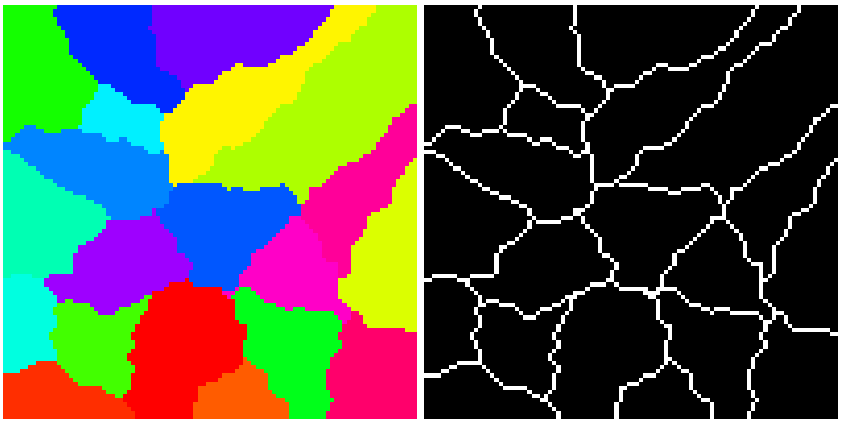


Figure 17: User interface.

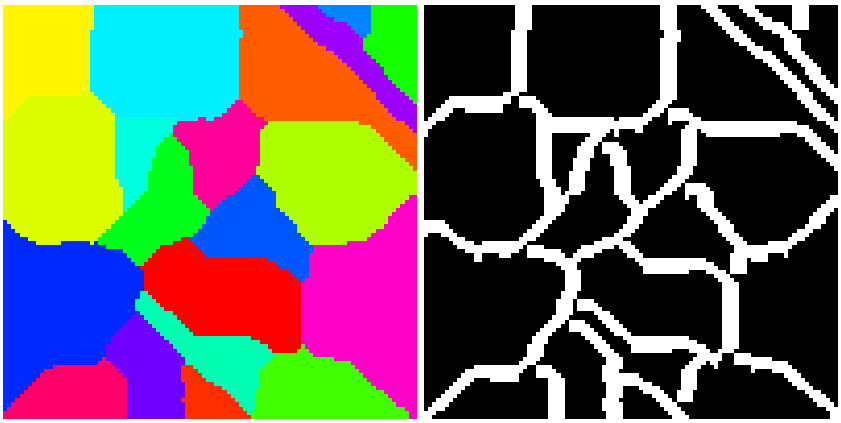
Analysis of results and conclusions

Figures below show micro-structures before and after boundary coloring. Program has reported space occupied by boundaries. For micro-structure shown on figure (19) we have used larger contour width.



(a) Before boundary coloring. (b) Boundary coloring applied.

Figure 18: Micro-structure with 7.05% of space occupied by boundaries.



(a) Before boundary coloring. (b) Boundary coloring applied.

Figure 19: Micro-structure with 22.36% of space occupied by boundaries.

Program also allows us to find a contour of a single grain, which can be seen on figure (20). For this we use locking feature from previous classes.

(a) Micro-structure. (b) Selection of a grain. (c) Clear space.

Figure 20: Single grain coloring.

Issues and possible improvements

Obviously boundary coloring algorithm could be improved to produce better contours. To do this properly we should use state of the art computational geometry algorithms. This was beyond time constraints, so we could only afford a simple algorithm, which merely does its job.