**Intro**

This is a project plan for continuing the migration of grid editing functions from Fortran prototype code to clean C++ code. The goal of the first phase was to prove the efficiency of C++ implementation when performing mesh orthogonalization, which is one of the most computing-intensive grid operation algorithms.

The efficiency was proven to be comparable to Fortran, and even better in multithread settings. This is an important achievement and allows to speed up computations for large grids. Moreover, the code communication with the C#/C++ GUIs is easier, the code has a better structure, it is unit-testable and implements the algorithms in fewer lines of code.

By rewriting the algorithm is also possible to re-design the interfaces (API) and improve the Grid Editor interactivity. An example is orthogonalization, where the user is now able to stop the process while visualizing the results. Another good candidate for interactive behavior is the “curvilinear grid generation (orthogonal)” algorithm, where the grid is grown from a central spline towards the river banks.

**The plan**

In this phase small, large and medium-size functions (denoted with letters S, M and L) will be ported, unit tested and connected to the Grid Editor. These are the same functions currently used by the Grid Editor. The proposed steps are the following:

1. Implement “curvilinear grid (orthogonal)“ (L, orthogonal version from Sander, with the desired interactivity in the GUI to stop the algorithm). The complete algorithm also offers the possibility to copy the central spline to polygon. This should also be made available.
2. Implement generate “curvilinear grid (transfinite)“ (M, transfinite version from Herman).
3. Implement merge nodes. A Kd-tree implementation needs to be added (S).
4. Implement “curvilinear to unstructured grid converter” (S).
5. Add a third-party triangulation library (S, triangle.c from R. Shewchuk).
6. Implement triangular mesh generation within a polygon (S).
7. Implement polygon edges refinement. This functionality is needed to produce triangulations in polygons with user-defined resolutions (S).
8. Implement the algorithm to convert mesh boundaries to polygon (S).
9. Implement “Make Rectangular grid” (S)
10. Implement “Make Rectangular grid within polygon” (S)
11. Implement “Make Grid from samples” (S)
12. Grid refinement from samples (L)
13. Edge/nodes manipulations: move nodes, delete edges (S)
14. Offset of a polygon perimeter (S)

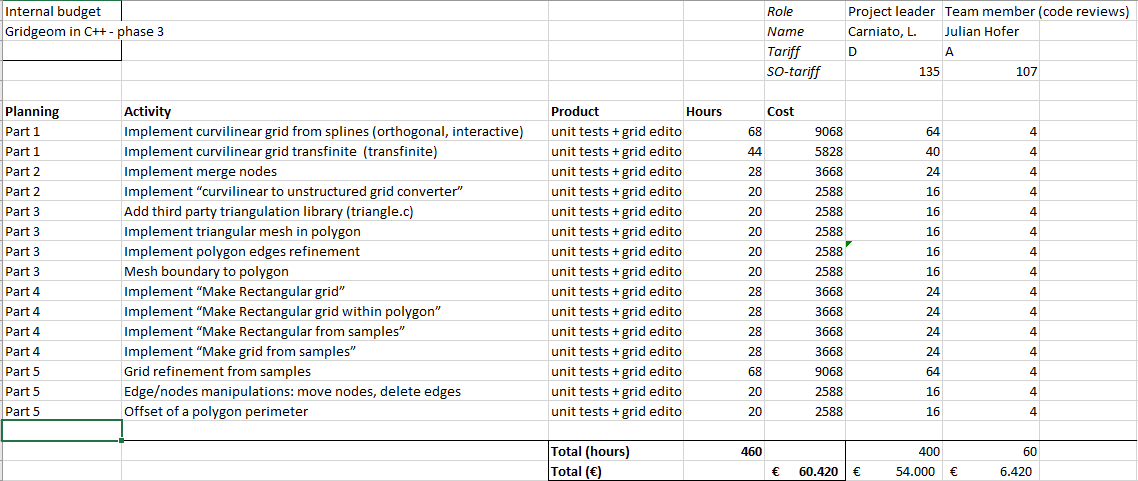
The number of functionalities that can be coded in the second phase is larger than phase one because some of the algorithms rely on data structures filled during the administration phase, which has been already coded.

After implementing the points above the new Grid Editor will be able to use only the new C++ backend. In the re-writing process, the ported functions will be hooked up to the Grid Editor and tested. Thoughtful testing is advised and might be considered in the next step.

It is also advised to involve another team member in the rewiring process, both for sharing the knowledge of the new component and improving the quality of the developed software through code reviews. Julian Hofer is proposed as developer/reviewer (his availability needs to be discussed, in principle he could be involved one day a week while the project is running).

**Time / budget / planning / team**

See the table below, which is also available as a spreadsheet:



The requested budget for the project is € 60 420,--. At the end of the project, the results with the new library will be demonstrated in the Grid Editor.

**Risks and measures**

Based on the previous experience with phase one, the following risks are anticipated:

|  |  |
| --- | --- |
| Risk | Grid refinement from sample (a large function) cannot be completed due to budget constraints. |
| Measure(s) | The porting of this function will be postponed. |

|  |
| --- |
| Copy for  To be defined. |
|  |