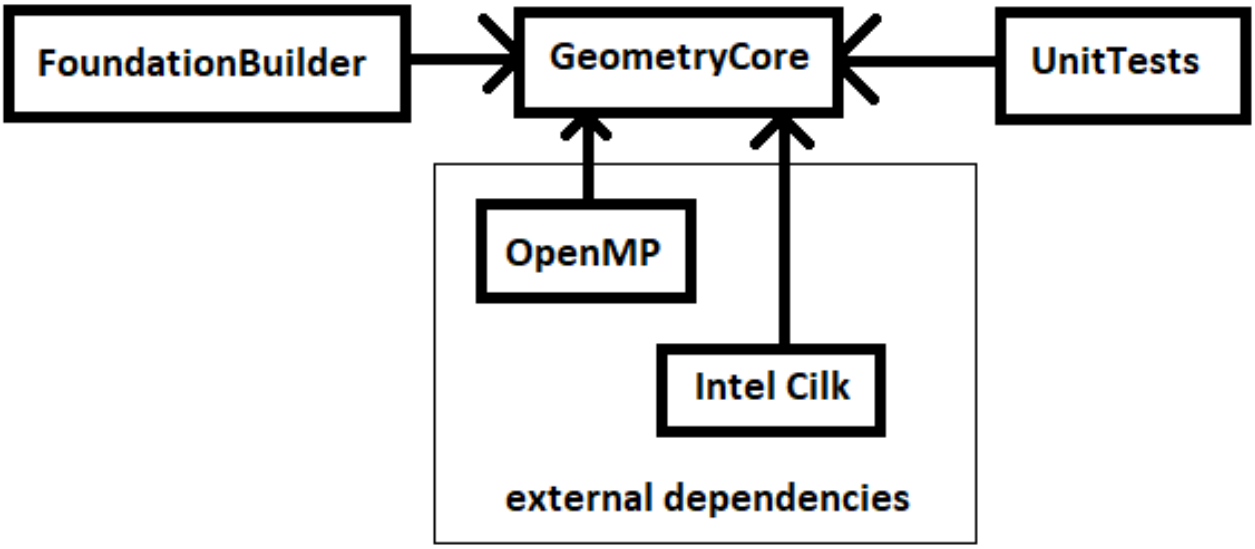
FoundationBuilder - Software Architecture

Software Implementation in a High Level

FoundationBuilder is Visual Studio 2019 Solution and consists of three projects:

- **GeometryCore**
- **UnitTests**
- **FoundationBuilder**

Here is the dependencies graph of the solution:



- **GeometryCore**
  C++ static library - provides geometrical primitives as well as linear algebra functionality. At the start of the project, this library has only classes' declarations while the functionality of major primitives should be implemented. If necessary, library may be extended with new primitives and functions.
- **UnitTests**
  C++ lightweight unit test framework. It is fully completed and should be used to cover all GeometryCore functionality with unit tests.
- **FoundationBuilder**
  C++ application, at the start of the project contains classes only for basic architecture.

Additional Software Implementation Details

As the FoundationBuilder-Spec says, Application should support multithreading capabilities. Thus, **GeometryCore** functionality should support single threaded as well as multithreaded calculations. For that purpose, library defines the global `environment` variable, which is supposed to be used as a runtime environment selector. The available options should by SingleThreaded, MultiThreadedByOpenMP, MultiThreadedByIntelCilk. For example, if some function can be improved by parallelizing the code, its implementation should look like this:

```
void someFunction()
{
  if (env == single_threaded)
    { plain single threaded code }
  else if (env == multi_threaded_openmp)
    { improved implementation using multithreaded OpenMP technology }
  else if (env == multi_threaded_intel_cilk)
    { improved implementation using multithreaded Intel Cilk technology }
}
```

Note that currently, the OpenMP and Inter Cilk are not linked with the projects.

Each developer **SHOULD TIGHTLY FOLLOW** these **THREE** main principles during software implementation:

1. Keep code as simple as possible. Simpler code, better quality.
2. Cover code as much as possible with unit tests. Provide, at least, one test case scenario for every method/function.
3. Before coding, try to research info as much as possible. For example, firstly understand the intent of specific function, research the approach/algorithm and search for possibilities for improving performance by parallelizing the code. Only then start coding.

… and these rules for code style:

- only spaces for tabulation (no tabs);
- `CamelCase` style for class, `camelCase` for variables/methods/functions;
- `const` in every place where possible;
- meaningful names for each entity;
- keep solution clear from compile errors and warnings.

Development of software is not just coding, but the research process as well. All necessary geometric algorithm are finely described in different Internet resources.

The start point of **GeometryCore** and **FoundationBuilder** projects provide only basic classes, which should be implemented. However, if necessary, extend the functionality with new classes and functions. Note, developing further architecture or even redesigning of the basic classes is the decision left up to the Team.

These classes/functions are not fully implemented: `AABB2D`, `AABB3D`, `LinearAlgebra`, `Mesh`, `Polygon`, `ModelHandler`, `CommandLine`, `Application`

Those classes can be expended with new functionality, if needed, as well as new classes can be created.

Project Roles and Responsibilities

Person	Role	Responsibility
Vlad Hrybynchuk	Customer	Define software requirements
Vlad Hrybynchuk	Project Manager	Prepare documentation and the ground for further development
Vlad Hrybynchuk	Tech Lead	Set up project, design main software architecture, tech consultation
…	Team Lead	Coordinate teamwork, report to Project Manager
…	Team Member	Write good code in time

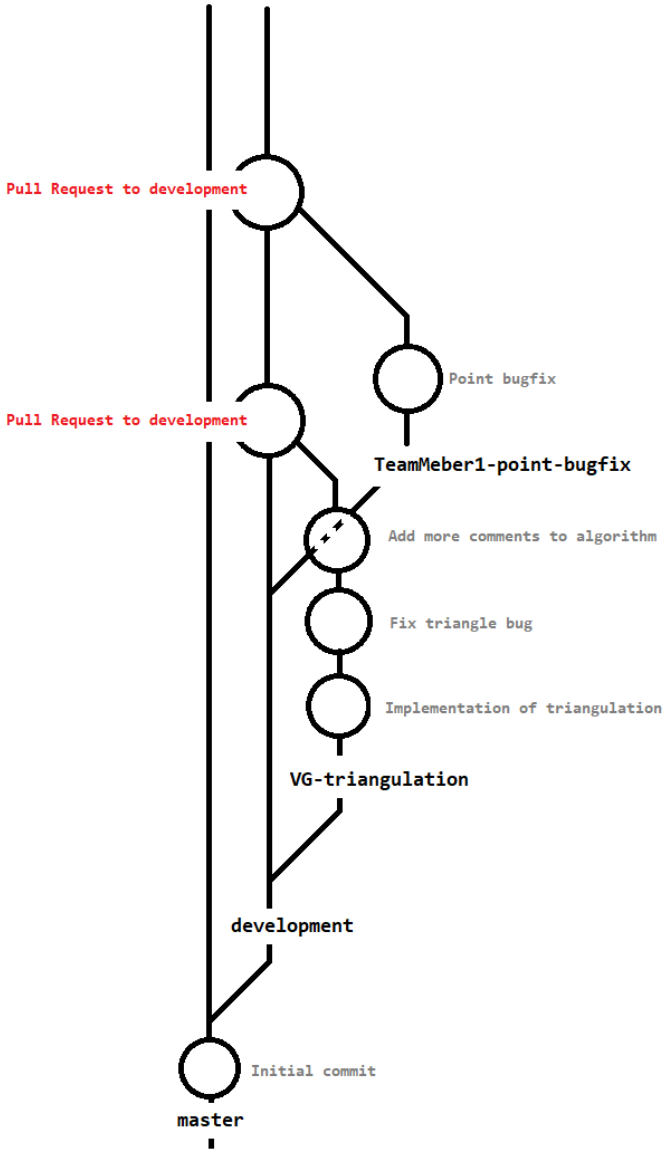
Git Workflow

This section describes basic workflow rules. Every Team Member should be aware and familiarized with the basic git terminology. Firstly, `git clone` the FoundationBuilder repository. It has two main branches: ‘master’ and ‘development. ‘Master’ is used only for stable releases. While ‘development’ is the main branch for coding. Below are some rules:

- Each feature should be implemented in separate developer-branch, branch name should correspond the pattern [DeveloperNickname]-[feature-name]. Tip: firstly, pull the recent changes: `git pull`, then move to development: `git checkout development`, create your branch: `git checkout -b your-branch-name`, commit your changes and push: `git push -u origin your-branch-name.`
- Team Member commits only into his current developer-branch. Whole solution at each commit should be in working state, i.e. it is not allowed to commit broken code in non-working state.
- After implementation is ready, Team Member pushes his branch to the origin and create pull request from his branch into the ‘development’ branch. Pull request should contain only some logically completed code snippet, preferably, only one feature. It is not allowed to stack all your code changes and pull request all your work at one time.
- Whole Team, as well as Tech Lead should be requested for the review of pull request.
- After each Team Member reviewed the pull request and, if all is OK, approved it, the final approving and merge will be done only by Tech Lead. Tech Lead starts review only after all other Team Members have approved pull request.

- Please, review the code of your teammates meticulously. If code contains some mistakes or errors or you see possible improving, please report your observations and, if necessary, request the Author to fix and refine its code. Remember, these development iterations are totally normal.

  For example, the development process may look as in the image below:



In this example, two developers created branches VG-triangulation and TeamMember1-point-bugfix. After they completed the implementations to some logical end, they opened pull requests to development branch. Once these pull requests were reviewed, they were merged into the development branch. Tech Lead will merge stable versions to master from time to time.

Start of Developing

Work on FoundationBuilder project is divided into 3 stages:

1st stage - familiarization with the project documentation and codebase; estimation of the scope of the work. The output: Excel table with planned tasks. *To be in a good shape, deadline to accomplish this stage is no longer than 2-3 day from the official beginning of the project.*

2nd stage - developing. Output: Application, which satisfy all requirements described in FoundationBuilder-Spec. At this point, each Team Member already has a workload. The whole team are working together on the implementation in accordance to the workflow (as described at the Git Workflow section). *This is the major part of the work, it should take about 3-4 weeks.*

3rd stage - finalization. Output: report. Once 2nd stage is completed, Team Lead should report the Tech Lead about the finishing of the developing. Each Team Member should prepare brief report about his work: what have you implemented, what problems have you faced with and how fixed them + suggestions/questions, if any.

Note: Team Lead is responsible to ensure that all stages are done in time; he should report to Tech Lead about any issues or if some work cannot be done in time.