Software Requirements Specification

Prepared by SMESHERS

for the project MESHTIKA*

METU - Department of Computer Engineering

CENG 491 Senior Design Project I

Fall 2015-2016

*** Project code-name; subject to change when release.**

Table of contents

Index of Tables

1. Introduction

This documentation is intended for anyone who wishes to understand how software Meshtika works. All system constraints, functionalities and elements included in the system are explained in detail. By using this documentation, developers can collaborate in order to shape the project to the final specification or they can extend the boundaries of the system for further demand. The documentation consists of four main parts, each targets a different audience. The first chapter is an introduction to the project where the software components and their purpose are described; the second chapter is the overall description of the project where the functionality is briefly described along with various interfaces used in the system; third chapter serves to a deeply elaborated requirements analysis and the final chapter; describes the data model of the project. One is free to read any chapter in any order. However, it is recommended to read the first one as the very first chapter defines the abbreviations required to understand this document, with some assumptions and constraints which play the key point when describing requirements of software components.

1.1 Problem Definition

Processing of digital geometries and of images, or modeling neural networks all require extensive visual debugging in development due to their nature of dealing with large amount of data, which is mostly nonsense for a naked-eye. Therefore development in these subfields of computer science usually goes parallel with a viewer, plotter or any kind of data visualizer.

Meshtika is a digital geometry processing toolkit, which offers a rich application programming interface with a fully integrated development environment which are both equipped and crafted for DGP specialists in target. In API, many DGP specific algorithms and data structures are offered; on the other hand in IDE, GUI supported development tools are offered. Although there exist tools in the software industry for geometry processing, they are frequently in the form of only programming interfaces. Thus, specialists of the field suffer from integrating 3rd party development tools on their own.

The basic idea behind the project is, therefore, to provide an all-in-one, rich and robust environment and tools for the target audience.

1.2 System Overview

We can split the products Meshtika offered in two parts. One is the API, the other is the IDE.

1.2.1 Meshtika API

API holds a rich digital geometry processing algorithms and data structures frequently used in DGP. The following packages are offered in product API: sampling, distance, voxelisation, descriptors, space-partitioning, mesh and visual.

Sampling	This package contains sampling algorithms.
Distance	This package contains distance algorithms.
Space-partitioning	This package contains space-partitioning algorithms.
Voxelisation	This package contains voxelisation algorithms.
Descriptors	This package contains structures for mesh recognition; i.e. to determining that two meshes are similar.
Mesh	This package contains relatively lower lever, but still user friendly mesh related API
Preferences	This package contains user preferences for IDE to use
Visual	This package contains visual debugger related API

Table 1: Packages in Meshtika API

See software design documentation for further information like the contents of each package.

1.2.2 Meshtika IDE

IDE holds several tools with GUI support. Following tools are offered in IDE: Code Editor, Interactive 3D Canvas, Info, Header, Step-by-step Debugger and Profiler.

Code Editor	Default code editor with capabilities: syntax highlighting, code completion, tab and basic text editor operations.
Interactive 3D Canvas	Default 3D canvas with trackball navigation, panning, zooming, peripheral event propagation capabilities.
Info	Default output panel with three channel output capability.
Header	Default component for communication between IDE and end-user.
Step-by-step Debugger	Default step-by-step classic debugger.
Profiler	Default profiler with memory consumption and timing measure capabilities.

Table 2: Tools in Mestika IDE

Please note that all six tools of IDE are GUI - supported.

1.3 Definitions, acronyms, and abbreviations

Below is a list of definitions, acronyms and abbreviations required to interpret this document properly.

DGP	Digital Geometry Processing
CGI	Computer Generated Imagery
API	Application Programming Interface
IDE	Integrated Development Environment
GUI	Graphical User Interface
SRS	Software Requirement Specification
SDD	Software Design Document
SSD	Step by Step Debugger
WYSIWYI	What you see is what you implemented
EU	End User
UML	Unified Modeling Language
editor	Code Editor (Text Editor, are used interchangeably) (software component)
canvas	Interactive 3D Canvas (software component)
info	Info (software component)
header	Header (software component)
profiler	Profiler (software component)
debugger	Step by Step Debugger (software component)
output	Anything written to Info is output
mesh	A data structure in computer graphics
vertex	A data structure in computer graphics

1.4 Assumptions and dependencies

Project Meshtika is built upon Blender ™ v.2.76. [1] However, maintenance of the project does not follow the new versions of the legacy code. Therefore, there is literally no dependencies for Meshtika.

2. Overall description

This section consists of inclusive explanations of main factors of Meshtika, i.e. product functionalities, user characteristics and limitations regarding the system.

2.1 Product functions

This section includes the brief information related to the functions provided by Meshtika. The information is divided into two subcategories regarding the explanations of the use cases and their actors, respectively.

2.1.1 Use-case model survey

The use-case model survey section consists of the functions that the project offers. In order to provide a better insight, an entire use case diagram is given below.B

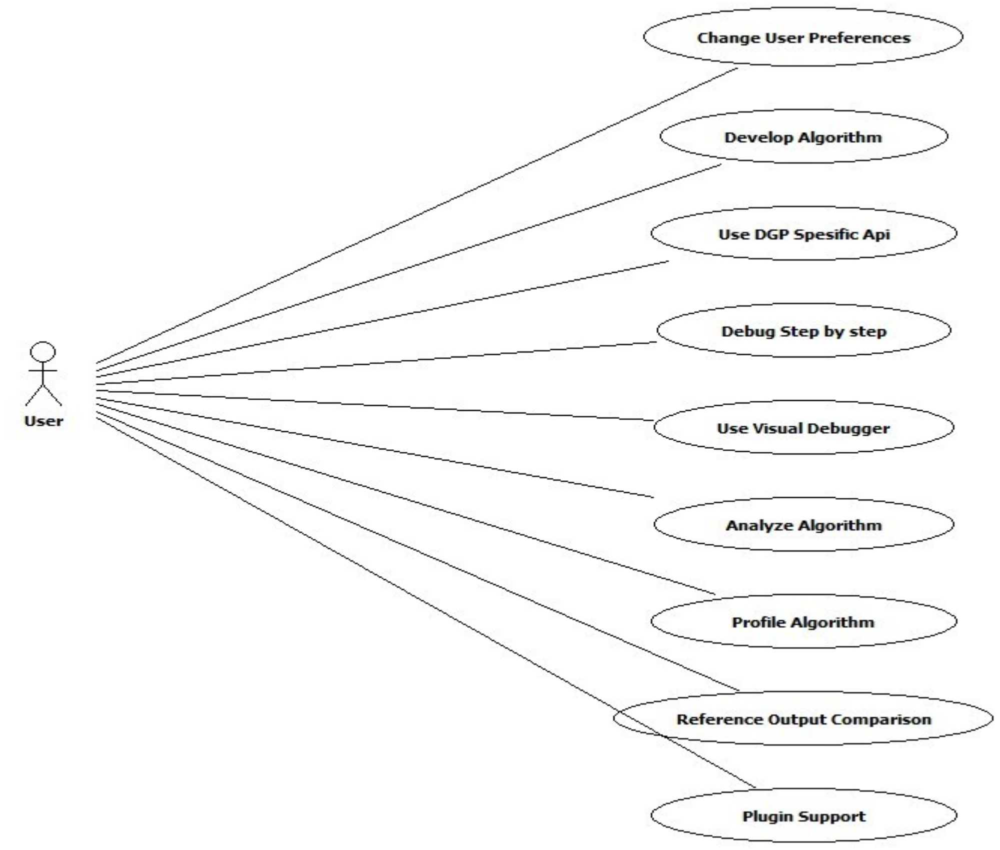


Figure 1: Use-case Diagram

In order to present what these use cases are actually for, their overall roles in the project, the actors related to them and the relations between them are briefly explained in the table below. Please also note that detailed explanations of these use-cases can be found under the section 3.1, regarding functional requirements.

Use-case Name	Description
Develop	Develop a custom DGP algorithm in the environment. IDE offers a robust text editor for this purpose.
Step-by-step Debug	Detect the logical errors the custom algorithm may have by classic step-by-step debugging technique
Visual Debug	Detect the logical errors the custom algorithm may have by visual debugging technique
Compare algorithm	Compare the behavior of the custom algorithm with a known, already implemented one
Use DGP API	Develop a custom DGP algorithm in the environment by an offered, easy-to-use API
Develop Plug-in *	Develop plug-ins that enhance the capabilities of Meshtika, rather than developing only custom algorithms for test purposes.
Customize IDE	Change predefined settings for the IDE (through preferences) for the suitable working environment

Table 3: Use Case Descriptions

* Plug-ins are supported by the project and licenses do not withhold the end-user for commercial benefit.

2.1.2 Actor survey

There is only one human actor for Meshtika, excluding software components. Throughout the document, he is called the end-user, which is sometimes abbreviated to EU. The target audience of Meshtika is people who are interested in digital geometry processing, therefore EU corresponds to any DGP specialist who can make use of the software.

2.2 Interfaces

Below are the interfaces the project relies on. The first subsection defines graphical user interfaces, second subsection defines hardware interfaces, and the last subsection, subsection the third, defines software interfaces required for Meshtika to operate.

2.2.1 User Interfaces

Meshtika provides seven main graphical user interface elements, each is a graphical representative of underlying software component. They are Header, Info, Preferences, Code Editor, 3D Canvas, Step-by-step debugger and Profiler.

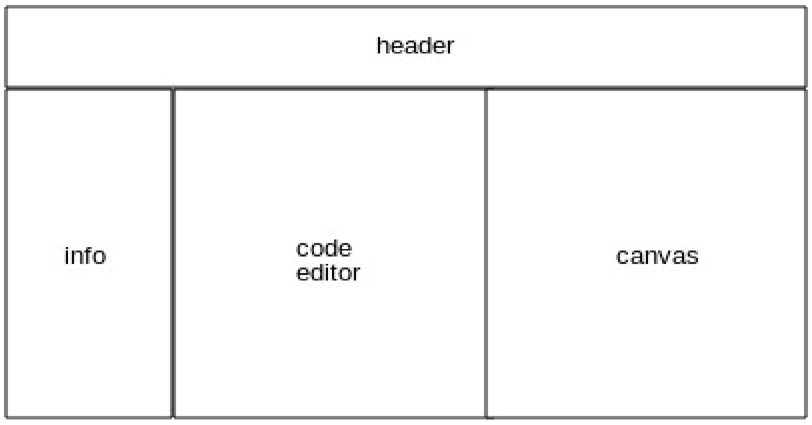


Figure 2: Depiction of the overall GUI

The default window layout consists of four of those elements. Since a profiler and a debugger is not frequently used, they are usually hidden. User can access profiler and debugger through Header. Tables including the detailed information related to these GUIs individually are given below.

GUI Name	Header
Description	Interface for end user to communicate with the IDE. For example, application specific files are overwritten, or the application is quited through Header.
Sub-Widgets	Three menus: File, Window, Help Two buttons: Debugger, Profiler A hidden label: Notifier (usually hidden)
Notes	File: A menu that contains easy-access to file r/w related operations. User also quits the IDE from this menu. Debugger: A button that opens up step-by-step debugger panel. Profiler: A button that opens up profiler. Window: A menu that contains easy-access to GUI related operations. User can select predefined window layouts. Help: A menu that contains easy-access to documentations about software use, or to software abouts. Notifier: Reports exceptions of IDE to the EU. Notifier is only visible when necessary; most of the time, this label is hidden. One example for the exceptions that Notifier reports is when a file is failed to write to file system.

Table 4: Header GUI

GUI Name	Info
Description	Interface for outputs. Whenever a print statement is executed, the output appears on Info, which has three output channels.
Sub-Widgets	None
Notes	There are three output channels. They are: 1. Output channel: Represents standard output. Anything on this channel is displayed as black text. 2. Error channel: Represents standard error. Anything on this channel is displayed as red text. 3. Warning channel: Represents standard warning, which is usually used for warnings that come from legacy code. Anything on this channel is displayed as orange text.

Table 5: Info GUI

GUI Name	Preferences
Description	Interface for user's customisations of Meshtika.
Sub-Widgets	Required settings for software components.
Notes	None

Table 6: Preferences GUI

GUI Name	Code Editor
Description	Code editor is where the end-user write scripts and this is the part of GUI that the end user is expected to interact with the most. It provides a robust environment to develop DGP algorithms by supporting necessary functionalities that an editor should provide.
Sub-Widgets	Two Menus: View and Text A Button: RunScript
Notes	View: A menu that contains view operations Text: A menu that contains basic text editor operations RunScript: A button that runs the current script Code Editor supports working with multiple scripts at a time via tab windows. The theme for Code Editor is customizable by the user in order to boost the user experience.

Table 7: Code Editor GUI

GUI Name	Interactive 3D Canvas
Description	Interface for visualizing 3D content. Uses trackball navigation, panning and zooming.
Sub-Widgets	None
Notes	Trackball navigation: Rotate the 3D camera around a pivot point, at a fixed length called zoom. Panning: Move camera w.r.t. its current orientation in a fixed plane. Zooming: Increase/decrease zoom length.

Table 8: Interactive 3D Canvas GUI

GUI Name	Step-by-step Debugger
Description	Interface for debugging commands
Sub-Widgets	Four buttons: Break, Step, Continue, Stop
Notes	Break: A button for putting a breakpoint for a line. When a line is selected in Code Editor (there is always one), press this button to put a breakpoint Step: A button that is used to jumping to the next line Continue: A button that is used to jump to the next registered breakpoint Stop: Stop debugging Step-by-step Debugger is integrated with the Code Editor to have the functionality of adding a breakpoint by simply clicking on a line.

Table 9: Step-by-step debugger GUI

GUI Name	Profiler
Description	Interface for algorithm Profiler. The Profiler inspects time and memory consumption of the custom algorithm.
Sub-Widgets	Two Line Charts: Time, Memory
Notes	Time: A line chart for time consumption of the custom algorithm, denoted in green. Memory: A line chart for memory consumption of the custom algorithm, denoted in yellow. Charts should be zoomable.

Table 10: Profiler GUI

2.2.2 Hardware Interfaces

In order for Meshtika to operate properly, only basic peripheral devices: a keyboard, a mouse and a display screen are required. Although graphics tablets and 3D mice are supported by the legacy software, they are not a requirement for this project.

2.2.3 Software Interfaces

Below is a list for required software packages for Meshtika to operate properly. Most of those libraries are requirements for Blender ™ v.2.76. However, since Meshtika is built upon that application, the legacy software interfaces are inherently requirement for the project.

Interface	Version	Description
ffmpeg	2.1.5	Decode-encode and stream algorithms for image and audio
OpenCOLLADA	1.3	Decode and encode algorithms for Collada 3D file format
OpenImageIO	1.4.16	Decode and encode algorithms for image
OpenColorIO	1.0.9	Color space conversions, algorithms for solving color problem on different hardware ie. Gamma correction
OpenSubdiv	3.0.2	A very adaptive subdivision algorithm which suppressed traditional Cutmull-Clark method, found by Pixar's R&D
OpenShadingLanguage	1.5.11	A shading language by Sony Pictures Imageworks, used in Blender's Cycles engine and visual debugger of Meshtika
OpenEXR	2.2.0	Operations on HDR images
ILMBase	2.2.0	Dependency for OpenEXR, found by Industrial Light & Magic
Boost	1.58	Utilities for C++, pioneered many lacks be compensated in C++11/14/17
Python	3.5.0	Python 3.5 interpreter to drive Meshtika
Numpy	1.10.1	Numerical Python, efficient python library for numeric operations. Most parts are implemented natively via cpython
Build-essential package	14.10	Software package of Ubuntu OS, which has OpenGL[2] and other core libraries

Table 11: Software Interfaces

2.3 Constraints

There are several constraints the project Meshtika should follow. These constraints apply for API and IDE parts of the software, and they are listed respectively as follows.

- API should let event based modeling; the project supports peripheral device events.
- API should let a development environment for developing algorithms for multiple meshes.
- API should not depend on operating system.
- API should only support development in Python programming language.
- IDE should be a use-once environment. That is, only user preferences are hold in file system, therefore only the changes in preferences are remembered between two successive runs.
- IDE should not let access to calls for all legacy libraries through its lifetime.
- IDE should have one scene at a time.
- IDE should always be initialized to an empty scene.
- IDE should not depend on operating system.

3. Specific requirements

This section holds the detailed information related to functional and non-functional requirements of the system and provides UML diagrams in order to give a strong understanding.

3.1. Functional Requirements

There are eight functional requirements for Meshtika, each of which is for the eight major use-cases, as stated in section 2.1. These will be elaborated individually, including their sequence diagrams, in the following subsections.

3.1.1. Change User Preferences

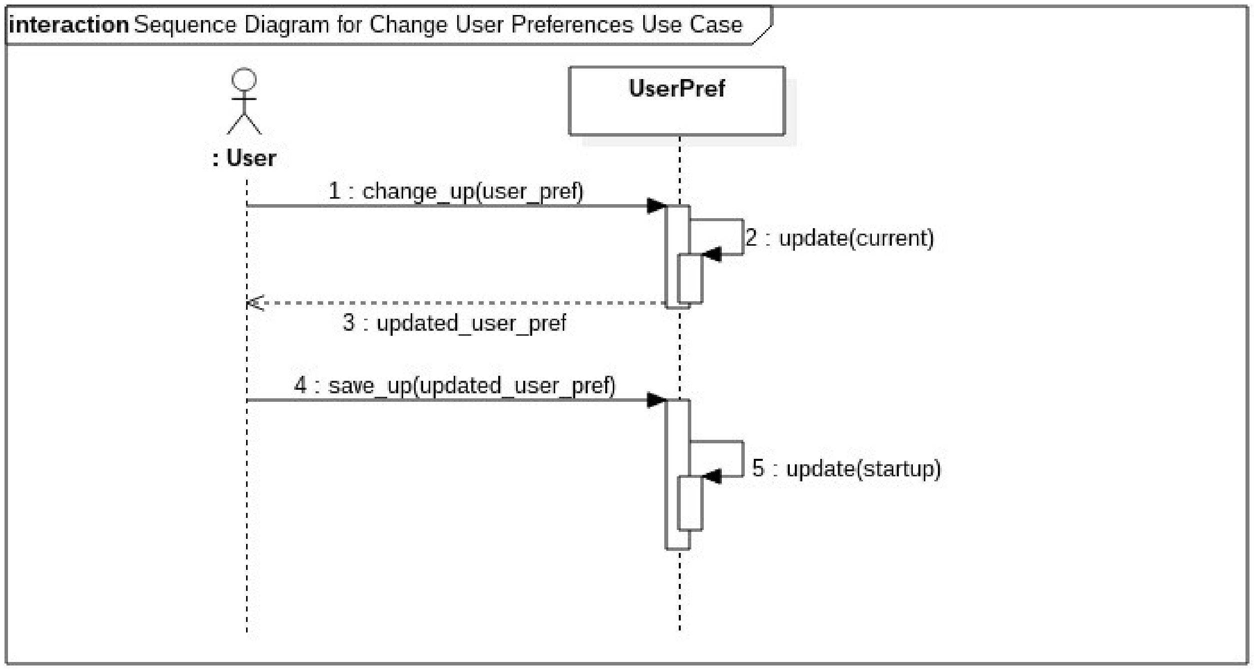


Figure 3: Sequence diagram of Change User Preferences Use Case

Use Case Name	Change User Preferences
Description	Users of Meshtika will have the power of customizing the platform, respecting to certain limits, via user preferences. They can perform actions, such as personalizing the theme of text editor, installing/ enabling/disabling add-ons, which are the correspondents of classic plug-ins in the scope of Blender
Primary Actor	End user
Secondary Actors	None
Preconditions	None
Postconditions	None
Trigger	Choosing the User Preferences from the File menu in the header GUI of the program
Scenario	As the user chooses User Preferences from the File menu, a pop-up window appears. Then, - user can change any of the given preferences in the window - observe its effect immediately, especially if the preference is related to the GUI. Later, if the user decides to stick with the changed version, - the user can save the updated preferences by clicking the button provided in the pop-up window -the updated version will now be used as startup version, which means that until the user makes a new change on preferences, the application will be started and used with the latest updated version of preferences.
Alternate Flow	None
Exception Flow	None
Extensions	None
Use case Notes	None

Table 12: Change User Preferences Use Case

3.1.2. Develop Algorithm

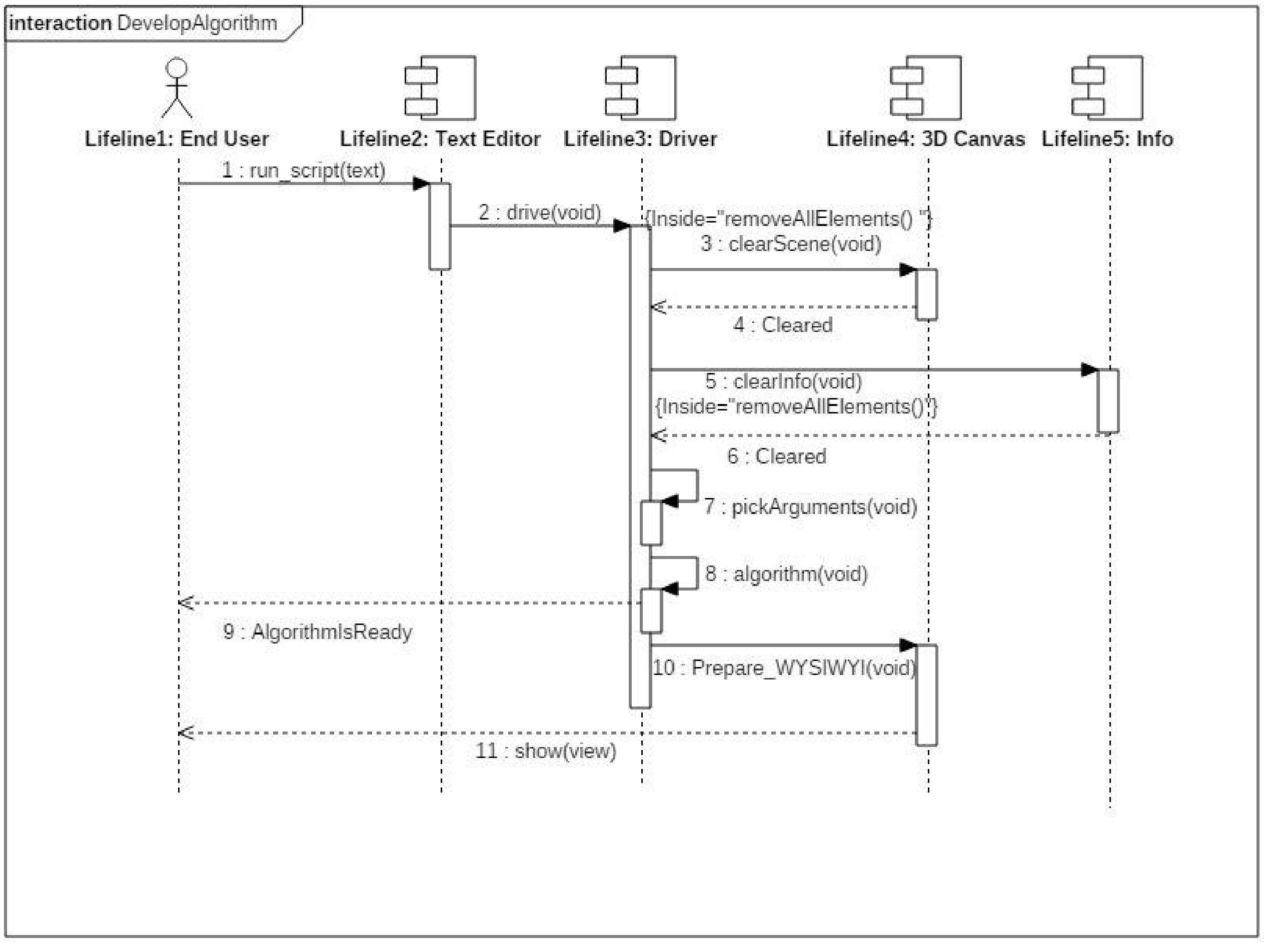


Figure 4: *Sequence diagram of Develop Algorithm Use Case*

Use Case Name	Develop Algorithm
Description	This is one of the four main use-cases of Meshtika: develop, profile, debug and compare an algorithm. User runs his implementation and sees the result in the interactive 3D canvas.
Primary Actor	End user
Secondary Actors	None
Preconditions	SSD is not running Comparison flag is not set
Postconditions	SSD is available to be run
Trigger	Pressing the 'RunScript' button of the code editor (text editor).
Scenario	After the primary actor writes a DGP script in the code editor, he presses the button 'Run Script'. This invokes the back-end driver, which makes some necessary preparations to run the code. Those preparations are as follows in order: 1. Initialize the scene by removing anything in the scene. 2. Initialize the output panel by removing any report in info. 3. Pick over the arguments of the custom algorithm. 4. Run the custom algorithm. 5. Invoke WYSIWYI of 3D Canvas. 6. Unless another algorithm is developed, i.e. 'RunScript' button is not pressed a second time, current algorithm is the active one.
Alternate Flow	None
Exception Flow	None
Extensions	If the step-by-step debugger is already running, this use-case is not available.
Use case Notes	None

Table 13: Develop Algorithm Use Case

3.1.3. Use DGP Specific API

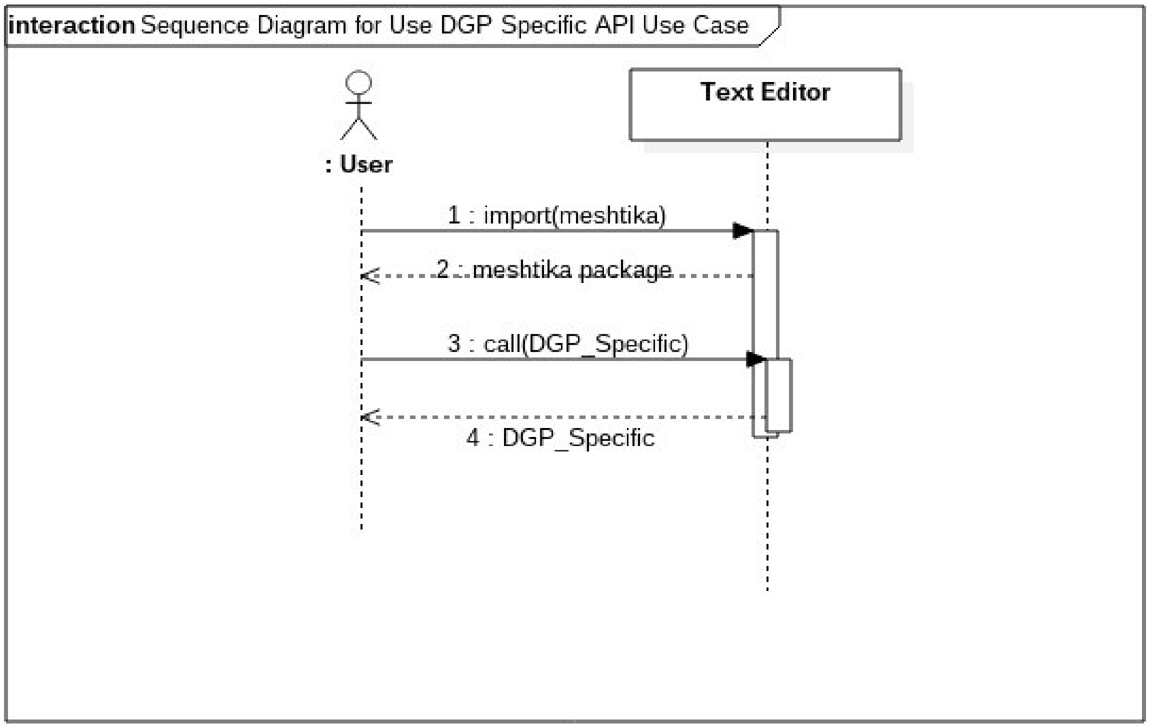


Figure 5: *Sequence diagram of Change User Preferences Use Case*

Use Case Name	Use DGP Specific API
Description	Most powerful side of Meshtika is its API product. That API has DGP specific algorithms and data structures. (see section 1.2.1)
Primary Actor	End user
Secondary Actors	None
Preconditions	None
Postconditions	None
Trigger	Importing meshtika packages
Scenario	Actor calls desired functions and uses desired data structures both defined in appropriate packages. They are DGP_Specific in the diagram above.
Alternate Flow	None
Exception Flow	None
Extensions	None
Use case Notes	None

Table 14: Use DGP Specific API Use Case

3.1.4. Analyze Algorithm

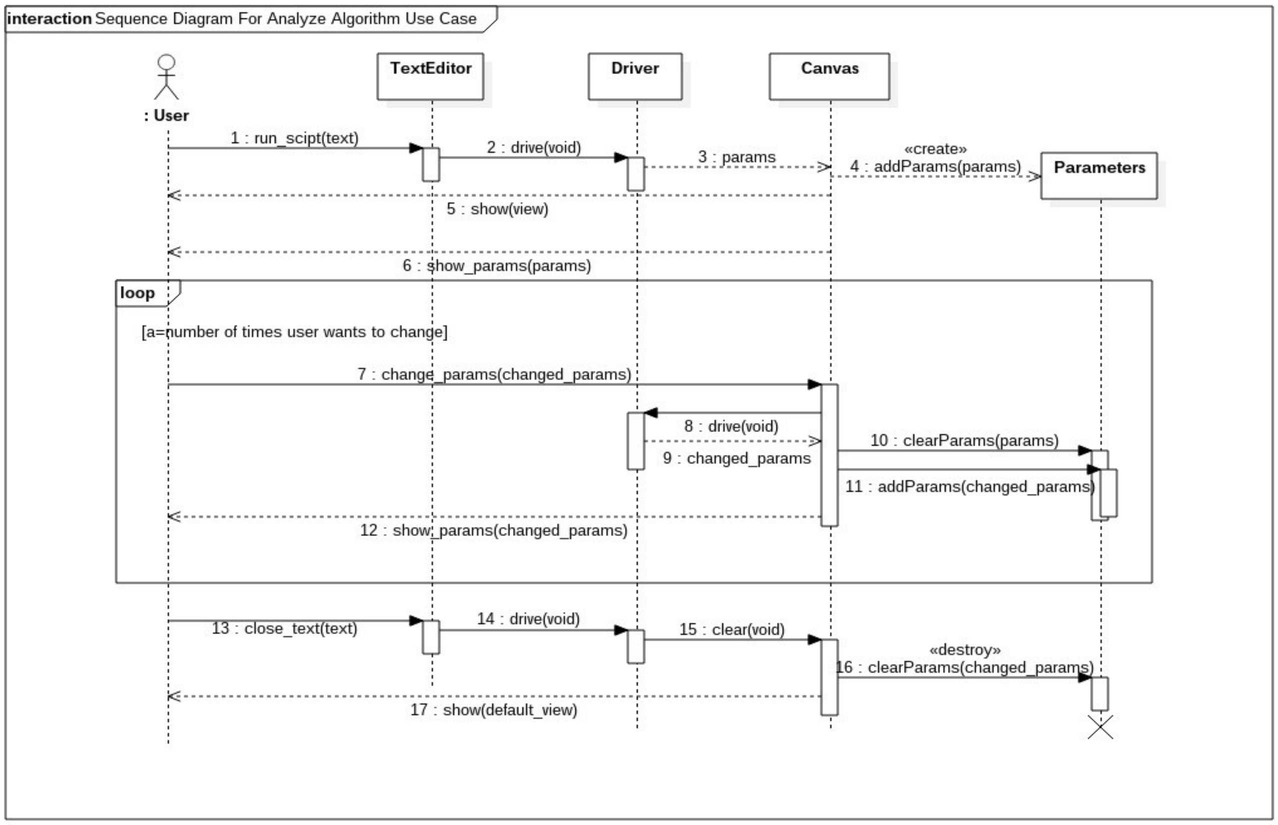


Figure 5: Sequence diagram of Analyze Algorithm Use Case

Use Case Name	Analyze Algorithm
Description	Meshtika lets its users to experience a “What you see is what you implement” environment. This implies that users are able to track the outcomes of changing dependents of their algorithms, by serving them the parameters of their own scripts as editable entities and update the consequent view accordingly.
Primary Actor	End user
Secondary Actors	None
Preconditions	A script should be existing and ready-to-run in the text editor.
Postconditions	None
Trigger	Pressing the run script button in the text editor.
Scenario	-First the script is run from the text editor -Accordingly, the back-end-driver is triggered for taking some necessary actions, which are already explained in section 3.1.2. An addition to those is providing parameters in the canvas. -The driver deducts the parameters used in the user's algorithm and shows them as editable entities in a side panel of the canvas. Once the parameters are provided, -The user can make changes on as much parameters and as many times as he/she wants. As a result of this action, -The driver realizes this change and recomputes them, provides the mesh on the canvas corresponding to the updated version of parameters. The flow here is similar with user running his/her script with updating the parameters manually in the text editors, but the distinction is that the user is provided the parameters individually in a part of GUI and does not need to rerun the script him/herself. Therefore, this use case is a quite beneficial one for the users in order to track their changes, especially to observe which effect is caused by which parameter adjustment.
Alternate Flow	None
Exception Flow	None
Extensions	None
Use case Notes	None

Table 15: Analyze Algorithm Use Case

3.1.5. Debug Step by Step

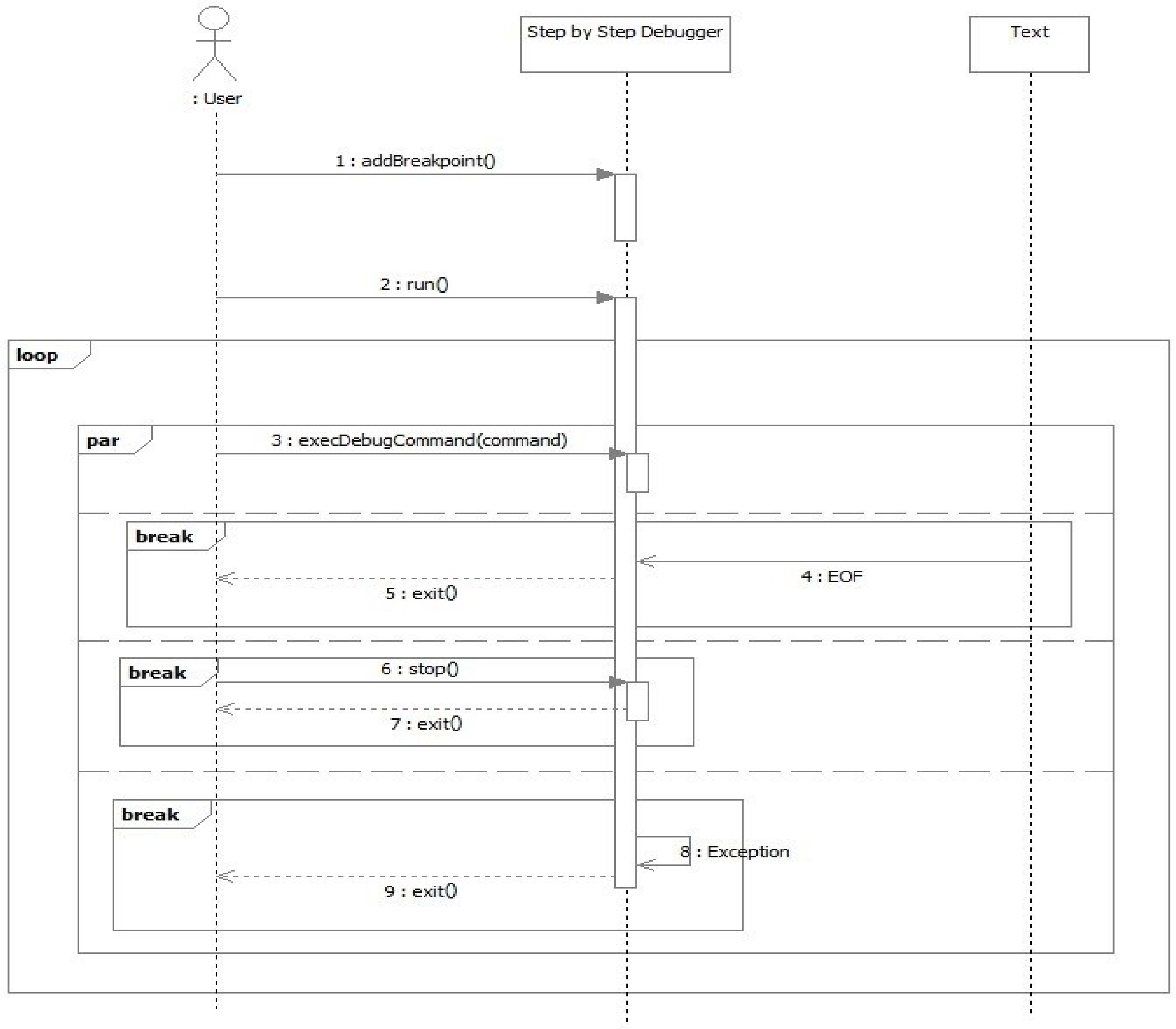


Figure 6: Sequence diagram of Debug Step-by-Step Use Case

Use Case Name	Debug Step by Step
Description	User debugs by classical step by step approach
Primary Actor	End User
Secondary Actor	None
Trigger	End user puts at least one breakpoint and clicks Start
Postconditions	'Run Script' becomes enabled
Preconditions	'Run Script' should be enabled
Scenario	- User puts at least one breakpoint for a line-by-line - User press 'run' - Debugger stops at breakpoint(s) and prints to local info when appropriate - Debugger is killed if EOF is reached
Alternate Flow	Debugger is killed if User presses 'stop' button
Exception Flow	Debugger is killed if an exception is thrown
Extensions	Can work cooperatively with visual debugger
Notes	None

Table 16: Debug Step by Step Use Case

3.1.6. Use Visual Debugger

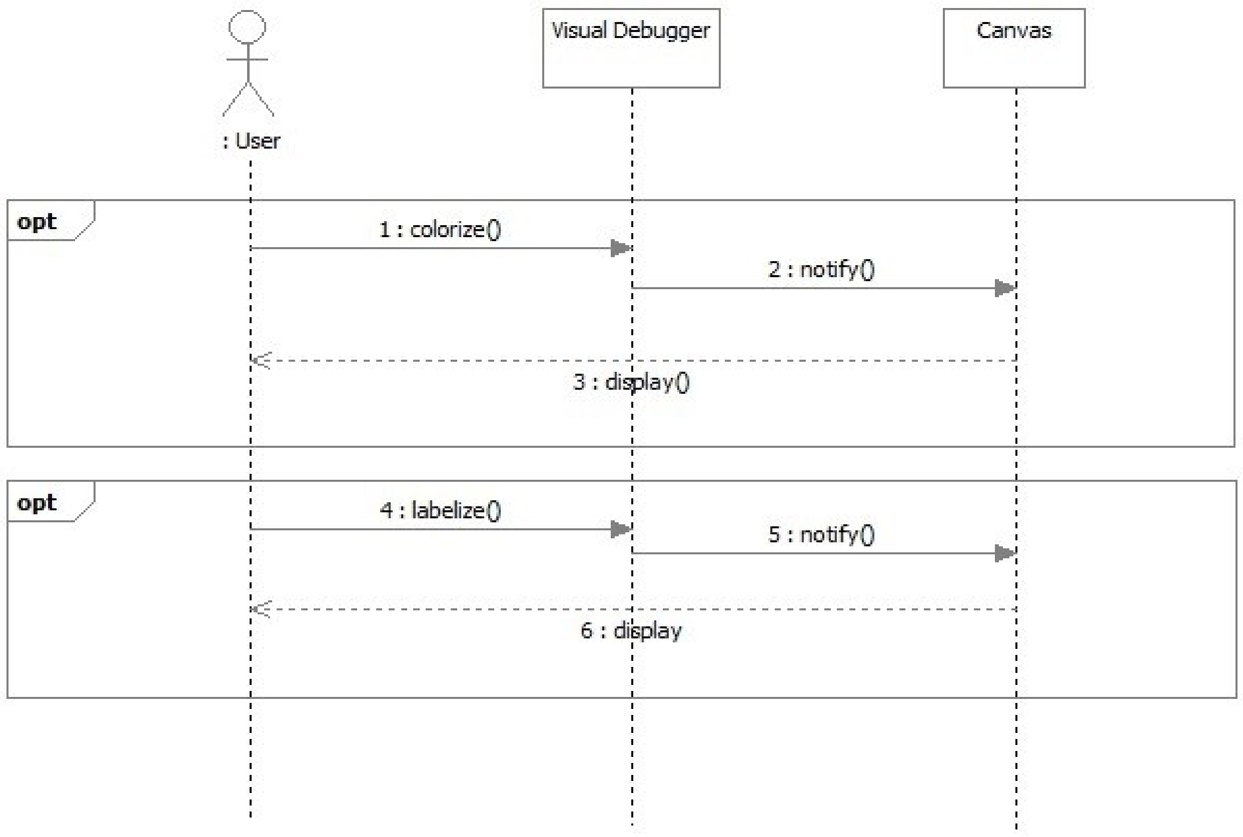


Figure 7: Sequence diagram of Use Visual Debugger Use Case

Use Case Name	Use Visual Debugger
Description	User debugs algorithm visually
Primary Actor	End User
Secondary Actor	None
Trigger	End user makes appropriate API calls
Postconditions	None
Preconditions	None
Scenario	- User imports Meshtika packages - User calls visual debugger functions with intended visualization arguments - User presses 'Run Script'
Alternate Flow	None
Exception Flow	None
Extensions	Can work cooperatively with step-by-step debugger
Notes	None

Table 17: Use Visual Debugger Use Case

3.1.6. Profile Algorithm

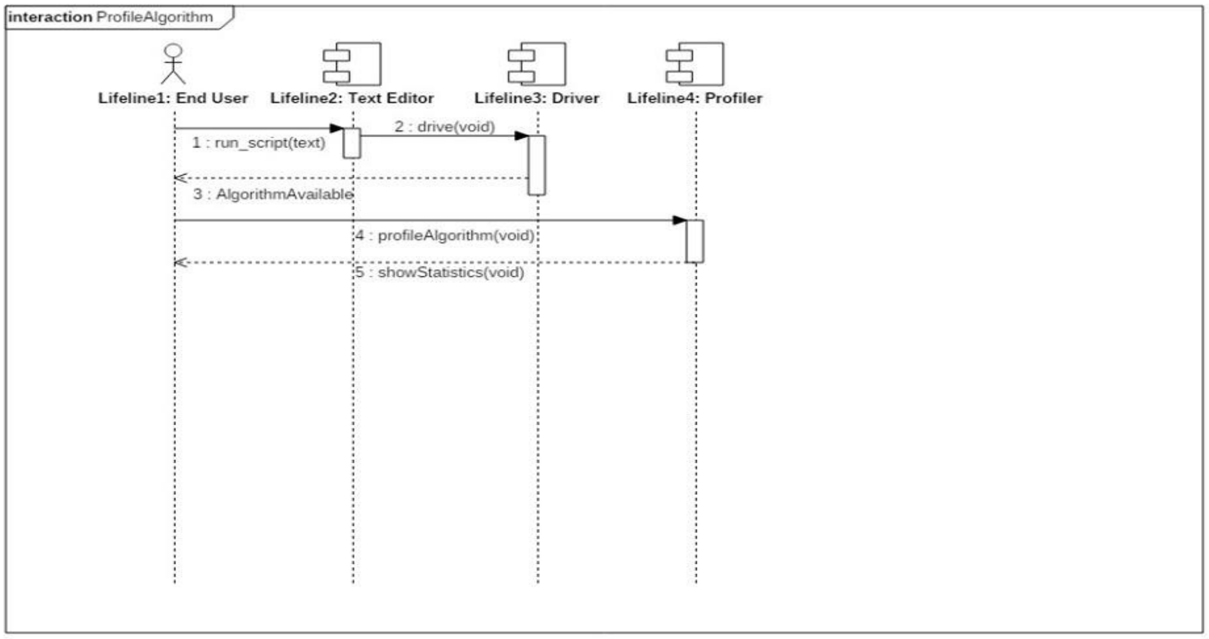


Figure 8: *Sequence diagram of Profile Algorithm Use Case*

Use case name	Profile Algorithm
Description	The user can make performance measurements and observe memory and time usages of his/her work .
Primary Actor	End User
Secondary Actor	None
Preconditions	At least one work (i.e algorithm) has to be runned.
Postconditions	None.
Trigger	Pressing “Run Script” button of the code editor. Opening the Profiler pop-up window.
Scenario	This use case is very bounded to the use case named “Develop Algorithm.”. After pressing “Run Script” button, the steps are the same up to and including “Run the custom algorithm” step. After this step : Step1 : The user opens the Profiler pop-up window. Step2 : The memory and time consumption of the algorithm is available in the Profiler window.
Alternate Flow	None
Exception Flow	If the written script has compile-time or run-time errors, the algorithm intended to be run will not be run and there will not be any statistics. The user must debug his/her source code.
Extensions	None
Use Case Notes	None

Table 18: Profile Algorithm Use Case

3.1.7. Reference Output Comparison

There is no sequence diagram for this use case as it is quite similar to the Develop Algorithm use case depicted in 3.1.2.

Use Case Name	Reference Output Comparison
Description	This is one of the four main use-cases of Meshtika: develop, profile, debug and compare an algorithm. User runs his implementation and sees the result in the interactive 3D canvas.
Primary Actor	End user
Secondary Actors	None
Preconditions	SSD is not running Comparison flag is set Comparison base algorithm is assigned
Postconditions	SSD is available to be run
Trigger	Pressing RunScript button of code editor.
Scenario	After the primary actor writes a DGP script in the code editor, he presses the button 'Run Script'. This invokes the back-end driver, which makes some necessary preparations to run the code. Those preparations are as follows in order: 1. Initialize the scene by removing anything in the scene. Duplicate the loaded meshes and assign them ghost shader. * 2. Initialize the output panel by removing any report in info. 3. Pick over the arguments of the custom algorithm. 4. Run the custom algorithm. 5. Invoke WYSIWYI of 3D Canvas. 6. Unless another algorithm is compared, i.e. 'RunScript' button is not pressed a second time, current algorithm is the active one.
Alternate flow	None
Exception flow	None
Extensions	If the step-by-step debugger is already running, this use-case is not available.
Use case Notes	* This will let the actor compare the meshes loaded originally and the meshes on which the custom algorithm is applied.

Table 19: Reference Output Comparison Use Case

3.1.8. Plug-in Support

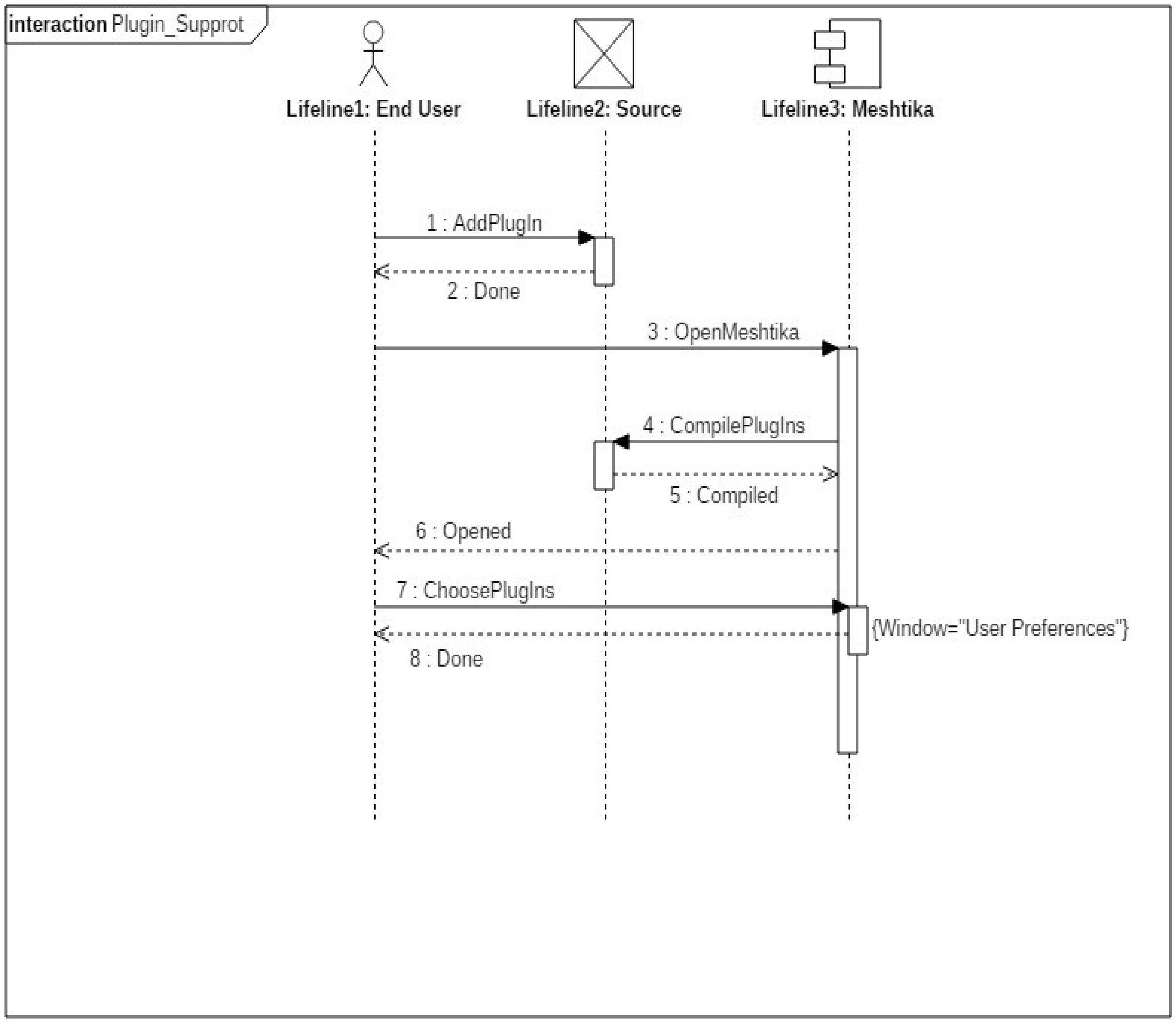


Figure 10: Sequence diagram of Plug-in Support Use Case

Use case name	Plug-in Support
Description	The user can implement new features like functionalities or new data types and register them to Meshtika. The user can also make use of extra features which are implemented by other Meshtika users.
Primary Actor	End User
Secondary Actor	None
Preconditions	The user has to know how to extend Meshtika.
Postconditions:	If the user choose newly added plug-in(s) for using them in his/her work, these preferences will be kept.
Trigger	None
Scenario:	Step1: The user implements functionalities. Step2: The user registers the functionalities newly written. Step3: The user opens Meshtika. When it is opened, the plug-in source codes will be compiled and available in the application. Step4: The user opens “User Preferences” window. Step5: The user activates the new plug-in from application. Step6: The plug-in is ready to use.
Alternate Flow	Instead of Step1 and Step2, the user can use other plug-ins which are implemented by other users. The user puts the plug-in source codes to the related directories and the scenario continues with Step3.
Exception Flow	None
Extensions	None
Use Case Notes	None

Table 20: Plug-in Support Use Case

3.2 Nonfunctional Requirements

3.2.1 Usability

Although the precise usability can be observed once the target audience starts to use Meshtika, it is possible to state the following requirements in this scope:

- The user does not need to tackle with any problems regarding the GUI as it aims to be a simple but functional one.

- It is expected that the user would not need to use any broad documentation or any other 3rd party services related to GUI usage, which is quite a big change for this field's enthusiasts. For instance, our base Blender requires several documentation and tutorial aids in order to be able to make use of its GUI appropriately. [3]

- All API has a learning curve. However, API in Meshtika requires relatively shorter time when compared to other competitors. There are two reasons for that. First, API in Meshtika is designed in a way that learning it takes a little time for EU. Second, the target audience of the project is people who are already interested in DGP, they are already accustomed to using much harder environments, therefore Meshtika is an easy toolkit for them.

- User is not retained to develop a custom algorithm for any time limit.

- The time for providing the resulting mesh, parameters, debugging and profiling services are aimed to be kept minimum.

3.2.2 Reliability

As the application is standalone, does not depend on any server or database, there is no concern of keeping that kind of components up and running all the time. This indicates that there will not be any availability problems either.

Accordingly, as long as there is no problem with the user's own operating system, it is expected that the program will go on processing smoothly.

3.2.3 Performance

Performance requirements for Meshtika can be categorized into three. They are as follows:

1. Response Time:

   For a mesh with 40K vertices, a naive algorithm should response in 0.5 seconds in average.

2. Capacity:

   The application is standalone, and accepts one user at a time. There is no capacity requirement.

3. Resource utilization:

   • 32-bit dual core 2Ghz CPU with SSE2 support.

   • 2 GB RAM

   • OpenGL 2.1 compatible graphics card with 512 MB RAM

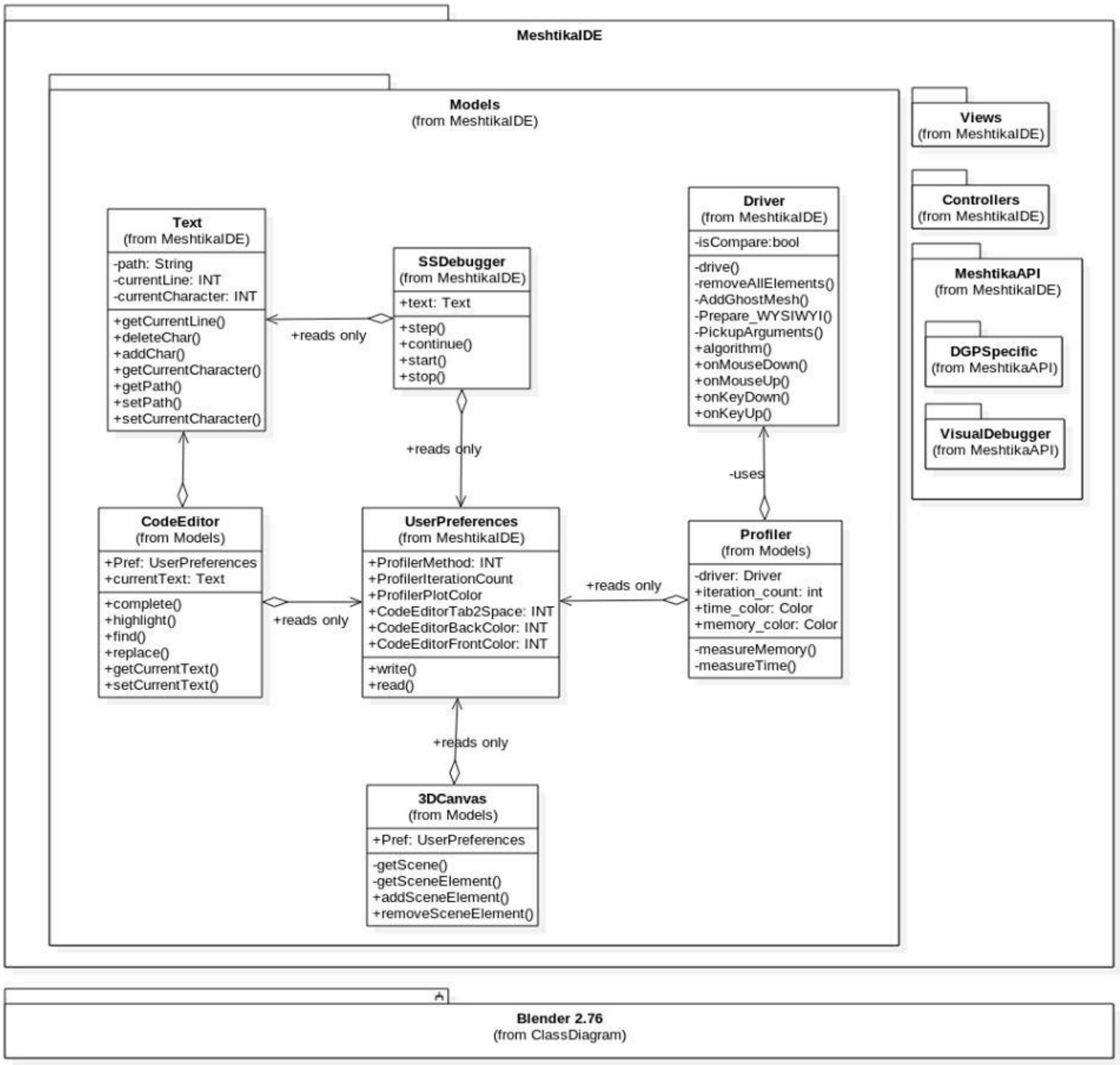
3.2.4 Supportability

The users will write their scripts in Python, which involves huge support and maintenance itself. Moreover, the naming conventions supplied the API of Meshtika will be as much easy-to-use as possible, in order not to damage that.

Support for any kind of problem can be provided by the forums involving developers, i.e. community, as it is expected for any open-source project, and the mean time to repair is tried to be kept at minimum via supplying and maintaining the most beneficial information.

4. Data Model and Description

This section will give the information related to the data objects that will be managed by out software. In order to achieve this, a class diagram representing the main parts of our toolkit and their relations is provided. Moreover, a class dictionary that explains these classes in detail, including their attributes, methods and relationships with other classes, in other words their roles in Meshtika, is given underneath the diagram.


MeshtikaIDE
Models
(from MeshtikaIDE)
Text
(from MeshtikaIDE)
-path: String
-currentLine: INT
-currentCharacter: INT
+getCurrentLine()
+deleteChar()
+addChar()
+getCurrentCharacter()
+getPath()
+setPath()
+setCurrentCharacter()
SSDebugger
(from MeshtikaIDE)
+text: Text
+step()
+continue()
+start()
+stop()
+reads only
Driver
(from MeshtikaIDE)
-isCompare:bool
-drive()
-removeAllElements()
-AddGhostMesh()
-Prepare_WYSIWYI()
-PickupArguments()
+algorithm()
+onMouseDown()
+onMouseUp()
+onKeyDown()
+onKeyUp()
+reads only
-uses
CodeEditor
(from Models)
+Pref: UserPreferences
+currentText: Text
+complete()
+highlight()
+find()
+replace()
+getCurrentText()
+setCurrentText()
+reads only
UserPreferences
(from MeshtikaIDE)
+ProfilerMethod: INT
+ProfilerIterationCount
+ProfilerPlotColor
+CodeEditorTab2Space: INT
+CodeEditorBackColor: INT
+CodeEditorFrontColor: INT
+write()
+read()
+reads only
Profiler
(from Models)
-driver: Driver
+iteration_count: int
+time_color: Color
+memory_color: Color
-measureMemory()
-measureTime()
+reads only
3DCanvas
(from Models)
+Pref: UserPreferences
-getScene()
-getSceneElement()
+addSceneElement()
+removeSceneElement()
Views
(from MeshtikaIDE)
Controllers
(from MeshtikaIDE)
MeshtikaAPI
(from MeshtikaIDE)
DGPSpecific
(from MeshtikaAPI)
VisualDebugger
(from MeshtikaAPI)
Blender 2.76
(from ClassDiagram)

UserPref	UserPref class holds the attributes of what user can actually change on the GUI of Meshtika. It is mostly similar to Blender, but involving changes in Themes section etc. It only has the save_up() method as the user would like to experience with the same settings for the upcoming sessions as well. We should note that as our tool does not provide any account/profile creation, no User class in the system either, these settings are kept in the tool itself,i.e. the installed programme, rather than actually being tied to the user.
TextEditor	This is one of the main parts of Meshtika, as its goal is to provide a development environment. TextEditor class holds the methods similar to other famous multi-functional text editors, that developers tend to use. It lets the developers to make use of actions such as higlighting, line numbering, find&replace mechanism and of course code completion, on Text objects. This class is not solely composed of Text objects, but rather performs on them as if the Text class is the part and TextEditor class is the whole, as this can be driven from the given aggregation.
Text	Text class is the script that the user writes on the TextEditor. The user can make use of the methods of TextEditor to manipulate the Text. We should point out that a Text Editor class can obtain more than one Text objects, thanks it its tab mechanism, but can works, perform its methods, on only one at a time.
SsDebugger	SsDebugger class represents a classic step-by-step debugger for Python, including the methods step(), next(), continue() and kill(). With the help of these methods the user can debug his/her script, i.e. the current Text object, line-by-line or block-by-block.
Canvas	Canvas class demonstrates the 3d View space of the tool. The user will be provided by the visual output of his/her script, i.e. the current Text Object on the Canvas. Whenever a script is run, the parameters belonging to it will be shown on the canvas and the user will be able to change them to see their effects on the visual output immediately, by making use of addParams() and clearParams().
DGP_Specific_API	Built-in DGP related functions and data structures in API.
VisualDebugger	Other than the SsDebugger, Meshtika supplies its users also a VisualDebugger class. Users can make use of this class by calling the VisualDebugger module inside the DGP_Specific_API, in their scripts, and the visual output, corresponding to the position where the module is called, will appear with appropriate colors and labels. This lets the user to keep track of visual changes easily.

Mesh	Mesh class is actually the minimalist output a DGP specialist will drive. So, it is safe to assume that whole script, the Text, will be written in order to create Meshes and perform distinct algorithms related to them via benefiting from the DGP_Specific_API of Meshtika. The necessary attributes such as its vertices, edges and faces are kept as the actions, related to the algorithms, will be performed on these.
Driver	Handles the algorithm phases, provides the output to canvas.
Profiler	Developing algorithm requires consideration of time and memory usage. Although, both of these can be estimated hypothetically, Meshtika aims to give its users to performance measurements in a real world environment. The profiler class has methods to inspect time and memory usage of the implemented algorithm as well as its subroutines. Moreover, Profiler plots a chart of the running time of the algorithm for meshes with different sizes, so user can observe time complexity of the algorithm.

5. References

[1] Blender.org - Home of the Blender project - Free and Open 3D Creation Software. (n.d.). Retrieved January 13, 2016, from https://www.blender.org/

[2]OpenGL 2.1 Reference Pages. (n.d.). Retrieved January 13, 2016, from https://www.opengl.org/sdk/docs/man2/

[3]Blender Manual Contents¶. (n.d.). Retrieved January 13, 2016, from https://www.blender.org/manual/