MIDDLE EAST TECHNICAL UNIVERSITY
COMPUTER ENGINEERING
DEPARTMENT

SOFTWARE TEST DOCUMENT

Group Name : Smeshers
Group Members : Uğur Yanıkoğlu
Furkan Odluyurt
Dicle Ayzit
Emre Barış

Advisors : Yusuf Sahillioğlu
Çağlar Seylan

Tests of the components of Digital Geometry Processing Toolkit, Meshtika.

Table of Figures

List of Tables

1. INTRODUCTION

1.1 Document identifier

This document can be used as a reference for the methods used to test our system, Meshtika and also can be viewed as a user guide, as users of the application may deal with the same cases while making use of our GUI and API.

1.2 Scope

This document will give information about the test cases by explaining architecture, subject and constraints of the test plans. Furhermore, the results that are are achieved from our experiments will be shown seperately to show how well the system is handling the situations that may occur in real world. In addition, the scope of the document includes the strategies and techniques that are needed to conduct those test cases.

1.3 Definitions, Acronyms & Abbreivations

DGP	Digital Geometry Processing
GUI	Graphical User Interface.
API	Application Programming Interface
IEEE	Institute of Electrical and Electronics Engineers.
IDE	Integrated Development Environment
SRS	Software Requirements Specification

SDD	Software Design Description
STD	Software Test Document
WYSIWYI	What You See is What You Implemented
CE	Code Editor (Software Component)
Mesh	A data structure in Computer Graphics
Output	Anything written to Info is output

Table 1: Table of Abbreviations

1.4 References

- IEEE 10162009, IEEE Standard for InformationTechnology-Systems Design-Software Design Descriptions
- Software Requirements Specification Document for Meshtika (2015), from http://senior.ceng.metu.edu.tr/2016/smeshers/
- Software Design Description Document for Meshtika (2016), from http://senior.ceng.metu.edu.tr/2016/smeshers/
- Blender Manual Contents (n.d.). Retrieved January 13, 2016, from https://www.blender.org/manual/
- Blender.org - Home of the Blender project - Free and Open 3D Creation Software. (n.d.). Retrieved January 13, 2016, from https://www.blender.org/

1.5 Level in the overall sequence

The system is not a part of any larger system but a small scale one, as we tried to achieve by scaling down the platform we build Meshtika on, which is Blender. Therefore, only one level of test planning is enough for this system and it is the test of the product itself.

1.6 Test classes and overall test conditions

For this level of testing, whole system is needed to be considered. Since the system should be tested to validate and verify the features of itself, the functionailities which were described in SRS will be tested. Hence, testing for this system should be conducted to meet its requirements. Test cases are divided into main test classes. Those classes are listed below:

-Manual testing

This method is used for Text Editor and Info Space to see if any errors happen, if all properties working correctly.

-Black Box Testing

Used for testing the algorithms provided in the Meshtika API, to see if the outcomes meets the expected ones.

2. Details for system test plan

2.1 Test items and their identifiers

We have built an integrated developing platform that does not include any harware components. Its software components that are tested defined in the next sections in detail. In this project, we use black box testing methods. We will test the use cases and features that are defined in the SRS document by generating different test scenarios. Then we decided to outputs according to pass/fail criterias. And at some point we also use bare eyes to test some conditions, as we are dealing with a branch of Computer Graphics. The contents of test approaches will become more clear in following sections.

2.2 Details of the System Test Design

For each test case we prepared tables to give details of each test procedure.

Test Case ID	T01
Test Case Name	Testing of Code Editor
Approach	Perform development on Code Editor, try to make use of its properties and run the script
Pass/Fail Criteria	Code editor needs to properly suffice its properties and does not get stuck during developing
Deliverables	Possibly the result on 3D View

Table 2: Test Case 1

Test Case ID	T02
Test Case Name	Testing of Info Space (The Log Space)
Approach	Perform on Code editor and catch the corresponding logs on Info Space
Pass/Fail Criteria	The log should meet the actions taken the developer
Deliverables	Its log tiself

Table 3: Test Case 2

Test Case ID	T03
Test Case Name	Testing of Meshtika API (The provided algorithms package)
Approach	Try to import any module from meshtika inside the script
Pass/Fail Criteria	It should be able to import and be used inside the script with no error meddling
Deliverables	-

Table 4: Test Case 3

Testing of Meshtika API’s individual algortihms will actually divide itself into many parts, as we needed to test every algorithm we integrated into it, but this table will give the general approach and its details we took while performing those individual tests.

Test Case ID	T04
Test Case Name	Testing of individual provided algorithms in Meshtika API
Approach	Specific inputs given and checking the output mesh/numeric value if it meets the expected outcome (pointed by the supervisor and the corresponding paper)
Pass/Fail Criteria	It should match the expected output mesh/numeric value
Deliverables	Result mesh on 3D View or numeric value on Info Space

Table 5: Test Case 4

Test Case ID	T05
Test Case Name	Testing of Profiler
Approach	Comparing the resource and time parameters of top command in terminal with the output of profiler
Pass/Fail Criteria	If the graphic that the profiler provides meet the parameters of top command, it works appropriately.
Deliverables	Graphic displaying memory used vs time passed

Table 6: Test Case 5

Test Case ID	T06
Test Case Name	Testing of Debugger
Approach	Checking if the basic deugger functions, such as run, next, break etc., work properly
Pass/Fail Criteria	The debugger functions should correspond to the expected actions during the traversal of the script
Deliverables	Real-time debug log

Table 7: Test Case 6

Test Case ID	T07
Test Case Name	Testing of WYSIWYI environment
Approach	Comparing the result of manual changes of parameters in the code editor itself with the result of making the changes in WYSIWYI environment. Also, making sure that all registered parameters appear in the environment.
Pass/Fail Criteria	If the results match and there is no problem of accessing the registered parameters in WYSIWYI environment, the test is passed.
Deliverables	-

Table 8: Test Case 7

3. Test management

3.1 Planned activities and tasks

The test activities considered for this system is listed below.
- Unit testing
- Integration testing
- Validation testing

Among those test activities, unit testing must be done first, then integration testing and finally validation testing. In unit testing phase, each component of the system must be tested separately. Unit testing must be done again and again until each unit succeeds in those tests. In this way, we make sure that the problem does not arise from the units when something wrong occurs during other test activities. After all units pass the tests successfully, integration test activity must be done. In this one, units are added to the system one by one. When one unit is added to the system, this merged version will be tested and if it passes the test, new one is added and this procedure will be carried out until all units are merged together. After integration test is over,validation test activity must be conducted. Therefore, the results of tests will be compared with those requirements in validation test.

3.2 Environment/infrastructure

Our project is developed on Ubuntu 14.04 and our programming language preference is mostly Python and we needed to deal with C on the way as well. We have only one test environment, which is Meshtika itself, by also making use of Blender’s basis.

4 Test case details

4.1 Testing of Code Editor

Actually making use Code Editor during development and try to break by making use of its promised properties. Eye inspection mostly for checking properties.

4.1.1 Objective

To make sure that Code Editor meets its requirements specified in SRS.

4.1.2 Input(s)

Any Python script.

4.1.3 Outcome(s)

Assuming no errors caused by the script itself, no errors actually happened beacuse the editor itself. Also cecking if an outcome supposed to appear on 3D view.

4.1.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.1.5 Special procedural requirements

The user interface should be running.

4.2 Testing of Info Space (The Log Space)

We need to make sure that we get required explanations corrsponding to the actions taken in Code Editor, as in any IDE.

4.2.1 Objective

To make sure that the log displayed in Info space matches the actions taken in Code Editor.

4.2.2 Input(s)

Any Python script in Code Editor and running it.

4.2.3 Outcome(s)

Displayed possible wirtten outcomes, any errors or exceptions caused by the script itself.

4.2.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.2.5 Special procedural requirements

The user interface should be running.

4.3 Testing of Meshtika API (The provided algorithms package)

We need to make sure that the developers using our platform can make use of the API we are providing to boost the user experience and make our contribution work as well.

4.3.1 Objective

To check if one can import any algorithm from the API.

4.3.2 Input(s)

None. The API is already integrated into the platform. The user just needs to make use of this snippet.

```
from meshtika import *
```

4.3.3 Outcome(s)

- (No errors should be displayed in the log.)

4.3.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.3.5 Special procedural requirements

The user interface should be running.

4.4 Testing of individual provided algorithms in Meshtika API

We also definitely have to make sure that the algorithms we provided produce specified outcomes, which are specified in the corresponding academic papers and also via the supervisor.

4.4.1 Objective

The objective is valid for every algorithm; make sure they are providing expected results.(meshes/numeric values)

4.4.2 Input(s)

For each algortihm we provide a suitable non-manipulated triangle mesh, such as bunny, man and centour. They are provided below, respectively.

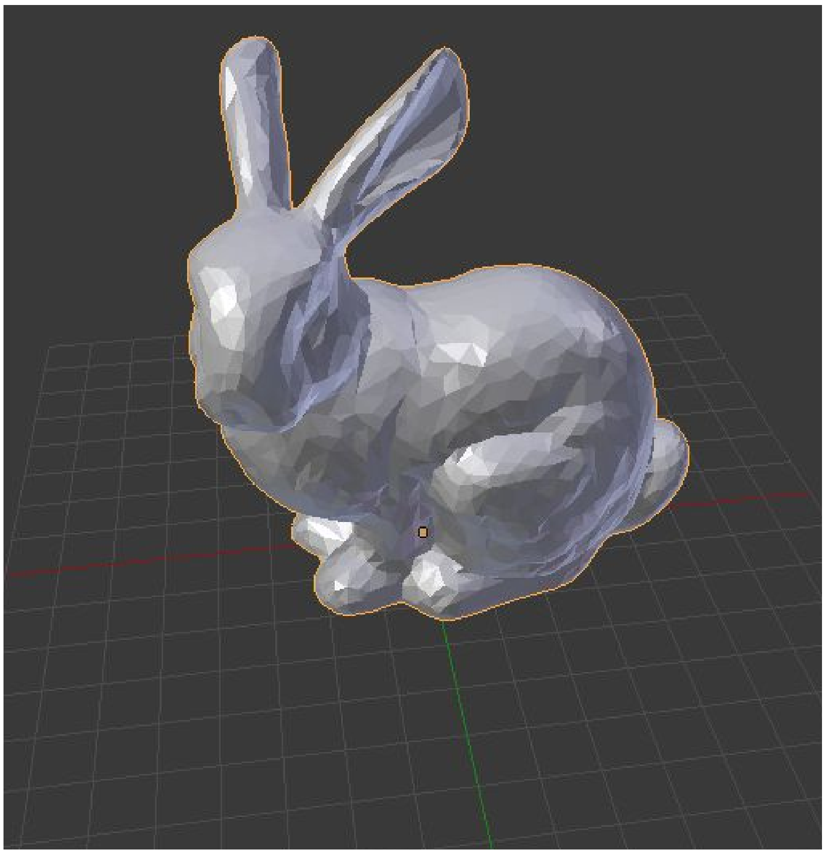

Figure 1: Triangular bunny mesh

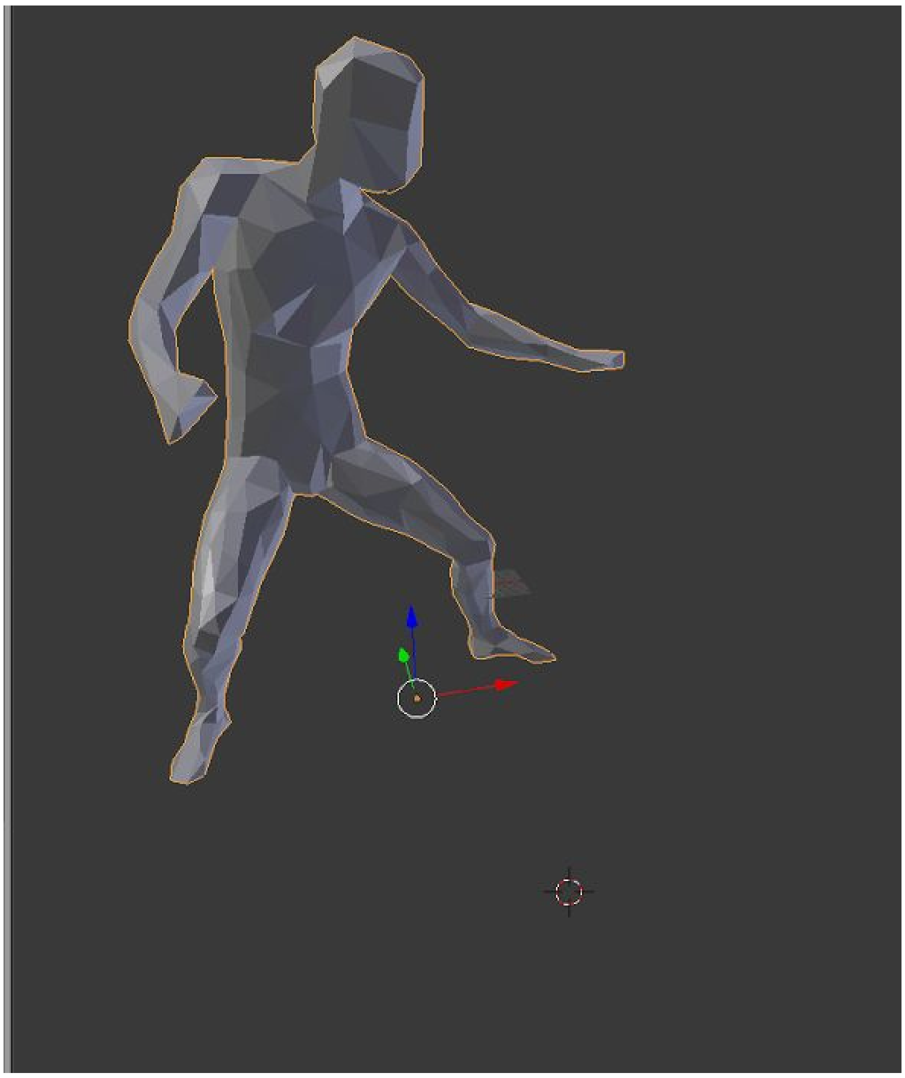

Figure 2: Triangular man mesh

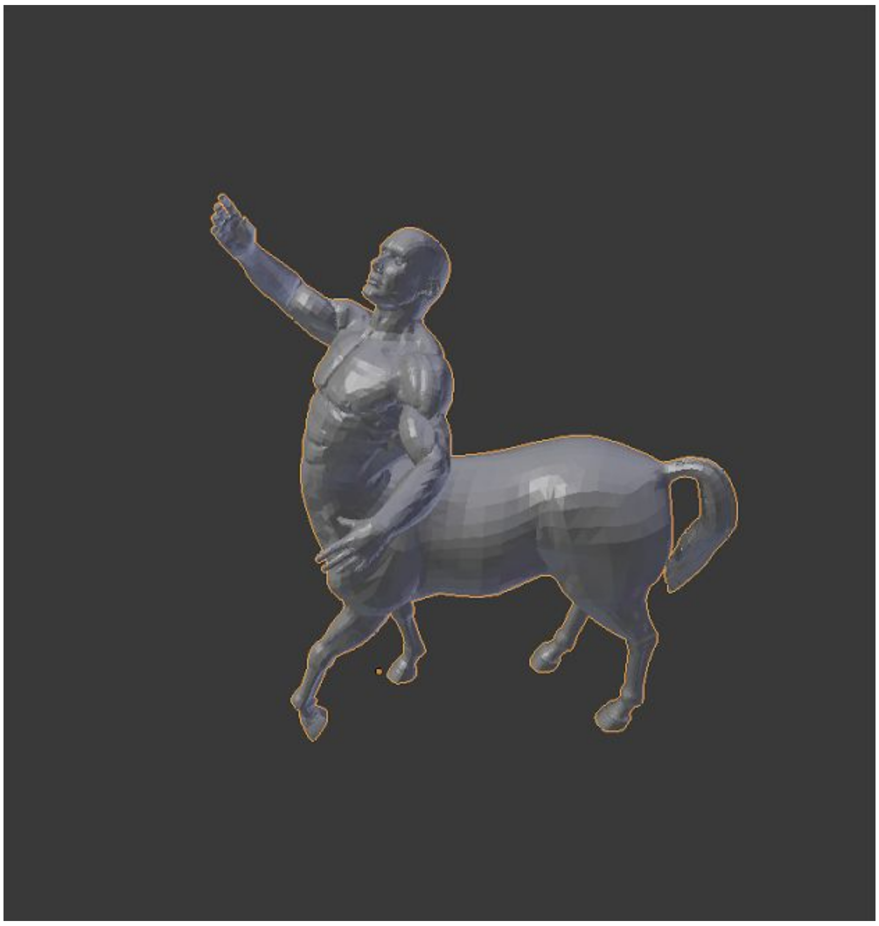
Figure 3: Triangular centour mesh

4.4.3 Outcome(s)

Resulting manipulated mesh displayed on 3D view, including corresponding displaying method, such as mesh coloring, sphere adding etc.

For Average Geodesic Distance, Curvature and Fathest Point Sampling algorithms, numeric values are returned, though in order to inspect these better, we used mesh coloring and adding sphere to resulting points methods are used. The outcoming visuals are shown below for each algorithm.

- Average Geodesic Distance

Mesh coloring according to the ranges of average geodesic distance values, which highlights the differences between the densities of vertices along the mesh, is used for a better inspection of the outcome.

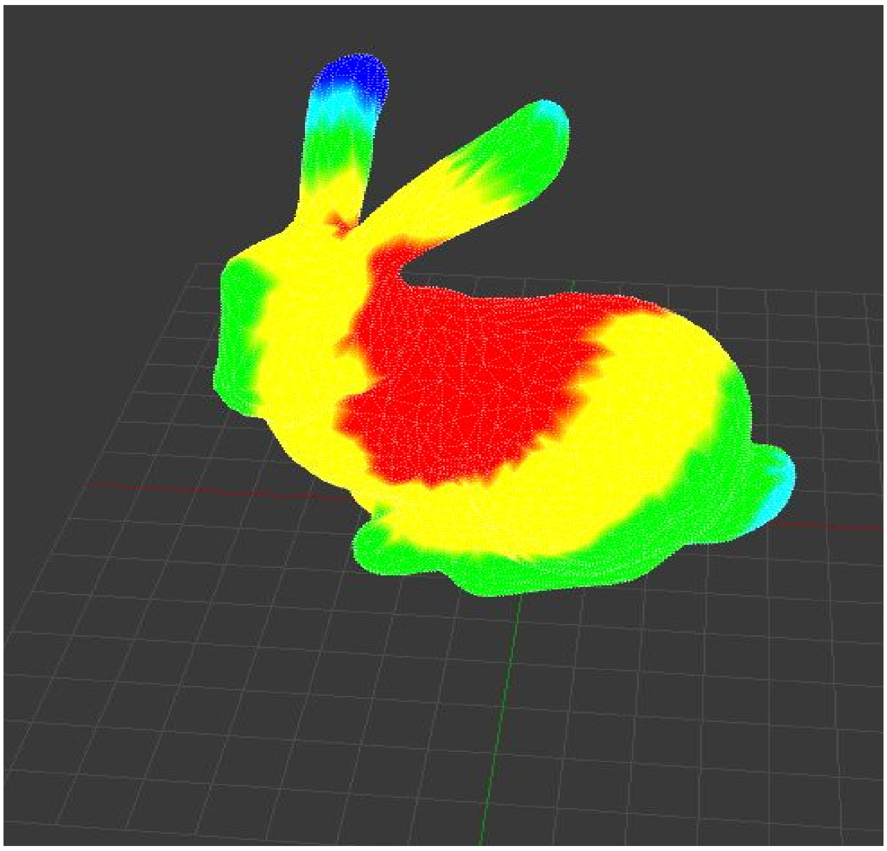

Figure 4: Result bunny mesh of average geodesic distance algorithm

- Biharmonic Distance

Only numeric value is returned and tested if they stay in an expected range for several mesh inputs.

- Curvature

Mesh coloring according to the ranges of curvature values, which highlights the differences between most and least curved faces of the mesh, is used for a better analysis of the outcome.

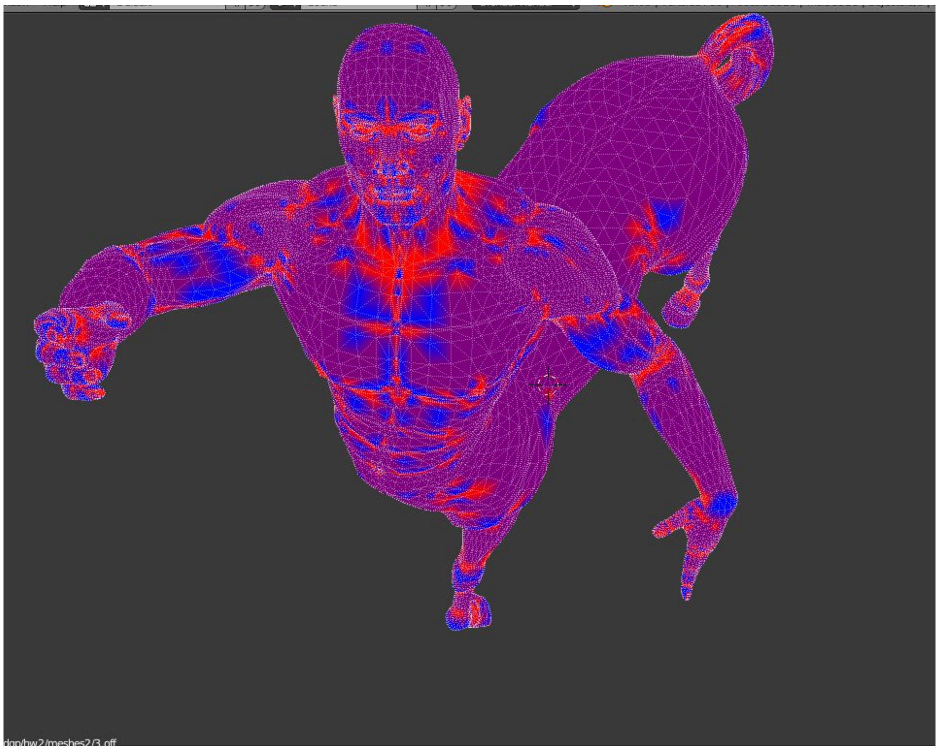

Figure 5: Result 1 man mesh of curvature algorithm

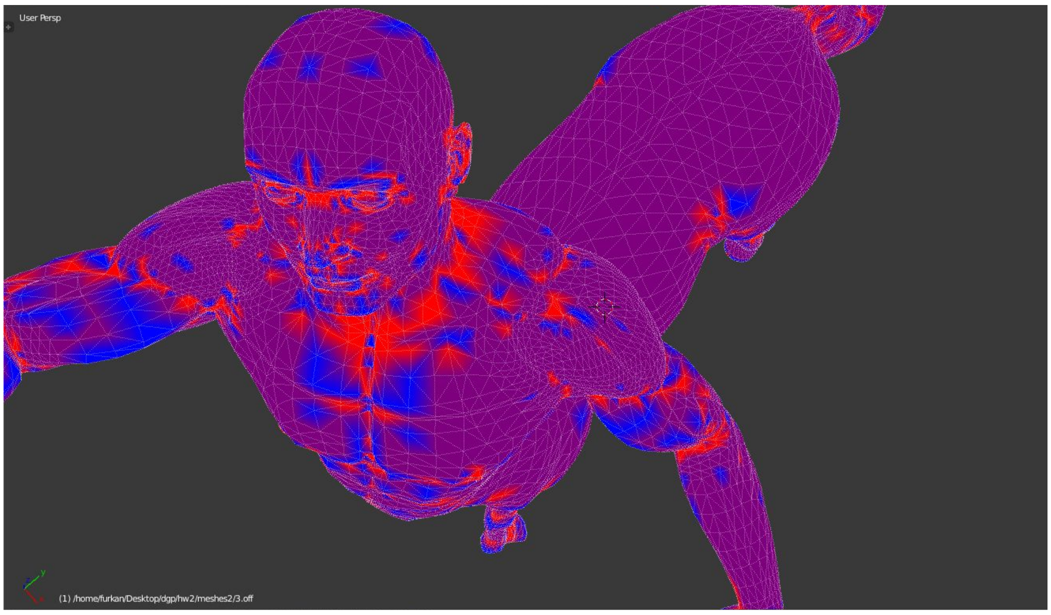

Figure 6: Result 2 man mesh of curvature algorithm

- Dijkstra

Only numeric value is returned and compared with expected outcomes of famous meshes.

- Farthest Sampling Point

  Sphere meshes are added at the points, which are found as the farthest sampling points, to be able to visually receive the outcome of the algorithm. The number of points is identified as a paramter of the algorithm’s own function, which will be determined by the user in his/her own development, as you can spot the difference between the two following output meshes.

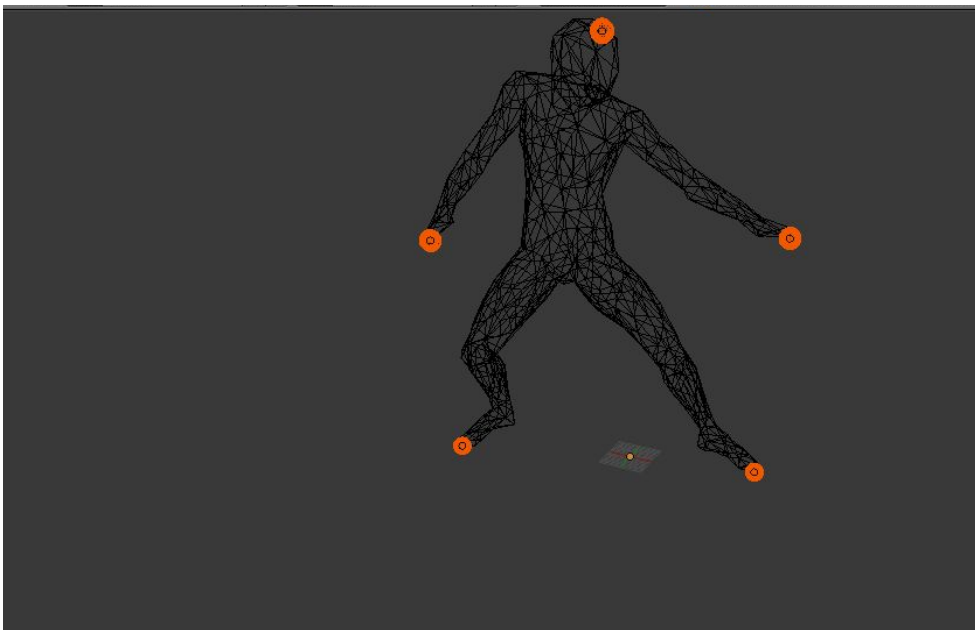

Figure 7: Result man mesh of farthest sampling point algorithm

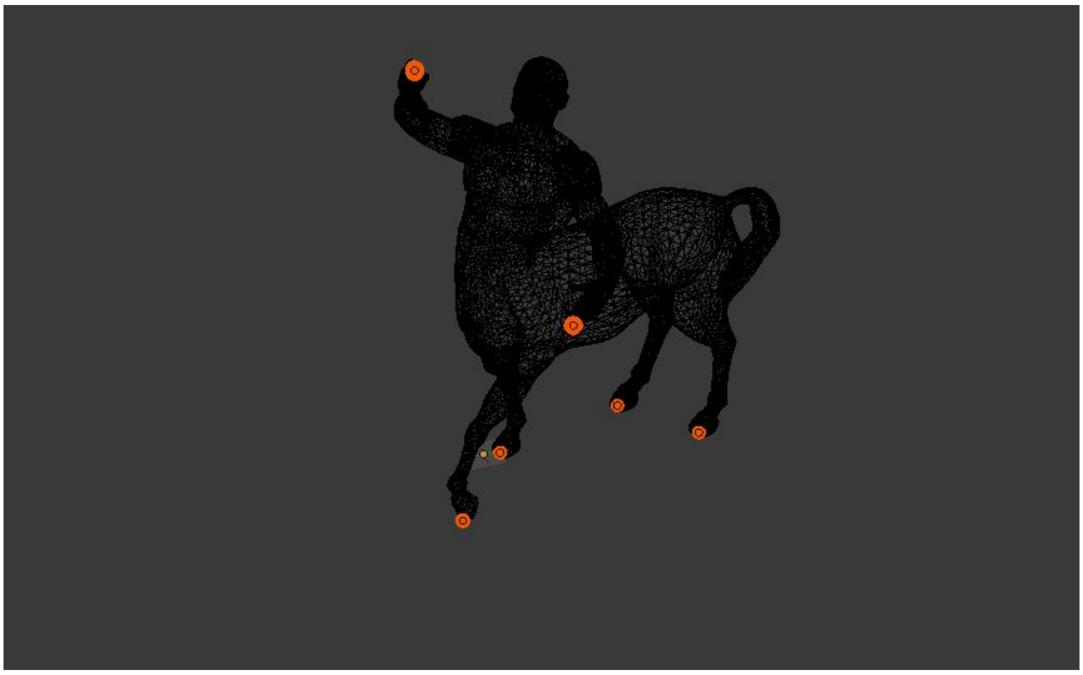

Figure 8: Result centour mesh of farthest point sampling algorithm

- Geodesic Distance

Numeric value is returned and cross-checked with expected values for famous meshes.

- Iterative Closest Point

Visual inspection is used to see if matching meshes connect to each other, such as using two of the same bunny meshes.

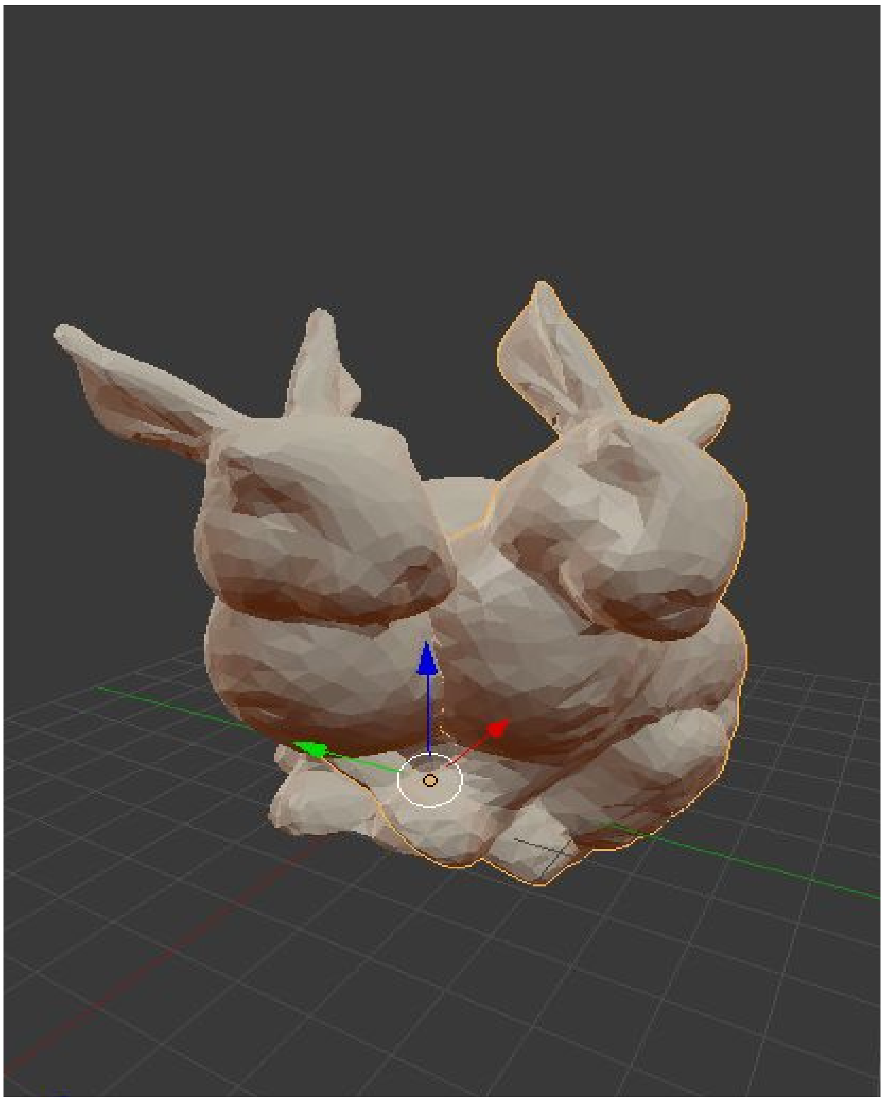

Figure 9: Result bunny mesh of iterative closest point algorithm

- Shape Diameter Function

Visual demonstration of the rays sent from the particular face to compute the Shape Diameter Function’s return value(numeric) is added.

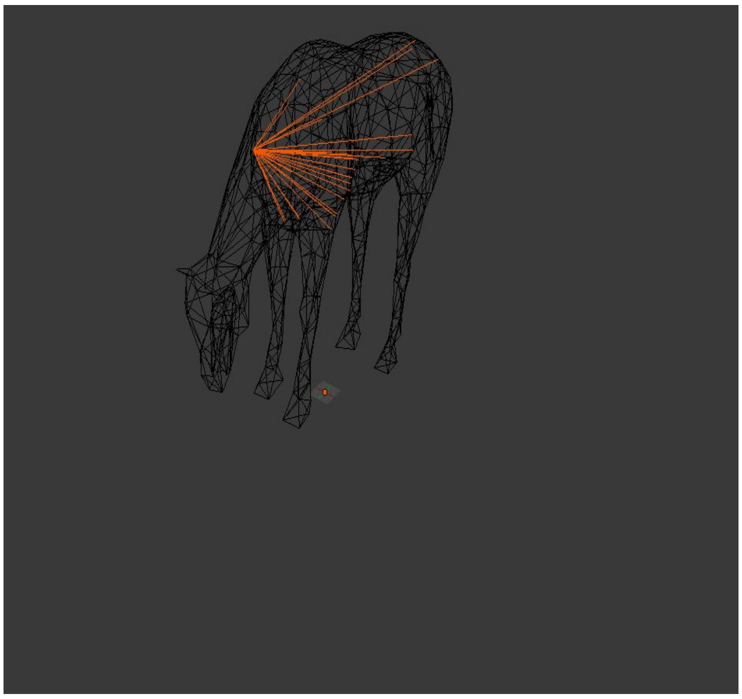
Figure 10: Result horse mesh of shape diameter function algorithm

4.4.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.4.5 Special procedural requirements

The user interface should be running.

4.5 Testing of Profiler

It is needed to be assured that the outcome of the profiler, which is a graphic estimating used memory versus the passed time, provides a correct one.

4.5.1 Objective

To make sure that profiler provides a reasonable graphical output. This is performed by comparing the resouces and time values returned by top command in terminal.

4.5.2 Input(s)

The script itself in the Code Editor.

4.5.3 Outcome(s)

Memory versus Time graphical result.

4.5.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.5.5 Special procedural requirements

The user interface should be running.

4.6 Testing of Debugger

Basic functionalities of any debugger, such as run, next, break etc., should be able to be performed.

4.6.1 Objective

It is expected that debugger meets the requirements and works in the way that every basic debugger does.

4.6.2 Input(s)

The script in the Code Editor.

4.6.3 Outcome(s)

The corresponding debug log to the called debug functions.

4.6.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.6.5 Special procedural requirements

The user interface should be running.

4.7 Testing of WYSIWYI environment

As WYSIWYI is the environment holding the registered parameters of the script and makes it possible to manipulate them easily, we need to make sure it performs these accordingly.

4.7.1 Objective

To check if all registered parameters appear in WYSIWYI environment and the result of mainpulating them in that environment provides the same outcome with manipulating them manually inside the script in the Code Editor.

4.7.2 Input(s)

Parameters registered by the developer in the script.

4.7.3 Outcome(s)

The alterable list of these registered parameters and when a chane is made, possibly corresponding visual change in 3D View or numeric value cahnge in result.

4.7.4 Environmental needs

The Meshtika application needs to be built correctly with its provided dependencies.

4.7.5 Special procedural requirements

The user interface should be running.

5 System test report details

5.1 Overview of test results

System passed all the tests conducted successfully by now.

5.2 Detailed test results

Code Editor	PASSED
Info Space	PASSED
Meshtika API (as the whole package)	PASSED
Average Geodesic Distance	PASSED
Biharmonic Distance	PASSED
Curvature	PASSED
Dijkstra	PASSED
Farthest Sampling Point	PASSED
Geodesic Distance	PASSED
Iterative Closest Point	PASSED
Shape Diameter Function	PASSED
Profiler	PASSED
Debugger	PASSED
WYSIWYI environment	PASSED

5.3 Rationale for decisions

Black box testing technique and visual inspection are used for the tests.

5.4 Conclusions and recommendations

In this document, definition of test cases is given so that making sure that system is working properly and meeting the requirements. The system is expected to pass all those test cases in order to be able to be ready to use.