**West Bohemia University**

Faculty of Applied Sciences

Department of Informatics and Computer Techniques

Semestral work for

# Graphics Systems and Data Visualization

Work number 5

# DisplaceMap

a set of modules for MVE2

Verification report

Jiří Skála
A01216
js.1@seznam.cz

# 1 Common tests of all modules

## 1.1 Module settings

Each module has some internal settings which may be modified in the module setup dialog. The aim of this test was to proof that module settings are shown correctly in the dialog window and that entered settings are accepted and stored by the module. Further attention was paid to verify that settings are correctly saved to XML file and later correctly loaded.

Every module was repeatedly set up to many various states and it was proved that module settings are always accepted, correctly saved as XML and loaded back.

## 1.2 Module duplicating

This test should verify module *DeepCopy* function. Each module was duplicated using *Ctrl+C*, *Ctrl+V* and it was proved that the new module has the same setting as its original.

Nevertheless some strange behaviour of newly created modules has been discovered. It seems that modules inserted using *Ctrl+V* are not properly integrated in the map. It was not possible to rename them and they caused confusion in module map levels so that the map couldn't be run. When the map was saved duplicated modules were not saved. This behaviour was observed with all modules even those that use base inherited *DeepCopy* method.

## 1.3 XML configuration parsing

Module configuration parsing should be sufficiently robust to accept even files not created by the module itself. That means parser must ignore unnecessary white characters, character cases etc. In case of error some meaningful message should be displayed.

Each module was tested to accept XML configuration reformed in some way. Modules can ignore keyword cases (this holds for configuration keywords only, not for XML tags), additional white characters and can process all valid number formats.

Another test was done with damaged configuration in XML files. There may be several types of such damage:

- Incorrect XML file structure
- Wrong configuration tag name
- Wrong tag attribute name or missing attribute
- Incorrect number or logical value format
- Incorrect configuration keyword

In case of incorrect XML file structure an exception is thrown already at lower layer. If there is a configuration tag missing then default values are set instead. For unknown tag (possibly typing mistake) a warning is printed into console window and tag is ignored. If file structure is OK but module configuration is unrecognizable an *MveException* is thrown with understandable message what went wrong.

## 1.4 Processing huge amount of data

There is no formal limit for amount of data that modules manage to process. A test was carried out with 1600×1200 image *large_1600x1200.jpg* passed to standard module pipeline defined in map *CalculateDisplace.xml*. After several seconds the processing was successfully done.

## 1.5 Extracting data attributes

Each tested module needs to work with some data attributes. The goal of this test is to check how modules behave when attributes are set incorrectly or missing. Input data may not contain any attribute of given name or the attribute may have different type than needed. In this case the module tries to find some other suitable color attribute. If it succeeds a message is printed with name of substitutive attribute. If no attribute of required type could be found an exception is thrown.

## 1.6 Consistency of produced data

All modules were tested to produce consistent data from various inputs. Testing map is available in *DataCheck.xml*. More tests were carried out while testing function of separate modules, see *Test*.xml* maps.

## 1.7 Common sense test

If we pass a greyscale image (*grayscale_80x60.png* for example) through the standard pipeline in *CalculateDisplace.xml* map with *Elevation* module set to greyscale color mapping we should get the same picture. Actually the result is slightly different because of deformations during cell-to-point color interpolation and renderer perspective projection.

# 2 *CalculateDisplace* module testing

## 2.1 Color processing

Correct color processing may be proved by *RGBA_80x60.tif* image. There are 4 color spots, one for each RGB color and one white. Please note that in the corners of the image there are 100% saturated colors but have some transparency. A test was performed by passing the image to standard *ClaculateDisplace.xml* map and modifying *CalculateDisplace* module settings. In the result one can see that color channels are masked correctly and alpha channel is supported.

## 2.2 Color interpolation

In order to get some small amount of data that may be verified by hand *RegGrid2DSource* and *RegGrid2DWatch* modules have been created. Using *TestCalcDispl.xml* map we can create some random input data and hand-check the output. All point colors were interpolated correctly including those lying at grid edges or in corners.

Further tests may be done on *CalculateDisplace.xml* map. Using *interpolate_10x10.png* image as input we should get a planar mesh at the end. The image consists of 3 chessboards and a 50% grey board. Each of the 4 segments should interpolate into uniform 50% intensity. Actually the result contains some artefacts in segment corners, where the interpolation of course yields different (but still correct) intensity. Interpolation effects may be studied more on *interp_effects_80x60.png* image.

## 2.3 Very small input data

We may use *TestCalcDispl.xml* map to check how the module deals with small data amounts which may be singular cases. Tests have shown that all meaningful data (down to 1×1 input) are processed correctly. In case of senseless input like $X\times0$ or $0\times Y$ module throws an exception with a message that explains the problem.

# 3 *DisplaceMap* module testing

## 3.1 Regular 2D grid displace mapping

Tests were carried out with *TestDisplMapGrid.xml* map. Input regular 2D grid was generated by *RegGrid2DSource*. Created triangle mesh point coordinates were checked by hand using *RegGrid2DWatch* and *TriMeshWatch*. It was proved that $x$ and $y$ coordinates are computed correctly according to grid size and position settings. Also $z$ coordinates equals to point elevation multiplied by elevation factor. Triangle mesh structure may be validated visually using a wireframe display in *TriMeshRenderer*.

Of course some singular cases may occur. If the elevation factor is zero a planar triangle mesh is generated with $z$ point coordinates set to zero. In case of zero size setting resulting triangle mesh degrades into a plane (for $X\times0$ or $0\times Y$ size) or into a line (for $0\times0$ size) due to point overlaying. But it still remains consistent. If the input grid has $X\times0$ or $0\times Y$ cells the triangle mesh consists of a linear sequence of points without any triangles. If the input grid has $0\times0$ cells the triangle mesh has only a single point. With respect to *DataChecker* such mesh without any triangles is still consistent.

## 3.2 Triangle mesh displace mapping

This test was performed with *TestDisplMapMesh.xml* map. Sample data were loaded from *triMeshSample.tri* and *TriMeshAttrAdder* module was used to add point attributes and discard $z$ coordinates. Correct displace mapping was proved by hand using *TriMeshWatch* modules. If the elevation factor is set to zero new triangle mesh is created planar.

# 4 *Elevation* module testing

## 4.1 Point elevation computing

Tests were performed in *TestElevation.xml* map. Sample triangle mesh was loaded from *triMeshSample.tri*. Module was tested both with internal elevation vector setting and with external *VectorSource* connected. Results were checked by hand using *TriMeshWatch* module output.

The elevation vector should define a direction in which point elevation will be computed. When the vector is zero it defines no direction. In this singular case all point elevations are set to the same level which is established to be the lowest one. All points are then painted with one color corresponding to the lowest elevation. This was validated on *triMeshPlanar.tri* mesh where all points lie in one plane. It is defined by the equation $-1.5x - y + 4z - 4 = 0$. With elevation vector set to (–1.5; –1; 4) all points have the same level of elevation and are painted in the same color.

## 4.2 Color mapping

Point color mapping was tested in *TestElevation.xml* map. Colors may be roughly checked visually using *TriMeshRenderer*. Exact verification is possible with *TriMeshWatch* module. However this is quite a challenge because it is necessary to find point elevation minimum and maximum, calculate the elevation of examined point and solve the color mapping scheme. As long as point elevation computing is proved to be correct, it is better to verify color mapping on some trivial case. Therefore verification was done on *triMeshSample.tri* mesh with elevation vector set to (0; 0; 1).

# 5 Conclusion

One significant error has been discovered during testing. *DisplaceMap* module computed wrong point coordinates when set to preserve input grid aspect ratio. The issue was fixed successfully. There were some more minor bugs concerned mainly with XML configuration saving / loading. All of them were fixed.